

LOWER BOUNDS ON THE COMPLEXITY OF SOME OPTIMAL DATA STRUCTURES*

MICHAEL L. FREDMAN†

Abstract. A technique is presented for deriving lower bounds on the complexity of optimal data structures which permit insertions and deletions of records, and queries of the form

$$\text{Query (Region);} \quad \text{Return } \sum_{\substack{\text{key}(r) \\ \in \text{Region}}} \text{value}(r) \quad (\text{Region} \in \Gamma),$$

where $\text{value}(r)$ (the value associated with a record r) lies in a commutative semi-group S , and Γ denotes a set of regions of the space of possible keys. This technique is illustrated with several examples.

Key words. Data structures, range queries, complexity, algorithms, lower bounds

1. Introduction. A widely studied area of concrete computational complexity concerns the existence of efficient data structures for dynamically maintaining a set of records T and permitting various types of range queries to be performed. Recently, a number of sophisticated data structures and general design techniques have been developed (see the References for a representative listing) leading in some instances to very good upper bounds. This paper attempts to provide a theoretical framework for the derivation of lower bounds on the inherent complexity of these kinds of computational problems.

Given a record r , we let $\text{key}(r)$ denote the key associated with r . We assume that the keys of records are members of a space of keys K . We refer to a subset of K as a region. We let $\text{value}(r)$ denote the value associated with a record r , which is assumed to lie in a commutative semi-group S (set of elements closed under a commutative and associative addition operation). Let Γ denote a set of regions of K . A *range query problem*, defined by a pair (K, Γ) , concerns the design of a data structure for representing a set of records T and facilitating efficient implementation (on-line) of the following computational tasks:

- Insert (k, x) ; insert a record r , with $\text{key}(r) = k$ and $\text{value}(r) = x$, into T (assuming T does not contain a record whose key is k). ($k \in K$).
- Delete (k) ; remove from T the record r with $\text{key}(r) = k$, should one be present. ($k \in K$).
- Query (R) ; compute the sum of the values associated with all records in T whose keys lie in the region R . ($R \in \Gamma$).

We concern ourselves only with data structures and manipulation algorithms which are structurally and functionally independent of the semi-group S . Hence, the pair (K, Γ) suffices to specify uniquely a range query problem.

The choice of the semi-group S depends on the intended purpose of a data structure. For example, if a query is used to count the number of records whose keys lie in a specified region, then S would be chosen to be the integers with the usual addition operation. If a query is used to list the records whose keys lie in a specified region, then S would be chosen to be a collection of sets of records under the operation of set union. We assume that the manipulation algorithms of a data structure interpret the operation

* Received by the editors April 30, 1979, and in final form March 11, 1980.

† Department of Electrical Engineering and Computer Science, University of California at San Diego, La Jolla, California 92093. This research was supported in part by the National Science Foundation under grant MCS 76-08543.

symbol “+” appropriately within the context of the choice for S . Similarly, the symbol “=”, which denotes equality testing between two registers storing values in S , is assumed to be appropriately interpreted. No other operations involving elements of S , apart from input, output and assignment statements, are permitted; consistent with our assumption concerning functional independence from S . (A precise model of computation is provided in § 2.)

A popular range query problem, referred to as the orthogonal range query problem, is defined by choosing K to be the d -dimensional vector space over the reals and Γ to be the set of hypercubes which can be expressed as a cross-product of d intervals, one in each dimension. In [4] we have derived a lower bound for this problem, showing that for each data structure and for each n there exists an intermixed sequence of n manipulations (insertions, deletions, and queries) having complexity $\Omega(n(\log n)^d)$. This result is best possible, as there exist data structures (see [8] and [9]) which achieve an $O(n(\log n)^d)$ upper bound. The method used in [4] can be regarded as an application of the general lower bound technique described in this paper (§ 3). Three new applications of the technique are provided in § 4. In each of these applications, K denotes the Euclidean plane (vector space of dimension two over the reals). In the first application, Γ consists of the set of all half-planes. In the second application, Γ consists of all circular regions (of the form $\{(x, y) | (x - a)^2 + (y - b)^2 \leq d\}$). In the third application, Γ consists of all parabolic regions of the form $\{(x, y) | y \leq a - (x - b)^2\}$. We derive an $\Omega(n^{4/3})$ lower bound on the inherent worst case complexity of a sequence of n manipulations in each of these three cases. This bound is significant in that the average cost $\Omega(n^{1/3})$ per manipulation is much larger than any fixed power of $\log n$. Hence, the orthogonal range query problem is intrinsically much easier than these other problems.

In § 5 we discuss a class of data structure problems closely related to the range query problems. Besides being of interest in their own right, they shed considerable light on the proof technique used to analyze the range query problems.

2. Computational model. A data structure will utilize registers, v_0, v_1, v_2, \dots , which store elements in the commutative semi-group S . The allowable operations on these registers are $v_i := v_j + v_k$; $v_i := cv_j$, where c is a positive integer constant; OUTPUT v_i ; and $v_i := \text{INPUT}$. Since we require that a data structure work correctly, independently of the choice for S , no other kinds of operations, tests, etc. on the v_i are allowed. (We could allow equality testing between two registers, v_i and v_j , but choose not to in favor of keeping the exposition of the computational model concise. This omission does not affect the validity of our subsequent theorems or their proofs.) A data structure will have an associated set of control states, Z . At any given instant, the structure will be in a control state in Z . When performing an insertion of a record having a specified key k , and an associated value $x \in S$, we assume that x is placed in v_i by means of the operation $v_i := \text{INPUT}$. The effect of an insertion results in a change of the control state, and the execution of a sequence of operations on the v_i registers. The resulting control state q and the choice of the operation sequence σ are a function $(q, \sigma) = F_I(p, k)$ of the current control state p and the key k . When performing a deletion of the record whose key is k , a change in control states takes place and an operation sequence on the v_i is performed. The resulting state q and the operation sequence σ are function $(q, \sigma) = F_D(p, k)$ of the current state p and k . When performing a query specified by a region $R \in \Gamma$, a (possible) change in control state takes place, and an operation sequence on the v_i is performed. The resulting state q and the sequence σ are given by $(q, \sigma) = F_O(p, R)$. (If no record lies in R , then no output will be generated by the operation sequence σ .) We assume that there is an initial state p_0 which corresponds to the empty set.

(As an example, consider the following data structure for one dimensional range queries, which involves a balanced binary tree scheme with records stored in the leaves. Each internal node has a key field in which a discriminator between the left and right subtrees is stored; as well as a value field v_i , in which gets stored the sum of the semi-group values of the records in the leaves beneath the node. A control state consists of a tree with specified keys stored in the key fields of its nodes, and a specified correspondence between the value fields of its nodes and the v_i registers. Insertions and deletions structurally alter the tree. The resulting operation sequences are chosen to appropriately update the quantities stored in the semi-group value fields of the nodes, taking into account rotations, etc. of the tree structure.)

We define complexity to be the number of operations performed on the v_i registers, not charging for the cost of computing the functions F_b , F_D , and F_Q . This measure underestimates "real" complexity, but for the purpose of proving lower bounds, this fact is of no consequence.

A data structure and its algorithms are defined by specifying functions F_b, F_D, F_Q . It can be shown (we leave the proof to the reader) that particular functions F_b, F_D , and F_Q correctly implement their intended semantics, as defined by (K, Γ) , for every commutative semi-group S ; if and only if they are correct when S is chosen to be the integers under the usual addition operation.

3. Lower bound technique. Let G be a graph with vertex set $V(G) = U \cup R$ where $U = \{u_1, u_2, \dots, u_m\}$ and $R = \{r_1, r_2, \dots, r_l\}$, and an edge set $E(G)$ which is a subset of $U \times R$, so that G is bipartite. A complete bipartite subgraph P of G is any subgraph of G satisfying $E(P) = (V(P) \cap U) \times (V(P) \cap R) \subseteq E(G)$; i.e., for each pair of vertices u_i and r_j in $V(P)$, the edge (u_i, r_j) is in both $E(P)$ and $E(G)$. Our method for deriving lower bounds is based on the following theorem.

THEOREM 1. *Let (K, Γ) be a range query problem and let n be a positive integer. Let k_1, \dots, k_m be keys in K and R_1, \dots, R_m be regions in Γ , $m \leq n/3$. Let G be the bipartite graph with $V(G) = \{u_1, \dots, u_m\} \cup \{r_1, \dots, r_m\}$ and $E(G) = \{(u_i, r_j) | 1 \leq i, j \leq m \text{ and } k_i \in R_j\}$. Assume that each edge $e \in E(G)$ has a nonnegative weight w_e , and that these weights satisfy*

$$\sum_{e \in E(P)} w_e \leq |V(P)|,$$

for each complete bipartite subgraph P of G . Then for each data structure which solves (K, Γ) , there exists a sequence of n manipulation operations which, when executed from the initial state p_0 representing the empty set, has total complexity at least $\sum_{e \in E(G)} w_e/4$.

The proof of Theorem 1 uses the following lemma.

LEMMA 1. *Let G be a bipartite graph of the type described at the beginning of this section, and let $A_1, \dots, A_m, B_1, \dots, B_l$ be sets satisfying*

$$(1) \quad A_i \cap B_j \neq \emptyset \quad \text{if and only if} \quad (u_i, r_j) \in E(G).$$

Let $w_e, e \in E(G)$, be nonnegative weights associated with the edges of G which satisfy

$$(2) \quad \sum_{e \in E(P)} w_e \leq |V(P)|,$$

for each complete bipartite subgraph P of G . Then

$$\sum_{e \in E(G)} w_e \leq \sum_{i=1}^m |A_i| + \sum_{i=1}^l |B_i|.$$

Proof. Let $A_1 \cup \dots \cup A_m \cup B_1 \cup \dots \cup B_l = \{x_1, x_2, \dots, x_t\}$. Let P_i , $1 \leq i \leq t$, denote the complete bipartite subgraph of G with vertices $V(P_i) = \{u_j | 1 \leq j \leq m \text{ and } x_i \in A_j\} \cup \{r_j | 1 \leq j \leq l \text{ and } x_i \in B_j\}$. Given an edge $e = (u_j, r_k) \in E(G)$, then $A_j \cap B_k \neq \emptyset$. If we choose i so that $x_i \in A_j \cap B_k$, then $e \in E(P_i)$. Hence

$$E(G) = \bigcup_{i=1}^t E(P_i).$$

Therefore,

$$(3) \quad \sum_{e \in E(G)} w_e \leq \sum_{i=1}^t \sum_{e \in E(P_i)} w_e.$$

Applying (2) to the inner sum in (3), we obtain

$$(4) \quad \sum_{e \in E(G)} w_e \leq \sum_{i=1}^t |V(P_i)|.$$

However,

$$(5) \quad \sum_{i=1}^t |V(P_i)| = \sum_{i=1}^m |A_i| + \sum_{i=1}^l |B_i|.$$

The proof is completed by combining (4) and (5).

Proof of Theorem 1. Choose S to consist of sets of pairs (k, i) , where k is a key and i is a positive integer, under the operation of set union. Assume that the semi-group value of the record having key k , when inserted for the i th time (having been deleted $(i-1)$ times), is $\{(k, i)\}$. Then the value returned by Query(R) is a set of pairs θ , and the set, $\{k | (k, i) \in \theta \text{ for some } i\}$, consists precisely of the keys which lie in R of the records currently in T (T is the set represented by the data structure). Without loss of generality, we may assume that the value initially stored in each v_i (in state p_0) is the empty set. We now argue that if an instruction to delete the record with key k is performed, then without loss of generality we may assume that at the commencement of this deletion instruction, each v_i whose contents contain a pair (k, j) for some j , is reset to ϕ (this resetting *not* regarded as an operation). This follows from the fact that under the stated assumptions, the value stored in each such v_i immediately prior to the deletion will never again be of any use; or stated differently, v_i cannot be used to advantage on the right hand side of an operation until after it has subsequently been reassigned a value. (Even if a record having key k is later inserted, its associated value $\{(k, j')\}$ will be different from the value $\{(k, j)\}$ associated with the record having key k just prior to its deletion.)

In any state p consistent with T having records whose keys precisely comprise $\{k_1, \dots, k_m\}$, let $B_j(p)$ denote the set of v_h whose values are summed to produce the output generated by Query(R_j) ($1 \leq j \leq m$), and let $A_i(p)$ denote the set of v_h whose values contain a pair of the form (k_i, l) , for some l ($1 \leq i \leq m$). Then

$$(6) \quad A_i(p) \cap B_j(p) \neq \emptyset \quad \text{if and only if} \quad k_i \in R_j.$$

(This follows from the manner in which $R_j \cap \{k_1, \dots, k_m\}$ can be deduced from the output generated by Query(R_j), as explained in the previous paragraph.) By the hypothesis of Theorem 1, Lemma 1, and (6), we conclude that for some j , $1 \leq j \leq m$, either

$$(7) \quad |A_j(p)| \geq \sum_{e \in E(G)} \frac{w_e}{(2m)} \quad \text{or} \quad |B_j(p)| \geq \sum_{e \in E(G)} \frac{w_e}{(2m)}.$$

We now construct, using the following algorithm, a sequence of at most n manipulations, which, when executed from state p_0 , require at least $\sum_{e \in E(G)} w_e/4$ operations.

- A. The first m manipulations are insertions to “load” into T records whose keys comprise $\{k_1, \dots, k_m\}$.
- B. Repeat the next step m times and stop.
- C. (Comment: After having executed from state p_0 the manipulations constructed thus far, the set of keys of the records in T is precisely $\{k_1, \dots, k_m\}$. With p denoting the state at this point, (7) implies that either $|A_j(p)|$ or $|B_j(p)| \geq \sum_{e \in E(G)} w_e/(2m)$ for some $j, 1 \leq j \leq m$.)

Let p denote the state resulting from having executed the manipulations constructed thus far. Find a j (should one exist) such that $1 \leq j \leq m$ and $|B_j(p)| \geq \sum_{e \in E(G)} w_e/(2m)$, and (C1) choose $\text{Query}(R_j)$ to be the next manipulation. Otherwise, find a j such that $1 \leq j \leq m$ and $|A_j(p)| \geq \sum_{e \in E(G)} w_e/(2m)$, and (C2) choose $\text{Delete}(k_j)$, $\text{Insert}(k_j, \{(k_j, l)\})$ (l chosen appropriately) to be the next two manipulations.

We now verify that the total number of operations t , involving the v_i registers, that are performed when executing the manipulations constructed by the above algorithm, exceeds $\sum_{e \in E(G)} w_e/4$. Let m_1 denote the number of iterations of step C in which the branch (C1) is followed. Then $t \geq m_1 \sum_{e \in E(G)} w_e/(2m)$. Let e denote the total number of times some v_i register with a nonempty value is reset to \emptyset as a consequence of performing deletions (as explained above), and let m_2 denote the number of times branch (C2) is followed. Then $e \geq m_2 \sum_{e \in E(G)} w_e/(2m)$, and so (since $m_1 + m_2 = m$)

$$(8) \quad t + e \geq \sum_{e \in E(G)} \frac{w_e}{2}.$$

Because all v_i initially have empty values, the number of times a v_i storing a nonempty value is reset to \emptyset cannot exceed the number of times it has appeared on the left-hand side of an operation. Therefore $t \geq e$. Combining this with (8) completes the proof of Theorem 1.

4. Applications. In this section we illustrate the use of Theorem 1 with several applications. These applications involve the Euclidean plane as a key space; we choose K to be $\{(x, y) | x, y \in \text{Reals}\}$. Given a data structure which solves (K, Γ) , we let C_n denote the worst case total complexity of an arbitrary sequence of n manipulations, implemented with the structure, and initiated from state p_0 representing the empty set.

4.1. Half-planes. In this application, our set of query regions consists of all half-planes of K ; we define $\Gamma_H = \{(x, y) | ax + by \geq c\} | (a, b) \neq (0, 0)\}$.

THEOREM 2. *A data structure which solves (K, Γ_H) has complexity $C_n = \Omega(n^{4/3})$.*

The following lemmas are required to prove Theorem 2.

LEMMA 2. *Let $\phi(j)$ denote Euler’s function: $\phi(j) = |\{i | 1 \leq i \leq j \text{ and } \gcd(i, j) = 1\}|$.*

Then

$$(a) \quad \sum_{j \leq m} \phi(j) = \frac{3}{\pi^2} m^2 + O(m \log m),$$

and

$$(b) \quad \sum_{j \leq m} j\phi(j) = \frac{2}{\pi^2} m^3 + O(m^2).$$

The formula (a) is a classical result of number theory (see [6, page 268]). The proof of (b) is only a slight modification of that of (a), given in [6].

Let $[m]^2$, $m \geq 1$, denote the set of m^2 points, $\{(i, j) | 1 \leq i, j \leq m\}$. Given a line l in the Euclidean plane, we refer to the number of points of $[m]^2$ through which l passes as the rank of l relative to $[m]^2$.

LEMMA 3. *There exist m^2 distinct lines in the Euclidean plane, the sum of whose ranks, relative to $[m]^2$, is $\Omega(m^{8/3})$.*

Proof. Let $l(i, j, a, b)$ denote the line passing through the points (i, j) and $(i + a, j + b)$. First, we show that the lines in $F_m = \{l(i, j, a, b) | 1 \leq a \leq \lfloor m^{1/3} \rfloor, 1 \leq i \leq a, 1 \leq j \leq m/2, 1 \leq b \leq a, \text{ and } \gcd(a, b) = 1\}$ are distinct. For suppose that $l(i, j, a, b) = l(i', j', a', b')$. Since $\gcd(a, b) = 1$ and $\gcd(a', b') = 1$, we conclude that $a = a'$ and $b = b'$ (otherwise the slopes, b/a and b'/a' , would differ). Since $(i, j) \in l(i', j', a', b')$ and $\gcd(a', b') = 1$, we conclude that $i' \equiv i \pmod{a}$, and therefore, since $1 \leq i, i' \leq a$, it follows that $i = i'$. With $i = i'$, $a = a'$ and $b = b'$, it must hold that $j = j'$.

The number of lines in F_m is given by $\lfloor m/2 \rfloor \sum_{a=1}^{\lfloor m^{1/2} \rfloor} a \phi(a) \leq m^2/\pi^2 + O(m^{5/3}) \leq m^2$ (when m is sufficiently large). The rank of $l(i, j, a, b)$ relative to $[m]^2$ is at least $\min(m/a, m/2b) \geq m/2a$. Hence, the sum of the ranks of the lines in F_m , relative to $[m]^2$, is at least $\lfloor m/2 \rfloor (m/2) \sum_{a=1}^{\lfloor m^{1/3} \rfloor} \phi(a) = \Omega(m^{8/3})$. By arbitrarily extending F_m to a family of m^2 lines, we end up with a collection of lines which satisfy the lemma.

Proof of Theorem 2. Let $A = \lfloor \sqrt{n/3} \rfloor$, so that $\lfloor [A]^2 \rfloor \leq n/3$. Choose keys k_1, \dots, k_m , $m = A^2$, so that $\{k_1, \dots, k_m\} = [A]^2$. Choose a set of $m = A^2$ lines, $\{l_1, \dots, l_m\}$, the sum of whose ranks relative to $[A]^2$ is $\Omega(A^{8/3}) = \Omega(n^{4/3})$ (the existence of which follows from Lemma 3). Let R_1, \dots, R_m be closed half-planes chosen so that the boundary of R_i is l_i , $1 \leq i \leq m$. Using the terminology of Theorem 1, applied to k_1, \dots, k_m and R_1, \dots, R_m , define w_e , $e = (u_i, r_j)$, as follows:

$$(9) \quad w_e = \begin{cases} 1 & \text{if } k_i \text{ lies on } l_j, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that $\sum_{e \in E(G)} w_e = \Omega(n^{4/3})$, in accordance with our choice for the lines l_1, \dots, l_m . Theorem 2, therefore, follows from Theorem 1 provided that $\sum_{e \in E(P)} w_e \leq |V(P)|$ for each bipartite subgraph P of G , which we now demonstrate.

We can associate with a complete bipartite subgraph P a polygonal region Π formed by the intersection of the half-planes associated with the r_i vertices in $V(P)$. The edges adjacent to a vertex u_i in $V(P)$ contribute a positive amount to $\sum_{e \in E(P)} w_e$ if and only if k_i lies on the boundary of Π . If k_i lies in the interior of a one-dimensional face of Π , then this contribution is 1 (only one edge adjacent to u_i has positive weight.) Now consider those vertices u_i in $V(P)$ for which k_i is a zero-dimensional face of Π . The edges (u_i, r_j) adjacent to u_i , which have positive weight, are of two types: (I) the line l_j associated with r_j contributes a one-dimensional face to Π , and (II) the line l_j associated with r_j has no contact with Π except at k_i . There are exactly two edges (u_i, r_j) adjacent to u_i of type (I). Moreover, the total number of edges of type (I), as u_i ranges over the vertices in $V(P)$ for which k_i is a zero-dimensional face of Π , is bounded by $|\{u_i | u_i \in V(P) \text{ and } k_i \text{ is a zero-dimensional face of } \Pi\}| + |\{r_j | r_j \in V(P) \text{ and } l_j \text{ contributes a one-dimensional face to } \Pi\}|$. The total number of edges of type (II) is bounded by $|\{r_j | r_j \in V(P) \text{ and } l_j \text{ intersects } \Pi \text{ in only one point}\}|$. Therefore, we conclude that $\sum_{e \in E(P)} w_e \leq |\{u_i | u_i \in V(P) \text{ and } k_i \text{ lies on the boundary of } \Pi\}| + |\{r_j | r_j \in V(P) \text{ and } l_j \text{ intersects } \Pi \text{ in at least one point}\}| \leq |V(P)|$. This completes the proof of Theorem 2.

4.2. Regions with curved boundaries. Given any half-plane H , there exists a circular region C_H such that $H \cap [m]^2 = C_H \cap [m]^2$. (We simply choose a circle with a sufficiently large diameter so that its boundary is well approximated by its tangent lines.) This observation yields the following corollary to Theorem 2.

COROLLARY 1. Let $\Gamma_c = \{(x, y) | (x - a)^2 + (y - b)^2 \leq d\} | a, b \text{ arbitrary, } d \geq 0\}$. A data structure which solves (K, Γ_c) has complexity $C_n = \Omega(n^{4/3})$.

Proof. The bipartite graph induced by our choice of keys and regions in the proof of Theorem 2, can be realized by picking, instead, regions of Γ_c , while retaining the same set $[A]^2$ of keys. The choice of regions is based on the mapping $H \rightarrow C_H$, mentioned above. Since the same graph can be realized, the same lower bound can also be derived.

The method used to prove Corollary 1 can be generalized as follows.

THEOREM 3. Let R be a subset of the plane, let C be an open disk, and assume that $C \cap R - \text{Boundary}(R)$ is open. Assume further than $C \cap \text{Boundary}(R)$ is given by $\{(x, f(x)) | a < x < b\}$ where $f(x)$ has a continuous derivative in (a, b) which is not constant. Let Γ_R contain R and all of its translations in the plane. Then any data structure which solves (K, Γ_R) has complexity $C_n = \Omega(n^{4/3})$.

Proof. Let s_1 and $s_2, s_1 \neq s_2$, be two values in the range of $f'(x)$. Let \bar{a} and \bar{b} be two nonzero vectors with slopes s_1 and s_2 respectively. Given $m \geq 1$, let $[m]^2(\bar{a}, \bar{b})$ denote $\{i\bar{a} + j\bar{b} | 1 \leq i, j \leq m\}$. The vectors \bar{a} and \bar{b} can be chosen to have sufficiently small magnitudes so that for each half-plane H , there exists a translate R_H of R such that $H \cap [m]^2(\bar{a}, \bar{b}) = R_H \cap [m]^2(\bar{a}, \bar{b})$. Lemma 3 easily generalizes to allow substitution of $[m]^2(\bar{a}, \bar{b})$ for $[m]^2$. The remainder of the proof of Theorem 3 closely follows that of Theorem 2 and Corollary 1.

A special case of Theorem 3 is given in the following corollary.

COROLLARY 2. Let $\Gamma_p = \{(x, y) | y \leq b - (x - a)^2\} | a, b \text{ arbitrary}\}$. (Γ_p contains regions having a parabolic shape.) Any data structure which solves (K, Γ_p) has complexity $C_n = \Omega(n^{4/3})$.

5. Semi-dynamic data structures. Our lower bound method of § 2 ultimately reduces to an analysis of the cardinalities of sets which satisfy the intersection conditions described in (1). In this section we define a computational problem whose analysis, relative to a particular computational model to be presented, is *equivalent* to an analysis of collections of sets satisfying (1).

Let S be an arbitrary commutative semi-group, and let $V = \{v_1, v_2, \dots, v_m\}$ be a set of variables which store values in S . Let T_1, T_2, \dots, T_n be finite subsets of $\{1, 2, \dots, m\}$. We consider data structures for representing V which facilitate efficient implementation of the following on-line tasks:

$$(10) \quad \begin{aligned} \text{Update}(j, x); & \quad v_j := v_j + x, & (x \in S, 1 \leq j \leq m), \\ \text{Retrieve}(j); & \quad \text{Return } \sum_{i \in T_j} v_i, & (1 \leq j \leq n). \end{aligned}$$

We can view (10) as a type of range query problem by means of the following correspondence. Let $K = \{1, 2, \dots, m\}$ and $\Gamma = \{T_1, \dots, T_n\}$. The set V corresponds to a set of records whose keys comprise K . The semi-group value associated with the record with key i is stored in v_i . The task Retrieve (j) is equivalent to the task Query (T_j) of the range query problem (K, Γ) . Instead of allowing insertions and deletions, the option of changing the value associated with a record is provided by implementing Update (j, x). Hence, the set of record keys remain static, but the values associated with records may be modified.

5.1. Computational model. We wish to consider straight line algebraic programs which work correctly, independently of the choice of the commutative semi-group S . With this in mind, we define the following model. Our data structures are to consist of registers Z_1, Z_2, \dots which store values in S . We consider programs which have the

following formats (U_j and R_j are finite subsets of $\{Z_1, Z_2, \dots\}$).

$$(11) \quad \begin{array}{ll} \text{Update } (j, x) & \text{Retrieve } (j) \\ \text{For each } Z_i \in U_j \text{ do} & \text{Return } \sum_{Z_i \in R_j} Z_i \\ Z_i := Z_i + x & \end{array}$$

We define the complexities of these programs to be the set cardinalities $|U_j|$ and $|R_j|$, respectively. The following lemma provides a condition on the sets U_j and R_j equivalent to correctness of the programs in (11).

LEMMA 4. *The programs in (11) are semantically equivalent to the programs in (10) if and only if*

$$(12) \quad |U_i \cap R_j| = \begin{cases} 1 & \text{if } i \in T_j, \\ 0 & \text{otherwise.} \end{cases}$$

Proof. Necessity. Consider an execution of the program sequence Retrieve (j), Update (i, x), Retrieve (j), and let W_1 and W_2 denote the two respective outputs. Then (10) implies that

$$(13) \quad W_2 = \begin{cases} W_1 + x & \text{if } i \in T_j, \\ W_1 & \text{otherwise.} \end{cases}$$

Choose S to be the set of integers and $x = 1$. Then (13) holds only if (12) holds.

Sufficiency. Because the Retrieve programs in (11) generate no side effects, it follows that (13) implies equivalence. But (13) holds if (12) holds, completing the proof.

The average complexity of the programs in (11) is given by

$$(14) \quad \frac{\left(\sum_{i=1}^m |U_i| + \sum_{i=1}^n |R_i| \right)}{(m+n)}.$$

Lower bounds on this average can be derived by making use of Lemma 1. Upper bounds can be derived by explicit construction. The average complexity defined in (14) loosely corresponds to the measure C_m/m associated with data structures which solve range query problems.

We illustrate the theory set forth in this section with a semi-dynamic variant of the parabola problem discussed in Corollary 2. Another example, which concerns a semi-dynamic variant of the orthogonal range query problem, appears in [4].

5.2. Parabolic regions of $[n]^2$. For the purpose of this example, we prefer to use a notation different from, but logically equivalent to that used in (10), namely, one based on the use of multiple subscripts, e.g.,

$$(15) \quad \begin{array}{ll} \text{Update } (i, j, x); & v_{ij} := v_{ij} + x, \quad ((i, j) \in [m]^2), \\ \text{Retrieve } (k, l); & \text{Return } \sum_{(i,j) \in T_{k,l}} v_{ij}, \quad ((k, l) \in [m]^2). \end{array}$$

Our problem is specified by setting $T_{kl} = \{(i, j) | (i, j) \in [m]^2 \text{ and } j \leq l - (i - k)^2\}$. We refer to this problem as the semi-dynamic parabola problem. Given a solution in the framework of (11), we define its average and worst case complexities to be, respectively,

$$C = \sum_{i,j=1}^m \frac{(|U_{ij}| + |R_{ij}|)}{2m^2} \quad \text{and} \quad \hat{C} = \max (\{|U_{ij}\} \cup \{|R_{ij}\})$$

(again, using double subscripts instead of single subscripts). Our results are the following.

THEOREM 4. *There exists a data structure within the framework of (11), which solves the semi-dynamic parabola problem, such that*

$$(16) \quad \hat{C} = O(m^{1/2}).$$

For every such data structure,

$$(17) \quad C = \Omega(m^{1/2}).$$

Proof. First, we prove the lower bound (17). Let G denote the graph with $V(G) = \{u_{ij} \mid (i, j) \in [m]^2\} \cup \{r_{kl} \mid (k, l) \in [m]^2\}$, and $E(G) = \{(u_{ij}, r_{kl}) \mid j \leq l - (k - i)^2\}$. For an edge $e = (u_{ij}, r_{kl}) \in E(G)$, define

$$(18) \quad w_e = \begin{cases} 1 & \text{if } j = l - (k - i)^2 \text{ (i.e., } (i, j) \text{ lies on the boundary of the parabolic region } T_{kl}), \\ 0 & \text{otherwise.} \end{cases}$$

By Lemma 4, $(u_{ij}, r_{kl}) \in E(G)$ if and only if $U_{ij} \cap R_{kl} \neq \emptyset$. Hence, if we can show that $\sum_{e \in E(G)} w_e \leq |V(P)|$ for each complete bipartite subgraph P of G , then Lemma 1 implies that $\sum_{e \in E(G)} w_e / (2m^2)$ is a lower bound for C . By (18), we have that

$$\begin{aligned} \sum_{e \in E(G)} w_e &= \sum_{\substack{(i,j) \in [m]^2 \\ (k,l) \in [m]^2 \\ l = j - (k-i)^2}} 1 = \sum_{1 \leq i, j \leq m} \sum_{\substack{1 \leq k \leq m \\ 1 \leq j - (k-i)^2 \leq m}} 1 \\ &\cong \sum_{\substack{1 \leq i \leq m/2 \\ m/2 < j \leq m}} \sum_{i \leq k \leq i + \sqrt{(m/2)}} 1 \cong \frac{m^2}{8} \sqrt{\left(\frac{m}{2}\right)}. \end{aligned}$$

Hence (17) follows once we have shown that $\sum_{e \in E(P)} w_e \leq |V(P)|$. We observe that the parabolic regions T_{kl} are convex, and that the parabolic boundaries of two such regions intersect in at most one point. This observation allows us to argue the inequality involving P in essentially the same way it was argued in the proof of Theorem 2.

Next, we prove the upper bound (16). In [5] a data structure referred to as a semi- (u, r) system is described. This data structure solves the problem in (10) specified by setting $m = n = \binom{u+r}{r} - 1$ and $T_j = \{1, 2, \dots, j\}$ for $1 \leq j \leq m$. A semi- (u, r) system falls within the framework of (11) and its maximum update complexity, $C_{\text{up}} = \max \{|U_i| \mid 1 \leq i \leq m\}$, is u ; and its maximum retrieve complexity, $C_{\text{re}} = \max \{|R_i| \mid 1 \leq i \leq m\}$, is r .

A data structure which solves the semi-dynamic parabola problem can be designed as follows. Organize each column $\{v_{i1}, v_{i2}, \dots, v_{im}\}$ of the v_{ij} variables in (15) as a semi- (u, r) system with $r = 2$ and $u = \lfloor \sqrt{2m} \rfloor$ (for a fixed column i , $T_j = \{(i, 1), (i, 2), \dots, (i, j)\}$, for $1 \leq j \leq m$). To perform Update (i, j, x) , perform Update (j, x) in the semi- (u, r) system corresponding to column i . The complexity of this task is no greater than $u = O(m^{1/2})$. To perform Retrieve (k, l) , perform Retrieve $(l - (k - i)^2)$ in the semi- (u, r) system corresponding to column i , for each i such that $1 \leq i \leq m$ and $l - (k - i)^2 > 0$; and sum the outputs generated. At most $2\sqrt{l+1} = O(m^{1/2})$ Retrieve commands within semi- (u, r) systems are performed when executing Retrieve (k, l) , each with complexity at most 2. Hence Retrieve (k, l) has complexity $O(m^{1/2})$. This establishes (16), completing the proof of Theorem 4.

REFERENCES

- [1] J. BENTLEY, *Decomposable searching problems*, Information Processing Letters 8, 5 (1979), pp. 244–251.
- [2] J. BENTLEY, J. FRIEDMAN, AND H. MAURER, *Two papers on range searching*, Report CMU-CS-78-136, Dept. of Computer Science, Carnegie-Mellon University (1978).
- [3] J. BENTLEY AND J. SAXE, *Transforming static data structures to dynamic data structures*, 20th Annual Symposium on Foundations of Computer Science (1979), pp. 148–168.
- [4] M. FREDMAN, *A lower bound on the complexity of orthogonal range queries*, J. Assoc. Comput. Mach., to appear.
- [5] ———, *On the complexity of maintaining an array and computing its partial sums*, J. Assoc. Comput. Mach., to appear.
- [6] G. HARDY AND E. WRIGHT, *The Theory of Numbers, Fourth Edition*, Oxford University Press, London, 1965.
- [7] D. KNUTH, *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
- [8] G. LUEKER, *A transformation for adding range restriction capability to dynamic data structures for decomposable searching problems*, UCI Technical Report no. 129 (1979).
- [9] D. WILLARD, *Predicate oriented data base search algorithms*, Research Report TR-20-78, Aiken Computation Laboratory, Harvard University (1978).

SOME NP-COMPLETE PROBLEMS SIMILAR TO GRAPH ISOMORPHISM*

ANNA LUBIW†

Abstract. The GRAPH ISOMORPHISM problem has so far resisted attempts at determining its complexity status—it has not been shown to be NP-complete nor in P. In this paper several altered or generalized versions of the ISOMORPHISM problem are presented and shown to be NP-complete. One of these is the problem of determining whether a given graph has a fixed-point-free automorphism. Some speculation is made on the possible implications of these results on deciding the complexity status of ISOMORPHISM. Various classes and hierarchies of problems in NP are discussed.

Key words. graph isomorphism, graph automorphism, NP-complete, isomorphism-complete, fixed-point-free automorphism

1. Introduction. The problem of determining whether two graphs are isomorphic—GRAPH ISOMORPHISM or simply ISOMORPHISM—has attracted considerable interest for both practical and theoretical reasons. Because any efficient algorithm for GRAPH ISOMORPHISM is of value in practical applications, much effort has gone into the search for polynomial time algorithms. Surveys of methods attempted can be found in [14] and [4]. In spite of the fact that no such algorithm has been found, ISOMORPHISM has not been shown to be NP-complete either, and there is some evidence that it is essentially different from the known NP-complete problems (see [1] or [12]). GRAPH ISOMORPHISM is of theoretical interest because of this uncertainty: the problem is in NP but no one has shown that it is in P or is NP-complete. A possible explanation for the lack of results is offered by Ladner's proof in [9] that if $P \neq NP$ then there are problems in NP which are neither in P nor NP-complete. ISOMORPHISM is a prime candidate.

Many problems have been shown to be polynomial-time equivalent to GRAPH ISOMORPHISM—these are called ISOMORPHISM-complete problems, and a list of them can be found in [2]. One approach to investigating the complexity of GRAPH ISOMORPHISM is to try to “map the boundary” between P and the class of ISOMORPHISM-complete problems by considering the ISOMORPHISM problem on restricted sets of graphs. For example, ISOMORPHISM restricted to bipartite graphs, to regular graphs, or to line graphs remains ISOMORPHISM-complete (see [2]), but polynomial algorithms have been found for ISOMORPHISM restricted to trees, or planar graphs [7], or interval graphs [11].

A second approach is to explore the boundary between ISOMORPHISM-complete and NP-complete problems. This can be done by finding two problems, one ISOMORPHISM-complete and the other NP-complete, which are related in a way that makes comparing them worthwhile, or by finding a chain of related problems containing such a pair. The most obvious example is to compare the NP-complete problem of SUBGRAPH ISOMORPHISM with its subproblem GRAPH ISOMORPHISM.

A second example involves LARGEST COMMON SUBGRAPH: given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, and an integer k , do there exist subsets $E'_1 \subseteq E_1$ and $E'_2 \subseteq E_2$ with $|E'_1| = |E'_2| \geq k$ such that the two subgraphs $G'_1 = (V_1, E'_1)$ and $G'_2 = (V_2, E'_2)$ are isomorphic? This problem is NP-complete (see [5]). GRAPH ISOMORPHISM is the subproblem with $|E_1| = |E_2| = k$.

* Received by the editors May 10, 1979, and in revised form November 27, 1979.

† Department of Computer Science, University of Toronto, Toronto, Canada M6S 2H3.

The third example is substantially different in that neither of the two problems is transparently an isomorphism problem—a rarity for ISOMORPHISM-complete problems. It was shown by Levi [10] and recently rediscovered by Kozen [8] that the CLIQUE problem restricted to a certain family of graphs is ISOMORPHISM-complete. Comparisons can be made between this and the general CLIQUE problem.

In this paper NP-completeness results will be proved for several problems similar to GRAPH ISOMORPHISM. Most of the problems are generalizations of a subproblem of ISOMORPHISM which is not known to be ISOMORPHISM-complete. This is the [GRAPH] AUTOMORPHISM problem: given a graph, does it have a nontrivial automorphism? AUTOMORPHISM-complete problems are those problems which are polynomial-time equivalent to GRAPH AUTOMORPHISM.

The observation that ISOMORPHISM is polynomially equivalent to the problem of determining whether a graph has an automorphism which does not fix a particular vertex leads to other generalizations. One is AUTOMORPHISM WITH RESTRICTIONS: given a graph $G = (V, E)$ and a set of restrictions $R \subseteq V \times V$, is there a nontrivial automorphism ψ of G which uses none of the restricted pairs, that is, an automorphism ψ such that if $(u, v) \in R$ then $\psi(u) \neq v$? Both this problem and the more general ISOMORPHISM WITH RESTRICTIONS are NP-complete, as will be shown. In fact, AUTOMORPHISM WITH RESTRICTIONS remains NP-complete when it is simplified to FIXED-POINT-FREE AUTOMORPHISM: given a graph, does it have a fixed-point-free automorphism? In this case the only restrictions on the automorphism are that it may not map any vertex to itself. NP-completeness is preserved under the further simplification to ORDER 2 FIXED-POINT-FREE AUTOMORPHISM: given a graph, does it have a fixed-point-free automorphism of order 2? One thing to note about this last result is that the corresponding problem for *groups* rather than graphs is solvable in polynomial time.

The second section of this paper consists mainly of the formal definitions and NP-completeness proofs for the above problems. In the following section some further results are presented. A chain of problems consisting of AUTOMORPHISM WITH RESTRICTIONS and some of its subproblems provides a nice characterization of the hierarchy of four (possibly equivalent) classes of NP shown in Fig. 1. A large class of

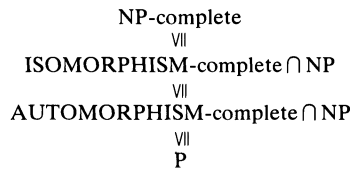


FIG. 1

problems containing FIXED-POINT-FREE AUTOMORPHISM is also given. The generalized problem of determining whether a graph of n vertices has an automorphism moving (i.e., *not fixing*) at least $\varepsilon \cdot n^{1/k}$ vertices is shown to be NP-complete for arbitrarily small fixed ε and arbitrarily large fixed k .

The final section of the paper contains a discussion of the differences between ISOMORPHISM and its various NP-complete generalizations, with the objective of explaining why ISOMORPHISM may not be NP-complete. Both SUBGRAPH ISOMORPHISM and LARGEST COMMON SUBGRAPH involve generalizing ISOMORPHISM in a way which allows *more* freedom in mapping the initial graph to the target graph, but the results in this paper show that ISOMORPHISM can also be made

hard—i.e., NP-complete—by *restricting* the possible maps from the initial graph to the target graph, as in ISOMORPHISM WITH RESTRICTIONS. The common factor in these two cases seems to be a destruction of the essential structure of the ISOMORPHISM problem. Perhaps this structure makes ISOMORPHISM easier than the NP-complete problems.

The notation “ $u \leftrightarrow v$ ” for an automorphism ψ of a graph $G = (V, E)$, with $u, v \in V$, will be used to mean “ $\psi(u) = v$ and $\psi(v) = u$ ”.

2. Generalizations of the AUTOMORPHISM problem. The AUTOMORPHISM problem and some NP-complete generalizations of it are defined in this section. The fact that AUTOMORPHISM WITH 1 RESTRICTION is ISOMORPHISM-complete has been part of the folklore for some time; a proof is included here for completeness.

AUTOMORPHISM.

Instance: a graph $G = (V, E)$.

Question: Does G have a nontrivial automorphism?

AUTOMORPHISM WITH 1 RESTRICTION.

Instance: a graph $G = (V, E)$ and a vertex $v \in V$.

Question: Does G have an automorphism ψ with $\psi(v) \neq v$?

PROPOSITION. AUTOMORPHISM WITH 1 RESTRICTION is ISOMORPHISM-complete.

Proof. Cook reducibility is used. To solve AUTOMORPHISM WITH 1 RESTRICTION in polynomial time using an ISOMORPHISM oracle:

Let $G = (V, E)$ and $v \in V$ be the given inputs to the problem. Let G_1 be a copy of G with a unique label attached to the vertex v . Take a second copy of G and attach the same label, in turn, to each of the vertices in $V \setminus \{v\}$. At least one of the resulting graphs is isomorphic to G_1 if and only if there is an automorphism ψ of G such that $\psi(v) \neq v$.

To solve ISOMORPHISM in polynomial time using an oracle for AUTOMORPHISM WITH 1 RESTRICTION:

Let G_1 and G_2 be the given graphs. Assume that they are connected (otherwise consider their complements). Uniquely label a vertex v of G_1 and give each of the vertices of G_2 , in turn, the same label, asking each time if the disjoint union of the two resulting graphs has an automorphism ψ satisfying $\psi(v) \neq v$. The answer will be “yes” at least once if and only if $G_1 \cong G_2$. \square

The problems ISOMORPHISM WITH RESTRICTIONS and AUTOMORPHISM WITH RESTRICTIONS both contain FIXED-POINT-FREE AUTOMORPHISM as a subproblem; so to prove that these three problems are NP-complete it is sufficient to prove that the last one is. The proof given will in fact support the claim that ORDER 2 FIXED-POINT-FREE AUTOMORPHISM is NP-complete (but note that this problem may not be a subproblem of the other three).

ISOMORPHISM WITH RESTRICTIONS.

Instance: two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, and a set $R \subseteq V_1 \times V_2$.

Question: Is there an isomorphism ψ of G_1 onto G_2 such that $\psi \cap R = \emptyset$?

(Here ψ is regarded in the formal sense, as a set of ordered pairs of vertices.)

AUTOMORPHISM WITH RESTRICTIONS.

Instance: a graph $G = (V, E)$ and a set $R \subseteq V \times V$.

Question: Does G have a nontrivial automorphism ψ such that $\psi \cap R = \emptyset$?

FIXED-POINT-FREE AUTOMORPHISM.Instance: a graph $G = (V, E)$.Question: Does G have a fixed-point-free automorphism?**ORDER 2 FIXED-POINT-FREE AUTOMORPHISM.**Instance: a graph $G = (V, E)$.Question: Does G have a fixed-point-free automorphism of order 2?**THEOREM.** FIXED-POINT-FREE AUTOMORPHISM is NP-complete.*Proof.* The problem is in NP.

THREE SATISFIABILITY (3SAT) will be used for the transformation. Let $U = \{u_1, \dots, u_n\}$ be the set of variables and $C = \{c_1, \dots, c_m\}$ be the set of clauses in an instance of 3SAT. Let $L = U \cup \{\bar{u}_i : u_i \in U\}$ be the set of literals. Any truth-value assignment $T : U \rightarrow \{t, f\}$ can be extended in the obvious way to L . Assume without loss of generality that the clauses of C are distinct and that each clause contains exactly three distinct literals. (The clauses may be padded with new variables to achieve this.) Let $c_i = \{q_i, r_i, s_i\}$, with $q_i, r_i, s_i \in L$.

A graph G must be constructed so that G has a fixed-point-free automorphism if and only if C is satisfiable. G will have two interconnected parts: a “truth-value setting component” and a “satisfiability component”. Fixed-point-free automorphisms of the truth-value setting component will correspond to truth-value assignments to L . The two components will be connected in such a way that the action of an automorphism of G on the satisfiability component is completely determined by its action on the truth-value setting component. The satisfiability component will contain vertices corresponding to the clauses in C , and an automorphism which fixes no points of the truth-value setting component will continue to fix no points of the satisfiability component if and only if the corresponding truth-value assignment satisfies all the clauses.

The truth-value setting component G_1 will contain the subgraph $G'_1 = (V'_1, E'_1)$ specified as follows:

$$V'_1 = \bigcup_{i=1}^n \{u_i(0), u'_i(0), u_i(1), u'_i(1), \bar{u}_i(0), \bar{u}'_i(0), \bar{u}_i(1), \bar{u}'_i(1), x_i, y_i\},$$

$$E'_1 = \bigcup_{i=1}^n \{(x_i, u'_i(1)), (u'_i(1), \bar{u}'_i(1)), (\bar{u}'_i(1), \bar{u}'_i(0)), (\bar{u}'_i(0), u_i(1)),$$

$$(u_i(1), y_i), (y_i, u'_i(0)), (u'_i(0), \bar{u}_i(1)), (\bar{u}_i(1), \bar{u}_i(0)), (\bar{u}_i(0), u_i(0)), (u_i(0), x_i)\}.$$

Figure 2 shows one component of G'_1 . To form G_1 , enough vertices and edges must be added to G'_1 to ensure that for each i , x_i and y_i can map only to each other. Attaching

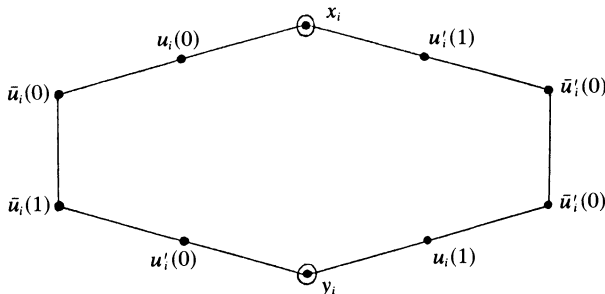


FIG. 2

one copy of a unique graph with no nontrivial automorphisms (an *asymmetric* graph) to x_i and one copy to y_i will accomplish this. These n “label” graphs must be asymmetric, so that the number of truth-value assignments to L is equal to the number of fixed-point-free automorphisms of the truth-setting component G_1 .

G_1 contains a vertex $s(0)$ for each literal $s \in L$, and vertices $s(1)$ and $s'(0)$ to which $s(0)$ can be mapped by fixed-point-free automorphisms of G_1 . A fixed-point-free automorphism ψ of G_1 and a truth-value assignment T to L will be said to *correspond* if, $\forall s \in L$,

$$s(0) \xleftrightarrow{\psi} s(1) \quad \text{iff} \quad T(s) = t;$$

and

$$s(0) \xleftrightarrow{\psi} s'(0) \quad \text{iff} \quad T(s) = f.$$

Any fixed-point-free automorphism ψ of G_1 must satisfy $\psi(x_i) \neq x_i$, $1 \leq i \leq n$. Then $x_i \xleftrightarrow{\psi} y_i$. For each i there are exactly two possibilities:

$$\begin{array}{ll} u_i(0) \xleftrightarrow{\psi} u_i(1) & u_i(0) \xleftrightarrow{\psi} u'_i(0) \\ \bar{u}_i(0) \xleftrightarrow{\psi} \bar{u}'_i(0) & \bar{u}_i(0) \xleftrightarrow{\psi} \bar{u}_i(1) \\ \text{OR} & \\ u'_i(0) \xleftrightarrow{\psi} u'_i(1) & u_i(1) \xleftrightarrow{\psi} u'_i(1) \\ \bar{u}_i(1) \xleftrightarrow{\psi} \bar{u}'_i(1) & \bar{u}'_i(0) \xleftrightarrow{\psi} \bar{u}'_i(1). \end{array}$$

The first possibility corresponds to $T(u_i) = t$, $T(\bar{u}_i) = f$, and the second to $T(u_i) = f$, $T(\bar{u}_i) = t$, so the fixed-point-free automorphism ψ of G_1 determines a unique corresponding truth-value assignment $T: L \rightarrow \{t, f\}$. Conversely, every such truth-value assignment determines a unique corresponding fixed-point-free automorphism of G_1 .

Note that $\forall s \in L$, $s(0) \xleftrightarrow{\psi} s'(0)$ iff $s(1) \xleftrightarrow{\psi} s'(1)$.

G_1 will now be expanded to the graph $G = \{V, E\}$ as follows:

$$\begin{aligned} V &= V_1 \cup \{c_i(j) : 1 \leq i \leq m, 0 \leq j \leq 7\} \\ &\quad \cup \{c'_i(j) : 1 \leq i \leq m, 0 \leq j \leq 7\}, \\ E &= E_1 \cup \{(c_i(j), c'_i(j)) : 1 \leq i \leq m, 0 \leq j \leq 7\} \\ &\quad \cup \bigcup_{i=1}^m \{(c_i(0), q_i(0)), (c_i(0), r_i(0)), (c_i(0), s_i(0)), \\ &\quad (c_i(0), q'_i(0)), (c_i(0), r'_i(0)), (c_i(0), s'_i(0)), \\ &\quad (c_i(1), q_i(0)), (c_i(1), r_i(0)), (c_i(1), s_i(1)), \\ &\quad \vdots \\ &\quad (c_i(7), q'_i(1)), (c_i(7), r'_i(1)), (c_i(7), s'_i(1))\}. \end{aligned}$$

See Fig. 3. G can be constructed in polynomial time.

Note that if abc is the binary representation of j then $c_i(j)$ is connected to $q_i(a)$, $r_i(b)$, $s_i(c)$, and to their primed counterparts $q'_i(a)$, $r'_i(b)$, $s'_i(c)$. The vertices $c'_i(j)$ ensure that no automorphism of G can map a vertex $c_i(j)$ to a vertex of G_1 , so that if ψ is an automorphism of G then ψ restricted to G_1 is an automorphism of G_1 . Because of the

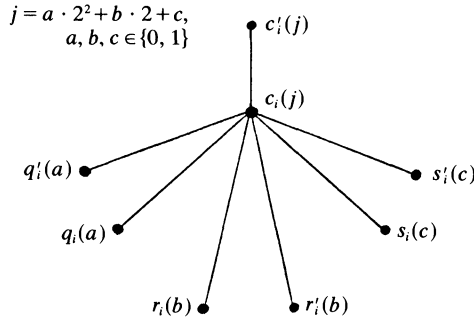


FIG. 3

assumptions that the c_i 's are distinct and that each c_i contains three distinct literals, any automorphism of G_1 can be extended in one and only one way to an automorphism of G .

Let ψ_1 and T be a corresponding fixed-point-free automorphism of G_1 and truth-value assignment to L , respectively. Let ψ be the extension of ψ_1 to G . Then T satisfies $c_i \in C$ if and only if $\psi(c_i(j)) \neq c_i(j)$, $0 \leq j \leq 7$, since:

$$\psi(c_i(j)) = c_i(j) \text{ for some } j \in \{0, \dots, 7\}, j = abc \text{ in binary representation,}$$

$$\text{with } a, b, c \in \{0, 1\}$$

$$\Leftrightarrow q_i(a) \xleftrightarrow{\psi} q'_i(a), r_i(b) \xleftrightarrow{\psi} r'_i(b), s_i(c) \xleftrightarrow{\psi} s'_i(c) \text{ for some } a, b, c \in \{0, 1\}$$

$$\Leftrightarrow q_i(0) \xleftrightarrow{\psi} q'_i(0), r_i(0) \xleftrightarrow{\psi} r'_i(0), s_i(0) \xleftrightarrow{\psi} s'_i(0)$$

$$\Leftrightarrow T(q_i) = T(r_i) = T(s_i) = f$$

$$\Leftrightarrow c_i \text{ is not satisfied.}$$

To complete the proof it must be shown that C is satisfiable if and only if G has a fixed-point-free automorphism.

Suppose C is satisfied by $T: L \rightarrow \{t, f\}$. There is a corresponding fixed-point-free automorphism of G_1 , which can be extended uniquely to an automorphism ψ of G . T satisfies each $c_i \in C$ so $\psi(c_i(j)) \neq c_i(j)$ for $1 \leq i \leq m$, $0 \leq j \leq 7$. Hence ψ is a fixed-point-free automorphism of G .

Conversely, suppose ψ is a fixed-point-free automorphism of G . Then ψ restricted to G_1 is a fixed-point-free automorphism of G_1 and has a corresponding truth-value assignment $T: L \rightarrow \{t, f\}$. T must satisfy each $c_i \in C$ since $\psi(c_i(j)) \neq c_i(j)$ for $1 \leq i \leq m$, $0 \leq j \leq 7$. Hence T satisfies C . \square

For the graph G constructed in the above proof, $\text{Aut } G$ is an Abelian 2-group, of order 2, with the set of generators consisting of the automorphisms ψ_i , $i = 1, \dots, n$, where $\psi_i(u_i(0)) = u_i(1)$, and $\psi_i(u_i(1)) = u_i(0)$, for $j \neq i$. The order partition of $\text{Aut } G$ is also easy to determine.

Since all fixed-point-free automorphisms of G have order 2, a consequence of the above proof follows.

COROLLARY 1. ORDER 2 FIXED-POINT-FREE AUTOMORPHISM is NP-complete. \square

In the proof, truth-value assignments satisfying C and fixed-point-free automorphisms of G correspond one-to-one (the transformation is *parsimonious*), so as a second consequence:

COROLLARY 2. *The problem of counting the number of fixed-point-free automorphisms of a graph is #P-complete.*

Proof. The problem of counting the number of satisfying truth-value assignments for an instance of THREE-SATISFIABILITY is #P-complete (see [15]). \square

The theorem also suffices to prove that any problem containing FIXED-POINT-FREE AUTOMORPHISM as a subproblem is NP-complete:

COROLLARY 3. ISOMORPHISM WITH RESTRICTIONS and AUTOMORPHISM WITH RESTRICTIONS are NP-complete. \square

The fact that AUTOMORPHISM WITH RESTRICTIONS is NP-complete can be proved independently. The proof is a more pleasing one and is outlined here. Once again 3SAT is used for the transformation. Let $U = \{u_1, \dots, u_n\}$ and $C = \{c_1, \dots, c_m\}$ be the variables and clauses of an instance of 3SAT as in the proof above. The graph G to be constructed will again have a truth-setting component G_1 , and a satisfiability component. The main changes are in G_1 which will be built up of copies of the graph S shown in Fig. 4.

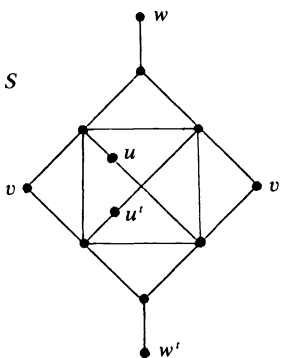


FIG. 4

If u, v, w are variables and S is labelled as shown, with vertices u, v, w, u', v', w' , then a truth-value assignment $T : \{u, v, w\} \rightarrow \{t, f\}$ and a (possibly trivial) automorphism ψ of S will be said to *correspond* if, $\forall x \in \{u, v, w\}$,

$$T(x) = t, \text{ iff } x \xleftrightarrow{\psi} x'.$$

S is an “exclusive-or gadget” in the sense that every automorphism of S corresponds to a truth-value assignment satisfying $u = v \oplus w$, where \oplus is the exclusive-or operator.

For the present purpose n copies of S are needed, one copy S_i , shown in Fig. 5, for each $u_i \in U$. Requiring that $\psi(x_i) \neq x_i$, for an automorphism ψ of S_i , forces $x_i \xleftrightarrow{\psi} y_i$, in turn forcing $u_i \xleftrightarrow{\psi} u'_i$ or $\bar{u}_i \xleftrightarrow{\psi} \bar{u}'_i$ but not both—that is, the corresponding truth-value assignment must set exactly one of u_i, \bar{u}_i true.

As before, the satisfiability component will contain 8 vertices $c_i(j)$, $0 \leq j \leq 7$, for each clause $c_i \in C$. If $c_i = \{q_i, r_i, s_i\}$, $c_i(0)$ will have edges to q_i, r_i, s_i in G_1 ; $c_i(1)$ will have edges to q_i, r_i, s'_i ; \dots ; $c_i(7)$ will have edges to q'_i, r'_i, s'_i . Additional vertices and edges are needed in G to structure it sufficiently enough that automorphisms of G take each S_i to itself.

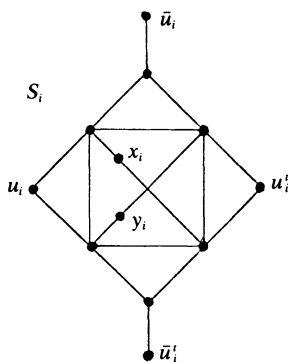


FIG. 5

The set of restrictions R is defined as follows:

$$R = \{(x_i, x_i) : 1 \leq i \leq n\} \cup \{(c_i(0), c_i(0)) : 1 \leq i \leq m\}.$$

By an argument similar to the one in the previous proof, we can show that C is satisfiable if and only if G has an automorphism ψ with $\psi \cap R = \emptyset$.

3. Extended classes of problems. Both AUTOMORPHISM WITH RESTRICTIONS and FIXED-POINT-FREE AUTOMORPHISM can be considered as single problems from larger classes of related problems with varying complexity standings.

AUTOMORPHISM WITH RESTRICTIONS has a range of subproblems from the hierarchy of four classes of NP shown in Fig. 1—NP-complete, ISOMORPHISM-complete \cap NP, AUTOMORPHISM-complete \cap NP, and P. Let $G = (V, E)$ and $R \subseteq V \times V$ be an instance of the problem.

If for some fixed number k , inputs to the problem are restricted to those for which $|E| \leq k$, then the problem is solvable in polynomial time. (In essence it becomes BIPARTITE MATCHING.)

If R is constrained to being the null set the problem is exactly automorphism.

If for some fixed number $k \geq 1$, R is restricted by $|R| \leq k$, the problem is ISOMORPHISM-complete.

In full generality the problem is NP-complete, and the corresponding counting problem is $\#P$ -complete.

FIXED-POINT-FREE AUTOMORPHISM also has a large range of related problems. For g a function $g: \mathbb{N} \rightarrow \mathbb{R}$ with $g(n) \leq n$, $\forall n \in \mathbb{N}$, define the problem:

g-FIXED-POINT-FREE AUTOMORPHISM.

Instance: a graph $G = (V, E)$ with $|V| = n$.

Question: Does G have an automorphism which moves (i.e., *does not fix*) at least $g(n)$ vertices?

This problem is in NP if g satisfies the requirement that the time needed to compute $\lceil g(n) \rceil$ is bounded by a polynomial in n for all $n \in \mathbb{N}$. Henceforth only functions satisfying this requirement will be considered.

When $g(n) = n$, $\forall n$, the problem is FIXED-POINT-FREE AUTOMORPHISM. The problem is trivial for $g \equiv 0$, and for $g(n) = 1$, $\forall n$, the problem is AUTOMORPHISM. If $g(n)$ is bounded by some constant $K \geq 1$, the problem is at least as easy as ISOMORPHISM and at least as hard as AUTOMORPHISM. The following results are more substantial.

THEOREM. *If $g(n) = \varepsilon \cdot n$ for any fixed ε , $0 < \varepsilon \leq 1$, g -FIXED-POINT-FREE AUTOMORPHISM is NP-complete.*

Proof. The problem is in NP.

FIXED-POINT-FREE AUTOMORPHISM will be used for the transformation. Let $G = (V, E)$, with $|V| = n$, be a graph given as an instance of **FIXED-POINT-FREE AUTOMORPHISM**. Assume that G is connected (otherwise consider the complement of G). Also assume that $n \geq 2$. If $n = 1$, G has no fixed-point-free automorphisms.

A new graph $G' = (V', E')$ with $|V'| = n'$ must be constructed so that there is an automorphism of G' moving at least $\varepsilon \cdot n'$ vertices if and only if G has a fixed-point-free automorphism. Let n' be any integer such that $n - 1 < n' \cdot \varepsilon \leq n$, i.e., $(n - 1)/\varepsilon < n' \leq n/\varepsilon$. Since $\varepsilon \leq 1$ this interval will contain at least one integer.

If $n' = n$, let $G' = G$.

If $n' = n + 1$, let $G' = (V', E')$ with $V' = V \cup \{v_1\}$, $v_1 \notin V$, and $E' = E$.

For all other cases let $G' = (V', E')$ with $V' = V \cup \{v_1, \dots, v_{n'-n}\}$, $v_i \notin V$, and $E' = E \cup \{(u, v_1) : u \in V\} \cup \{(v_i, v_{i+1}) : 1 \leq i \leq n' - n - 1\}$. See Fig. 6.

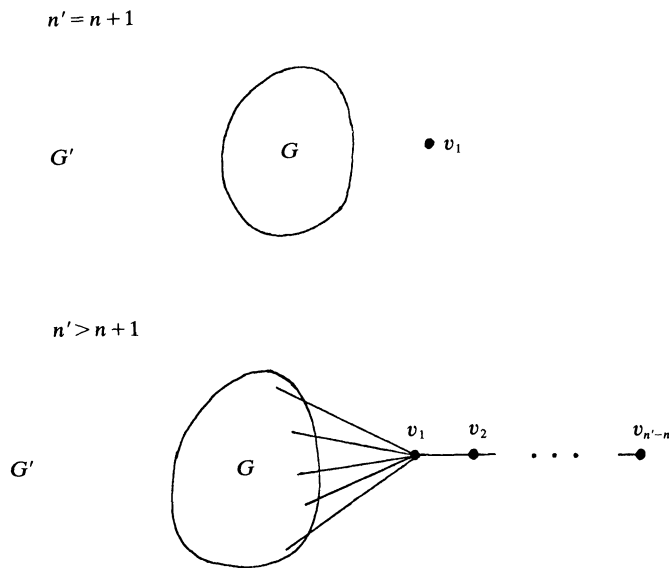


FIG. 6

G' can be constructed in polynomial time. Any automorphism ψ of G must fix the vertices $v_1, \dots, v_{n'-n}$, so ψ restricted to G must be an automorphism of G .

If G' has an automorphism ψ moving at least $n' \cdot \varepsilon$ vertices then, since $n' \cdot \varepsilon > n - 1$, ψ moves at least n vertices. But ψ must fix the $n' - n$ vertices $v_1, \dots, v_{n'-n}$. Hence ψ moves all n vertices of G , and ψ restricted to G is a fixed-point-free automorphism of G .

Conversely, if G has a fixed-point-free automorphism it can be extended to an automorphism ψ of G' fixing exactly the vertices $v_1, \dots, v_{n'-n}$. Then ψ moves n vertices and $n \geq n' \cdot \varepsilon$, so G' has an automorphism moving at least $n' \cdot \varepsilon$ vertices. \square

The class of functions for which g -FIXED-POINT-FREE AUTOMORPHISM is NP-complete can easily be expanded.

THEOREM. *If $g(n) = \varepsilon \cdot n^{1/m}$, for any fixed ε , $0 < \varepsilon \leq 1$, and fixed $m \in \mathbb{N}$, then g -FIXED-POINT-FREE AUTOMORPHISM is NP-complete.*

Proof. Using the same proof as above, the requirement for n' will be $n - 1 < (n')^{1/m} \cdot \varepsilon \leq n$; i.e., $[(n - 1)/\varepsilon]^m < n' \leq (n/\varepsilon)^m$. Since m and ε are fixed, n' is bounded

by a polynomial in n , so G' can be constructed (exactly as above) in polynomial time. \square

The next question is: what is the complexity of g -FIXED-POINT-FREE AUTOMORPHISM when g is a function which grows more slowly than $g(n) = \varepsilon \cdot n^{1/m}$ and yet is unbounded, for example, $g(n) = \log_2 n$?

4. Conclusions and open problems. Although SUBGRAPH ISOMORPHISM allows more freedom than ISOMORPHISM in mapping one graph to another, and AUTOMORPHISM WITH RESTRICTIONS allows less freedom, the common factor is a destruction of the essential structure of the ISOMORPHISM problem. It is possible that because of its structure GRAPH ISOMORPHISM is not NP-complete.

The concept of “structure” in a problem is a difficult one to define precisely. In ISOMORPHISM one facet of the structure of the problem is that the automorphisms of a graph induce an equivalence relation on the vertices of the graph— $u \sim v$ if there is an automorphism ψ of G with $\psi(u) = v$. The set of equivalence classes of this relation constitutes the *automorphism partition* of the graph. The problem of finding the automorphism partition of a graph is ISOMORPHISM-complete (see [14]). This result and the fact that the number of automorphisms of a graph $G = (V, E)$ is equal to the number of automorphisms of G which fix $v \in V$ multiplied by the number of elements in v 's cell of the automorphism partition of G are the essential steps in the proof that the problem of counting the number of isomorphisms between two graphs is ISOMORPHISM-complete. This result is due to Babai [1] and Mathon [12].

The counting versions of most (perhaps all) known NP-complete problems belong to the class of #P-complete problems (see [15] and [16], or [5] for definitions and results). #P-complete problems do not appear to be polynomial time equivalent to NP-complete problems. This evidence of a difference between ISOMORPHISM and the known NP-complete problems depends strongly on the “structure” of ISOMORPHISM.

When ISOMORPHISM is generalized to an NP-complete problem, the structure is destroyed. In SUBGRAPH ISOMORPHISM it is not so much broken down as hidden under a wealth of other possibilities—there is a choice not only of how to map one graph onto another, but also of which subgraph of the larger graph to consider. In AUTOMORPHISM WITH RESTRICTIONS the structure is simply destroyed; specifically, the relation \sim on the vertices induced by the set of allowable automorphisms may no longer be reflexive, symmetric, or transitive. For example, if R is a set of restrictions for a graph $G = (V, E)$ and $(x, y) \in R$, for some $x, y \in V$, there may be an automorphism ψ of G with $\psi \cap R = \emptyset$ and $\psi(y) = x$, so that $y \sim x$, but $x \not\sim y$ since there can be no automorphism ψ of G with $\psi \cap R = \emptyset$ and $\psi(x) = y$. Symmetry is lost. In FIXED-POINT-FREE AUTOMORPHISM the only restriction is that no vertex may map to itself so only the reflexive property is destroyed (though the transitive property suffers as a consequence). In all of these problems destroying the structure of ISOMORPHISM causes the counting version of the problem to become #P-complete.

It is quite possible that ISOMORPHISM is neither in P nor NP-complete. If this is the case there will be problems in NP strictly between P and ISOMORPHISM-complete, and problems in NP strictly between ISOMORPHISM-complete and NP-complete (Ladner proves more general statements than these in [9]). Some possible candidates for problems in these “inbetween” classes are described below.

Open Problems:

1. Is the AUTOMORPHISM problem ISOMORPHISM-complete? In P?
2. Is fixed-point-free automorphism of *groups* NP-complete?

This seems doubtful. Order 2 fixed-point-free automorphism of groups is solvable in

polynomial time since a group G has a fixed-point-free automorphism ψ if and only if G is abelian and $\psi(x) = x^{-1}$, $\forall x \in G$ (see [6]). It is still an open question whether isomorphism of groups is ISOMORPHISM-complete or in P. An $O(n^{\log n})$ algorithm is known (see [13]).

The problem of fixed-point-free automorphism on lattices can easily be shown to be NP-complete. (The maximum and minimum elements are allowed to remain fixed of course.)

3. What is the complexity of g -FIXED-POINT-FREE AUTOMORPHISM for $g(n) = \log_2 n$? The problem would seem to be at least as hard as ISOMORPHISM but it is not obviously ISOMORPHISM-complete or NP-complete.

4. For $K \in \mathbb{N}$ consider the problem:

*K*th ISOMORPHISM.

Instance: two graphs G_1 and G_2 , and $K - 1$ distinct isomorphisms ψ_i , $1 \leq i \leq K - 1$, of G_1 onto G_2 .

Question: Is there another isomorphism? That is, is there an isomorphism ψ of G_1 onto G_2 such that $\psi \neq \psi_i$, $1 \leq i \leq K - 1$.

For $K = 1$ the problem is ISOMORPHISM; for $K = 2$ it is AUTOMORPHISM. The general problem in which K is allowed to vary as part of the problem is ISOMORPHISM-complete, so *K*th ISOMORPHISM is always at least as easy as ISOMORPHISM, but is it ISOMORPHISM-complete? AUTOMORPHISM-complete? Do the problems get easier or harder as K increases?

Acknowledgments. I would like to thank Charles Colbourn for his enthusiasm, and Derek Corneil for his encouragement and considerable assistance.

REFERENCES

- [1] L. BABAI, *On the isomorphism problem*, Proc. FCT Conf., Poznań-Kórnik, 1977.
- [2] K. S. BOOTH AND C. J. COLBOURN, *Problems polynomially equivalent to graph isomorphism*, Tech. Report CS-77-04, Dept. of Computer Science, Univ. Waterloo, 1979.
- [3] C. J. COLBOURN, *A bibliography of the graph isomorphism problem*, TR-123, Dept. of Computer Science, Univ. Toronto, 1978.
- [4] D. G. CORNEIL, *Recent results on the graph isomorphism problem*, Proc. 8th Manitoba Conference on Numerical Mathematics and Computing, 1978, pp. 13-31.
- [5] M. R. GAREY AND D. S. JOHNSON, *Computers And Intractability: A Guide To The Theory Of NP-completeness*, Freeman, San Francisco, 1979.
- [6] P. GORENSTEIN, *Finite Groups*, Harper & Row, New York, 1968.
- [7] J. E. HOPCROFT AND J. K. WONG, *Linear time algorithm for isomorphism of planar graphs*, Proc. of Sixth Annual ACM Symp. on Theory of Computing, 1974, pp. 172-184.
- [8] D. KOZEN, *A clique problem equivalent to graph isomorphism*, SIGACT News, 10, 2 (1978), pp. 50-52.
- [9] R. E. LADNER, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach., 22 (1975), pp. 155-171.
- [10] G. LEVI, *A note on the derivation of maximal common subgraphs of two directed or undirected graphs*, Calcolo, 9 (1972), pp. 341-354.
- [11] G. S. LUEKER AND K. S. BOOTH, *A linear time algorithm for deciding interval graph isomorphism*, J. Assoc. Comput. Mach., 26 (1979), pp. 183-195; also Tech. Report 109, Univ. Calif. Irvine, 1977.
- [12] R. A. MATHON, *A note on the graph isomorphism counting problem*, preprint, 1978.
- [13] G. L. MILLER, *On the $n^{**} \log n$ isomorphism technique*, Proc. Tenth SIGACT Symp. on the Theory of Computing, 1978, pp. 51-58; also Tech. Report 17, Univ. Rochester, 1976.
- [14] R. C. READ AND D. C. CORNEIL, *The graph isomorphism disease*, J. Graph Theory, 1 (1977), pp. 339-363.
- [15] L. G. VALIANT, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189-201.
- [16] ———, *The complexity of enumeration and reliability problems*, this Journal, 8 (1979), pp. 410-421.

CHARACTERIZATIONS OF PRESBURGER FUNCTIONS*

OSCAR H. IBARRA[†] AND BRIAN S. LEININGER[†]

Abstract. Let \mathcal{F} be the smallest class of functions on the natural numbers containing the functions $U_i^n(x_1, \dots, x_n) = x_i$, $S(x) = x + 1$, $A(x, y) = x + y$, $D(x, y) = x \div y$, $C(x, y) = (1 \div y)x$, $T_k(x) = \lfloor x/k \rfloor$ and closed under composition. It is shown that \mathcal{F} is exactly the class of functions definable by Presburger formulas. Moreover, for Presburger functions with finite output range, $A(x, y)$ and $C(x, y)$ can be deleted from the list of initial functions. Characterizations of \mathcal{F} and its subclasses in terms of simple programs are also given. An example is the following: A function is in \mathcal{F} if and only if it is computable by a program which contains only instructions of the form $x \leftarrow x + 1$, $x \leftarrow x - 1$, $x \leftarrow y$, and **do** $x \dots$ **end**, where **do**'s cannot be nested.

Key words. Characterization, Presburger formula, Presburger function, simple program, straight-line program

1. Introduction. For a set F of initial functions, let $\text{Comp}(F)$ be the smallest class of functions containing F and closed under function composition. Let F be the following list of functions:

- (1) $U_i^n(x_1, \dots, x_n) = x_i$ for each positive integer n and $1 \leq i \leq n$
- (2) $Z(x) = 0$
- (3) $S(x) = x + 1$
- (4) $A(x, y) = x + y$
- (5) $P(x, y) = x \div 1^1$
- (6) $C(x, y) = (1 \div y)x$
- (7) $T_k(x) = \lfloor x/k \rfloor$ for each positive integer k
- (8) $R_k(x) = \text{remainder}(x/k)$ for each positive integer k
- (9) $D(x, y) = x \div y$.

In [11], it is shown that $\text{Comp}(1-8)$ (called simple functions in [11]) is precisely the class of functions computable by L_1 programs. (L_1 is the "loop(1)" language consisting only of instructions $x \leftarrow 0$, $x \leftarrow x + 1$, $x \leftarrow y$, **do** $x \dots$ **end** where **do**'s cannot be nested [10].) Our main results in this paper are the following:

(a) $\text{Comp}(1-9) = \text{Comp}(1, 3, 4, 6, 7, 9)$ is exactly the class of functions definable by Presburger formulas.

(b) The functions 1, 3, 4, 6, 7, 9 are independent in the sense that none of the functions can be obtained from the others by composition.

(c) The set of functions in $\text{Comp}(1, 3, 7, 9)$ with finite output range (i.e., the output can only assume values $0, 1, \dots, m$ for some m) is exactly the class of Presburger functions with finite output range. Moreover, the functions 1, 3, 7, 9 are independent.

The proof of (a) uses a result in [6] (see also [7]) which relates Presburger formulas to simple programs. Before we can state this result we need a few definitions.

DEFINITION. Presburger formulas are defined inductively as follows (see, e.g., [4]):

- (a) $a_0 + \sum_{i=1}^m a_i x_i = b_0 + \sum_{i=1}^m b_i x_i$ is a Presburger formula for every integer $m \geq 1$, variables x_1, \dots, x_m and nonnegative integers $a_0, a_1, \dots, a_m, b_0, b_1, \dots, b_m$.
- (b) If F_1 and F_2 are Presburger formulas, then so are their conjunction $F_1 \wedge F_2$ and their disjunction $F_1 \vee F_2$.
- (c) If F is a Presburger formula, then so is its negation $\neg F$.

* Received by the editors June 11, 1979 and in final revised form April 18, 1980. This research was supported in part by the National Science Foundation under Grant No. MCS78-01736.

[†] Department of Computer Science, Institute of Technology, University of Minnesota, Minneapolis, Minnesota 55455.

¹ $u - v = u - v$ if $u \geq v$, 0 otherwise.

(d) If x_i is a free variable in a Presburger formula F , then $(\exists x_i)F$ and $(\forall x_i)F$ are Presburger formulas.

(e) Only expressions derivable using rules (a)–(d) are Presburger formulas. A Presburger formula with $m \geq 1$ free variables will be denoted by $F(x_1, \dots, x_m)$.

DEFINITION. Let N be the set of nonnegative integers. A *total* function $f: N^n \rightarrow N^m$ ($n, m \geq 1$) is a *Presburger function* if there is a Presburger formula $F(x_1, \dots, x_n, y_1, \dots, y_m)$ such that for each (i_1, \dots, i_n) in N^n , if $f(i_1, \dots, i_n) = (j_1, \dots, j_m)$ then $F(i_1, \dots, i_n, j_1, \dots, j_m)$ is true and $F(i_1, \dots, i_n, k_1, \dots, k_m)$ is false for all $(k_1, \dots, k_m) \neq (j_1, \dots, j_m)$.

Next, we give the syntax of a simple programming language which has been shown to characterize Presburger functions [6], [7].

Let XL be the programming language which has the following instruction set:

- (1) $x \leftarrow x + 1$
- (2) $x \leftarrow x \div 1$
- (3) **if** $x = 0$ **then exit**
- (4) **do** x
- ⋮
- end**
- (5) $x \leftarrow 0$
- (6) $x \leftarrow y$
- (7) **go to** l
- (8) **if** $x = 0$ **then go to** l

An XL program P is any finite sequence of (possibly labeled) instructions of the form (1)–(8) satisfying the following conditions:

- (a) **do** \dots **end** pairs only enclose instructions of the form (1)–(3) and (5)–(8). Thus, nested **do**'s are not allowed.
- (b) **if** $x = 0$ **then exit** instructions can only appear inside **do** \dots **end** constructs.
- (c) Labels in instructions of the form (7) and (8) are forward labels.
- (d) No instruction in the scope of a **do** \dots **end** construct can be labeled. (The **do** itself can be labeled.)

Each program variable can hold any nonnegative integer. The variable x controlling the **do** $x \dots$ **end** construct can be modified inside the **do** without changing the number of iterations. The **if** $x = 0$ **then exit** causes an exit out of the **do** containing it if $x = 0$. The program halts after processing the last instruction. Two fixed (not necessarily disjoint) sets of program variables are designated input variables and output variables, respectively. Before the start of program execution, all noninput variables are initialized to zero while the input variables are set to some input values. Assume that the program P has input variables x_1, \dots, x_n and output variables y_1, \dots, y_m ($n, m \geq 1$). The function f_P defined by P is given by: $f_P(i_1, \dots, i_n) = (j_1, \dots, j_m)$ if P with x_1, \dots, x_n set to i_1, \dots, i_n halts with values j_1, \dots, j_m in y_1, \dots, y_m . The subset of XL consisting only of instructions $x \leftarrow x + 1$, $x \leftarrow x \div 1$, **if** $x = 0$ **then exit**, and **do** $x \dots$ **end** is called the SL language. The language consisting only of instructions $x \leftarrow 0$, $x \leftarrow x + 1$, $x \leftarrow y$, and **do** $x \dots$ **end** is called the loop(1) language (L_1 , for short) [10].

The connection between SL programs and Presburger formulas is given by the following theorem which was proved in [6], [7]. (See also [1] for related results.)

THEOREM 1.

- (a) A function is a Presburger function if and only if it is computable by an SL program.
- (b) Every XL program can be converted into an equivalent SL program.

We will use Theorem 1 to prove our characterization results. For convenience, we first prove an intermediate result about SL programs which is of independent interest. This is done in the next section.

2. A straight-line program characterization of SL. Let Q be the programming language consisting only of instructions:

$$\begin{aligned} x &\leftarrow x + 1 \\ x &\leftarrow x + y \\ x &\leftarrow x \div y \\ x &\leftarrow (1 \div y)x \\ x &\leftarrow \lfloor x/k \rfloor, \quad k \text{ a positive integer} \end{aligned}$$

The variables x and y in the above instructions need not be distinct. We will show that every SL program can be transformed into an equivalent Q program, and conversely.

Notation. If E is an arithmetic expression computable by a Q program and x is a variable, $x \leftarrow E$ will denote a Q program for assigning to x the value of the expression E . For example, we can write $x \leftarrow (\lfloor (1 \div y)x + 1 \rfloor \div x) + y$.

The following lemma will simplify proofs of later results.

LEMMA 1. *Each of the following instructions can be coded in Q : $x \leftarrow 0$, $x \leftarrow c$, $x \leftarrow x + c$, $x \leftarrow x \div c$, $x \leftarrow ay$, $x \leftarrow ax + by$, $x \leftarrow ax \div by$, **if** p_1 **then** $x \leftarrow E$, **if** p_1 **and** p_2 **then** $x \leftarrow E$, where a, b, c are positive integers, p_1, p_2 are predicates of the form $y > 0$, $y = 0$, $y < 0$, $y > z$, $y = z$, $y < z$, and E is an expression computable in Q .*

Proof. The codings in Q are straightforward. For example, **if** $y > 0$ **then** $x \leftarrow E$ can be coded as

$$\begin{aligned} z &\leftarrow E \\ x &\leftarrow (1 \div y)x + (1 \div (1 \div y))z \end{aligned}$$

The instruction **if** p_1 **and** p_2 **then** $x \leftarrow E$ can be coded as

$$\begin{aligned} z &\leftarrow E \\ h &\leftarrow 2 \\ \mathbf{if} \ p_1 \ \mathbf{then} \ h &\leftarrow h \div 1 \\ \mathbf{if} \ p_2 \ \mathbf{then} \ h &\leftarrow h \div 1 \\ x &\leftarrow (1 \div h)z + (1 \div (1 \div h))x \quad \square \end{aligned}$$

The proof that every SL program can be converted into an equivalent Q program is given in four lemmas.

Notation. \hat{Q} will denote the language consisting only of instructions $x \leftarrow x + 1$, $x \leftarrow x + y$, and $x \leftarrow x \div y$.

LEMMA 2. *Let P be an SL program without **do** \cdots **end** constructs (and hence no **if** instructions) which uses only variable x . Then P can be reduced to an equivalent program P' which takes one of the following forms:*

- (1) empty
- (2) $x \leftarrow x + c$
- (3) $x \leftarrow x \div c$
- (4) $x \leftarrow x \div c$
 $x \leftarrow x + d$

where c and d are positive integers. Hence, P can be converted into an equivalent \hat{Q} program.

Proof. Apply the following algorithm to P :

Step 1. First reduce P into an equivalent program P' whose instructions are of the types $x \leftarrow x + c_1$ and $x \leftarrow x \dot{-} c_2$ and they *alternate* (c_1 and c_2 are positive integers).

Step 2. If P' has one of the forms (1)–(4), stop.

Step 3. Find two instructions in P' of the form

$$x \leftarrow x + c_1$$

$$x \leftarrow x \dot{-} c_2$$

where c_1 and c_2 are positive integers. If $c_1 = c_2$, delete these two instructions and go to step 2. Otherwise, replace these two instructions by the single instruction $x \leftarrow x + (c_1 - c_2)$ if $c_1 > c_2$ or by the single instruction $x \leftarrow x \dot{-} (c_2 - c_1)$ if $c_2 > c_1$ and go to step 2.

Clearly, the algorithm terminates with P' taking one of the forms (1)–(4). Moreover, such a P' is equivalent to P . \square

Notation. Let α be a program containing only instructions of the form $x_i \leftarrow x_i + 1$ or $x_i \leftarrow x_i \dot{-} 1$. For any variable x , $[x, \alpha]$ will denote the program obtained from α by deleting all instructions not involving variable x .

LEMMA 3. *Let P be an SL-**{if}** program of the form*

do z

α

end

Let x_1, \dots, x_n be the variables appearing in α . Assume that if z appears in α , then $x_n = z$. Then P can be converted into an equivalent \hat{Q} program.

Proof. Clearly, P is equivalent to the program

do z

$[x_1, \alpha]$

end

\vdots

do z

$[x_n, \alpha]$

end

By Lemma 2, each $[x_i, \alpha]$ can be reduced to one of the following forms:

- (1) empty
- (2) $x_i \leftarrow x_i + c_i$
- (3) $x_i \leftarrow x_i \dot{-} c_i$
- (4) $x_i \leftarrow x_i \dot{-} c_i$
 $x_i \leftarrow x_i + d_i$

Hence, each

do z
 $[x_i, \alpha_i]$
end

reduces to one of the following forms:

- (5) empty
 (6) $x_i \leftarrow x_i + c_i z$
 (7) $x_i \leftarrow x_i \div c_i z$
 (8) $y \leftarrow z \div (z \div 1)$
 $x_i \leftarrow (x_i \div c_i y) + (d_i - c_i)(z \div 1) + d_i y$
 (for the case $d_i \geq c_i$)
 (9) $y \leftarrow z \div (z \div 1)$
 $x_i \leftarrow (x_i \div c_i y) \div (c_i - d_i)(z \div 1) + d_i y$
 (for the case $c_i > d_i$)

The result follows. \square

THEOREM 2.

(i) Every $\text{SL} - \{\mathbf{if}\}$ program can be converted into an equivalent $\hat{\mathcal{Q}}$ program, and conversely.

(ii) Every $(\text{SL} - \{\mathbf{if}\}) \cup \{x \leftarrow 0\}$ program can be converted into an equivalent $\hat{\mathcal{Q}} \cup \{x \leftarrow (1 \div y)x\}$ program, and conversely.

Proof. (i) follows from Lemmas 2 and 3 and the fact that the instructions $x \leftarrow x + 1$, $x \leftarrow x + y$, and $x \leftarrow x \div y$ are clearly computable by $\text{SL} - \{\mathbf{if}\}$ programs. The proof of (ii) is similar. In Lemma 3, $x_i \leftarrow b_i$ is a possible form (b_i a nonnegative integer). The reduced form is: **if** $z > 0$ **then** $x_i \leftarrow b_i$. This uses the instruction $x \leftarrow (1 \div y)x$ when translated (see Lemma 1). \square

Next, we consider SL **do** \dots **end** constructs with **if** instructions.

LEMMA 4. Let P be an SL program of the form

do z
 α_1
 if $x_1 = 0$ **then exit**
 α_2
 if $x_2 = 0$ **then exit**
 α_3
 \vdots
 if $x_m = 0$ **then exit**
 α_{m+1}
end

where $m \geq 1$ and $\alpha_1, \dots, \alpha_{m+1}$ contain only instructions of the form $x \leftarrow x + 1$ or $x \leftarrow x \div 1$. (Some α_i 's may be empty.) The variables x_1, \dots, x_m need not be distinct. Assume that $z > 0$ and one of the **if**'s causes the exit. Then P can be converted into an equivalent Q program.

Proof. The construction uses a technique in [6]. For $i = 1, \dots, m$, consider the program

```

do w
     $\alpha_1$ 
     $\vdots$ 
     $\alpha_i$ 
    if  $x_i = 0$  then exit
     $\alpha_{i+1}$ 
     $\vdots$ 
     $\alpha_{m+1}$ 
end
    
```

where w is a new variable. Let t_i = the least number of times the “**if** $x_i = 0$ **then exit**” instruction is encountered if it causes the exit for some value of w . If the **if** instruction cannot cause an exit for any value of w , let $t_i = z$. Then the following program is equivalent to P:

<pre> $t \leftarrow z$ compute t_1 if $t > t_1$ then $t \leftarrow t_1$ \vdots compute t_m if $t > t_m$ then $t \leftarrow t_m$ </pre>	}	$t \leftarrow \min \{t_1, \dots, t_m\}$
<pre> $h_1 \leftarrow 0$ \vdots $h_m \leftarrow 0$ if $t_1 = t$ then $h_1 \leftarrow h_1 + 1$ $u \leftarrow t_1$ if $t_2 = t$ and $u > t_2$ then $h_2 \leftarrow h_2 + 1$ if $u > t_2$ then $u \leftarrow t_2$ \vdots if $t_m = t$ and $u > t_m$ then $h_m \leftarrow h_m + 1$ if $u > t_m$ then $u \leftarrow t_m$ </pre>	}	Sets $h_i = 1$ where i is the smallest index such that $t_i = t$. All other h_j 's are set to 0.

```

w ← 0
if  $h_i > 0$  then  $w \leftarrow w + 1$ 
do w
     $\alpha_1$ 
     $\vdots$ 
     $\alpha_i$ 
end
w ← 0
if  $h_i > 0$  then  $w \leftarrow t \div 1$ 
do w
     $\alpha_{i+1}$ 
     $\vdots$ 
     $\alpha_{m+1}$ 
     $\alpha_1$ 
     $\vdots$ 
     $\alpha_i$ 
end

```

} repeat this code for $i = 1, 2, \dots, m$

The next lemma shows that the task “compute t_i ” can be computed by a Q program. Hence, by Lemmas 1–3, the above program can be reduced to an equivalent Q program. \square

LEMMA 5. *Let P be an SL program of the form*

```

do z
     $\alpha_1$ 
    if  $x_i = 0$  then exit
     $\alpha_2$ 
end

```

where $z > 0$ and α_1, α_2 contain no **if** instruction and only reference variable x_i . Then t_i can be computed by a Q program.

Proof. We consider two cases.

Case 1. The program

```

 $\alpha_2$ 
 $\alpha_1$ 

```


reduces to the empty program or to the form $x_i \leftarrow x_i + c_i$ or the form

$$x_i \leftarrow x_i \dot{-} c_i$$

$$x_i \leftarrow x_i + d_i$$

where c_i and d_i are positive integers. Then t_i can be computed by the code

$$\alpha_1$$

$$t_i \leftarrow z$$

if $x_i = 0$ **then** $t_i \leftarrow 1$

Case 2. The program

$$\alpha_2$$

$$\alpha_1$$

reduces to the form $x_i \leftarrow x_i \dot{-} c_i$. Then the code to compute t_i is

$$\alpha_1$$

$$t_i \leftarrow \left\lfloor \frac{x_i + (c_i - 1)}{c_i} \right\rfloor + 1$$

By Lemma 1, t_i can be computed by a Q program. \square

We are now ready to prove the main result of this section.

THEOREM 3. *Every SL program can be converted into an equivalent Q program, and conversely.*

Proof. By Theorem 2, we only have to show that every SL program of the form

do z

$$\alpha_1$$

if $x_1 = 0$ **then exit**

$$\alpha_2$$

\vdots

$$\alpha_m$$

if $x_m = 0$ **then exit**

$$\alpha_{m+1}$$

end

can be reduced to an equivalent Q program. Now the above program is equivalent to the

program

```

 $z' \leftarrow z + 1$ 
do  $z'$ 
   $z' \leftarrow z' \div 1$ 
  if  $z' = 0$  then exit
   $\alpha_1$ 
  if  $x_1 = 0$  then exit
   $\alpha_2$ 
   $\vdots$ 
   $\alpha_m$ 
  if  $x_m = 0$  then exit
   $\alpha_{m+1}$ 
end

```

where z' is a new variable. Then $z' > 0$ and the exit will be caused by an **if** instruction. Then, by Lemma 4, the **do** $z' \cdots$ **end** construct can be reduced to an equivalent Q program.

The converse is obvious. \square

From Theorems 1 and 3, we have

COROLLARY 1. *Every XL program can be converted into an equivalent Q program, and conversely.*

COROLLARY 2. *A one-output function $f(x_1, \dots, x_n)$ is computable by an XL program if and only if it can be written as an arithmetic expression with operands x_1, \dots, x_n and nonnegative integer constants, and operations: plus 1, integer division by a positive integer constant, addition, proper subtraction, and the operation $(1 \div y)x$.*

Let CL be the language consisting only of instructions $x \leftarrow x + 1$, $x \leftarrow x \div 1$, $x \leftarrow y$, and **do** $x \cdots$ **end**, where **do**'s cannot be nested. CL was introduced in [2] and was shown to have a complete and consistent Hoare axiomatics. Clearly, every Q instruction can be coded in CL. Then by Corollary 1, we also have

COROLLARY 3. *Every CL program can be converted into an equivalent Q program, and conversely.*

We will show that the language Q is minimal in that all instructions in Q are independent. First, we prove the following more general result.

LEMMA 6. *Let R be the language consisting of instructions $x \leftarrow x + 1$, $x \leftarrow x + y$, $x \leftarrow x \div 1$, $x \leftarrow x \div y$, $x \leftarrow \lfloor x/k \rfloor$, $x \leftarrow \text{rem}(x/k)$, and $x \leftarrow (1 \div y)x$. (k is a positive integer, $\text{rem}(x/k) = \text{remainder of } x \text{ divided by } k$, x and y need not be distinct.) Then each of the following instructions cannot be expressed in terms of the remaining instructions: $x \leftarrow x + 1$, $x \leftarrow x + y$, $x \leftarrow \lfloor x/k \rfloor$, $x \leftarrow x \div y$, and $x \leftarrow (1 \div y)x$.*

Proof.

(a) $x \leftarrow x + 1$ cannot be deleted from R:

Clearly, every function $g(x)$ computable by an $R - \{x \leftarrow x + 1\}$ program has the property that $g(0) = 0$. Hence, the function $f(x) = 1$ for all x is not $R - \{x \leftarrow x + 1\}$ computable.

(b) $x \leftarrow x + y$ cannot be deleted from R:

If $g(x)$ is $R - \{x \leftarrow x + y\}$ computable by a program with r instructions, then $g(x) \leq x + r$. Hence, $f(x) = 2x$ is not computable in $R - \{x \leftarrow x + y\}$.

(c) $x \leftarrow \lfloor x/k \rfloor$ cannot be deleted from R:

Suppose $g(x)$ is computable by an $R - \{x \leftarrow \lfloor x/k \rfloor\}$ program P with r instructions. Let $K =$ product of all k 's (possibly with repetition) appearing in instructions of the form $x \leftarrow \text{rem}(x/k)$. If no such instruction appears in P, let $K = 1$. Then one can easily verify (by induction on r) that for all x of the form $x = Kx_0$ with $x_0 > 2^r$, $g(x)$ can uniquely be written in one of the following forms: $g(x) = 0$, $g(x) = c$, $g(x) = ax$, $g(x) = ax + c$, $g(x) = ax - c$, where a and c are positive integers satisfying $1 \leq a$, $c \leq 2^r K$. It follows that $f(x) = \lfloor x/2 \rfloor$ is not $R - \{x \leftarrow \lfloor x/k \rfloor\}$ computable.

(d) $x \leftarrow x \div y$ cannot be deleted from R:

Let $g(x, y)$ be computable by an $R - \{x \leftarrow x \div y\}$ program with r instructions. Let $K =$ product of all k 's appearing in instructions of the form $x \leftarrow \lfloor x/k \rfloor$ or $x \leftarrow \text{rem}(x/k)$. Then by induction on r , we can show that for all x and y of the form $x = Kx_0$, $y = Ky_0$ with $x_0, y_0 > 2^r K$, $g(x, y)$ can be written uniquely in one of the following forms: $g(x, y) = 0$, $g(x, y) = c$, $g(x, y) = ax_0$, $g(x, y) = ax_0 + c$, $g(x, y) = ax_0 - c$, $g(x, y) = ay_0$, $g(x, y) = ay_0 + c$, $g(x, y) = ay_0 - c$, $g(x, y) = ax_0 + by_0$, $g(x, y) = ax_0 + by_0 + c$, $g(x, y) = ax_0 + by_0 - c$, where a , b , and c are positive integers satisfying $1 \leq a, b, c \leq 2^r K$. It follows that $f(x, y) = x \div y$ is not computable by an $R - \{x \leftarrow x \div y\}$ program.

(e) $x \leftarrow (1 \div y)x$ cannot be deleted from R:

Suppose $g(x, y) = (1 \div y)x$ is computed by an $R - \{x \leftarrow (1 \div y)x\}$ program P with r instructions. Without loss of generality, we may assume that y is restricted to values 0 and 1. Let z be the output variable and K be as defined in part (d). Then one can verify (by induction on r) that for all x of the form $x = Kx_0$ with $x_0 > 2^r K$, the value of z at the end of the program can be written uniquely in one of the following forms: $z = 0$, $z = c(y)$, $z = ax_0$, $z = ax_0 + c(y)$, $z = ax_0 - c(y)$, where a is a positive integer satisfying $1 \leq a \leq 2^r K$ and $c(y)$ is an arithmetic expression involving only variable y and $0 \leq c(y) \leq 2^r K$ for $y = 0, 1$. It follows that z cannot output $(1 \div y)x$ for all $y = 0, 1$ and $x = Kx_0$, $x_0 > 2^r K$. Hence, $f(x, y) = (1 \div y)x$ is not computable in $R - \{x \leftarrow (1 \div y)x\}$. \square

We also need the following lemma.

LEMMA 7.

(a) The instruction $x \leftarrow x \div 1$ cannot be expressed in terms of $x \leftarrow x + 1$, $x \leftarrow x + y$, $x \leftarrow \lfloor x/k \rfloor$, $x \leftarrow \text{rem}(x/k)$, and $x \leftarrow (1 \div y)x$.

(b) The instruction $x \leftarrow \text{rem}(x/k)$ cannot be expressed in terms of $x \leftarrow x + 1$, $x \leftarrow x + y$, $x \leftarrow x \div 1$, $x \leftarrow \lfloor x/k \rfloor$, and $x \leftarrow (1 \div y)x$.

Proof.

(a) Suppose $g(x)$ is a function computed by a program P without instructions of the form $x \leftarrow x \div 1$. Let r be the number of instructions in P and $K =$ product of all k 's appearing in instructions of the form $x \leftarrow \lfloor x/k \rfloor$ or $x \leftarrow \text{rem}(x/k)$. Then, clearly, for all $x_0 \geq 2^r K + 2$, $g(Kx_0)$ can be written uniquely as $g(Kx_0) = c$ or $g(Kx_0) = ax_0 + c$ (c is a nonnegative integer $\leq 2^r K$ and a is some positive integer). If $g(Kx_0) = c$ then, obviously, P does not compute $x \div 1$. If $g(Kx_0) = ax_0 + c$, we have two cases. If $a \geq K$, then $g(Kx_0) \geq Kx_0$. If $a < K$ then $g(Kx_0) = ax_0 + c \leq (K - 1)x_0 + c = Kx_0 + (c - x_0) \leq Kx_0 - 2$. In either case, P does not compute $x \div 1$.

(b) Let $g(x)$ be computed by a program P without instructions of the form $x \leftarrow \text{rem}(x/k)$. Let r be the number of instructions of P and $K =$ product of all k 's appearing in instructions of the form $x \leftarrow \lfloor x/k \rfloor$. One easily checks that $g(x)$ is either a constant for all $x \geq Kr$ or $g(x)$ is unbounded for $x \geq Kr$. It follows that P cannot compute $\text{rem}(x/k)$. \square

We can now state the following theorem.

THEOREM 4.

(a) *Every function computable by an R program is computable by a Q program.*

Thus, R and Q are equivalent languages.

(b) *Q is minimal.*

Proof. Part (a) is obvious since the instructions $x \leftarrow x \div 1$ and $x \leftarrow \text{rem}(x/k)$ are expressible in terms of Q instructions. The independence of the instructions in Q follows from Lemma 6. \square

In [11], it is shown that a function is computable by an L_1 program if and only if it is computable by an $(R - \{x \leftarrow x \div y\}) \cup \{x \leftarrow 0\}$ program. From Lemmas 6 and 7 and the fact that $x \leftarrow 0$ is computed by the code: $x \leftarrow \text{rem}(x/2); x \leftarrow x \div 1$, we have

THEOREM 5. *A function is computable by an L_1 program if and only if it is computable by an $R - \{x \leftarrow x \div y\}$ program. Moreover, $R - \{x \leftarrow x \div y\}$ is minimal.*

3. Characterizations of Presburger functions. Let $\mathcal{F} = \text{Comp}(B)$, where B are the functions: $U_i^n(x_1, \dots, x_n) = x_i$, $S(x) = x + 1$, $A(x, y) = x + y$, $D(x, y) = x \div y$, $C(x, y) = (1 \div y)x$, and $T_k(x) = \lfloor x/k \rfloor$. Let \mathcal{F}^* be the class of multiple-output functions obtained from \mathcal{F} as follows: An n -input, m -output function $f: N^n \rightarrow N^m$ is in \mathcal{F}^* if there are one-output functions f_1, \dots, f_m in \mathcal{F} such that $f(x_1, \dots, x_n) = (f_1(x_1, \dots, x_n), \dots, f_m(x_1, \dots, x_n))$. Let \mathcal{P} be the class of one-output Presburger functions and \mathcal{P}^* be the class of multiple-output Presburger functions.

The following lemma is immediate from the definition of language Q.

LEMMA 8. *A one-output (multiple-output) function is computable by a Q program if and only if it is in $\mathcal{F}(\mathcal{F}^*)$.*

We can now prove the main result of this section.

THEOREM 6. *$\mathcal{F} = \mathcal{P}$ and $\mathcal{F}^* = \mathcal{P}^*$. Moreover, B is minimal in the sense that none of the functions in B can be obtained from the others by composition.*

Proof. The first statement follows from Theorems 1 and 3 and Lemma 8. The minimality of B follows from Theorem 4 and the fact that the function $U_i(x_1, \dots, x_n) = x_i$ is needed to specify the input parameters \square

Our next result concerns the characteristic functions of Presburger formulas.

THEOREM 7. *Let \mathcal{F}_1 be the set of functions in \mathcal{F} with output range $\{0, 1\}$. Then f is in \mathcal{F}_1 if and only if it is the characteristic function of some Presburger formula.*

Proof. Let $F_1(x_1, \dots, x_n)$ be a Presburger formula. The characteristic function λ_{F_1} of F_1 is defined by

$$\lambda_{F_1}(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } F_1(x_1, \dots, x_n) \text{ is true} \\ 0 & \text{if } F_1(x_1, \dots, x_n) \text{ is false.} \end{cases}$$

Define a Presburger formula F over the variables x_1, \dots, x_n, y by

$$F(x_1, \dots, x_n, y) = (F_1(x_1, \dots, x_n) \wedge y = 1) \vee (\neg F_1(x_1, \dots, x_n) \wedge y = 0).$$

Then the function defined by F is $f(x_1, \dots, x_n) = \lambda_{F_1}(x_1, \dots, x_n)$. By Theorem 6 $\lambda_{F_1}(x_1, \dots, x_n)$ is in \mathcal{F}_1 .

Now suppose $f(x_1, \dots, x_n)$ is in \mathcal{F}_1 . Then by Theorem 6, there is a Presburger formula $F(x_1, \dots, x_n, y)$ defining f . Construct the formula

$$F_1(x_1, \dots, x_n) = \exists y((y = 1) \wedge F(x_1, \dots, x_n, y)).$$

Now $F_1(x_1, \dots, x_n)$ has the following property:

- (i) If $F_1(x_1, \dots, x_n)$ is true, then $y = 1$ and $F(x_1, \dots, x_n, 1)$ is true.

(ii) If $F_1(x_1, \dots, x_n)$ is false, then $F(x_1, \dots, x_n, 1)$ is false and, hence, $F(x_1, \dots, x_n, 0)$ is true.

It follows that $f(x_1, \dots, x_n)$ is the characteristic function of F_1 . \square

The next theorem characterizes $\mathcal{F}^* = \text{Comp}^*(B)$ in terms of simple programs:

THEOREM 8. *The following statements are equivalent for a function f .*

- (i) f is in \mathcal{F}^* .
- (ii) f is XL computable.
- (iii) f is SL computable.
- (iv) f is CL computable.

Moreover, SL and CL are minimal languages.

Proof. The equivalences follow from Lemma 8, Theorem 3, and Corollaries 1 and 3. The independence of the instructions in SL and CL are easily verified using the techniques of Lemmas 6 and 7 (see [6]). \square

Similarly, we have the following result using Theorem 2.

THEOREM 9.

- (i) f is in $\text{Comp}^*(B - \{C(x, y), T_k(x)\})$ if and only if it is SL-**{if}** computable.
- (ii) f is in $\text{Comp}^*(B - \{T_k(x)\})$ if and only if it is (SL-**{if}**) $\cup \{x \leftarrow 0\}$ computable.

Moreover, SL-**{if}** and (SL-**{if}**) $\cup \{x \leftarrow 0\}$ are minimal languages.

Remark. We should mention here a related work of Harrow [8] (see also [9]). Let C be the smallest class of functions containing the functions $U_i^n(x_1, \dots, x_n) = x_i$, $Z(x) = 0$, $S(x) = x + 1$, $A(x, y) = x + y$, $D(x, y) = x \dot{-} y$, and closed under the operations of composition and strict limited minimum (see [5], [8] for the definition of the last operation).² In [8], the following result was shown: A set $R \subseteq N^n$ is a Presburger set (i.e., there is a Presburger formula $F(x_1, \dots, x_n)$ such that $R = \{(i_1, \dots, i_n) \mid F(i_1, \dots, i_n) \text{ is true}\}$) if and only if there exists a function $f(x_1, \dots, x_n)$ in C such that $R = \{(i_1, \dots, i_n) \mid f(i_1, \dots, i_n) = 0\}$. Thus, by Theorem 7, $\mathcal{F}_1 =$ the set of characteristic functions of Presburger formulas = the set of functions in C with output range $\{0, 1\}$.

4. Presburger functions with finite output range. In this section, we prove a somewhat surprising result: Any Presburger function with a finite output range can be defined as a composition of functions $U_i^n(x_1, \dots, x_n) = x_i$, $S(x) = x + 1$, $D(x, y) = x \dot{-} y$, and $T_k(x) = \lfloor x/k \rfloor$. Thus, for such functions, the initial functions $A(x, y) = x + y$ and $C(x, y) = (1 \dot{-} y)x$ are not needed. The proof involves looking at the computing properties of a very simple programming language V which consists only of the following instructions:

- $x \leftarrow y + 1$
- $x \leftarrow x \dot{-} y$
- $x \leftarrow \lfloor x/k \rfloor$

where x and y are (not necessarily distinct) variables and k is a positive integer. Clearly, any V computable function is Q computable. Note that $x \leftarrow y$ can be coded in V as: $x \leftarrow y + 1$; $z \leftarrow z \dot{-} z$; $z \leftarrow z + 1$; $x \leftarrow x \dot{-} z$.

We will show that any Q program whose output variables can only assume values $0, 1, \dots, m$ (for some m) can be transformed into an equivalent V program. This will follow from a sequence of lemmas. The first six show the V computability of some “special” functions. To simplify the proofs, we will freely use programming constructs (and/or arithmetic expressions) which can easily be coded in V, leaving the translation to the reader.

² Actually, the class was called $g^1(x \dot{-} y)$ in [8].

LEMMA 9. Let x and y be nonnegative integer variables. Then the following functions are V computable:

- (1) $\neg x = \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{otherwise;} \end{cases}$
- (2) $x \wedge y = \begin{cases} 1 & \text{if } x > 0 \text{ and } y > 0, \\ 0 & \text{otherwise;} \end{cases}$
- (3) $x \vee y = \begin{cases} 1 & \text{if } x > 0 \text{ or } y > 0, \\ 0 & \text{otherwise;} \end{cases}$
- (4) $x < y = \begin{cases} 1 & \text{if } x < y, \\ 0 & \text{otherwise;} \end{cases}$
- (5) $x > y = \begin{cases} 1 & \text{if } x > y, \\ 0 & \text{otherwise;} \end{cases}$
- (6) $(x = y) = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{otherwise.} \end{cases}$

Proof. The V programs are easily constructed. For example, $z \leftarrow 1 \div x$ computes $\neg x$ while $w \leftarrow (1 \div (1 \div x)) \div (1 \div y)$ computes $x \wedge y$. Part (3) follows from (1) and (2). Similarly, $u \leftarrow 1 \div (1 \div (y \div x))$ computes $x < y$, etc. \square

LEMMA 10. $f(x) = \text{rem}(x/k)$ is V computable for any positive integer k .

Proof. The following program puts in y the value $\text{rem}(x/k)$:

$$(*) \quad \left\{ \begin{array}{l} w \leftarrow k \div 1 \\ y \leftarrow \left\lfloor \frac{x+1}{k} \right\rfloor \div \left\lfloor \frac{x}{k} \right\rfloor \\ w \leftarrow w \div y \\ y \leftarrow \left\lfloor \frac{x+2}{k} \right\rfloor \div \left\lfloor \frac{x}{k} \right\rfloor \\ w \leftarrow w \div y \\ \vdots \\ y \leftarrow \left\lfloor \frac{x+(k-1)}{k} \right\rfloor \div \left\lfloor \frac{x}{k} \right\rfloor \\ w \leftarrow w \div y \\ y \leftarrow k \div 1 \\ y \leftarrow y \div w \end{array} \right.$$

Clearly, if $\text{rem}(x/k) = l$, then the smallest integer c such that $\lfloor (x+c)/k \rfloor = \lfloor x/k \rfloor + 1$ is $c = k - l$. Hence at the end of (*), $w = (k-1) - \text{rem}(x/k)$. It follows that the value of y at the end of the program is $\text{rem}(x/k)$. \square

LEMMA 11. Suppose $x \leq k_1$ and $y \leq k_2$ for some nonnegative integer constants k_1 and k_2 . Then $x + y$ and cx are V computable for any nonnegative integer constant c .

Proof. The code for $v \leftarrow x + y$ is

$$\begin{aligned} w &\leftarrow k_1 + k_2 \\ w &\leftarrow w \dot{-} x \\ w &\leftarrow w \dot{-} y \\ v &\leftarrow k_1 + k_2 \\ v &\leftarrow v \dot{-} w \end{aligned}$$

The code for $v \leftarrow cx$ is

$$\begin{aligned} w &\leftarrow ck_1 \\ \left. \begin{array}{l} w \leftarrow w \dot{-} x \\ \vdots \\ w \leftarrow w \dot{-} x \end{array} \right\} & c \text{ times} \\ v &\leftarrow ck_1 \\ v &\leftarrow v \dot{-} w \quad \square \end{aligned}$$

DEFINITION. A *term* is an expression of the form $a_0 + a_1x_1 + \cdots + a_nx_n$, where each a_i is a nonnegative integer, $n \geq 0$, and x_1, \cdots, x_n are distinct nonnegative integer variables.

LEMMA 12. *The following function is V computable:*

$$r < s = \begin{cases} 1 & \text{if } r < s, \\ 0 & \text{otherwise,} \end{cases}$$

where r and s are terms.

Proof. The proof is an induction on n , where n is the number of variables in r and s . Let $r = a_0 + a_1x_1 + \cdots + a_nx_n$ and $s = b_0 + b_1x_1 + \cdots + b_nx_n$. Without loss of generality, we may assume that for each $0 \leq i \leq n$, $a_i = 0$ or $b_i = 0$ since $r < s$ if and only if $r' < s'$ where $r' = a'_0 + a'_1x_1 + \cdots + a'_nx_n$, $s' = b'_0 + b'_1x_1 + \cdots + b'_nx_n$, $a'_i = a_i \dot{-} b_i$, and $b'_i = b_i \dot{-} a_i$. The reader can easily verify that if $a_1 = a_2 = \cdots = a_n = 0$ or $b_1 = b_2 = \cdots = b_n = 0$ then $r < s$ is computable in V.

Assume that if r and s involve less than n variables then $r < s$ is computable in V. Now consider terms r and s involving n variables. Clearly, we need only take care of the case when $a_i \neq 0$, $b_i = 0$ for some $1 \leq i \leq n$ and $a_j = 0$, $b_j \neq 0$ for some $1 \leq j \leq n$. Without loss of generality, we may assume that $a_1 \neq 0$, $b_1 = 0$, $a_2 = 0$ and $b_2 \neq 0$. It is easy to verify that $r < s$ if and only if one of the following three conditions holds:

$$\begin{aligned} (1) \quad u &= \left\lfloor \frac{x_2}{a_1} \right\rfloor \dot{-} \left(\left\lfloor \frac{x_1}{b_2} \right\rfloor + 1 \right), \quad \text{and} \\ &\left(\left\lfloor \frac{x_1}{b_2} \right\rfloor < \left\lfloor \frac{x_2}{a_1} \right\rfloor \right) \wedge \left(a_0 + a_3x_3 + \cdots + a_nx_n < b_0 \right. \\ &\quad \left. + \left(a_1b_2 + b_2 \operatorname{rem} \left(\frac{x_2}{a_1} \right) \dot{-} a_1 \operatorname{rem} \left(\frac{x_1}{b_2} \right) \right) + a_1b_2u + b_3x_3 + \cdots + b_nx_n \right). \end{aligned}$$

$$(2) \quad w = \left\lfloor \frac{x_1}{b_2} \right\rfloor \dot{-} \left(\left\lfloor \frac{x_2}{a_1} \right\rfloor + 1 \right), \quad \text{and}$$

$$\left(\left\lfloor \frac{x_1}{b_2} \right\rfloor > \left\lfloor \frac{x_2}{a_1} \right\rfloor \right) \wedge \left(a_0 + \left(a_1 b_2 + a_1 \operatorname{rem} \left(\frac{x_1}{b_2} \right) \dot{-} b_2 \operatorname{rem} \left(\frac{x_2}{a_1} \right) \right) \right. \\ \left. + a_1 b_2 w + a_3 x_3 + \cdots + a_n x_n < b_0 + b_3 x_3 + \cdots + b_n x_n \right).$$

$$(3) \quad \left(\left\lfloor \frac{x_1}{b_2} \right\rfloor = \left\lfloor \frac{x_2}{a_1} \right\rfloor \right) \wedge \left(a_0 + \left(a_1 \operatorname{rem} \left(\frac{x_1}{b_2} \right) \dot{-} b_2 \operatorname{rem} \left(\frac{x_2}{a_1} \right) \right) + a_3 x_3 + \cdots + a_n x_n \right. \\ \left. < b_0 + \left(b_2 \operatorname{rem} \left(\frac{x_2}{a_1} \right) \dot{-} a_1 \operatorname{rem} \left(\frac{x_1}{b_2} \right) \right) + b_3 x_3 + \cdots + b_n x_n \right).$$

Now each of these three cases can be split up, so at most $n-1$ variables will appear in the right inequality. For example, consider case (1). Let

$$h = a_1 b_2 + b_2 \operatorname{rem} \left(\frac{x_2}{a_1} \right) \dot{-} a_1 \operatorname{rem} \left(\frac{x_1}{b_2} \right).$$

Clearly, $0 \leq h \leq a_1 b_2 + b_2(a_1 - 1)$.

Hence, $a_0 + a_3 x_3 + \cdots + a_n x_n < (b_0 + h) + a_1 b_2 u + b_3 x_3 + \cdots + b_n x_n$ if and only if

$$\begin{aligned} & [(h = 0) \wedge (a_0 + a_3 x_3 + \cdots + a_n x_n < b_0 + a_1 b_2 u + b_3 x_3 + \cdots + b_n x_n)] \\ & \vee [(h = 1) \wedge (a_0 + a_3 x_3 + \cdots + a_n x_n < (b_0 + 1) + a_1 b_2 u + b_3 x_3 + \cdots + b_n x_n)] \\ & \vee \cdots \vee [(h = a_1 b_2 + b_2(a_1 - 1)) \wedge (a_0 + a_3 x_3 + \cdots + a_n x_n < (b_0 + a_1 b_2 + b_2(a_1 - 1)) \\ & \quad + a_1 b_2 u + b_3 x_3 + \cdots + b_n x_n)]. \end{aligned}$$

Similar formulas can be found for cases (2) and (3). This finishes the proof since we have reduced the inequalities involving n variables to inequalities involving at most $n-1$ variables. \square

LEMMA 13. *The function*

$$(r = s) = \begin{cases} 1 & \text{if } r = s \\ 0 & \text{otherwise,} \end{cases}$$

where r and s are terms, is \mathbb{V} computable.

Proof. This is \mathbb{V} computable by Lemmas 9 and 12. \square

LEMMA 14. *The function*

$$r \equiv_k s = \begin{cases} 1 & \text{if } r \equiv_k s \\ 0 & \text{otherwise,} \end{cases}$$

where r and s are terms and k is a positive integer, is \mathbb{V} computable.³

Proof. Let $r = a_0 + a_1 x_1 + \cdots + a_n x_n$ and $s = b_0 + b_1 x_1 + \cdots + b_n x_n$. For $1 \leq i \leq n$, let $y_i = \operatorname{rem}(x_i/k)$. By Lemma 10, y_i is \mathbb{V} computable. Now $y_i \leq k-1$ so $a_i y_i$ and $b_i y_i$ are \mathbb{V} computable by Lemma 11.

³ $r \equiv_k s$ denotes $r = s$ (modulo k).

Let $w = a_0 + a_1y_1 + \cdots + a_ny_n$ and $u = b_0 + b_1y_1 + \cdots + b_ny_n$. Again, by Lemma 11, w and u are V computable. Clearly, $r \equiv_k s$ if and only if $w \equiv_k u$. Let $w' = \text{rem}(w/k)$ and $u' = \text{rem}(u/k)$. Then $w \equiv_k u$ if and only if $w' = u'$ which is V computable by Lemma 9. \square

The next lemma concerns Q programs whose output variables can only assume values 0, 1.

LEMMA 15. *Every Q program whose output variables can only assume values 0, 1 can be converted into an equivalent V program.*

Proof. Let P be a Q program satisfying the hypothesis of the lemma. Assume without loss of generality that P has only one output variable. By Lemma 8 and Theorem 7, there is a Presburger formula F whose characteristic function is exactly the function computed by P. Now the formula F can be transformed into an equivalent quantifier-free formula $F'[3]$, and F' has the form:

$$\bigvee_{j \neq l} \bigvee_{q < m} \left[\bigwedge_{i \leq l} r_i + s_j < r_j + s_i + q \wedge \bigwedge_{i < k} r_j + u_i + q < t_i + s_j \wedge \bigwedge_{i < n} r_j + w_i + q \equiv_{m_i} v_i + s_j \right]$$

where l, q, m, k, n, m_i are nonnegative integer constants, and $r_i, s_i, t_i, u_i, v_i, w_i$ are terms. By Lemmas 9–14, we can construct a V program P' which computes the characteristic function of F' . \square

Generalizing Lemma 15, we have

THEOREM 10. *Let m be a positive integer. Every Q program whose output variables can only assume values 0, 1, \dots , m can be converted into an equivalent V program.*

Proof. Let P be a Q program. Again, we only consider the case when P has one output variable z whose range is 0, 1, \dots , m . Let the input variables of P be x_1, \dots, x_n . For each $1 \leq i \leq m$, construct the following program P_i (y and z_i are new variables):

$$\begin{aligned} &P \\ &y \leftarrow z \\ &y \leftarrow y \div i \\ &z_i \leftarrow (1 \div y)z \\ &z_i \leftarrow z_i \div (i - 1) \end{aligned}$$

Clearly, at the end of P_i , $z_i = 1$ if $z = i$ and $z_i = 0$ otherwise. Now P_i can be coded in Q. Hence, by Lemma 15, P_i can be transformed into an equivalent V program P'_i . Assume that the variables in P, P'_1, \dots, P'_m are distinct. For $1 \leq i \leq m$, let the input variables of P'_i be x_{i1}, \dots, x_{in} and its output variable be z_i . Then the following program P' is

equivalent to $P(z_0$ and z are new variables):

$$\begin{array}{l}
 \left. \begin{array}{l}
 x_{i1} \leftarrow x_1 \\
 x_{i2} \leftarrow x_2 \\
 \vdots \\
 x_{in} \leftarrow x_n
 \end{array} \right\} \text{repeat for } i = 1, 2, \dots, m \\
 P'_1 \\
 \vdots \\
 P'_m \\
 z_0 \leftarrow 1 \\
 z_0 \leftarrow z_0 \dot{-} z_1 \\
 \vdots \\
 z_0 \leftarrow z_0 \dot{-} z_m \\
 z \leftarrow m + 1 \\
 \left. \begin{array}{l}
 z \leftarrow z \dot{-} z_0 \\
 \vdots \\
 z \leftarrow z \dot{-} z_0
 \end{array} \right\} m + 1 \text{ times} \\
 \left. \begin{array}{l}
 z \leftarrow z \dot{-} z_1 \\
 \vdots \\
 z \leftarrow z \dot{-} z_1
 \end{array} \right\} m \text{ times} \\
 \vdots \\
 \left. \begin{array}{l}
 z \leftarrow z \dot{-} z_j \\
 \vdots \\
 z \leftarrow z \dot{-} z_j
 \end{array} \right\} m - j + 1 \text{ times} \\
 \vdots \\
 \left. \begin{array}{l}
 z \leftarrow z \dot{-} z_{m-1} \\
 z \leftarrow z \dot{-} z_{m-1}
 \end{array} \right\} 2 \text{ times} \\
 z \leftarrow z \dot{-} z_m
 \end{array}$$

P' has input variables x_1, \dots, x_n (which are also the input variables of P) and output variable z . It is straightforward to transform P' into an equivalent V program P'' . \square

Theorem 11 below shows that V is minimal.

THEOREM 11. *The instructions $x \leftarrow y + 1$, $x \leftarrow x \dot{-} y$, $x \leftarrow \lfloor x/k \rfloor$ are independent.*

Proof. One can easily verify the following statements using the ideas in the proofs of Lemmas 6 and 7:

- (1) Without $x \leftarrow y + 1$, the function $f(x) = 1$ for all x cannot be computed.

(2) Without $x \leftarrow x \dot{-} y$, the function

$$x < y = \begin{cases} 1 & \text{if } x < y \\ 0 & \text{otherwise} \end{cases} \quad \text{cannot be computed.}$$

(3) Without $x \leftarrow \lfloor x/k \rfloor$, the function $f(x) = \text{rem}(x/k)$ cannot be computed. \square

Notation. Let \mathcal{C} be a class of (possibly multiple-output) functions and m be a positive integer. Then \mathcal{C}_m will denote the set of functions in \mathcal{C} whose outputs can only assume the values $0, 1, \dots, m$.

From Lemma 8 and Theorems 6, 10 and 11, we have the main result of this section:

THEOREM 12. *For each positive integer m , $\mathcal{F}_m = \mathcal{P}_m = \text{Comp}_m(\hat{B})$ and $\mathcal{F}_m^* = \mathcal{P}_m^* = \text{Comp}_m^*(\hat{B})$, where \hat{B} consists only of the functions $U_i^n(x_1, \dots, x_n) = x_i$, $S(x) = x + 1$, $D(x, y) = x \dot{-} y$, and $T_k(x) = \lfloor x/k \rfloor$. Moreover, the functions in \hat{B} are independent.*

REFERENCES

- [1] J. CHERNIAVSKY, *Simple programs realize exactly Presburger formulas*, this Journal, 5 (1976), pp. 666–677.
- [2] J. CHERNIAVSKY AND S. KAMIN, *A complete and consistent Hoare axiomatics for a simple programming language*, J. Assoc. Comput. Mach., 26 (1979), pp. 119–128.
- [3] H. ENDERTON, *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.
- [4] S. GINSBURG AND E. SPANIER, *Semigroups, Presburger formulas, and languages*, Pacific J. Math., 16 (1966), pp. 285–296.
- [5] A. GRZEGORCZYK, *Some classes of recursive functions*, Rozprawy Matemat., 4 (1953), pp. 1–45.
- [6] E. GURARI AND O. IBARRA, *The complexity of the equivalence problem for simple programs*, Tech. Rep. No. 78–17, University of Minnesota, August, 1978, J. Assoc. Comput. Mach., to appear.
- [7] E. GURARI AND O. IBARRA, *The complexity of the equivalence problem for counter machines, semilinear sets, and simple programs*, Proc. of the Eleventh Annual ACM Symposium on Theory of Computing, 1979, pp. 142–152.
- [8] K. HARROW, *Small Grzegorzczk classes and limited minimum*, Z. Math. Logik Grundlag. Math., 21 (1975), pp. 417–426.
- [9] K. HARROW, *Sub-elementary classes of functions and relations*, Ph.D. Thesis, New York University, 1973.
- [10] A. MEYER AND D. RITCHIE, *The complexity of loop programs*, Proceedings of the Twenty-Second National Conference of the ACM, Thompson Book Co., Washington, DC, 1967, pp. 465–469.
- [11] D. TSICHRITZIS, *The equivalence problem of simple programs*, J. Assoc. Comput. Mach., 17 (1970), pp. 729–738.

ON ETOL SYSTEMS WITH FINITE TREE-RANK*

A. EHRENFEUCHT†, G. ROZENBERG‡ AND D. VERMEIR§

Abstract. This paper studies an extension of the notion of a finite index ETOL system. It turns out that by setting some quite natural restrictions on the set of bare derivation trees of an ETOL system (that is derivation trees stripped of labels) one can characterize languages of finite rank. Several properties of the new class of ETOL systems are investigated; in particular their relationship to ETOL systems of finite rank and ETOL systems of finite index is investigated.

Key words. formal language theory, ETOL systems, finite index, tree complexity

Introduction. The notion of finite index is a classical one in formal language theory. Imposing the finite index restriction is an established way to investigate the structural properties of a rewriting system.

One of the essential differences between sequential and parallel rewriting systems, i.e., between context-free grammars and ETOL systems, is that ETOL systems allow for a meaningful and interesting extension of the finite index notion, namely the notion of ETOL systems with rank. (Indeed, it was shown in [1] that the rank restriction, when applied to context-free grammars, is equivalent to the finite index restriction.) In [1], [8] and [9] it was demonstrated that the notion of rank in ETOL systems forms an important extension of the finite index concept and allows us to learn quite a bit about the structure of ETOL systems.

However, when looked upon carefully, the definition of (finite) rank seems to be too detailed: it talks too much about the actual form of the words belonging to the set of sentential forms.

Hence a very natural question is whether one can give an alternative definition which would set the minimal restrictions on ETOL systems and hence would allow one to see the essentials behind the property of rank.

The aim of this paper is to present such an alternative characterization. We feel that the characterization we have provided is surprisingly simple. It relies on the structure of the bare derivation trees (i.e., derivation trees with labels erased) only! The paper is organized as follows.

In the first section we recall some basic terminology and notation concerning trees and ETOL systems. In § II we define the notion of the rank of a tree and we investigate several of its properties. In § III we investigate the effect of imposing the finite tree-rank restriction on ETOL systems. In particular we show that the finite tree-rank property is decidable, and establish an infinite hierarchy of classes of languages associated with ETOL systems of finite tree-rank. The class of ETOL languages of finite tree-rank is then characterized in several ways in § IV: it is shown to be equal to the class of languages generated by strongly nonexpansive ETOL systems and to the class of ETOL languages with rank. In § V we further investigate the relation between the notion of rank and tree-rank of ETOL systems by comparing the respective hierarchies of language families. Finally, in § VI we mention some closure properties of the classes of ETOL languages with finite tree-rank.

* Received by the editors August 9, 1978, and in revised form October 24, 1979.

† Department of Computer Science, University of Colorado at Boulder, Boulder, Colorado.

‡ Department of Computer Science, University of Leiden, Leiden, Holland.

§ Department TEW, University of Leuven, Leuven, Belgium.

I. Preliminaries. The reader is assumed to be familiar with basic formal language theory and in particular with the basic theory of L systems, e.g., in the scope of [3]. In addition to the standard notation, we will also use the following.

First of all, we often do not distinguish between a singleton and its element, e.g., $\{\Lambda\}$ will often be denoted by Λ . Let V be an alphabet and let x be a word over V .

(1) For $k > 0$, we use $V^{\leq k}$ to denote the set of all nonempty words over V with length not exceeding k ; i.e., $V^{\leq k} = \{x \in V^+ : |x| \leq k\}$.

(2) For every alphabet $\Delta \subset V$, we define a homomorphism $\text{Pres}_\Delta : V^* \rightarrow \Delta^*$ by

$$\text{Pres}_\Delta(a) = \begin{cases} a & \text{if } a \in \Delta, \\ \Lambda & \text{otherwise.} \end{cases}$$

(3) For a set $\Delta \subset V$, we use $\#_\Delta(x)$ to denote the number $|\text{Pres}_\Delta x|$ and $\text{alph } x$ to denote the set $\{a : \#_a x > 0\}$.

(4) For $1 \leq i \leq |x|$, we use $x(i)$ to denote the i th symbol in x .

To establish some terminology and notation we recall now several notions concerning trees.

DEFINITION 1.

(1) A (finite) *tree* is a pair $T = (A, R)$ where A is a finite nonempty set of *nodes* and $R \subseteq A \times A$ is a set of *edges* with the following properties:

(i) there exists a special node $r \in A$, called the *root* of T , which is such that $(x, r) \notin R$ for all $x \in A$,

(ii) for every $a \in A \setminus \{r\}$, there exists precisely one b in A such that $(b, a) \in R$.

(iii) for every $a \in A \setminus \{r\}$, there exists a unique sequence $a = b_0, b_1, \dots, b_n = r$, $n \geq 1$, of nodes such that $(b_{i+1}, b_i) \in R$ for all $0 \leq i < n$.

T is said to be ordered if, for every $a \in A \setminus \{r\}$, the set $\{b : (a, b) \in R\}$ is linearly ordered. Let $T = (A, R)$ be a tree.

(2) A *path* in T is a word $b_0 \cdots b_n \in A^+$, $0 \leq n$, which is such that $(b_i, b_{i+1}) \in R$ for all $0 \leq i < n$. We say that $b_0 \cdots b_n$ is a path from b_0 to b_n in T . The *length* of a path $b_0 \cdots b_n$ is n .

(3) The *height* of T , denoted as $\text{Hgt}(T)$, is the length of a path p in T such that no other path in T is longer than the path p .

(4) For $l \in \mathbb{N}$, let $P_T(l)$ be the set of all paths of length l starting at the root. The *width* of T , denoted $\text{Wdh}(T)$, is defined by $\text{Wdh}(T) = \max_{l \geq 0} \{\#P_T(l)\}$.

(5) For a node $a \in A$, the set of *direct descendants* of a , denoted as $\text{Desc}_T a$, is defined by $\text{Desc}_T a = \{b \in A : (a, b) \in R\}$.

(6) A node $a \in A$ is called a *leaf* if $\text{Desc}_T a = \emptyset$.

(7) T is called *full binary* if $\#\text{Desc}_T a \in \{0, 2\}$ for all $a \in A$.

(8) T is called *parallel full binary*, abbreviated as p.f.b., if T is full binary and all paths from the root to a leaf have the same length.

(9) For a node $a \in A$, the *subtree* of a , denoted by T_a , is defined by $T_a = (A_a, R_a)$ where A_a is the set of all nodes x such that there is a path from a to x and $R_a = R \cap A_a \times A_a$.

(10) For a node $a \in A$, the *level* of a in T , denoted $\text{level}_T(a)$, is defined by $\text{level}_T a = \text{Hgt}(T_a)$.

DEFINITION 2. Let $T = (A, R)$ be a tree and let $f : A \rightarrow A$ be a partial function such that the root of T is in the domain of f and $f(a) = a$ for all $a \in \text{Dom } f$. Then $f(T)$ is a tree defined by $f(T) = (f(A), R_f)$ where $(a, b) \in R_f$ if and only if there exists a path $ac_1 \cdots c_n b$, $n \geq 0$, in T such that $c_i \notin \text{Dom } f$ for $1 \leq i \leq n$. We then say that T *contains* $f(T)$.

DEFINITION 3. Let $T = (A, R)$ be a tree and let \mathcal{D} be a set of trees. The set of trees $\text{Hang}(T, \mathcal{D})$ is defined as follows. A tree $T' = (A', R') \in \text{Hang}(T, \mathcal{D})$ if and only if there exist an integer $n \geq 0$ and a subset $\{T_i = (A_i, R_i) : 0 \leq i \leq n\}$ of \mathcal{D} such that the following hold (we assume that A and A_i , $0 \leq i \leq n$, are mutually disjoint):

- (i) $A' = \bigcup_{i=0}^n A_i \cup A$, and
- (ii) for every $0 \leq i \leq n$ there exists exactly one node $a_i \in A$ such that $R' = \bigcup_{i=0}^n R_i \cup R \cup \{(a_i, r_i) : r_i \text{ is the root of } T_i\}$. Thus, intuitively, the set $\text{Hang}(T, \mathcal{D})$ contains all trees obtained by attaching some trees from \mathcal{D} to T .

DEFINITION 4. A *labeled tree* is a 4-tuple $T = (A, R, V, f)$ where (A, R) is a tree, V is a finite alphabet, and f is a total function from A into V . For each node $a \in A$, $f(a)$ is called the *label* of a .

Next we recall some basic definitions and notation concerning ETOL systems.

(0) An ETOL system is a construct $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ where V is a finite alphabet, $\Sigma \subseteq V$ is the set of terminal symbols, $S \in V$ is the start symbol and \mathcal{P} is a finite set of finite substitutions $T : V \rightarrow 2^V$. An element T of \mathcal{P} is usually called a table and denoted as a set of productions

$$T = \{a \rightarrow \alpha : a \in V, \alpha \in T(a)\}.$$

For words $x \in V^*$ we write $x \xrightarrow{G} y$ (or $x \Rightarrow y$ if G is understood) if $y \in T(x)$ for some $T \in \mathcal{P}$. We may also write $x \xrightarrow{T} y$ to stress the fact that we use T . As usual the “derives” relation \xrightarrow{G}^* (or \Rightarrow^*) is the reflexive and transitive closure of G . The language of G , denoted $L(G)$, is defined by $L(G) = \{w \in \Sigma^* : S \Rightarrow^* w\}$.

Let $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ be an ETOL system.

- (1) G is a *deterministic* ETOL system, abbreviated EDTOL system, if every table is a homomorphism.
- (2) G is *synchronized* if for every terminal symbol $a \in \Sigma$, $a \Rightarrow^* x$ implies $x \notin \Sigma^*$.
- (3) We use $\text{maxr } G$ to denote the length of the longest right hand side of any production in G .
- (4) For Δ a subset of V , we say that G is *deterministic in* Δ if $\#\{\alpha : a \xrightarrow{T} \alpha\} = 1$ for all $a \in \Delta$ and $T \in \mathcal{P}$.
- (5) The *deterministic version* $(G)_D$ of G is the unique EDTOL system $(G)_D = \langle V, \bar{\mathcal{P}}, S, \Sigma \rangle$ where $P \in \bar{\mathcal{P}}$ if P is a homomorphism and $P \subseteq T$ for some $T \in \mathcal{P}$.
- (6) A symbol $a \in V$ is called *nonactive* if $a \xrightarrow{G} \alpha$ implies that $\alpha = a$. We say that

G is in *Active Normal Form* (abbreviated as ANF) if every terminal is nonactive. Let $D : x_0 \xrightarrow{G} x_1 \xrightarrow{G} \cdots \xrightarrow{G} x_n \in V^*$ be a derivation in G with T_D being the corresponding derivation tree.

(7) Let α be a subword of some x_i ($0 \leq i \leq n$). Then, for $j \geq 0$, we use $\text{contr}_D(\alpha, j)$ to denote the subword of x_j formed by the descendants of α ($\text{contr}_D(\alpha, j)$ is undefined if $j < i$.) We also use $\text{contr}_D(\alpha)$ to denote $\text{contr}_D(\alpha, n)$. If $\text{contr}_D(A) \neq \Lambda$ for some occurrence A , then A is said to be *productive* and we also say that A *produces* $\text{contr}_D(A)$.

(8) D is said to be *proper* if $x_i \neq x_j$ for all $i \neq j$.

(9) For a subword z of an intermediate word x_i ($0 \leq i \leq n$) we use $\langle z \rangle$ to denote the word obtained from z by erasing all nonproductive occurrences.

(10) D is said to be *deterministic* if $x_0 \Rightarrow x_1 \Rightarrow \cdots \Rightarrow x_n$ is also a derivation in $(G)_D$.

(11) Let y be a subword of x_n . For every $0 \leq i \leq n$, $\text{anc}_D(y, i)$ denotes the minimal subword y_i of x_i which is such that $\text{contr}_D(y_i)$ contains y as a subword.

(12) We use $\mathcal{U}(G)$ to denote set $\{\text{alph } x : S \xrightarrow[G]{*} x \xrightarrow[G]{*} y \in \Sigma^*\}$ of *useful alphabets* of G . We also use $\text{Succ } G$ to denote the language $\{x \in V^* : x \xrightarrow[G]{*} y \text{ for some } y \in \Sigma^*\}$.

In [6] it has been shown that both $\mathcal{U}(G)$ and $\text{Succ } G$ can be effectively constructed.

(13) We say that G is *nonexpansive* if for every $a \in V$, $a \xrightarrow[G]{*} \alpha$ implies that $\#_a \alpha \leq 1$.

(14) We say that G is of *index* k for some $k \geq 1$ if for every word w in $L(G)$ there exists a derivation D of it such that no word in D contains more than k occurrences of active symbols. G is said to be of *finite index* if G is of index k for some $k \geq 1$. We use $\mathcal{L}(\text{ETOL})_{\text{FIN}}$ to denote the class of languages generated by ETOL systems of finite index.

Finally we recall the definition of an ETOL system with rank (see [1]).

DEFINITION 5. Let $G = \langle V, \mathcal{R}, S, \Sigma \rangle$ be an ETOL system.

(1) For a word $\alpha \in V^*$ and a subset Z of V we define the sets $\text{SUCC}_{G,Z}(\alpha)$ and $\text{NSUCC}_{G,Z}(\alpha)$, denoted as $\text{SUCC}_Z(\alpha)$ and $\text{NSUCC}_Z(\alpha)$ if G is understood, as follows.

(i) $\text{SUCC}_{G,Z}(\alpha) = \{\text{Pres}_Z(x) : \alpha \xrightarrow[G]{*} x \xrightarrow[G]{*} y \text{ for some } y \in \Sigma^*\}$ where Pres_Z is a homomorphism on V^* defined by $\text{Pres}_Z a = a$ if $a \in Z$ and $\text{Pres}_Z a = \Lambda$ if $a \in V \setminus Z$.

(ii) $\text{NSUCC}_{G,Z}(\alpha) = \{|x| : x \in \text{SUCC}_{G,Z}(\alpha)\}$.

(2) The (in general partial) function $\text{rank}_G : V \rightarrow \mathbb{N}$, denoted as *rank* if G is understood, is defined recursively as follows.

(i) Let $Z_0 = V$. Then $\text{rank}_G(a) = 0$ if and only if $\text{SUCC}_{G,Z_0}(a)$ is a finite set.

(ii) For $i \geq 0$, let $Z_{i+1} = V \setminus \{a \in V : \text{rank}_G a \leq i\}$. Then, for $a \in Z_{i+1}$, $\text{rank}_G(a) = i + 1$ if and only if $\text{SUCC}_{G,Z_{i+1}}(a)$ is a finite set. For $i \geq 0$, $R_i(G)$ denotes the set $\text{rank}_G^{-1}(i)$.

(3) We say that G is an *ETOL system with rank* if and only if rank_G is a total function on V . Moreover we say that G is of rank m , $m \geq 0$, denoted as $\text{rank } G = m$, if $R_m(G) \neq \emptyset$ and $R_i(G) = \emptyset$ for all $i > m$.

We will use $(\text{ETOL})_{\text{RAN}(i)}$, $i \geq 0$, and $(\text{ETOL})_{\text{RAN}}$ to denote the set of ETOL systems with rank not bigger than i and the set of ETOL systems with rank respectively. As usual, $\mathcal{L}(\text{ETOL})_{\text{RAN}(i)}$ and $\mathcal{L}(\text{ETOL})_{\text{RAN}}$ denote the corresponding classes of languages.

II. Rank on trees. In this section the notion of tree-rank is introduced and its basic properties are investigated. Informally speaking, the rank of a tree T is computed in a bottom-up fashion as follows. Every leaf has rank 0. Let c be a node in T and let i be the maximal rank of the descendants of c . If c has at least two descendants with rank i , then the rank of c is $i + 1$, otherwise the rank of c is i . The rank of T is then set equal to the rank of its root. Here is a formal definition.

DEFINITION 6. Let $T = (A, R)$ be a tree with root r .

(1) The *tree-rank function* of T , denoted r_T , is a mapping from A into \mathbb{N} which is defined as follows.

(i) If $a \in A$ is a leaf, then $r_T(a) = 0$.

(ii) If $a \in A$ is not a leaf then we consider the number $M_T(a) = \max \{r_T(b) : b \in \text{Desc } a\}$. Then

$$r_T(a) = \begin{cases} M_T(a) & \text{if } \# r_T^{-1}(M_T(a)) \cap \text{Desc } a = 1, \text{ and} \\ M_T(a) + 1 & \text{otherwise.} \end{cases}$$

(2) The *rank* of T , denoted $\text{rank } T$, is defined by $\text{rank } T = r_T(r)$. We will often refer to $r_T(a)$ as the *tree-rank* of a (in T).

Obviously, the rank of a p.f.b. tree equals its height. This observation is generalized in the following theorem which provides a characterization of trees of rank k .

THEOREM 1. *A tree has rank k if and only if it contains a p.f.b. tree of height k and it does not contain a p.f.b. tree of height bigger than k .*

Proof. The proof goes by induction on k .

(1) First we show that the theorem holds for $k = 0$. To show the *if*-part, let T be a tree which contains a p.f.b. tree of height 0 but does not contain a p.f.b. tree of height more than 0. This immediately implies that T has but one leaf, and from this and Definition 6 it readily follows the rank $T = 0$. To show the *only-if* part, let $T = (A, R)$ be a tree of rank 0. This implies that $\# \text{Desc}(a) \leq 1$ for every a in A and thus T does not contain a p.f.b. tree of height larger than 0. Together with the fact that every tree contains a p.f.b. tree of height 0 this completes the proof for the case $k = 0$.

(2) Let us assume that the theorem holds for $k = l$.

(3) We show that the theorem holds for $k = l + 1$.

To show the *only-if* part, let T be a tree of rank $l + 1$. Let a_0 be a fixed node with the property that $r_T(a_0) = l + 1$ and $M_T(a_0) = l$. (Note that the hypothesis and the definition of rank T guarantee the existence of such a node a_0 .) From the definition of r_T it then follows that there exist two nodes a_1, a_2 in $\text{Desc}(a_0)$ such that $r_T(a_1) = r_T(a_2) = l$. Hence rank $T_{a_1} = \text{rank } T_{a_2} = l$. By the inductive assumption T_{a_1} and T_{a_2} both contain a p.f.b. tree of height l . Let f_1 and f_2 be the associated mappings and define $f = f_1 \cup f_2 \cup (a_0, a_0)$. It follows immediately that $f(T)$ is a p.f.b. tree of height $l + 1$. In order to show that T does not contain a parallel full binary tree of height greater than $l + 1$, we first prove the following.

CLAIM. *If T contains a p.f.b. tree T' and a is a node in T' with $\text{level}_{T'}(a) = l$, then $r_T(a) \geq l$.*

Proof of the claim. The proof is by induction on l .

(i) The claim holds trivially if $l = 0$.

(ii) Let us assume that the claim holds for $l = t$.

(iii) Let a be a node in T' with $\text{level}_{T'}(a) = t + 1$ and let $\text{Desc}_{T'}(a) = \{a_1, a_2\}$. Obviously, $\text{level}_{T'}(a_1) = \text{level}_{T'}(a_2) = t$ and thus $r_T(a_1) \geq t$ and $r_T(a_2) \geq t$. Let $ab_0 \cdots b_n a_1 (n \geq 0)$ and $ac_0 \cdots c_n a_2 (n \geq 0)$ be the paths in T from a to a_1 and to a_2 . It is then clear that $r_T(b_0) \geq t$ and $r_T(c_0) \geq t$. Since b_0 and c_0 are both in $\text{Desc}_T(a)$, this implies that $r_T(a) \geq t + 1$, thus completing the proof of the claim.

From the above claim it immediately follows that if T contains a p.f.b. tree of height greater than $l + 1$ then $\text{rank}_T > l + 1$, a contradiction. This completes the proof of the *only-if* part.

To show the *if* part, let T be a tree containing a p.f.b. tree B of height l but not containing a p.f.b. tree of height $l + 1$. From the claim it then follows that $\text{rank } T \geq l + 1$. On the other hand the *only-if* part of the theorem implies that if $\text{rank}_T > l + 1$, then T would contain a p.f.b. tree of height greater than $l + 1$, a contradiction. Thus $\text{rank } T = l + 1$ and the theorem holds. \square

Next we note that the rank of a tree is always strictly bounded by its width.

THEOREM 2. *Let T be a tree. Then $\text{rank}(T) < \text{Wdh}(T)$.*

The following lemma will be useful in the sequel; we omit the proof.

LEMMA 1. *Let T be a tree of rank not greater than k_1 , for some $k_1 \geq 0$, and let \mathcal{D} be a family of trees which is such that there exists an integer k_2 such that $\text{rank}(T') \leq k_2$ for all T' in \mathcal{D} . Then $\text{rank}(\bar{T}) \leq k_1 + k_2 + 1$ for all \bar{T} in $\text{Hang}(T, \mathcal{D})$.*

Up to now we have only considered the notion of rank on unlabeled trees. However, one can make some interesting observations on the relationship between

special kinds of labeled trees, namely nonexpansive trees, and their rank. Intuitively, a labeled tree is expansive if there exists a node a and two paths from a to two (different) nodes b and c such that a , b and c have the same label. Formally, we have the following definition.

DEFINITION 7. A labeled tree $T = (A, R, V, f)$ is called *expansive* if there exists a node $a \in A$ and two paths $ab_1 \cdots b_n$ and $ac_1 \cdots c_m$ ($m, n \geq 1$) in T such that $b_n \neq c_m$ and $f(b_n) = f(c_m) = f(a)$. A labeled tree T is called *nonexpansive* if T is not expansive.

LEMMA 2. *If $T = (A, R, V, f)$ is a nonexpansive labeled tree of rank k , then $\#V \geq k + 1$.*

Proof. By Theorem 1 it suffices to show that if $B = (A, R, V, f)$ is a nonexpansive labeled p.f.b. tree of height k then $\#V \geq k + 1$. The proof of this goes by induction on k .

(1) Let $k = 1$; thus $B = (\{a_0, a_1, a_2\}, \{(a_0, a_1), (a_0, a_2)\}, V, f)$. Since $f(a_0)$, $f(a_1)$ and $f(a_2)$ cannot be equal, it follows that $\#V > 1$.

(2) Let us assume that the lemma holds for all nonexpansive labeled p.f.b. trees of height t .

(3) Let $B = (A, R, V, f)$ be a nonexpansive labeled p.f.b. tree of height $t + 1$ with root a_0 and let $\text{Desc}(a_0) = \{a_1, a_2\}$. Hence $\text{Hgt}(T_{a_1}) = \text{Hgt}(T_{a_2}) = t$. Let A_1 and A_2 be the set of nodes of T_{a_1} and T_{a_2} respectively. By the induction hypothesis we know that $\#f(A_1) \geq t$ and $\#f(A_2) \geq t$. If $f(A_1) \neq f(A_2)$ then, trivially, $\#V \geq \#f(A) \geq t + 1$. If $f(A_1) = f(A_2)$ then $f(a_0) \notin f(A_1)$ since otherwise B would be expansive. Hence $\#V \geq \#f(A_1) + 1 = t + 1$. This completes the induction and thus the lemma holds. \square

The next example shows that the bound from Lemma 2 is optimal.

Example. Let $T = (A, R)$ be a tree of rank not greater than k . Take $V = \{0, 1, \dots, k\}$ and define $f: A \rightarrow V$ by $f(a) = r_T(a)$ for all a in A . It is then a straightforward matter to show that $\bar{T} = (A, R, V, f)$ is nonexpansive.

III. ETOL systems and tree-rank. Now we turn to ETOL systems and their (sets of) derivation trees. We start with the following definition.

DEFINITION 8. Let $G = \langle V, \mathcal{P}, \mathcal{S}, \Sigma \rangle$ be an ETOL system. By \mathcal{D}_G we denote the set of (labeled, ordered) *derivation trees* of G . The set \mathcal{B}_G of *bare derivation trees* of G is defined by $\mathcal{B}_G = \{(A, R) : (A, R, V, f) \in \mathcal{D}_G\}$.

Next we define the concept of tree-rank on ETOL systems. Intuitively, an ETOL system is of tree-rank k for some $k \geq 0$ if every derivation tree T from \mathcal{D}_G has a rank not greater than k , and at least one tree $T' \in \mathcal{D}_G$ has rank k .

DEFINITION 9. Let $G = \langle V, \mathcal{P}, \mathcal{S}, \Sigma \rangle$ be an ETOL system.

(1) We say that G is of *tree-rank* k for some $k \geq 0$, denoted as $\text{tr}(G) = k$, if *rank* $T \leq k$ for every T in \mathcal{B}_G and *rank* $T' = k$ for at least one tree T' from \mathcal{B}_G .

(2) We say that G is of *finite tree-rank* if $\text{tr}(G) = k$ for some $k \geq 0$.

We will use $(\text{ETOL})_{\text{TR}(k)}$ and $(\text{ETOL})_{\text{TR}}$ to denote the class of ETOL systems of tree-rank not bigger than k and of finite tree-rank respectively.

As usual, we denote the class of languages generated by a class of rewriting systems X by $\mathcal{L}X$, yielding expressions like $\mathcal{L}(\text{ETOL})_{\text{TR}(k)}$ and $\mathcal{L}(\text{ETOL})_{\text{TR}}$.

The following theorem shows that being of tree-rank k is a decidable property in the class of ETOL systems.

THEOREM 3. *There exists an algorithm which, given an arbitrary ETOL system G and a nonnegative integer k , decides whether or not G is of tree-rank k .*

Proof. It clearly suffices to show that for an arbitrary ETOL system G and a nonnegative integer t , it is decidable whether or not $\text{tr}(G) \leq t$. So let $G = \langle V, \mathcal{P}, \mathcal{S}, \Sigma \rangle$ be an ETOL system and let $t \geq 0$ be a nonnegative integer. (Clearly we can assume that $L(G) \neq \emptyset$.) We construct a ‘‘bracketed version’’ G' of G as follows.

Let $[$ and $]$ be new symbols. Define new alphabets $V' = V \cup \{[,]\}$ and $\Sigma' = \Sigma \cup \{[,]\}$. For every table T from \mathcal{P} we construct a new table

$$T' = T \cup \{a \rightarrow [\alpha] : a \xrightarrow{T} \alpha \text{ and } |\alpha| > 1\} \cup \{[\rightarrow [,] \rightarrow]\}.$$

Obviously, $\text{tr}(G') = \text{tr}(G)$.

For every $i \geq 0$ we inductively define a word w_i in $\{[,]\}^*$ as follows:

- (1) $w_0 = \Lambda$, and
- (2) $w_{i+1} = [w_i w_i]$ for every $i > 0$.

It is then not difficult to show (by induction on t) that G' is of tree-rank greater than t if and only if $L(G')$ contains a word u such that $\text{Pres}_{\{[,]\}} u = w_{t+1}$. One can easily construct an a -transducer M_{t+1} which is such that $M_{t+1}(L(G')) \neq \{\Lambda\}$ if and only if $L(G')$ contains a word u such that $\text{Pres}_{\{[,]\}} u = w_{t+1}$. But $\mathcal{L}ETOL$ is closed under a -transducer mappings [3]. Together with the obvious fact that it is decidable whether or not an ETOL system generates a nonempty word. This implies the theorem. \square

It is also decidable whether or not an arbitrary ETOL system is of finite tree-rank. This is shown in Theorem 11 (§ IV).

We will now prove a normal form theorem for ETOL systems of tree-rank k which will be useful in the sequel. This result will also imply that every ETOL language of finite tree-rank can be generated by a strongly nonexpansive ETOL system, i.e., an ETOL system which has only nonexpansive derivation trees. First we need a definition.

DEFINITION 10. An ETOL system G is called *strongly nonexpansive* if every tree in \mathcal{D}_G is nonexpansive.

We show that the property of an ETOL system being strongly nonexpansive is decidable.

THEOREM 4. *There exists an algorithm which given an arbitrary ETOL system, decides whether or not it is strongly nonexpansive.*

Proof. Let $G = \langle V, \mathcal{P}, \mathcal{S}, \Sigma \rangle$ be an ETOL system. We construct a “bracketed version” G' of G as follows. Define new alphabets,

$$Z = \left\{ \begin{bmatrix} [\\ a \end{bmatrix} : a \in V \right\},$$

$$V' = V \cup Z \cup \{\$, \}, \text{ where } \$ \text{ is a new symbol,}$$

and

$$\Sigma' = \Sigma \cup Z.$$

For every table T from \mathcal{P} we construct a new table

$$T' = T \cup \left\{ a \rightarrow \begin{bmatrix} \alpha \\ a \end{bmatrix} : a \xrightarrow{T} \alpha \right\} \cup \{X \rightarrow X : X \in Z\} \cup \left\{ \mathcal{S} \rightarrow \begin{bmatrix} \mathcal{S} \\ \mathcal{S} \end{bmatrix} \right\}.$$

Let $\mathcal{P}' = \{T' : T \in \mathcal{P}\}$. Consider the ETOL system $G' = \langle V', \mathcal{P}', \mathcal{S}, \Sigma' \rangle$. Clearly, G' is strongly nonexpansive if and only if G is. On the other hand, it is not difficult to show that G' is strongly nonexpansive if and only if

$$(*) \quad \text{Pres}_{\{[,]\}}(u) = \begin{bmatrix} [& [& [\\ a & a & a \\] &] &] \end{bmatrix} \text{ for some } u \in L(G'), a \in V.$$

One can then easily construct an a -transducer M which is such that $M(L(G)) \neq \{\Lambda\}$ if and only if $(*)$ holds. But $\mathcal{L}ETOL$ is closed under a -transducer mappings, and it is

decidable whether or not an arbitrary ETOL language contains a nonempty word; hence the theorem holds. \square

Next we can prove the “tree-rank Normal Form” theorem. Roughly speaking, an ETOL system is in tree-rank Normal Form if it is in ANF, strongly nonexpansive, every derivation can be extended to a successful one and each symbol “ a ” (except for the axiom) has a unique “tree-rank” t_a associated with it; i.e., every node with label a in every derivation tree has rank t_a . Here is a formal definition.

DEFINITION 11. Let $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ be an ETOL system of tree-rank k , ($k \geq 0$). We say that G is in *tree-rank Normal Form*, abbreviated as TRNF, if the following holds.

- (i) G is propagating.
- (ii) There exists a partition (V_0, \dots, V_k) of $V \setminus S$ such that if a is a node, different from the root, of a derivation tree $T \in \mathcal{D}_G$ then $r_T(a) = t$, $0 \leq t \leq k$, if and only if the label of a is in V_t . Moreover, G is deterministic in V_k and S does not appear at the right-hand side of any production in G .
- (iii) $\text{SUCC}(G) = V^*$.
- (iv) G is strongly nonexpansive.
- (v) G is in ANF.

The following lemma will be useful in the sequel.

LEMMA 3. *There exists an algorithm which, given an arbitrary ETOL system of tree-rank k , will produce an equivalent ETOL system with tree-rank not larger than k which is propagating and synchronized.*

Proof.

(1) Let $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ be an ETOL system of tree-rank k . We show that the standard algorithm (see, e.g., [3]) to produce an equivalent EPTOL system does not increase the tree-rank. This can be seen as follows.

Let $G' = \langle V', \mathcal{P}', S', \Sigma \rangle$ be the equivalent EPTOL system produced from G by the standard algorithm. It is then easy to show that every tree T' from $\mathcal{B}_{G'}$ is contained in some tree T from \mathcal{B}_G and thus, by Theorem 1,

$$\text{rank}(T') \leq \text{rank}(T).$$

Consequently,

$$\text{tr}(G') \leq \text{tr}(G).$$

(2) Let $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ be an EPTOL system of tree-rank k . Clearly we can assume that $S \in V \setminus \Sigma$. Define a new alphabet $V' = V \cup \{N_a : a \in \Sigma\} \cup \ell$ where $N_a, a \in \Sigma$ and ℓ are new symbols. Let $\psi : V^* \rightarrow V'^*$ be the homomorphism defined by

$$\psi(a) = \begin{cases} a & \text{if } a \in V \setminus \Sigma, \text{ and} \\ N_a & \text{if } a \in \Sigma. \end{cases}$$

For every table P from \mathcal{P} we define a new table

$$P' = \{\psi(a) \rightarrow \psi(\alpha), \psi(a) \rightarrow \alpha : a \xrightarrow{P} \alpha\} \cup \{a \rightarrow \ell : a \in \Sigma \cup \ell\}.$$

Let $\mathcal{P}' = \{P' : P \in \mathcal{P}\}$. Consider the EPTOL system $G' = \langle V', \mathcal{P}', S, \Sigma \rangle$. Obviously, $L(G') = L(G)$ and G' is synchronized. On the other hand, it is a straightforward matter to show that $\mathcal{B}_{G'} \subseteq \mathcal{B}_G$ and thus $\text{tr}(G') \leq \text{tr}(G)$. The lemma then follows from (1) and (2). \square

THEOREM 5. *There exists an algorithm which, given an arbitrary ETOL system of tree-rank k , will produce an equivalent ETOL system with tree-rank not bigger than k which is in tree-rank Normal Form.*

Proof. Let $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ be an ETOL system of tree-rank k for some $k \geq 0$. By Lemma 3 we can assume that G is propagating and synchronized. For every $0 \leq i \leq k$, let $V_i = \{[A, i] : A \in V \setminus \Sigma\}$ be a set of new symbols.

Define a new alphabet

$$V' = \bigcup_{i=0}^k V_i \cup \Sigma \cup \{\$, \ell\}, \quad \text{where } \$ \text{ and } \ell \text{ are new symbols.}$$

For every table P from \mathcal{P} we define a new table P' as follows:

$$(1.1) \text{ If } S \xrightarrow{P} \alpha \text{ for some } \alpha \in \Sigma^+, \text{ then } S \xrightarrow{P'} \alpha.$$

(1.2) If $S \xrightarrow{P} A_1 \cdots A_t$ for some $t \geq 1$ with $A_1, \dots, A_t \in V \setminus \Sigma$, then $S \rightarrow [A_1, i_1] \cdots [A_t, i_t]$ for every t -tuple (i_1, \dots, i_t) which is such that $1 \leq i_j \leq k$ for all $1 \leq j \leq t$ and, moreover, $\#\{j : i_j = k\} \leq 1$.

(1.3) If $A \xrightarrow{P} \alpha$ for some $a \in \Sigma^+$, $A \in V \setminus \Sigma$, then $[A, 0] \xrightarrow{P'} \alpha$ if $|\alpha| = 1$ and $[A, 1] \xrightarrow{P'} \alpha$ if $|\alpha| > 1$.

(1.4) If $A \xrightarrow{P} A_1 \cdots A_t$ for some $t \geq 1$, $A_1, \dots, A_t \in V \setminus \Sigma$ then $[A, i] \xrightarrow{P'} [A_1, i_1] \cdots [A_t, i_t]$ for every $0 \leq i \leq k$ and t -tuple (i_1, \dots, i_t) of nonnegative integers which is such that one of the following holds:

- (i) either $\#\{j : i_j = i\} = 1$ and $i_j \leq i$ for all $1 \leq j \leq t$,
- (ii) or $\#\{j : i_j = i - 1\} > 1$ and $i_j \leq i - 1$ for all $1 \leq j \leq t$.

$$(1.5) X \xrightarrow{P'} \ell \text{ for every } X \text{ in } V'.$$

Let $\mathcal{P}' = \{P' : P \in \mathcal{P}\}$. Consider the ETOL system $G' = \langle V', \mathcal{P}', S, \Sigma \rangle$. It should then be clear to the reader that $L(G') = L(G)$, $\mathcal{B}_G = \mathcal{B}_{G'}$ and thus $\text{tr}(G') = \text{tr}(G) = k$. Next we claim the following.

CLAIM. Let $T = (A, R, V, f)$ be a tree in \mathcal{D}_G and let $a \in A$ be such that $f(a) \neq S$. Then

$$r_T(a) = \begin{cases} 0 & \text{if } f(a) \in \Sigma, \text{ and} \\ i & \text{if } f(a) = [A, i] \text{ for some } A \in V \setminus \Sigma. \end{cases}$$

Proof of the claim. Let T and a be as in the statement of the claim. The proof goes by induction on the level of a in T .

(1) If $\text{level}_T(a) = 0$ then, obviously, $r_T(a) = 0$ and $f(a) \in \Sigma$.

(2) If $\text{level}_T(a) = 1$ then $f(a) \notin \Sigma$ because G' is synchronized. The claim then follows from the construction of G' , in particular the productions of type (1.3).

(3) Let us assume that the claim holds if $\text{level}_T(a) \leq t$, $t \geq 1$.

(4) Let $\text{level}_T(a) = t + 1$ and let $\text{Desc}_T(a) = \{b_1, \dots, b_n\}$ where $f(b_j) = [B_j, i_j]$ for all $1 \leq j \leq n$. From the induction hypothesis it then follows that $M_T(a) = \max_{1 \leq j \leq n} i_j$. The claim then follows from the definition of r_T and the construction of G' , in particular the productions of type (1.4). Hence the claim holds. \square

Next we show that G' is strongly nonexpansive. Assume the contrary; i.e., there exists a tree $T = (A, R, V', f)$ in $\mathcal{D}_{G'}$ and two paths $b_0 \cdots b_n$ and $c_0 \cdots c_m$, $m, n \geq 1$, in T such that $b_0 = c_0 = a$, $b_n \neq c_m$ and $f(b_n) = f(c_m) = f(a) = [A, i]$, $A \in V \setminus \Sigma$, $1 \leq i \leq k$. Let $r = \max_{0 \leq i \leq n} \{i : b_i = c_i\}$. From the claim it follows that $r_T(c_m) = r_T(b_n) = r_T(a) = i$. But $r_T(b_r) \geq i + 1$ because of the definition of r_T . This contradicts the fact that $r_T(a) \geq r_T(b_r)$ and thus G' is strongly nonexpansive.

Next we do the following construction. Define a new alphabet

$$V'' = \{A_\Delta : A \in \Delta \in \mathcal{U}(G')\} \cup \Sigma.$$

For every pair (Δ, Δ') of useful alphabets and for every table $T \in \mathcal{P}'$ which is such that $T(a) \cap \Delta'^+ \neq \emptyset$ for every $a \in \Delta$, we define a new table $T_{\Delta, \Delta'}$ as follows. (For $\Delta \in \mathcal{U}(G)$ and $\alpha \in \Delta^+$ we use α_Δ to denote the word obtained from α by adding subscript Δ to every occurrence of every symbol.)

$$(2.1) \text{ If } a \xrightarrow{T} \alpha \text{ for some } a \in \Delta, \alpha \in \Delta'^+ \text{ then } a_\Delta \xrightarrow{T_{\Delta, \Delta'}} \alpha_\Delta.$$

$$(2.2) \text{ } a_{\Delta''} \xrightarrow{T_{\Delta, \Delta'}} a_{\Delta''} \text{ for every } a \in \Delta'' \in \mathcal{U}(G') \text{ such that } \Delta'' \neq \Delta.$$

$$(2.3) \text{ } a \xrightarrow{T_{\Delta, \Delta'}} a \text{ for every } a \in \Sigma.$$

Also for every $\Delta \in \mathcal{U}(G')$ which is such that $\Delta \subseteq \Sigma$, we define a table

$$T_\Delta = \{a_\Delta \rightarrow a : a \in \Delta\} \cup \{a_{\Delta'} \rightarrow a_{\Delta'} : a \in \Delta' \neq \Delta\} \cup \{a \rightarrow a : a \in \Sigma\}.$$

Let \mathcal{P}'' be the set of all new tables $T_{\Delta, \Delta'}$ and T_Δ that can be defined in this manner. Consider the EPTOL system $G'' = \langle V'', \mathcal{P}'', \$\$, \Sigma \rangle$. Clearly, $L(G') = L(G)$ and $\text{tr}(G'') = \text{tr}(G)$. One observes that G'' has the following properties.

$$(3.1) \text{ Succ } G = V''^*.$$

(3.2) G'' is strongly nonexpansive (because G' is strongly nonexpansive).

(3.3) Consider the partition $\Omega = (V''_0, \dots, V''_k)$ of $V'' \setminus \$\$_$ which is defined by $V''_0 = \{[A, 0]_\Delta : [A, 0] \in \Delta \in \mathcal{U}(G')\} \cup \{a_\Delta : a \in \Delta \cap \Sigma, \Delta \in \mathcal{U}(G')\} \cup \Sigma$ and, for $1 \leq i \leq k$, $V''_i = \{[A, i]_\Delta : [A, i] \in \Delta \in \mathcal{U}(G')\}$.

It then follows from the claim and the construction of G'' that if a is a node, different from the root, of a tree T from $\mathcal{D}_{G''}$, then $r_T(a) = t$, $0 \leq t \leq k$, if and only if the label of a is in V''_t .

$$(3.4) \text{ If } \$\$_ \xrightarrow[G'']{*} x \text{ then } \#_{V''_k}(x) \leq 1.$$

(3.5) G'' is in ANF.

Because of (3.4) it is a straightforward matter to construct an equivalent EPTOL system $G''' = \langle V''', \mathcal{P}''', \$\$, \Sigma \rangle$ of tree-rank k which is deterministic in V''_k . (It suffices to “split up” every table in a suitable way.) Moreover, the properties (3.1), (3.2), (3.3) and (3.5) still hold for G''' . Hence G''' is in TRNF and the theorem holds. \square

We conclude this section by showing that the notion of tree-rank on ETOL systems gives rise to an infinite hierarchy of classes of languages, the union of which is strictly included in $\mathcal{L}\text{ETOL}$. First we need the following lemma.

LEMMA 4. For every $k \geq 0$, $\mathcal{L}(\text{ETOL})_{\text{TR}(k)} \subseteq \mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$.

Proof. Let $k \geq 0$ and let $K = L(G)$ be in $\mathcal{L}(\text{ETOL})_{\text{TR}(k)}$ where $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ is an ETOL system of tree-rank t , $0 \leq t \leq k$. By Theorem 5 we can assume that G is in TRNF. Let $\Omega = (V_0, \dots, V_t)$ be the associated partition of $V \setminus S$. From the definition of tree rank and the fact that $\text{Succ } G = V^*$ it follows that

$$(*) \text{ for every } 0 \leq i \leq t, \text{ if } a \in V_i \text{ then } a \xrightarrow[G]{*} y \text{ implies that } \#_{V_i}(y) \leq 1.$$

We will show by induction on i that, for $0 \leq i \leq t$, $\text{rank}_G(a) \leq i$ for every a in V_i .

(1) Let a be a symbol in V_0 . Trivially, $a \xrightarrow[G]{*} y$ implies that $|y| \leq 1$. Thus

$\text{SUCC}_{G, V}(a)$ is a finite set and consequently, $\text{rank}_G(a) = 0$.

(2) Let us assume that, for $0 \leq j \leq l$, $\text{rank}_G(a) \leq j$ for every a in V_j .

(3) Let a be a symbol in V_{l+1} .

Define the set $Z_l = \bigcup_{j=0}^l V_j$.

From the induction hypothesis it follows that $Z \subseteq \bigcup_{j=0}^t R_j(G)$. But (*) implies that $\text{SUCC}_{G, V \setminus Z_i}$ is a finite set and thus $\text{rank}_G(a) \leq l + 1$, completing the induction.

Because G is of tree-rank t and in TRNF, it must hold that $S \xrightarrow[G]{*} x$ implies $\#_{V_i}(x) \leq 1$. It readily follows that $\text{rank}_G(S) \leq t$ and thus $\text{rank}(G) \leq t$. Hence $L \in \mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ and the lemma holds. \square

THEOREM 6. $\mathcal{L}(\text{ETOL})_{\text{TR}(0)} \subsetneq \mathcal{L}(\text{ETOL})_{\text{TR}(1)} \subsetneq \cdots \subsetneq \mathcal{L}(\text{ETOL})_{\text{TR}} \subsetneq \mathcal{L}\text{ETOL}$.

Proof. We show that for every $i \geq 0$

$$(*) \quad \mathcal{L}(\text{ETOL})_{\text{TR}(i)} \subsetneq \mathcal{L}(\text{ETOL})_{\text{TR}(i+1)}.$$

Clearly, $\mathcal{L}(\text{ETOL})_{\text{TR}(i)} \subseteq \mathcal{L}(\text{ETOL})_{\text{TR}(i+1)}$ for every $i \geq 0$. To show that the inclusion is proper, let i be a nonnegative integer. Define an alphabet $V_i = \{a_1, \dots, a_{i+2}\}$ and a homomorphism $\delta_i: V_i^* \rightarrow V_i^*$ by

$$\delta_i(a_j) = \begin{cases} a_j a_{j+1} & \text{if } 1 \leq j \leq i+1, \text{ and} \\ a_j & \text{if } j = i+2. \end{cases}$$

Consider the DOL system $G_i = \langle V_i, \delta_i, a_1 \rangle$. It is then a straightforward matter to show that G_i is of tree-rank $i+1$; thus $L(G_i) \in \mathcal{L}(\text{ETOL})_{\text{TR}(i+1)}$. On the other hand it has been shown in [1] that $L(G_i) \notin \mathcal{L}(\text{ETOL})_{\text{RAN}(i)}$, and thus, by Lemma 4, $L(G_i) \notin \mathcal{L}(\text{ETOL})_{\text{TR}(i)}$ completing the proof of (*). From Lemma 4 it also follows that $\mathcal{L}(\text{ETOL})_{\text{TR}} \subseteq \mathcal{L}(\text{ETOL})_{\text{RAN}}$. Since $\mathcal{L}(\text{ETOL})_{\text{RAN}} \subsetneq \mathcal{L}\text{ETOL}$, (see [1]) this implies that $\mathcal{L}(\text{ETOL})_{\text{TR}} \subsetneq \mathcal{L}\text{ETOL}$, thus completing the proof of the theorem. \square

IV. Characterization results. In this section we will characterize the class of ETOL systems of finite tree-rank in various ways. In particular, we will prove that ETOL systems of finite tree-rank, ETOL systems with rank, strongly nonexpansive ETOL systems and expansive ETOL systems are equivalent as far as their language generating power is concerned. We start by showing that every strongly nonexpansive ETOL system has a finite tree-rank.

THEOREM 7. *If $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ is a strongly nonexpansive ETOL system then $\text{tr}(G) \leq \#V - 1$.*

Proof. Let $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ be a strongly nonexpansive ETOL system. From Lemma 2 it immediately follows that $\#V \geq \text{rank } T + 1$ for every tree T in \mathcal{D}_G .

Hence $\text{tr}(G) \leq \#V - 1$ and thus the theorem holds. \square

From the above theorem and the fact that every ETOL language of finite tree-rank can be generated by an ETOL system of finite tree-rank which is in TRNF, and thus strongly nonexpansive, one can conclude the following.

THEOREM 8. *An ETOL language has a finite tree-rank if and only if it can be generated by a strongly nonexpansive ETOL system.*

Proof. The theorem follows immediately from Theorem 5 and Theorem 7. \square

It is interesting to note here that, as we made use of the TRNF theorem, the above result states an equality of classes of languages, not of systems. Indeed, although every nonexpansive ETOL system has a finite tree-rank, the converse does not hold. This is further illustrated by the following example.

Example. Consider the ETOL system $G = \langle \{a, b, \ell\}, \{a \rightarrow aba, b \rightarrow \ell, \ell \rightarrow \ell\}, a, \{a, b\} \rangle$. Clearly, G is an ETOL system of finite tree-rank (as a matter of fact, $\# \mathcal{D}_G = 1$); however the derivation tree in Fig. 1 is expansive,

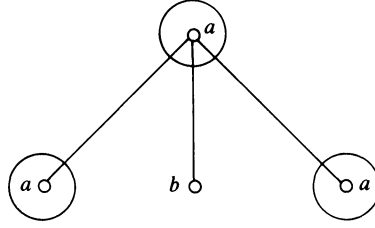


FIG. 1

and thus G is not strongly nonexpansive.

Next we will characterize ETOL systems of finite tree-rank by ETOL systems with rank. We know already from Lemma 4 that $\mathcal{L}(\text{ETOL})_{\text{TR}(k)} \subseteq \mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ for every $k \geq 1$, and thus the class of ETOL languages of finite tree-rank is included in the class $\mathcal{L}(\text{ETOL})_{\text{RAN}}$ of ETOL systems with rank.

To show the other inclusion, we prove the following result.

THEOREM 9. *Every ETOL system with rank has a tree-rank.*

Proof. Let $G = \langle V, \mathcal{P}, \mathcal{S}, \Sigma \rangle$ be an ETOL system of rank m for some $m \geq 0$. It suffices to show that for each $0 \leq l \leq m$, there exists an integer N_l such that if $T = (A, R, V, f)$ is in \mathcal{D}_G , then $r_T(a) \leq N_l$ for every node $a \in A$ which is such that $\text{rank}_G f(a) = l$.

The proof goes by induction on l .

(1) $l = 0$.

Let $N_0 = \max_{x \in R_0(G)} \{ \max \text{NSUCC}_{G, V} X \}$. It follows that $\text{Wdh}(T_a) \leq N_0$ for every $T = (A, R, V, f) \in \mathcal{D}_G$ and $a \in A$ such that $f(a) \in R_0(G)$. Together with Theorem 2, this ends the proof for $l = 0$.

(2) Let us assume that the proposition holds for every $l \leq k$ for some $k \geq 0$.

(3) $l = k + 1$.

Define

$$N_{k+1} = \max_{x \in R_{k+1}(G)} \{ \max \text{NSUCC}_{G, R_{k+1}(G)} X \} + \sum_{i=0}^k N_i + 1.$$

Let $T = (A, R, V, f) \in \mathcal{D}_G$ and let $a \in A$ be such that $\text{rank}_G f(a) = k + 1$. Consider $T_a = (A_a, R_a)$ and define a function g on A_a by

$$g(c) = \begin{cases} c & \text{if } \text{rank}_G f(c) = k + 1, \text{ and} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Obviously, $\text{rank } g(T_a) < \text{Wdh } g(T_a) \leq \max_{a \in R_{k+1}(G)} (\max \text{NSUCC}_{G, R_{k+1}(G)} a)$ and also $T_a \in \text{Hang}(g(T_a), \varepsilon)$ where ε is the set of trees corresponding to all successful derivations $X \Rightarrow^* x$ for some $x \in \Sigma^*$, $X \in \bigcup_{i=0}^k R_i(G)$. One observes that $\text{rank } \bar{T} \leq \sum_{i=0}^k N_i$ for all \bar{T} in ε and thus it follows from Lemma 1 that $r_T(a) = \text{rank } T_a \leq N_{k+1}$, thus completing the proof of the proposition. Hence the theorem holds. \square

It is interesting to note here that the above theorem cannot be reversed. This is illustrated by the following example.

Example. Consider the ETOL system

$$G = \langle \{a, b, \ell\}, \{P_1, P_2\}, b, \{a, b\} \rangle$$

where

$$P_1 = \{b \rightarrow ab, a \rightarrow a, \ell \rightarrow \ell\}$$

and

$$P_2 = \{b \rightarrow \varepsilon, a \rightarrow aa, \varepsilon \rightarrow \varepsilon\}.$$

It is a straightforward matter to show that $\text{tr}(G) = 1$. On the other hand $\text{SUCC}_{G,\{a\}}(a)$ is infinite and thus G has no rank.

From Theorem 9 and Lemma 4 we can conclude that $\mathcal{L}(\text{ETOL})_{\text{TR}} = \mathcal{L}(\text{ETOL})_{\text{RAN}}$. Combining this result with Theorem 8 and the fact (see Theorem 7 in [1]) that the class $\mathcal{L}(\text{ETOL})_{\text{RAN}}$ equals the class of languages generated by nonexpansive ETOL systems, we get the following result.

THEOREM 10. *The following classes of languages coincide:*

- (1) $\mathcal{L}(\text{ETOL})_{\text{RAN}}$.
- (2) *The class of languages that can be generated by nonexpansive ETOL systems.*
- (3) $\mathcal{L}(\text{ETOL})_{\text{TR}}$.
- (4) *The class of languages that can be generated by strongly nonexpansive ETOL systems.*

We end this section by showing that being of finite tree-rank is a decidable property in the class of ETOL systems. This can now be done easily.

THEOREM 11. *There exists an algorithm which, given an arbitrary ETOL system, decides whether or not it has a finite tree-rank.*

Proof. Let $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ be an ETOL system. We will construct an equivalent ETOL system $G' = \langle V', \mathcal{P}', S', \Sigma \rangle$ which is such that $\text{Succ}(G') = V'^*$.

Define a new alphabet $V' = \{A_\Delta : A \in \Delta \in \mathcal{U}(G)\} \cup \Sigma$. For every pair (Δ, Δ') of useful alphabets, and for every table $T \in \mathcal{P}$ which is such that $T(a) \cap \Delta'^+ \neq \emptyset$ for every $a \in \Delta$, we define a new table $T_{\Delta, \Delta'}$ as follows. (For $\Delta \in \mathcal{U}(G)$ and $\alpha \in \Delta^*$, we use α_Δ to denote the word obtained from α by adding subscript Δ to every occurrence of every symbol.)

- (1) If $a \xrightarrow{T} \alpha$ for some $a \in \Delta$, $\alpha \in \Delta'^*$, then $a_\Delta \xrightarrow{T_{\Delta, \Delta'}} \alpha_{\Delta'}$.
- (2) $a_{\Delta''} \xrightarrow{T_{\Delta, \Delta'}} a_{\Delta''}$ for every $a \in \Delta'' \in \mathcal{U}(G)$ with $\Delta'' \neq \Delta$.
- (3) $a \xrightarrow{T_{\Delta, \Delta'}} a$ for every $a \in \Sigma$.

Also for every $\Delta \in \mathcal{U}(G)$ which is a subset of Σ we define a table

$$T_\Delta = \{a_\Delta \rightarrow a : a \in \Delta\} \cup \{a_{\Delta'} \rightarrow a_{\Delta'} : \Delta' \neq \Delta\} \cup \{a \rightarrow a : a \in \Sigma\}.$$

Let \mathcal{P}' be the set of all newly defined tables. Consider the ETOL system $G' = \langle V', \mathcal{P}', S', \Sigma \rangle$. Clearly, $L(G') = L(G)$, $\text{Succ}(G') = V'^*$ and G' is of finite tree-rank if and only if G is an ETOL system with finite tree-rank. It follows that G' is of finite tree-rank if and only if G has a rank, a property which is known to be decidable (see [1]). Hence the theorem holds. \square

V. Tree-rank k versus rank k . In the preceding section we proved that the tree-rank hierarchy and the rank-hierarchy (of ETOL languages) have the same “limit”; i.e., $\bigcup_{k \geq 1} \mathcal{L}(\text{ETOL})_{\text{TR}(k)} = \bigcup_{k \geq 1} \mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$. On the other hand we also have proved (see Lemma 4) that each element of the tree-rank hierarchy is included in the corresponding element of the rank hierarchy, i.e., $\mathcal{L}(\text{ETOL})_{\text{TR}(k)} \subseteq \mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ for every $k \geq 1$. In this section we will further investigate the relationship between the elements of the two hierarchies, i.e., we will prove that the above inclusion is proper. As a matter of fact, we will show that for every $k \geq 1$, there exists a language M_k in $\mathcal{L}(\text{ETOL})_{\text{RAN}(1)}$ which is not in $\mathcal{L}(\text{ETOL})_{\text{TR}(k)}$.

To avoid cumbersome notation we will use from now on the same name for a derivation tree and its corresponding bare version and for an occurrence of a word at a level and the corresponding nodes in the tree. This should not lead to confusion. Next we need some definitions.

DEFINITION 12. Let $G = \langle V, \mathcal{P}, \mathcal{S}, \Sigma \rangle$ be an ETOL system, $D : S \xrightarrow{*} x_1 x_2 \xrightarrow{*} w \in \Sigma^*$ be a derivation in G and let k be a positive integer. We say that x is (D, k) -pure if one of the following holds:

- (i) either $\langle x \rangle = \beta B \beta'$ for some $\beta, \beta' \in V^+$, $B \in V$ such that $r_{T_D}(B) \geq k$,
- (ii) or $\langle x \rangle = B \gamma B'$ for some $\gamma \in V^*$, B and B' in V such that $r_{T_D}(B) \geq k$ and $r_{T_D}(B') \geq k$.

To prove that $\mathcal{L}(\text{ETOL})_{\text{TR}(k)} \subsetneq \mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$, we will consider a special subclass of the class of ETOL systems of tree-rank k , called (the class of) k -pure ETOL systems. Intuitively, an ETOL system is k -pure if for every positive integer n which is bigger than some constant C_G , there exists a word w_n such that every derivation D of w_n has the following property: every word of D , between the C_G th and the n th word, either contains a symbol of tree-rank (not smaller than) k that is embedded in the word, i.e., it is not at one of the extremes, or the leftmost and the rightmost occurrence of the word are *both* of tree-rank (not smaller than) k . Here is a formal definition.

DEFINITION 13. An ETOL system $G = \langle V, \mathcal{R}, \mathcal{S}, \Sigma \rangle$ is called k -pure (for some $k > 0$) if there exists a constant C_G such that the following holds. For every integer $n \geq C_G$, there exists a word w_n in $L(G)$ such that, if $D : S = x_0 \xrightarrow{G} x_1 \xrightarrow{G} \cdots \xrightarrow{G} x_m = w_n$, $m \geq 0$, is a proper derivation of w_n , then x_i is (D, k) -pure for each $C_G \leq i \leq n$.

DEFINITION 14. An ETOL language L is called k -pure, for some $k > 0$, if every ETOL system generating L is k -pure.

Note that the definition of a k -pure ETOL language is unusual in the sense that instead of calling a language k -pure if it *can* be generated by a k -pure ETOL system, we call an ETOL language k -pure if *every* ETOL system generating it is k -pure.

The following lemma will be useful in the sequel.

LEMMA 5. *Let G be an ETOL system and let k be a positive integer. If G is not k -pure then there exists an equivalent propagating system G' such that G' is not k -pure.*

Proof. Let G and k be as in the statement of the lemma. Let G' be the propagating ETOL system, equivalent to G , which is obtained from G using the standard construction (see, e.g., [3]). It is then a straightforward matter to show that G' is not k -pure if G is not k -pure.

DEFINITION 15. For every positive integer i we recursively define a language M_i as follows.

- (i) $M_1 = \{a^n b^n : n \geq 1\}$.
- (ii) Let \square , $\#$ and $\hat{\#}$ be new symbols.

Then $M_{i+1} = \{\alpha_1 \alpha_2 \cdots \alpha_n \square \alpha_n \cdots \alpha_2 \alpha_1 : n \geq 1, \alpha_i \in \# M_i \hat{\#} \text{ for every } 1 \leq i \leq n\}$. The sequence M_1, M_2, \dots will be used as follows. We will show that every language M_k is k -pure, thus implying that $M_k \notin \mathcal{L}(\text{ETOL})_{\text{TR}(k-1)}$ for every $k \geq 1$. We will also show that $M_k \in \mathcal{L}(\text{ETOL})_{\text{TR}(k)}$ for every $k \geq 1$. So we will span the tree-rank hierarchy on the sequence M_1, M_2, \dots . But, obviously, $M_k \in \mathcal{L}(\text{ETOL})_{\text{FIN}} = \mathcal{L}(\text{ETOL})_{\text{RAN}(1)}$ for every $k \geq 1$. From this it will then follow that $\mathcal{L}(\text{ETOL})_{\text{FIN}} \setminus \mathcal{L}(\text{ETOL})_{\text{TR}(k)} \neq \emptyset$ for every $k \geq 0$, implying that $\mathcal{L}(\text{ETOL})_{\text{TR}(k)} \subsetneq \mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ for every $k \geq 0$. First we prove the following lemma.

LEMMA 6. *For every $i \geq 1$, M_i is i -pure and $M_i = L(G_i)$ for some EDTOL system G_i of tree-rank not greater than i and is in ANF.*

Proof. The proof goes by induction on i .

(1) $i = 1$. Clearly, $M_1 \in \mathcal{L}(\text{EDTOL})_{\text{TR}(1)}$ as it is generated by the EDTOL system

$$G_1 = \langle \{S, a, b\}, \{P_1, P_2\}, S, \{a, b\} \rangle,$$

where

$$P_1 = \{S \rightarrow aSb, a \rightarrow a, b \rightarrow b\}$$

and

$$P_2 = \{S \rightarrow ab, a \rightarrow a, b \rightarrow b\},$$

which is of tree-rank 1 and in ANF. It follows from the well-known fact that M_1 is not regular that M_1 is 1-pure.

(2) Assume that the lemma holds for $i \leq k$.

(3) $i = k + 1$. First we show that M_{k+1} can be generated by an EDTOL system of tree-rank not greater than $k + 1$. By the induction hypothesis, we know that there exists an EDTOL system $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ in ANF of tree-rank not greater than k such that $L(G) = M_k$. Define new alphabets

$$V' = V \cup \{\$, \ell, \square, \#, \hat{\#}\}$$

and

$$\Sigma' = \Sigma \cup \{\square, \#, \hat{\#}\},$$

where $\$, \ell, \square, \#$ and $\hat{\#}$ are new symbols. We define two special tables

$$P_0 = \{\$ \rightarrow \# S \hat{\#} \$ \# S \hat{\#}\} \cup \{X \rightarrow \ell : X \notin \Sigma' \cup S\} \cup \{a \rightarrow a : a \in \Sigma'\}$$

and

$$P'_0 = \{\$ \rightarrow \square\} \cup \{X \rightarrow \ell : X \notin \Sigma' \cup \$\} \cup \{a \rightarrow a : a \in \Sigma'\}.$$

For every table T from \mathcal{P} we define a new table

$$T' = T \cup \{X \rightarrow X : X \in V' \setminus V\}.$$

Let $\mathcal{P}' = \{P_0, P'_0\} \cup \{T' : T \in \mathcal{P}\}$. Consider the EDTOL system $G' = \langle V', \mathcal{P}', S, \Sigma' \rangle$. It should then be clear to the reader that $\text{tr}(G') = \text{tr}(G) + 1 \leq k + 1$ and $L(G') = M_{k+1}$.

Next we show that every ETOL system generating M_{k+1} is $(k + 1)$ -pure. By Lemma 5, it suffices to show that every EPTOL system generating M_{k+1} is $(k + 1)$ -pure. So let $G = \langle V, \mathcal{P}, S, \Sigma \rangle$ be an EPTOL system generating M_{k+1} .

CLAIM I. *There exists a constant $C_1 > 0$ such that for every integer $n > C_1$, there exists a word $x_n \in M_k$ which is such that if $D : A = u_0 \xrightarrow{G} u_1 \xrightarrow{G} \cdots \xrightarrow{G} u_m = \alpha_1 \# x_n \hat{\#} \alpha_2$ is a derivation in G where $A \in V$, $m \geq 0$ and $\alpha_1 \alpha_2 \in \Sigma^*$ then, for all $C_1 \leq i \leq n$, $\text{anc}_D(\# x_n \hat{\#}, i)$ is (D, k) -pure.*

Proof. Define the set

$$AX = \{A \in V : A \xrightarrow{G}^* \alpha_1 \# x \hat{\#} \alpha_2 \text{ for some } x \in M_k, \alpha_1 \alpha_2 \in V^*\},$$

and let $V' = V \cup \{Z\}$ be a new alphabet where Z is a new symbol. Define a table $P_0 = \{Z \rightarrow X : X \in AX\} \cup \{X \rightarrow X : X \in V\}$, and, for every table P from \mathcal{P} , let $P' = P \cup \{Z \rightarrow Z\}$. Let $\mathcal{P}' = \{P_0\} \cup \{P' : P \in \mathcal{P}\}$. Consider the ETOL system $H = \langle V', \mathcal{P}', Z, \Sigma \rangle$. Given H one can easily construct an ETOL system H' such that $L(H') = M_k$, and, moreover, H is k -pure if H' is. But from the induction hypothesis, it

follows that H' , and thus also H , is k -pure. From this the claim immediately follows. \square

Define a constant $C_2 = \sum_{i=1}^N s^i$ where $s = \#V$ and $N = (\maxr G)^{2^{s+1}}$. For every integer $p \geq C_2$ we define a word $w_p \in M_{k+1}$ as follows. Let $n = C_1 + 1$, $r = 2(\maxr G)^p + 1$ and let x_n be as in the statement of Claim I. Then $w_p = (\#x_n\hat{\#})^r \square (\#x_n\hat{\#})^r$. In the sequel, the i th copy of $\#x_n\hat{\#}$ in w_p , $1 \leq i \leq 2r$, will be denoted by $\#\alpha_i\hat{\#}$. The following statement is easy to prove.

CLAIM II. *If $E : S = u_0 \xrightarrow{G} u_1 \xrightarrow{G} \cdots \xrightarrow{G} u_m = w_p$, $m \geq 0$, is a derivation of w_p , $p \geq C_2$,*

and X is an occurrence in T_E such that $r_{T_E}(X) \leq k$, then $\text{contr}_E(X)$ contains at most one copy of $\#x_n\hat{\#}$.

Next we will show that for every integer $p \geq C_2$ if $D : S = y_0 \xrightarrow{G} y_1 \xrightarrow{G} \cdots \xrightarrow{G} y_l = w_p$, $l > 0$, is a proper derivation of w_p , then y_i is $(D, k+1)$ -pure for every $C_2 \leq i \leq p$. So let $D : S = y_0 \xrightarrow{G} y_1 \xrightarrow{G} \cdots \xrightarrow{G} y_l = w_p$, $l > 0$, be a proper derivation of w_p . Let ANC be the set of all lowest common ancestors A_i of $\#\alpha_i\hat{\#}$, $1 \leq i \leq 2r$. For $1 \leq i \leq 2r$, let $t_i = l\text{-level}_{T_D} A_i + C_1$, i.e., t_i is such that $y_{t_i - C_1}$ contains A_i . From Claim I it follows that, for $1 \leq i \leq 2r$, either

$$(3.1) \quad \text{anc}_D(\#\alpha_i\hat{\#}, t_i) = \beta_i B_i \beta'_i,$$

or

$$(3.2) \quad \text{anc}_D(\#\alpha_i\hat{\#}, t_i) = B'_i \gamma_i B''_i,$$

for some $\beta_i, \beta'_i \in V^+$, $\gamma_i \in V^*$, B_i, B'_i and B''_i in V such that $r_{T_D}(B_i) \geq k$, $r_{T_D}(B'_i) \geq k$ and $r_{T_D}(B''_i) \geq k$. The following is an immediate consequence of (3.1) and (3.2).

CLAIM III. *For no $1 \leq i, j \leq 2r$, $|i - j| > 1$, there is a path in T_D from B_i , or B'_i , or B''_i , to B_j , or B'_j , or B''_j .*

From Claim III it follows that T_D contains nodes of rank not smaller than $k+1$. For $1 \leq i \leq 2r$, let N_i be defined as follows:

(i) If (3.1) holds for i then N_i is the lowest ancestor of B_i with rank not smaller than $k+1$.

(ii) If (3.2) holds for i then N_i is the lowest common ancestor of B'_i and B''_i .

Thus, $r_{T_D}(N_i) \geq k+1$ for all $1 \leq i \leq 2r$. It also holds that, for $1 \leq i \leq 2r$, $\#\{j : N_j = N_i\} \leq 2 \maxr G$, and thus T_D contains at least

$$(3.3) \quad \frac{2r}{2 \maxr G} > 2^p > p$$

nodes of rank not smaller than $k+1$.

Next we show that y_{i_0} is $(D, k+1)$ -pure for every $C_2 \leq i \leq p$. Assume the contrary, i.e., y_{i_0} is not $(D, k+1)$ -pure for some $C_2 \leq i_0 \leq p$. From (3.3) and Definition 12 it follows that either

$$(3.4) \quad y_{i_0} = \beta B,$$

or

$$(3.5) \quad y_{i_0} = B\beta,$$

for some $B \in V$, $\beta \in V^*$ with $r_{T_D}(B) > k$ and $r_{T_D}(X) \leq k$ for every occurrence X in β . Both cases being symmetric, let us assume that (3.4) holds.

Next we will transform D into a deterministic derivation

$$\bar{D} : S = z_0 \xrightarrow[(G)_D]{T_1} z_1 \xrightarrow[(G)_D]{T_2} \cdots \xrightarrow[(G)_D]{T_l} z_l = w_p$$

of w_p in $(G)_D$ which is such that $z_{i_0} = \beta' B$ for some $B \in V$, $\beta' \in V^*$ such that $r_{T_D}(B) > k$ and $r_{T_D}(X) \leq k$ for every occurrence X in β' .

In order to do this we need some more results. First of all we note that if $uvxyz \in M_{k+1}$ for some $u, v, x, y, z \in \Sigma^*$ such that $v \neq y$ then $uyxyz \notin M_{k+1}$. From this we get the following.

CLAIM IV. *If $E: u_0 \xrightarrow{G} u_1 \xrightarrow{G} \cdots \xrightarrow{G} u_m \in \Sigma^*$, $m \geq 0$, is a derivation in G and if X and Y are two occurrences of the same symbol in the same word u_i , $0 \leq i \leq m$, then $\text{contr}_E(X) = \text{contr}_E(Y)$.*

CLAIM V. *For every $0 \leq i \leq i_0$, if X_i is the name of $\text{anc}_D(B, i)$ then $\#_{X_i}(y_i) = 1$.*

Proof. Assume the contrary, i.e., there exists an integer $0 \leq t \leq i_0$ such that y_t contains two occurrences Y_1 and $Y_2 = \text{anc}_D(B, t)$ of the same symbol X_i . From Claim IV it follows that we can replace T_{Y_2} by T_{Y_1} and obtain a new derivation $E: v_0 \xrightarrow{G} v_1 \xrightarrow{G} \cdots \xrightarrow{G} v_l = w_p$ of w_p . It follows from (3.4) that $r_{T_E}(X) \leq k$ for every occurrence X in v_{i_0} . But, obviously, $|v_{i_0}| \leq (\max r G)^p$. From Claim II and the definition of r it follows that v_l cannot contain more than r copies of $\# x_n \#$, a contradiction. Hence the claim holds. \square

For any l -step derivation $E: S = u_0 \xrightarrow{G} u_1 \xrightarrow{G} \cdots \xrightarrow{G} u_l = w_p$ of w_p and for any two occurrences X and Y of the same symbol in the same intermediate word u_i of E , we define a new l -step derivation $E' = \psi(E, X, Y): S = u'_0 \xrightarrow{G} u'_1 \xrightarrow{G} \cdots \xrightarrow{G} u'_l = w_p$ of w_p as follows.

- (i) If $i \leq C_2$ and $|\text{contr}_E(X, C_2)| \leq |\text{contr}_E(Y, C_2)|$, then replace T_X by T_Y ;
- (ii) otherwise replace T_Y by T_X .

The following property of ψ is a consequence of Claim II, Claim IV and Claim V.

CLAIM VI. *If $u_{i_0} = \gamma C$ for some $C \in V$, $\gamma \in V^*$, such that $r_{T_E}(C) > k$ and $r_{T_E}(X) \leq k$ for every occurrence X in γ , then $u'_{i_0} = \gamma' C$ for some $\gamma' \in V^*$ such that $r_{T_E}(C) > k$ and $r_{T_E}(X) \leq k$ for every occurrence X in γ' .*

Clearly by iterating ψ in a top-down fashion we can transform D into a deterministic (not necessarily proper!) derivation $\bar{D}: S = z_0 \xrightarrow[(G)_D]{T_1} z_1 \xrightarrow[(G)_D]{T_2} \cdots \xrightarrow[(G)_D]{T_l} z_l = w_p$ of w_p in $(G)_D$. Claim VI then implies that $z_{i_0} = \beta' B$ for some $\beta' \in V^*$ such that $r_{T_D}(B) > k$ and $r_{T_D}(X) \leq k$ for every occurrence X in β' . From the definition of r , Claim II and the obvious fact that $|z_{i_0}| \leq (\max r G)^p$, it immediately follows that $\text{contr}_{\bar{D}}(B) = x' \square (\# x_n \#)^r$ for some $x' \in \Sigma^+$.

From the definition of C_2 , the construction of \bar{D} and the fact that D is proper we get that $|z_{C_2}| \geq |y_{C_2}| > (\max r G)^{2^{s+1}}$. Hence there exist constants $0 \leq i_1 < i_2 \leq C_2$ such that $\text{alph } z_{i_1} = \text{alph } z_{i_2}$ and $|z_{i_2}| > |z_{i_1}|$. Define $\mu = T_1 \cdots T_{i_1}$, $\rho = T_{i_1+1} \cdots T_{i_2}$ and $\nu = T_{i_2+1} \cdots T_l$. The situation is best represented in Fig. 2.

It is then a straightforward matter to show that $\mu \rho^2 \nu(S)$ is a word in $L((G)_D) \subseteq L(G)$ which is not in M_{k+1} . Thus the lemma holds. \square

From the previous lemma, the main results of this section now easily follow.

THEOREM 12. *For every $k \geq 1$, there are ETOL languages of finite index which are not in $\mathcal{L}(\text{ETOL})_{\text{TR}(k)}$.*

Proof. It is a straightforward matter to show that, for $i \geq 1$, $M_i \in \mathcal{L}(\text{ETOL})_{\text{FIN}(2i+1)}$. But from Lemma 6 it follows that, for $i \geq 1$, $M_i \notin \mathcal{L}(\text{ETOL})_{\text{TR}(i-1)}$. Thus $M_{k+1} \in \mathcal{L}(\text{ETOL})_{\text{FIN}} \setminus \mathcal{L}(\text{ETOL})_{\text{TR}(k)}$ for every $k \geq 1$. Hence the theorem holds. \square

THEOREM 13. *For every $k \geq 1$, $\mathcal{L}(\text{ETOL})_{\text{TR}(k)} \subsetneq \mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$.*

Proof. This follows immediately from Theorem 12, Lemma 4, and Theorem 5 in [1], which says that $\mathcal{L}(\text{ETOL})_{\text{RAN}(1)} = \mathcal{L}(\text{ETOL})_{\text{FIN}}$. \square

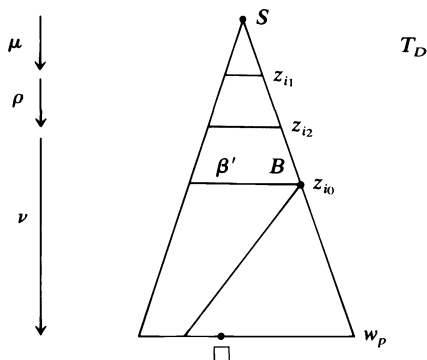


FIG. 2

Thus, although the two hierarchies of classes of ETOL languages, associated with ETOL systems of finite tree-rank and ETOL systems with rank respectively, have the same limit, the corresponding elements are different in the sense that one is strictly included in the other, i.e., $\mathcal{L}(\text{ETOL})_{\text{TR}(k)} \subsetneq \mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ for every $k \geq 0$. Moreover they are “very different” because there does not exist an integer function f such that $\mathcal{L}(\text{ETOL})_{\text{RAN}(k)} \subsetneq \mathcal{L}(\text{ETOL})_{\text{TR}(f(k))}$ for any $k \geq 1$.

VI. Closure properties. We end this paper by listing some results concerning closure properties of the classes $\mathcal{L}(\text{ETOL})_{\text{TR}(k)}$, $k \geq 1$, and $\mathcal{L}(\text{ETOL})_{\text{TR}}$. The somewhat tedious proofs of these results are omitted; however the interested reader can find them in [10].

THEOREM 14

- (1) For every $k \geq 1$, $\mathcal{L}(\text{ETOL})_{\text{TR}(k)}$ is a full-principal full semi-AFL.
- (2) $\mathcal{L}(\text{ETOL})_{\text{TR}}$ is a nonfull-principal full-substitution closed AFL.

The previous theorem and the results on the relation between $\mathcal{L}(\text{ETOL})_{\text{TR}(k)}$ and $\mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ allow us now to prove that $\mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ is nonfull-principal for every $k \geq 1$.

THEOREM 15. For every $k \geq 1$, $\mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ is a nonfull-principal full AFL.

Proof. By Theorem 8 in [1] we know that $\mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ is a full AFL for each $k \geq 1$. Assume the contrary, i.e., $\mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ is a full principal full AFL for some $k \geq 1$. From Corollary 5.4.1 in [2] it then follows that $\mathcal{L}(\text{ETOL})_{\text{RAN}(k)}$ is a full-principal full semi AFL. Let L'_k be a generator of it. By Theorem 9, $L'_k \in \mathcal{L}(\text{ETOL})_{\text{TR}(k')}$ for some $k' \geq 1$. By Theorem 14, $\mathcal{L}(\text{ETOL})_{\text{TR}(k')}$ is a full principal semi-AFL. Let $L_{k'}$ be its generator. Hence L'_k can be obtained from $L_{k'}$ using only semi-AFL operations.

But this implies that every ETOL language of finite index can be obtained from $L_{k'}$ using semi-AFL operations. Hence $\mathcal{L}(\text{ETOL})_{\text{FIN}} \subseteq \mathcal{L}(\text{ETOL})_{\text{TR}(k')}$, contradicting Theorem 13. Thus the theorem holds. \square

REFERENCES

- [1] A. EHRENFEUCHT, G. ROZENBERG AND D. VERMEIR, *On ETOL systems with rank*, J. Comput. System Sci., to appear.
- [2] S. GINSBURG, *Algebraic and Automata-Theoretic Properties of Formal Languages*, Fundamental Studies in Computer Sci. 2, North Holland/American Elsevier, Amsterdam, 1975.
- [3] G. ROZENBERG AND A. SALOMAA, *The Mathematical Theory of L Systems*, Academic Press, London-New York, 1980.

- [4] G. ROZENBERG AND D. VERMEIR, *On ETOL systems of finite index*, Information and Control, 38 (1975), pp. 103–133.
- [5] ———, *On metalinear ETOL systems*, Fundamenta Informaticae, 3 (1980), pp. 15–36.
- [6] ———, *On recursion in ETOL systems*, J. Comput. Systems Sci., 19 (1979), pp. 179–196.
- [7] ———, *On L systems of finite index*, in *Automata, Languages and Programming 4th Colloquium*, Lecture Notes in Computer Science, 52, Springer-Verlag, Berlin, 1977, pp. 430–440.
- [8] ———, *A hierarchy of ETOL languages with rank*, submitted for publication.
- [9] ———, *A note on the M-growth functions of ETOL systems with rank*, Fundamenta Informaticae, to appear.
- [10] D. VERMEIR, *Structural restrictions on ETOL systems*, Ph.D. Thesis, University of Antwerp, 1978.

ON THE SELECTION OF TEST DATA FOR RECURSIVE MATHEMATICAL SUBROUTINES*

JOHN H. ROWLAND† AND PHILIP J. DAVIS‡

Abstract. Let Y be a family of sequences defined by linear difference equations of the form $y(n+1) = P(n)y(n) + Q(n)$, where P and Q are restricted to be polynomials of limited degree, constant matrices, multinomials, and so forth. The central problem is to find a finite sample which uniquely identifies members of Y . It is shown that such a sample exists when P and Q are polynomials of limited degree. The sample size depends linearly on the degree limits. A similar result holds for systems of difference equations with P a constant matrix and Q a column vector with polynomial components. Testing procedures are also derived for the case where the coefficients of P and Q are multinomials in a vector parameter x , and y is considered to be a function of its initial value.

Key words. Testing, sampling, verification, difference equations

1. Introduction. A number of authors have recently noted the need for more research on the theory of program testing [11], [10], [3]. It is well known that no reasonable amount of testing can conclusively establish the correctness of a program. However, test data can often be selected which will guarantee that certain types of errors will be detected. A classical illustration of this situation is given by the fact that an error in the coefficients of a polynomial of degree $\leq n$ can be detected by sampling the polynomial at $n+1$ distinct points. In general, we will assume that the program with bugs and the desired function both belong to an appropriate class of functions. We then seek test data which will uniquely identify members of that class. This paper will be concerned with the selection of test data for functions which can be computed recursively by means of linear difference equations. The linearity restriction is a technical one which makes the mathematics more tractable. This restriction essentially limits the results to sequences y in which $y(n+1)$ depends only on the first power of $y(n)$, $y(n-1)$, \dots , $y(n-k)$ for some fixed integer k . Some examples of such sequences will be given in § 6.

The goal of this research is to provide a theoretical basis for testing certain programs involving loop constructs. However, the theory does not directly involve the actual program constructs. The important feature is that both the program to be tested and the desired function must satisfy difference equations of a given type. Subject to this restriction, the actual program might involve iteration, recursive subroutine calls, or neither of these. For example, the sequence $y(n) = 2^n + 5^n$ could be computed iteratively from the difference equation

$$\begin{aligned}y(0) &= 2, \\y(1) &= 7, \\y(n+2) &= 7y(n+1) - 10y(n), \quad n = 0, 1, 2, \dots,\end{aligned}$$

or it could be computed directly from the formula

$$y(n) = \alpha_1 e^{\beta_1 n} + \alpha_2 e^{\beta_2 n},$$

with $\alpha_1 = \alpha_2 = 1$ and $\beta_1 = \ln(2)$, $\beta_2 = \ln(5)$. Even though a program based on the latter formula would not involve iteration, the theory would still be applicable because for any

* Received by the editors May 15, 1979, and in revised form March 20, 1980.

† Department of Computer Science, University of Wyoming, Laramie, Wyoming 82071.

‡ Division of Applied Mathematics, Brown University, Providence, Rhode Island 02912.

choice of $\alpha_1, \alpha_2, \beta_1$, and β_2 the sequence would satisfy a second order difference equation with constant coefficients.

Howden [11] has described a very general method for testing recursive programs which is based directly on program constructs. His result does not have the linearity assumption of the present paper but does require the a posteriori demonstration that a certain matrix is nonsingular. Howden's paper does not discuss systems or higher-order equations.

Consider a family Y of sequences defined by difference equations of the form

$$(1.1) \quad y(n+1) = P(n)y(n) + Q(n), \quad y(0) = t,$$

$n = 0, 1, 2, \dots$, where P and Q will be restricted to certain classes such as polynomials of limited degree, constant matrices, multinomials, and so forth. We will be concerned with three general questions:

- (i) Does there exist a finite sample S such that if y and $z \in Y$ and $y(n) = z(n)$, for all $n \in S$, then $y(n) = z(n)$ for all n ?
- (ii) Let $y(\cdot, t)$ and $z(\cdot, t)$ be two sequences in Y considered as a function of the initial value t . Do there exist finite samples S of integers and T of complex numbers such that $y(n, t) = z(n, t)$, for all $n \in S$ and $t \in T$, implies that $y(n, t) = z(n, t)$, for all n and all t ?
- (iii) Suppose

$$\begin{aligned} y(n+1) &= P_1(n)y(n) + Q_1(n), \\ z(n+1) &= P_2(n)z(n) + Q_2(n). \end{aligned}$$

Does the fact that $y(n) = z(n)$ for all n imply that $P_1 = P_2$ and $Q_1 = Q_2$? If not, can the equation $y(n) = z(n)$ hold for only special types of sequences, or does this imply that certain relationships must hold between P_1, P_2, Q_1 , and Q_2 ?

Questions (ii) and (iii) are similar to the systems identification problem which arises, for example, in the theory of linear electrical and mechanical systems.

The mathematical background required for this paper is outlined in § 2. Section 3 will be concerned with scalar equations of the form (1.1) with P and Q polynomials of restricted degree, while § 4 is concerned with systems of equations with P a constant matrix and Q a polynomial vector of limited degree. This latter case includes higher-order difference equations with constant coefficients and polynomial forcing function. Systems of equations with P a polynomial matrix are still under investigation. In § 5 it is shown that one can prove certain identities involving sequences by using sampling in place of mathematical induction. The final section contains several examples which illustrate the theory.

2. Mathematical background. This section contains some notation, terminology, and facts which will be used later. The symbol N will denote the nonnegative integers, \mathcal{C} the complex numbers, and \mathcal{C}_ν complex ν -dimensional space. By an initial segment of N we will mean a finite set of consecutive integers starting with zero.

The forward difference operator Δ and forward shift operator E are defined by

$$\begin{aligned} \Delta y(n) &= y(n+1) - y(n), \\ Ey(n) &= y(n+1). \end{aligned}$$

The set of polynomials of degree $\leq m$ will be denoted by \mathcal{P}_m . It can be shown that a sequence $y \in \mathcal{P}_m$ if and only if $\Delta^{m+1}y(n) = 0$, for all n .

The theory of linear difference equations is similar to that for differential equations and is described in greater detail by Brand [1], Milne-Thomson [13], and Fort [7]. Let us consider a linear difference equation of the form

$$(2.1) \quad \sum_{j=0}^k p_j(n)y(n+j) = q(n),$$

where p_1, \dots, p_k, q , and y are complex-valued sequences. A sequence y is a solution to (2.1) if (2.1) holds for all $n \in N$. The equation is said to be autonomous if p_1, \dots, p_k and q are independent of n . The sequence q is called the forcing function, and if $q = 0$ the equation is said to be homogeneous. An initial-value problem consists of (2.1) together with values for $y(0), y(1), \dots, y(k-1)$. If p_k never vanishes, then one can solve (2.1) recursively to show that the initial-value problem has a unique solution.

The sequences u_1, \dots, u_k are said to be linearly independent iff $c_1u_1(n) + \dots + c_ku_k(n) = 0$, for all $n \in N$, implies $c_1 = \dots = c_k = 0$. The Casorati determinant $K(n)$ associated with u_1, \dots, u_k is defined by

$$K(n) = \det(u_i(n+j)), \quad i = 1, \dots, k, \quad j = 0, \dots, k-1.$$

The Casorati determinant has properties which are similar to the Wronskian in the theory of differential equations. In particular, if u_1, \dots, u_k are solutions to the homogeneous equation associated with (2.1) and p_0p_k never vanishes, then a necessary and sufficient condition for the linear independence of u_1, \dots, u_k is that $K(0) \neq 0$.

The general solution to (2.1) can be expressed in the form

$$y = c_1u_1 + \dots + c_ku_k + y_p,$$

where u_1, \dots, u_k are linearly independent solutions to the homogeneous equation, c_1, \dots, c_k are constants, and y_p is a particular solution to (2.1). Solutions to autonomous homogeneous equations have explicit representations as sums of exponentials. Consider an equation of the form

$$(2.2) \quad \sum_{j=0}^k a_j y(n+j) = 0,$$

where $a_0a_k \neq 0$. Let $p(t) = a_0 + a_1t + \dots + a_kt^k$ be the corresponding characteristic polynomial with zeros $\lambda_1, \dots, \lambda_k$. If these zeros are all distinct, then linearly independent solutions to (2.2) are given by

$$(2.3) \quad \phi_j(n) = \lambda_j^n, \quad j = 1, 2, \dots, k.$$

Repeated zeros are handled by multiplying λ_j^n by powers of n ; for example, if λ_j has multiplicity m , we take

$$(2.4) \quad \phi_{j+i}(n) = n^i \lambda_j^n, \quad i = 0, 1, \dots, m-1$$

to be the linearly independent solutions corresponding to λ_j .

Finally, we will need the notion of component matrices. Let A be a ν by ν matrix having eigenvalues $\lambda_1, \dots, \lambda_k$ with multiplicities ρ_1, \dots, ρ_k in the minimal polynomial of A . Then [12, p. 173] there exist ν by ν matrices Y_{ij} with the property that for any polynomial f ,

$$(2.5) \quad f(A) = \sum_{i=1}^k \sum_{j=0}^{\rho_i-1} f^{(j)}(\lambda_i) Y_{ij}.$$

The Y_{ij} 's are called the component matrices of A . If t is a ν by 1 column vector, we will call the vectors Y_{ijt} the spectral components of t with respect to A .

3. Scalar equations. Let $Y(k, m)$ be the set of sequences satisfying an equation of the form (1.1) with $P \in \mathcal{P}_k$ and $Q \in \mathcal{P}_m$. This includes, for example, $n!$, t^n for any complex number t , and any sequence of the form

$$y(n) = q(0) + q(1) + \cdots + q(n),$$

where $q \in \mathcal{P}_m$. The example

$$\begin{aligned} y(n+1) &= y(n) + 1, & y(0) &= 0, \\ z(n+1) &= n + 1, & z(0) &= 0, \end{aligned}$$

shows that the same sequence can have two different representations of the form (1.1). Let us show that this can happen if and only if y is essentially a polynomial in n of degree $\leq m$.

THEOREM 3.1. *If $y \in \mathcal{P}_m$, then there exist polynomials $P_1, P_2 \in \mathcal{P}_k$ and $Q_1, Q_2 \in \mathcal{P}_m$ with $P_1 \neq P_2$ and $Q_1 \neq Q_2$ such that*

$$(3.1) \quad y(n+1) = P_j(n)y(n) + Q_j(n), \quad j = 1, 2,$$

for all $n \in \mathbb{N}$. Conversely, if y satisfies both equations (3.1) for all n , then either $P_1 = P_2$ and $Q_1 = Q_2$ or there exists a polynomial $\phi \in \mathcal{P}_m$ and an integer $n_0 \leq k$ such that $y(n) = \phi(n)$ for all $n \geq n_0$.

Proof. If $y \in \mathcal{P}_m$, then y satisfies the equations

$$\begin{aligned} y(n+1) &= 1 \cdot y(n) + \Delta y(n), \\ y(n+1) &= 0 \cdot y(n) + E y(n), \end{aligned}$$

where Δ is the forward difference operator and E is the forward shift operator. Both equations are of the proper form since 1 and $0 \in \mathcal{P}_k$, $\Delta y \in \mathcal{P}_{m-1} \subset \mathcal{P}_m$, and $E y \in \mathcal{P}_m$. The converse will follow from the proof of Theorem 3.2.

The next theorem shows that two sequences from $Y(k, m)$ are identical if they agree on a sufficiently large initial segment.

THEOREM 3.2. *Let y and $z \in Y(k, m)$. If $y(n) = z(n)$ for $n = 0, 1, \dots, 2k + m + 2$, then $y(n) = z(n)$, for all $n \in \mathbb{N}$.*

Proof. Suppose y and z are defined by the equations

$$(3.2) \quad y(n+1) = P_1(n)y(n) + Q_1(n),$$

$$(3.3) \quad z(n+1) = P_2(n)z(n) + Q_2(n).$$

If $P_1 = P_2$, then $Q_1(n) = Q_2(n)$, $n = 0, 1, \dots, m$. Thus $Q_1 = Q_2$ and the result follows easily. Henceforth we will assume that $P_1 \neq P_2$. Let $P_0 = P_2 - P_1$ and $Q_0 = Q_2 - Q_1$. Now subtract (3.2) from (3.3) and shift the index to obtain

$$(3.4) \quad P_0(n+1)y(n+1) + Q_0(n+1) = 0, \quad n = -1, 0, \dots, 2k + m.$$

Next, multiply (3.2) by $P_2(n)$, (3.3) by $P_1(n)$, and subtract. This gives

$$(3.5) \quad P_0(n)y(n+1) + P_1(n)Q_2(n) - P_2(n)Q_1(n) = 0,$$

$n = 0, 1, \dots, 2k + m + 1$. Now eliminate $y(n+1)$ from (3.4) and (3.5), and replace P_2 by $P_0 + P_1$, Q_2 by $Q_0 + Q_1$ to obtain

$$(3.6) \quad P_0(n)Q_0(n+1) + P_0(n+1)P_0(n)Q_1(n) - P_0(n+1)P_1(n)Q_0(n) = 0,$$

$n = 0, 1, \dots, 2k + m$. Note that the left side of (3.6), which is a polynomial of degree at most $2k + m$, vanishes at $2k + m + 1$ distinct points. It follows that (3.6) holds over the entire complex plane.

Let Q^*/P^* represent the rational function Q_0/P_0 after common factors, if any, have been removed. Also remove these common factors from (3.6) and replace n by t to obtain

$$(3.7) \quad P^*(t)Q^*(t+1) + P^*(t+1)P^*(t)Q_1(t) - P^*(t+1)P_1(t)Q^*(t) = 0,$$

for all complex numbers t . Now (3.7) implies that P^* is constant. To see this, suppose the contrary and let t_1 be a zero of P^* having the smallest real part. Then $Q^*(t_1) \neq 0$ because P^* and Q^* have no common factors, and $P^*(t_1 - 1) \neq 0$ because P^* has no zeros with real part less than $\text{Re}(t_1)$. The substitution of $t_1 - 1$ for t in (3.7) leads to a contradiction because the first term on the left does not vanish whereas the other two terms do vanish. Thus P^* is constant, say $P^*(t) = c \neq 0$, for all complex t .

Next, let $\phi = -Q^*/P^* = -Q^*/c$, and note that $\phi \in \mathcal{P}_m$. We wish to show that ϕ satisfies both of the difference equations (3.2) and (3.3). The fact that ϕ satisfies (3.2) can be seen by dividing both sides of (3.7) by $-c^2$ and rearranging terms. Now (3.4) implies that $y(n) = \phi(n)$, $n = 0, 1, \dots, 2k + m + 1$, except perhaps at zeros of P_0 . But $P_0\phi \in \mathcal{P}_{k+m}$ and P_0 has at most k zeros; so $P_0(n)\phi(n) + Q_0(n) = 0$, for all n . Then we have

$$\begin{aligned} \phi(n+1) &= P_1(n)\phi(n) + Q_1(n) = P_2(n)\phi(n) + Q_2(n) - P_0(n)\phi(n) - Q_0(n) \\ &= P_2(n)\phi(n) + Q_2(n), \end{aligned}$$

and ϕ satisfies (3.3) for all n . Finally, let n_0 be the first index for which $y(n_0) = \phi(n_0)$. Clearly $n_0 \leq k$. Then an inductive argument shows that $y(n) = \phi(n) = z(n)$, for all $n \geq n_0$, which completes the proof of Theorem 3.2.

The converse of Theorem 3.1 can now be proved by noting that its hypotheses are stronger than those for Theorem 3.2; hence, the polynomial ϕ produced above satisfies the given requirements.

It is tempting to conjecture that $k + m + 2$ samples (in addition to the initial value) should be sufficient in Theorem 3.2 since that would provide one equation for each coefficient of P and Q . The following example shows that this is not always sufficient.

Example 3.1. Let y and $z \in Y(1, 1)$ be defined by

$$\begin{aligned} y(n+1) &= -2y(n) + 3 + n, & y(0) &= 1, \\ z(n+1) &= -nz(n) + 1 + 2n, & z(0) &= 1. \end{aligned}$$

Then a direct calculation shows that $y(n) = z(n)$, for $n = 0, 1, 2, 3, 4$, but $y(5) \neq z(5)$. Here $k + m + 2 = 4$.

Theorem 3.2 shows that $2k + m + 2$ samples are sufficient to identify sequences in $Y(k, m)$, while Example 3.1 shows that $k + m + 2$ samples need not be sufficient. We have not resolved the question as to whether some number between $k + m + 2$ and $2k + m + 2$ would in general provide a large enough sample. Let us prove, however, that an equation requiring more than $k + m + 2$ samples to identify cannot have a polynomial from \mathcal{P}_m as a solution.

THEOREM 3.3. *Suppose y and $z \in Y(k, m)$. If $y \in \mathcal{P}_m$ and $y(n) = z(n)$, $n = 0, 1, \dots, k + m + 1$, then $y(n) = z(n)$ for all n .*

Proof. Suppose y and z are given by (3.2) and (3.3), respectively. Then $y(n) = z(n)$, $n = 0, 1, \dots, k + m + 1$ implies that

$$y(n+1) = P_2(n)y(n) + Q_2(n),$$

for $n = 0, 1, \dots, k + m$. This equation involves polynomials of degree at most $k + m$ and it holds for at least $k + m + 1$ distinct values of n . Thus it holds for all n . One can then show by induction that $y(n) = z(n)$, for all n .

Any sequence y defined by (3.2) has a companion sequence z defined by (3.3) such that $P_2 \neq P_1$ and $y(n) = z(n)$, for $n = 0, 1, \dots, k + m + 1$. To see this consider the system of equations

$$(3.8) \quad P_0(n)y(n) + Q_0(n) = 0, \quad n = 0, 1, \dots, k + m,$$

where $P_0 \in \mathcal{P}_k$ and $Q_0 \in \mathcal{P}_m$. This linear system has more unknowns (the coefficients of P_0 and Q_0) than equations; hence, there is a nontrivial solution [14]. Now let $P_2 = P_0 + P_1$, $Q_2 = Q_0 + Q_1$, $z(0) = y(0)$, and apply (3.2), (3.3), and (3.8) to obtain

$$(3.9) \quad \begin{aligned} y(n+1) &= P_1(n)y(n) + Q_1(n) + P_0(n)y(n) + Q_0(n) \\ &= P_2(n)y(n) + Q_2(n) = z(n+1), \end{aligned}$$

for $n = 0, 1, \dots, k + m$.

If $\deg(P_0) = k - \nu$, $\deg(Q_0) = m - \mu$, with μ and $\nu > 0$, we say that the rational interpolation problem (3.8) is degenerate with deficiency $\eta = \min(\mu, \nu)$. A deficient solution to (3.8) for a nonpolynomial y and $\eta \geq 2$ would provide an example of a sequence requiring more than $k + m + 3$ samples to identify; in fact, such a sequence would require at least $k + m + 2 + \eta$ samples.

Let us now turn our attention to question (ii) which was posed in the introduction. Agreement on an appropriate initial segment of the sequences $y(n, t)$ and $z(n, t)$ for two distinct initial values t implies agreement for all n and t .

THEOREM 3.4. *Let $y(n, t)$ and $z(n, t)$ be the sequences defined by (3.2) and (3.3), respectively, with $y(0) = z(0) = t$. If $t_1 \neq t_2$ and*

$$y(n, t_j) = z(n, t_j), \quad n = 0, 1, \dots, 2k + m + 2, \quad j = 1, 2,$$

then $y(n, t) = z(n, t)$, for all $n \in N$ and $t \in \mathcal{C}$.

Proof. Let n_0 be the first member of N which is a zero of P_1 (with $n_0 = \infty$ if there are none). For $n = 0, 1, \dots, 2k + m + 1$ and $j = 1, 2$ we have

$$(3.10) \quad y(n+1, t_j) = P_1(n)y(n, t_j) + Q_1(n) = P_2(n)y(n, t_j) + Q_2(n).$$

Let $n = 0$ and note that $y(0, t_j) = t_j$. Then one can infer from (3.10) and the fact that $t_1 \neq t_2$ that $P_1(0) = P_2(0)$ and $Q_1(0) = Q_2(0)$. Furthermore, if $P_1(0) \neq 0$, then $y(1, t_1) \neq y(1, t_2)$. An inductive argument then shows that

$$(3.11) \quad P_1(n) = P_2(n), \quad Q_1(n) = Q_2(n),$$

and $y(n, t_1) \neq y(n, t_2)$, for $n = 0, 1, \dots, \min(n_0, 2k + m + 1)$. If $n_0 \geq \max(k, m)$, then (3.11) implies $P_1 = P_2$ and $Q_1 = Q_2$, from which the result follows. Otherwise $n_0 < \max(k, m)$ and $P_1(n_0) = P_2(n_0) = 0$. Then for any t ,

$$y(n_0 + 1, t) = z(n_0 + 1, t) = Q_1(n_0),$$

so the sequences $y(n, t)$ and $z(n, t)$ are independent of t for $n > n_0$. Furthermore, Theorem 3.2 implies that $y(n, t_1) = z(n, t_1)$, for all n . Thus for any t we have from (3.11)

$$y(n, t) = z(n, t), \quad n = 0, 1, \dots, n_0 + 1.$$

For $n > n_0 + 1$,

$$y(n, t) = y(n, t_1) = z(n, t_1) = z(n, t),$$

and the proof is complete.

It is not always possible to identify P_1 and Q_1 by sampling with different initial

values. For example, let

$$y(n+1, t) = ny(n, t) + 1 + n - n^2, \quad y(0, t) = t,$$

$$z(n+1, t) = 0 \cdot z(n, t) + 1 + n, \quad z(0, t) = t.$$

Note that $y(n, t) = z(n, t) = n$, for all $n > 0$ and all $t \in \mathcal{C}$. Thus the two sequences are identical, but $P_1 \neq P_2$ and $Q_1 \neq Q_2$. From the proof of Theorem 3.4 one can see that this situation can arise only when $P_1(n) = 0$ for some $n < \max(k, m)$. This possibility can be ruled out by requiring that distinct initial values produce distinct sequence values on the appropriate initial segment.

COROLLARY 3.5. *Let y and z be defined as in Theorem 3.4. If $t_1 \neq t_2$ and*

$$y(n, t_j) = z(n, t_j), \quad n = 0, 1, \dots, \max(k, m) + 1, \quad j = 1, 2;$$

$$y(n, t_1) \neq y(n, t_2), \quad n = 0, 1, \dots, \max(k, m),$$

then $P_1 = P_2$ and $Q_1 = Q_2$.

Proof. This follows from the argument used to establish (3.11).

To complete this section let us consider the case where the coefficients P and Q in (1.1) depend on a vector parameter $x \in \mathcal{C}_v$. We will assume that these coefficients are multinomials in n and x ; that is,

$$(3.12) \quad P(n, x) = \sum_{j=0}^k a_j(x)n^j, \quad Q(n, x) = \sum_{j=0}^m b_j(x)n^j,$$

where the a_j 's and b_j 's are multinomials in the components of x . Let \mathcal{M} be a class of multinomials and let $Y(k, m, \mathcal{M})$ be the set of all sequences satisfying (1.1) with P and Q defined by (3.12) and $a_j, b_j \in \mathcal{M}$, for each j . We wish to consider question (ii) for this situation.

Our basic technique will be to apply Corollary 3.5 for a fixed x to identify the coefficients $a_j(x)$ and $b_j(x)$. Then we will let x vary over a set which uniquely identifies the multinomials a_j and b_j . Details concerning tests for multinomials can be found in [11], [4], and [15]. In the following theorem $y(n, x, t)$ will denote a sequence from $Y(k, m, \mathcal{M})$ with parameter x and initial value t .

THEOREM 3.6. *Let X be a subset of \mathcal{C}_v which uniquely identifies members of \mathcal{M} , and suppose $t_1 \neq t_2$. If y and $z \in Y(k, m, \mathcal{M})$ and*

$$(3.13) \quad y(n, x, t_j) = z(n, x, t_j), \quad n = 0, 1, \dots, \max(k, m) + 1, \quad j = 1, 2,$$

$$(3.14) \quad y(n, x, t_1) \neq y(n, x, t_2), \quad n = 0, 1, \dots, \max(k, m),$$

whenever $x \in X$, then $y(n, x, t) = z(n, x, t)$ for all $n \in \mathbb{N}$, $x \in \mathcal{C}_v$, and $t \in \mathcal{C}$.

Proof. Suppose y and z are given by (3.2) and (3.3), where $P_1, Q_1, P_2,$ and Q_2 have coefficients $a_j, b_j, c_j,$ and d_j , respectively. For any $x \in X$, we infer from (3.13), (3.14), and Corollary 3.5 that $a_j(x) = c_j(x)$ and $b_j(x) = d_j(x)$, for all j . But this holds for all $x \in X$, and X uniquely identifies members of \mathcal{M} ; so $a_j = c_j, b_j = d_j$, and the proof is complete.

This theorem should be contrasted with Theorem 8 of Howden [11]. The theorems are not strictly comparable, but Howden's result is essentially more general because the starting value can be a function of x , and y need not occur linearly in the difference equation. However, Theorem 3.6 is considerably simpler to state and prove, and does not require the a posteriori demonstration of the nonsingularity of a matrix as required by Howden's theorem.

We do not as yet have any answers to questions (i), (ii) and (iii) for $Y(k, m, \mathcal{M})$ without the restriction (3.14). It would be interesting to know whether the analogues of Theorems 3.1, 3.2, and 3.4 hold for this class without this restriction.

4. Higher-order equations and systems. Higher-order equations and systems have many applications in the physical, social, and biological sciences. For example, systems of difference equations are used in economic models, control and stability theory, physics, neural network models, anthropology, probability theory, and in the numerical solution of partial differential equations. Higher-order equations have applications in psychology, inventory models, communication theory, and in the numerical solution of ordinary differential equations. Many of these applications are described by Goldberg [9]. For other applications see Feller [6], Spitzer [17], Bridge [2], Dhrymes [5], and Smithies [16].

A linear difference equation of order greater than one can be converted to a system of first-order equations. In fact, if the leading coefficient p_k does not vanish on the nonnegative integers, then (2.1) can be converted to the system:

$$(4.1) \quad \begin{aligned} y_j(n+1) &= y_{j+1}(n), & j &= 1, 2, \dots, k-1, \\ y_k(n+1) &= -\frac{1}{p_k(n)} \sum_{j=0}^{k-1} p_j(n) y_{j+1}(n) + \frac{q(n)}{p_k(n)}. \end{aligned}$$

It is easy to show that y_1, \dots, y_k satisfy (4.1) if and only if y_1 satisfies (2.1). For this reason we will restrict our attention to systems of first-order equations.

Let us first consider two homogeneous systems of linear difference equations with constant coefficients:

$$(4.2) \quad y(n+1) = Ay(n), \quad y(0) = t,$$

$$(4.3) \quad z(n+1) = Bz(n), \quad z(0) = t,$$

where A and B are ν by ν constant matrices, and y and z are ν by 1 column vectors. The choice

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix}, \quad t = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

in (4.2) and (4.3) gives the solution $y(n) = z(n) = 2^n t$. Thus it is possible for a sequence to have two distinct representations of the type (4.2). The following theorem shows that this can happen only when the initial vector t has the same spectral decomposition with respect to A and B .

THEOREM 4.1. *Suppose y and z satisfy (4.2) and (4.3) respectively. Then $y(n) = z(n)$ for all $n \in \mathbb{N}$ if and only if the initial vector t has the same spectral components with respect to A and B .*

Proof. Note that $y(n) = A^n t$ and $z(n) = B^n t$. Let $\lambda_1, \dots, \lambda_k; \mu_1, \dots, \mu_m$ be the distinct eigenvalues of A and B , respectively, with multiplicities ρ_1, \dots, ρ_k and $\sigma_1, \dots, \sigma_m$ in the corresponding minimal polynomials. Let us further assume that any common eigenvalues are indexed first; so that $\lambda_i = \mu_i$, $i = 1, \dots, k_1$ and $\lambda_i \neq \mu_i$, for $i > k_1$. Let $f_n(x) = x^n$. Then according to (2.5) we have

$$(4.4) \quad \begin{aligned} y(n) &= f_n(A)t = \sum_{i=1}^k \sum_{j=0}^{\rho_i-1} f_n^{(j)}(\lambda_i) Y_{ij} t, \\ z(n) &= f_n(B)t = \sum_{i=1}^m \sum_{j=0}^{\sigma_i-1} f_n^{(j)}(\mu_i) Z_{ij} t, \end{aligned}$$

where the Y_{ij} 's and Z_{ij} 's are the component matrices of A and B , respectively. For convenience, let $Y_{ij} = 0$ if $j \geq \rho_i$ and $Z_{ij} = 0$ if $j \geq \sigma_i$. Define $\tau_i = \max(\rho_i, \sigma_i)$. With this

notation we obtain from (4.4) the identity

$$\begin{aligned}
 (4.5) \quad y(n) - z(n) &= \sum_{i=1}^{k_1} \sum_{j=0}^{\tau_i-1} f_n^{(j)}(\lambda_i)(Y_{ijt} - Z_{ijt}) \\
 &+ \sum_{i=k_1+1}^k \sum_{j=0}^{\rho_i-1} f_n^{(j)}(\lambda_i) Y_{ijt} \\
 &- \sum_{i=k_1+1}^m \sum_{j=0}^{\sigma_i-1} f_n^{(j)}(\mu_i) Z_{ijt}.
 \end{aligned}$$

Note that

$$\begin{aligned}
 f_n^{(0)}(\lambda_i) &= \lambda_i^n, & f_n^{(1)}(\lambda_i) &= \frac{n}{\lambda_i} \lambda_i^n, \\
 f_n^{(2)}(\lambda_i) &= \frac{n(n-1)\lambda_i^n}{\lambda_i^2}.
 \end{aligned}$$

It then follows from (2.3) and (2.4) that the sequences

$$\begin{aligned}
 f_n^{(j)}(\lambda_i), & \quad i = 1, \dots, k_1, \quad j = 0, 1, \dots, \tau_i - 1; \\
 f_n^{(j)}(\lambda_i), & \quad i = k_1 + 1, \dots, k, \quad j = 0, 1, \dots, \rho_i - 1; \\
 f_n^{(j)}(\mu_i), & \quad i = k_1 + 1, \dots, m, \quad j = 0, 1, \dots, \sigma_i - 1;
 \end{aligned}$$

form a linearly independent set. (In fact, they are independent solutions to the difference equation of the form (2.2) whose characteristic polynomial is the least common multiple of the minimal polynomials of A and B .) If $y(n) = z(n)$, for all $n \in N$, then the linear independence implies that the coefficients of the $f_n^{(j)}(\lambda_i)$'s and $f_n^{(j)}(\mu_i)$'s on the right side of (4.5) must vanish. Thus

$$\begin{aligned}
 Y_{ijt} &= Z_{ijt}, & i = 1, 2, \dots, k, \quad j = 0, 1, \dots, \tau_i - 1, \\
 Y_{ijt} &= Z_{ijt} = 0, & i > k_1,
 \end{aligned}$$

and the spectral components of t are the same for A and B . Conversely, if the spectral components agree, then the right side of (4.5) vanishes identically and $y(n) = z(n)$, for all $n \in N$.

A necessary condition for the agreement of y and z can be given in terms of the principal vector decomposition of t .

COROLLARY 4.2. *Let y and z satisfy (4.2) and (4.3), respectively. If $y(n) = z(n)$, for all n , then the initial vector t has the same principal vector decomposition with respect to A and B .*

Proof. Using the notation from Theorem 4.1, let U_i and V_i be the null spaces of $(A - \lambda_i I)^{\rho_i}$ and $(B - \lambda_i I)^{\sigma_i}$, respectively. The principal vector decompositions [8] of t are given (uniquely) by

$$\begin{aligned}
 t &= u_1 + \dots + u_k, \\
 t &= v_1 + \dots + v_m,
 \end{aligned}$$

where $u_i \in U_i$ and $v_i \in V_i$. It is shown in [12, pp. 177–178] that $Y_{i0t} \in U_i$, $Z_{i0t} \in V_i$, and

$$(4.6) \quad t = \sum_{i=1}^k Y_{i0t} = \sum_{i=1}^m Z_{i0t}.$$

Thus $u_i = Y_{i0t}$, $v_i = Z_{i0t}$ and the result now follows from Theorem 4.1.

We will call (λ, x) a common eigenvalue-vector pair of A and B if $x \neq 0$ and $Ax = Bx = \lambda x$. The next result relates the existence of a common eigenvalue-vector pair to the identity of y and z .

COROLLARY 4.3. *Let y and z satisfy (4.2) and (4.3), respectively. If A and B have a common eigenvalue-vector pair, then there is an initial vector t such that $y(n) = z(n)$ for all n . Conversely, if $y(n) = z(n)$ for all n and $t \neq 0$, then A and B have a common eigenvalue-vector pair.*

Proof. Let (λ, x) be a common eigenvalue-vector pair. Using x as the initial vector we get

$$y(n) = A^n x = \lambda^n x = B^n x = z(n),$$

for all n . For the converse, note that the existence of a common eigenvalue follows from the equality of the spectral components of t with respect to A and B . From (4.6) and the fact that $t \neq 0$, we conclude that there is an index $i \leq k_1$ such that $\lambda_1 = \mu_i$ and $Y_{i0}t = Z_{i0}t \neq 0$. Now $Y_{i0}t$ belongs to the nullspaces of $(A - \lambda_i I)^{\tau_i}$ and $(B - \lambda_i I)^{\tau_i}$. Let j_0 , $0 \leq j_0 < \tau_i$ be the largest index for which $(A - \lambda_i I)^{j_0} Y_{i0}t \neq 0$, and let $x = (A - \lambda_i I)^{j_0} Y_{i0}t$. Then x is an eigenvector of A corresponding to λ_i , since

$$(A - \lambda_i I)x = (A - \lambda_i I)^{j_0+1} Y_{i0}t = 0.$$

Now it is shown in [12, p. 177] that

$$Y_{ij} = \frac{1}{j!} (A - \lambda_i I)^j Y_{i0},$$

$$Z_{ij} = \frac{1}{j!} (B - \lambda_i I)^j Z_{i0}.$$

Then from the equality of the spectral components of t we see that $x = j_0! Y_{i0}t = j_0! Z_{i0}t$; hence

$$(4.7) \quad (B - \lambda_i I)x = (B - \lambda_i I)^{j_0+1} Z_{i0}t.$$

Now if $j_0 + 1 = \tau_i$, then the right side of (4.7) is zero because $Z_{i0}t$ is in the nullspace of $(B - \lambda_i I)^{\tau_i}$. Otherwise

$$\begin{aligned} (B - \lambda_i I)x &= (j_0 + 1)! Z_{i, j_0+1} t = (j_0 + 1)! Y_{i, j_0+1} t \\ &= (A - \lambda_i I)^{j_0+1} Y_{i0} t = 0. \end{aligned}$$

In either case x is an eigenvector of B and the proof is complete.

Next, we will give several sampling theorems for homogeneous systems. Let M be the smallest degree of the minimal polynomials for A and B .

THEOREM 4.4. *Let y and z satisfy (4.2) and (4.3), respectively. If $y(n) = z(n)$, for $n = 0, 1, \dots, M$, then $y(n) = z(n)$, for all $n \in N$.*

Proof. Let ϕ be the minimal polynomial for A . We may suppose without loss of generality that $M = \deg(\phi)$. Now the coefficient of A^M in $\phi(A)$ cannot be zero; so we can solve the equation $\phi(A) = 0$ to obtain the expression of the form

$$A^M = \sum_{j=0}^{M-1} a_j A^j.$$

Multiply both sides of this equation by $A^n t$ (using multiplication on the right), and note that $y(n) = A^n t$ to obtain

$$y(n+M) = \sum_{j=0}^{M-1} a_j y(n+j).$$

Proceeding by induction, let us suppose that $y(i) = z(i)$, for $i = 0, 1, \dots, n$, with $n \geq M$. Then

$$\begin{aligned} z(n+1) = Bz(n) = By(n) &= \sum_{j=0}^{M-1} a_j By(n+j-M) \\ &= \sum_{j=0}^{M-1} a_j y(n+1+j-M) = y(n+1). \end{aligned}$$

Thus $y(n) = z(n)$ for all n and the proof is complete.

In general one would not necessarily know the value of M . In this case it would suffice to use $0, 1, \dots, \nu$ for the sample because $M \leq \nu$.

Let us now turn our attention to nonhomogeneous systems of the form

$$(4.8) \quad y(n+1) = Ay(n) + Q(n), \quad y(0) = t,$$

$$(4.9) \quad z(n+1) = Bz(n) + R(n), \quad z(0) = t,$$

where A, B are ν by ν constant matrices, and Q, R are ν by 1 column vectors with polynomial components of degree at most m . The next theorem shows that y and z are identical if they agree on a sufficiently large initial segment. As before, let M be the smallest degree of the minimal polynomials for A and B .

THEOREM 4.5. *Let y and z satisfy (4.8) and (4.9), respectively. If $y(n) = z(n)$, for $n = 0, 1, \dots, M+m+1$, then $y(n) = z(n)$, for all $n \in \mathcal{N}$.*

Proof. Note that the forward difference operator Δ commutes with a constant matrix A because

$$A\Delta y(n) = A(y(n+1) - y(n)) = Ay(n+1) - Ay(n) = \Delta Ay(n).$$

Also note that $\Delta^{m+1}Q(n) = \Delta^{m+1}R(n) = 0$ because Q_i and $R_i \in \mathcal{P}_m$ for each i . Let $u(n) = \Delta^{m+1}y(n)$ and $v(n) = \Delta^{m+1}z(n)$. The application of Δ^{m+1} to both sides of (4.8) and (4.9) yields the systems

$$(4.10) \quad \begin{aligned} u(n+1) &= Au(n), & u(0) &= \Delta^{m+1}y(0), \\ v(n+1) &= Bv(n), & v(0) &= \Delta^{m+1}z(0). \end{aligned}$$

Now $y(n) = z(n)$, for $n = 0, 1, \dots, M+m+1$, implies that $u(n) = v(n)$, for $n = 0, 1, \dots, M$. According to Theorem 4.4, $u(n) = v(n)$ for all n ; thus $\Delta^{m+1}[y(n) - z(n)] = 0$ for all n . This can happen only if each component $y_i - z_i \in \mathcal{P}_m$. But $y(n) - z(n) = 0$ for $n = 0, 1, \dots, m$; hence $y(n) - z(n) = 0$ for all n and the proof is complete.

Note that if M is not known, it suffices to sample y and z , for $n = 0, 1, \dots, \nu + m + 1$, since $M \leq \nu$.

As in the homogeneous case, we can show that A and B must share certain spectral properties if $y(n) = z(n)$ for all n .

COROLLARY 4.6. *Let y and z satisfy (4.8) and (4.9), respectively. If $y(n) = z(n)$, for all n , then*

- (i) $\Delta^{m+1}y(0)$ has the same spectral components with respect to A and B ;
- (ii) $\Delta^{m+1}y(0)$ has the same principal vector representation with respect to A and B ;
- (iii) if $\Delta^{m+1}y(0) \neq 0$, then A and B have a common eigenvalue-vector pair.

Proof. These results follow directly from (4.10) and Theorems 4.1–4.3.

In view of (4.10), one can see that if $\Delta^{m+1}y(0) = 0$, then $\Delta^{m+1}y(n) = 0$, for all n , and $y_i \in \mathcal{P}_m$ for each i . Conversely, if $y_i \in \mathcal{P}_m$ for each i , then $\Delta^{m+1}y(0) = 0$. Thus the condition $\Delta^{m+1}y(0) \neq 0$ in (iii) is equivalent to requiring that at least one component of y not be a polynomial of degree $\leq m$.

Let us now consider the analogues of Theorems 3.4 and 3.6 for systems of difference equations.

THEOREM 4.7. *Let $y(n, t)$ and $z(n, t)$ be the sequences defined by (4.8) and (4.9), respectively, with $y(0) = z(0) = t$. Let T be a basis for the Euclidean space \mathcal{E}_v . If*

$$(i) \quad y(n, 0) = z(n, 0), \quad n = 1, 2, \dots, m+1, \text{ and}$$

$$(ii) \quad y(1, t) = z(1, t), \text{ for all } t \in T,$$

then $y(n, t) = z(n, t)$, for all $n \in \mathbb{N}$ and $t \in \mathcal{E}_v$.

$$(ii) \quad y(1, t) = z(1, t), \text{ for all } t \in T, \text{ then } y(n, t) = z(n, t), \text{ for all } n \in \mathbb{N} \text{ and } t \in \mathcal{E}_v.$$

Proof. Condition (i) implies that

$$y(1, 0) - z(1, 0) = Q(0) - R(0) = 0.$$

This combined with (ii) implies

$$y(1, t) - z(1, t) = (A - B)t = 0$$

for all $t \in T$. Thus $A = B$. The fact that $Q(n) - R(n) = 0$, $n = 0, 1, \dots, m$ now follows from (i). But $Q_i - R_i \in \mathcal{P}_m$; so $Q = R$ and the proof is complete.

To complete this section let us consider the case where the coefficients of Q, R, A , and B are multinomials in a vector parameter x . That is,

$$Q_j(n, x) = \sum_{j=0}^m q_j(x)n^j, \quad R_j(n, x) = \sum_{j=0}^m r_j(x)n^j,$$

$$A = (a_{ij}(x)), \quad B = (b_{ij}(x)),$$

where the q_j 's, r_j 's, a_{ij} 's and b_{ij} 's are multinomials from a class \mathcal{M} . Let X be a test set which uniquely identifies members of \mathcal{M} .

THEOREM 4.8. *If*

$$(i) \quad y(n, x, 0) = z(n, x, 0), \text{ for all } x \in X \text{ and } n = 1, 2, \dots, m+1, \text{ and}$$

$$(ii) \quad y(1, x, t) = z(1, x, t), \text{ for all } x \in X \text{ and } t \in T,$$

then $y(n, x, t) = z(n, x, t)$, for all n, x and t .

Proof. For a fixed $x \in X$, Theorem 4.7 implies that $q_j(x) = r_j(x)$, $a_{ij}(x) = b_{ij}(x)$. But X uniquely identifies members of \mathcal{M} ; so $q_j(x) = r_j(x)$ and $a_{ij}(x) = b_{ij}(x)$ for all x . It follows that $y(n, x, t) = z(n, x, t)$, for all n, x , and t .

5. Implications concerning mathematical induction. It is interesting to observe that the theory presented here can be used to prove the validity of certain formulas by sampling rather than mathematical induction. For example, consider the formula

$$(5.1) \quad 1 \cdot 2 + 2 \cdot 3 + \dots + n(n+1) = \frac{1}{3}n(n+1)(n+2).$$

Note that this formula is valid for $n = 0, 1, 2, 3, 4, 5$. This is sufficient to conclude that the formula is valid for all natural numbers n . To see this, let $y(n)$ represent the left side of (5.1), $z(n)$ the right side, and note that

$$y(n+1) = y(n) + (n+1)(n+2), \quad y(0) = 0,$$

$$z(n+1) = 0 \cdot z(n) + \frac{1}{3}(n+1)(n+2)(n+3), \quad z(0) = 0.$$

Thus y and $z \in Y(0, 3)$, and Theorem 3.2 implies that $y(n) = z(n)$, for all n .

This example appears to contradict the popular notion that a finite sample is never sufficient to establish the identity of two sequences defined on the natural numbers. This generalization is valid when there is no restriction on the sequences; however, finite sampling is sufficient for certain classes of sequences. Of course, we are not really avoiding mathematical induction in a fundamental sense since it was used in the proof of Theorem 3.2.

6. Examples. We will present several examples to illustrate our theory. No paper on testing or correctness would be complete without an application of the results to the factorial function!

Example 6.1. Let $y(n) = n!$ and note that y satisfies the difference equation

$$y(n+1) = (n+1)y(n), \quad y(0) = 1.$$

Thus $y \in Y(1, 1)$. If we assume that the function subroutine to be tested also produces a sequence $z \in Y(1, 1)$, then according to Theorem 3.2 it suffices to test the program for $n = 0, 1, 2, 3, 4, 5$. This agrees with the test set obtained by Howden in [11].

Example 6.2. Consider the function y defined by

$$y(n, x) = 1 + x + x^2 + \cdots + x^n.$$

For testing purposes it is convenient to consider y as a function of the initial value t as well as of n and x . Then the choice $t = 1$ in the following difference equation yields the desired function:

$$y(n+1, x, t) = xy(n, x, t) + 1, \quad y(0) = t.$$

Thus $y \in Y(0, 0, \mathcal{P}_1)$ and Theorem 3.6 suggests sampling $y(n, x, t)$ for the eight triples

$$(n, x, t), \quad n = 0, 1; \quad x = 0, 1; \quad t = 0, 1.$$

(Two values of x uniquely identify members of \mathcal{P}_1 .)

Note that this function could also be generated by the system of equations

$$\begin{aligned} y(n+1, x, t) &= y(n, x, t) + w(n), & y(0) &= t_1, \\ w(n+1, x, t) &= xw(n, x, t), & w(0) &= t_2. \end{aligned}$$

The choice $t_1 = 0, t_2 = 1$ yields the desired function with the index shifted by 1. Theorem 4.8 yields the test set formed of two parts:

- (i) $n = 1; x = 0, 1; t_1 = t_2 = 0;$
- (ii) $n = 1; x = 0, 1; t_1 = 1, t_2 = 0, 1.$

Example 6.3. A model for price competition [16] among 20 companies uses the model

$$y(n+1) = By(n) + Q,$$

where $y(n)$ is a vector whose coordinates represent the price for each competitor at time n , B is a 20 by 20 matrix, and Q is a vector. The components of B and Q are constants with respect to n which are related to the demands and costs associated with each competitor's product. For a fixed choice of B and Q Theorem 4.5 would suggest that a program for this model be tested on the values $n = 0, 1, 2, \dots, 21$.

Example 6.4. The second-order scalar equation

$$y(n+2) - \beta(1+\eta)y(n+1) + \eta\beta y(n) = v_0$$

has been used for an inventory control model, where y is the income function, β is the marginal propensity to consume, and η is a coefficient of expectation [9]. This equation can be converted to the system of two first-order equations:

$$(6.1) \quad \begin{aligned} y_1(n+1) &= y_2(n), \\ y_2(n+1) &= \beta(1+\eta)y_2(n) - \eta\beta y_1(n) + v_0. \end{aligned}$$

For fixed starting values and parameters β, η , and v_0 , Theorem 4.5 would suggest that a program for this model be tested for $n = 0, 1, 2, 3$. On the other hand, if the starting

value t and parameters β , η , and v_0 are input variables, we could apply Theorem 4.8. This theorem would imply that it would be sufficient to test on the 16 points obtained from the combinations of

$$n = 1; \quad \beta = 0, 1; \quad \eta = 0, 1; \quad v_0 = 0, 1; \quad t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Here we have taken $m = 0$ and assumed that the coefficients on the right side of (6.1) are multinomials of degree ≤ 1 in β , η , and v_0 .

Finally, we will give an example for which our theory does not apply.

Example 6.5. The Taylor polynomial about zero for the exponential function satisfies the system:

$$\begin{aligned} y(n+1, x) &= y(n, x) + w(n, x), & y(0) &= 1, \\ w(n+1) &= \frac{w(n, x)x}{n+2}, & w(0) &= x. \end{aligned}$$

This fails to fit our theory because the coefficient of $w(n, x)$ in the second equation is not a constant. This points out the desirability of extending the results of § 4 to allow the elements of the matrices A and B in (4.8) and (4.9) to be rational functions of n .

REFERENCES

- [1] L. BRAND, *Differential and Difference Equations*, John Wiley, New York, 1966.
- [2] J. L. BRIDGE, *Applied Econometrics*, North-Holland, Amsterdam, 1971.
- [3] T. A. BUDD, R. J. LIPTON, F. G. SAYWARD, AND R. A. DEMILLO, *The design of a prototype mutation system for program testing*, AFIPS Conference Proceedings, 47 (1978), pp. 623-627.
- [4] P. J. DAVIS, *Proof, completeness, transcendentals, and sampling*, J. Assoc. Comput. Mach., 24 (1977), pp. 298-310.
- [5] P. J. DHRYMES, *Econometrics: Statistical Foundations and Applications*, Harper and Row, New York, 1970.
- [6] W. FELLER, *Probability Theory and its Applications*, Vol. 1, John Wiley, New York, 1950.
- [7] T. FORT, *Finite Differences and Difference Equations in the Real Domain*, Oxford, London, 1948.
- [8] J. N. FRANKLIN, *Matrix Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1968.
- [9] S. GOLDBERG, *Introduction to Difference Equations*, John Wiley, New York, 1958.
- [10] J. B. GOODENOUGH AND S. L. GERHART, *Toward a theory of test data selection*, Proceedings International Conference on Reliable Software, IEEE, New York, 1975.
- [11] W. E. HOWDEN, *Elementary algebraic program testing techniques*, Technical Report 12, Computer Science Department, University of California at San Diego, La Jolla, Cal., 1976.
- [12] P. LANCASTER, *Theory of Matrices*, Academic Press, New York, 1969.
- [13] L. M. MILNE-THOMSON, *The Calculus of Finite Differences*, Macmillan, London, 1951.
- [14] T. J. RIVLIN, *An Introduction to the Approximation of Functions*, Blaisdell, Waltham, MA, 1969.
- [15] J. H. ROWLAND AND P. J. DAVIS, *On the use of transcendentals in program testing*, J. Assoc. Comput. Mach., to appear.
- [16] A. SMITHIES, *The stability of competitive equilibrium*, Econometrica 10 (1942), pp. 258-274.
- [17] F. SPITZER, *Principles of Random Walk*, Van Nostrand, Princeton, NJ, 1964.

GENERALIZATION OF VORONOI DIAGRAMS IN THE PLANE*

D. T. LEE† AND R. L. DRYSDALE, III‡

Abstract. In this paper we study the Voronoi diagram for a set of N line segments and circles in the Euclidean plane. The diagram is a generalization of the Voronoi diagram for a set of points in the plane and has applications in wire layout, facility location, clustering and contouring problems. We present an $O(N(\log N)^2)$ algorithm for constructing the diagram. It is an improvement of a previous known result which takes $O(Nc^{\sqrt{\log N}})$ time. The algorithm described in this paper is also shown to be applicable under a more general metric if certain conditions are satisfied.

Key words. Voronoi diagram, computational geometry, point location, computational complexity, divide-and-conquer, analysis of algorithms

1. Introduction. The Voronoi diagram [18] for a set of N points in the Euclidean plane has been studied by a number of people [8], [11], [12], [17], [20], [22]. Essentially, a Voronoi diagram is a partition of the plane into N polygonal regions, each of which is associated with a given point. The region associated with a point is the locus of points closer to that point than to any other given point. Shamos and Hoey [21] present a divide-and-conquer algorithm which computes the diagram in $O(N \log N)$ time and show that the all nearest-neighbor problem [10], the largest empty circle problem and other seemingly unrelated problems can all be solved very efficiently once the diagram is available. They point out that such problems arise in wire layout [1], facility location [5], cutting-stock problems and geometric optimization problems [19], clustering problems [7] and contouring problems [2].

A natural question to ask is whether the algorithm for computing the Voronoi diagram can be generalized to other figures and metrics. These generalized Voronoi diagrams would have many applications. Lee and Wong [13] study the Voronoi diagram for a set of points under the L_1 - and L_∞ -metrics¹ and point out that these diagrams speed up retrieval algorithms for two-dimensional storage systems. Shamos [20] poses the problem of finding minimum weight spanning trees for circles and line segments.

These spanning trees can be computed very quickly if the Voronoi diagram for the circles and line segments is known. A country's territorial waters consist of its Voronoi region intersected with the locus of all points within 200 miles of a point in the country, so Voronoi diagrams are important when computing territorial waters. Many of the applications mentioned above have analogues where the objects considered are better represented by polygons, circles, or other geometric figures than by points.

In a previous paper [4] we presented an $O(Nc^{\sqrt{\log N}})$ algorithm for constructing the Voronoi diagram for circles and line segments in the Euclidean plane. We showed that the algorithm could also be used for more general figures and metrics if certain conditions were satisfied. We now present an $O(N(\log N)^2)$ algorithm for computing the Voronoi diagram for circles or line segments in the Euclidean plane. This algorithm can be applied under a more general metric if certain conditions are met.

* Received by the editors March 2, 1978, and in final form April 18, 1980.

† Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60201. The research of this author was supported in part by the National Science Foundation under grant MCS76-17321 and in part by the Joint Services Electronics Program under Contract DAAB-07-72-0259.

‡ Department of Mathematics, Dartmouth College, Hanover, New Hampshire 03755. The research of this author was supported by the National Science Foundation under grant MCS77-05313.

¹F. Hwang [9] also studies the Voronoi diagram for a set of points under the L_1 -metric and independently discovers an $O(N \log N)$ time algorithm for constructing it.

2. Preliminaries. We begin by introducing some definitions and notations.

DEFINITION 1. A closed line segment $\overline{a, b}$ consists of two endpoints a and b , and a straight-line portion which is denoted by (a, b) and referred to as an open segment, or briefly a segment. Points or segments are called elements. The straight line containing $\overline{a, b}$ is denoted by $\overleftrightarrow{a, b}$. The same line directed from a to b is denoted by $\overrightarrow{a, b}$.

DEFINITION 2. The projection $p(q, \overline{a, b})$ of a point q onto a closed segment $\overline{a, b}$ is the intersection of the line $\overleftrightarrow{a, b}$ and the line perpendicular to $\overleftrightarrow{a, b}$ and passing through q .

DEFINITION 3. The distance $d(q, \overline{a, b})$ between a point q and a closed segment $\overline{a, b}$ in the Euclidean metric is defined as the distance $d(q, p(q, \overline{a, b}))$ between the point q and its projection onto $\overline{a, b}$ if $p(q, \overline{a, b})$ belongs to $\overline{a, b}$ and is $\text{MIN}(d(q, a), d(q, b))$ otherwise. In other words, $d(q, \overline{a, b}) = \text{MIN}_{u \in \overline{a, b}} d(q, u)$. The point of $\overline{a, b}$ which is closest to q is called the image $I(q, \overline{a, b})$ of q on $\overline{a, b}$.

DEFINITION 4. The bisector $B(e_i, e_j)$ of two elements e_i and e_j is the locus of points equidistant from e_i and e_j . The bisector $B(X, Y)$ of two sets of elements X and Y is defined to be the locus of points equidistant from X and Y , where the distance $d(q, X)$ between a point q and a set of elements X is defined to be $\text{MIN}_{e \in X} d(q, e)$. The bisector $B(e_i, e_j)$ is said to be oriented if a direction is imposed upon it so that elements e_i and e_j lie to the left and to the right of it respectively. An oriented bisector $B(X, Y)$ is defined similarly.

For example, in Fig. 1 the bisector $B(q, \overline{a, b})$ of a point q and a closed segment $\overline{a, b}$ has three components, i.e., $B(q, a)$, $B(q, b)$, and a portion of the parabola² whose focus and directrix are the point q and the line $\overleftrightarrow{a, b}$ respectively. Fig. 2 shows the bisector $B(\overline{a, b}, \overline{c, d})$ of two closed segments $\overline{a, b}$ and $\overline{c, d}$, where one of the components is a portion of the angular bisector of the angle formed by the lines $\overleftrightarrow{a, b}$ and $\overleftrightarrow{c, d}$. In what follows we shall also consider the bisector $B(q, (a, b))$ of a point q and a segment (a, b) to be the same as $B(q, \overline{a, b})$. Similarly, the bisector $B((a, b), (c, d))$ of two segments (a, b) and (c, d) is the same as $B(\overline{a, b}, \overline{c, d})$. We define the bisector $B(a, \overline{a, b})$ to be a line perpendicular to $\overleftrightarrow{a, b}$ and passing through a .

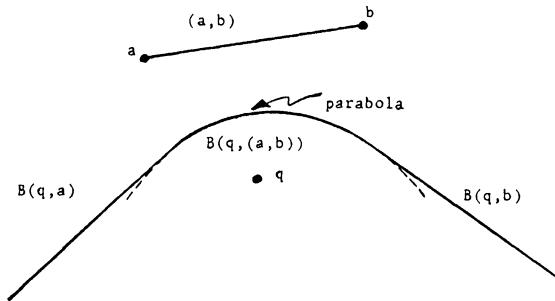


FIG. 1. Bisector of a point and a closed segment.

With these definitions we can define the half-plane $h(e_i, e_j)$ as the locus of points closer to element e_i than to element e_j . The Voronoi region $V(e_i)$ is the intersection of all the half-planes containing e_i ; i.e., $V(e_i) = \bigcap_{j \neq i} h(e_i, e_j)$, which is the locus of points

² We shall use the term parabola to mean the portion of the parabola whose projection belongs to the segment.

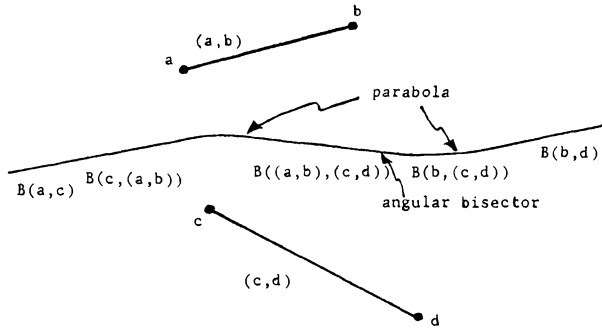


FIG. 2. Bisector of two closed segments.

closer to e_i than to any other element. The boundary edges of a Voronoi region are called *Voronoi edges*, and vertices of the region are called *Voronoi points*. Finally, we can define the Voronoi diagram $V(S)$ for a set S of disjoint closed segments $\{s_1, s_2, \dots, s_n\}$ as the union of Voronoi regions $V(s_i)$; each $V(s_i)$ in turn is the union of three Voronoi regions, each associated with an element of the closed segment s_i (two endpoints and an open segment). Fig. 3 shows the Voronoi diagram for two closed segments. Since

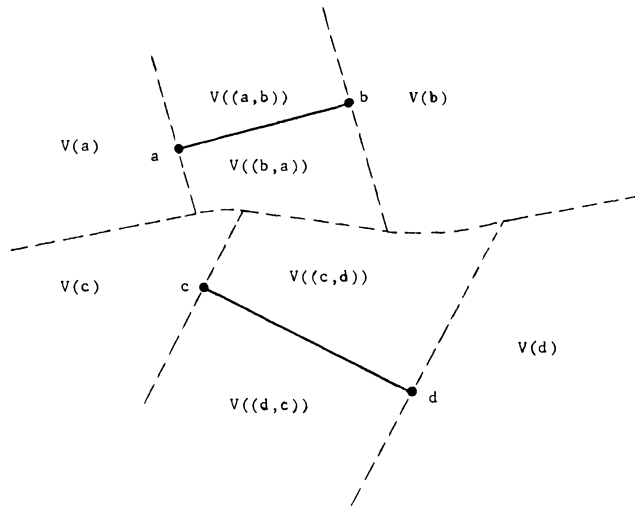


FIG. 3. Voronoi diagram for two closed segments.

associated with each segment there are two regions (one lying on each side of the segment), for ease of reference we shall denote the region that lies to the left of the segment (a, b) directed from a to b by $V((a, b))$ and the other by $V((b, a))$. From now on we shall consider the closed segment $\overline{a, b}$ to be composed of *four* elements, (instead of *three*), i.e., two endpoints of a and b and two directed segments (a, b) and (b, a) . As a consequence, the Voronoi region $V(\overline{a, b})$ associated with the closed segment $\overline{a, b}$ is then the union of $V(a)$, $V(b)$, $V((a, b))$ and $V((b, a))$. Unless otherwise stated, in what follows we shall consider the set S to be composed of elements; i.e., the Voronoi diagram $V(S)$ is a collection of Voronoi regions $V(e_i)$ which is the locus of points closer to element e_i than to any other element.

DEFINITION 5. A polygonal region R in the plane is *generalized-star-shaped* with nucleus C , $C \subseteq R$, if for any point $r \in R$ there exists a point $c \in C$ such that the line segment $\overline{r, c}$ lies completely within R .

DEFINITION 6. The *dual* $D(S)$ of the Voronoi diagram $V(S)$ for a set of N closed segments is a graph with $4N$ nodes each of which corresponds to an element of S ; two nodes are connected by an edge if their associated Voronoi regions share a Voronoi edge. The segment (a, b) is considered a Voronoi edge bordering the regions $V((a, b))$ and $V((b, a))$. (Fig. 4 shows the dual graph of the diagram in Fig. 3.)

Now let us state the properties of the Voronoi diagram for closed segments. Assume that the given set S of elements is $\{e_1, e_2, \dots, e_n\}$.

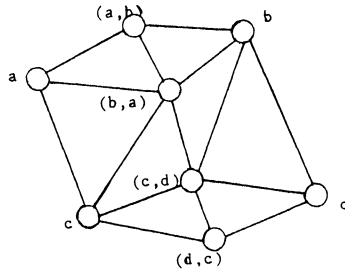


FIG. 4. Dual graph of the diagram of Fig. 3.

LEMMA 1. Given a point q , if $q \in V(e_i)$, then $\overline{q, q_i}$ lies completely in $V(e_i)$, where q_i is the image $I(q, e_i)$ of q on e_i . This shows that $V(e_i)$ is *generalized-star-shaped* with nucleus e_i .

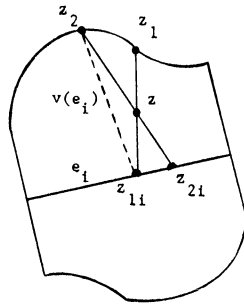
Proof. It suffices to show that any point z on $\overline{q, q_i}$ must belong to $V(e_i)$. Suppose $z \in V(e_j)$, i.e., $d(z, e_j) < d(z, e_i)$. Let z_j be the image $I(z, e_j)$ of z on e_j . By the triangle inequality, $d(q, z_j) \leq d(q, z) + d(z, z_j)$. Since $d(z, z_j) = d(z, e_j) < d(z, e_i)$, we have $d(q, z_j) < d(q, z) + d(z, e_i)$. Now $d(z, e_i) = d(z, q_i)$ and $d(q, z) + d(z, q_i) = d(q, q_i)$ imply that $d(q, z_j) < d(q, q_i) = d(q, e_i)$. Since $d(q, e_j) \leq d(q, z_j)$ by Definition 3, we have $d(q, e_j) < d(q, e_i)$ which contradicts the assumption that $q \in V(e_i)$. \square

LEMMA 2. The Voronoi regions $V(e_i)$ and $V(e_j)$ share an edge if and only if there exists a point q such that the circle centered at q with radius $d(q, e_i) = d(q, e_j)$ does not contain any point of other elements in its interior or on its boundary.

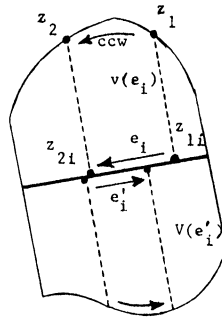
Proof. If $V(e_i)$ and $V(e_j)$ share an edge, then obviously the circle centered at any point q on the edge with radius $d(q, e_i) = d(q, e_j)$ will not contain any point of other elements. If the circle centered at q passes through the images $I(q, e_i)$ and $I(q, e_j)$ and does not include any point of other elements in its interior or on its boundary, then the nearest neighbor of q is either e_i or e_j ; i.e., $\overline{q, I(q, e_i)}$ and $\overline{q, I(q, e_j)}$ lie entirely in $V(e_i)$ and $V(e_j)$ respectively. Therefore, $V(e_i)$ and $V(e_j)$ must share an edge. \square

THEOREM 1. Let z_1 and z_2 be two points on the boundary of $V(e_i)$ and z_{1i} and z_{2i} the images of z_1 and z_2 on e_i , respectively. Either one of the two closed segments $\overline{z_1, z_{1i}}$ and $\overline{z_2, z_{2i}}$ properly contains the other or they do not intersect except possibly at endpoints. (z_{1i} and z_{2i} may coincide.)

Proof. Suppose that neither of $\overline{z_1, z_{1i}}$ and $\overline{z_2, z_{2i}}$ properly contains the other. Assume that they intersect at a point z . Without loss of generality we may assume that $d(z, z_{1i}) \leq d(z, z_{2i})$ (Fig. 5a). We have $d(z_2, z_{2i}) = d(z_2, z) + d(z, z_{2i}) \leq d(z_2, z) + d(z, z_{1i}) > d(z_2, z_{1i})$. Since z_{2i} is the image of z_2 on e_i , we have a contradiction. \square



(a)



(b)

FIG. 5. Illustration of the proof of Theorem 1.

What this theorem implies is that if we move a point z along the boundary of $V(e_i)$ in a counterclockwise or a clockwise direction, then the image $I(z, e_i)$ also moves in the same direction along e_i . If the element e_i is a point, e_i itself is the image $I(z, e_i)$ for all $z \in V(e_i)$. Fig. 5b illustrates this phenomenon.

THEOREM 2. *The element e_i is on the convex hull $CH(S)$ of S if and only if its associated Voronoi region is unbounded.*

Proof. We first prove that if an element e_i is on the convex hull, its associated Voronoi region is unbounded. Take a point z of e_i which is on the convex hull $CH(S)$. z being on $CH(S)$, there exists a supporting line $l(z)$ of $CH(S)$ which passes through z with the property that all the elements including e_i lie on one side of the line. Consider the closed segment $\overline{z, t}$ perpendicular to $l(z)$ and lying outside of $CH(S)$. We have $d(t, z) = \text{MIN}_{a \in e_i} d(t, a)$. Since the circle centered at t with radius $d(z, t)$ does not contain any other point of S in its interior, we have $d(t, z) = \text{MIN}_{a \in S} d(t, a)$, i.e., t must belong to the Voronoi region $V(e_i)$. Moreover, the endpoint t of the closed segment $\overline{z, t}$ can be extended arbitrarily, so the Voronoi region $V(e_i)$ is unbounded.

To prove the converse we shall show that for any element that is not on $CH(S)$, its associated Voronoi region must be bounded. Suppose e_i is not on $CH(S)$. Any line intersecting e_i will divide the set of elements into two nonempty subsets. Let l be a line intersecting e_i at a point z_i of e_i . Consider the line $\overline{z_i, z}$ perpendicular to the line l . The circle centered at z with radius $d(z, z_i)$ becomes the line l if we let $d(z, z_i)$

approach infinity. Therefore, when $d(z, z_i)$ is sufficiently large the circle will contain a point of some element e_j in its interior. That is, the nearest neighbor of z is not e_i . This shows that there are no half-lines contained in $V(e_i)$. But it is easy to show from Lemma 1 that an unbounded Voronoi region must contain a half-line. Therefore, $V(e_i)$ is bounded. \square

THEOREM 3. *The dual graph $D(S)$ of the Voronoi diagram $V(S)$ for a set of $N \geq 3$ elements is planar and has at most $3N - 5$ edges and $2N - 3$ faces.*

Proof. We first show that $D(S)$ is planar by exhibiting a planar embedding for it, and then obtain the bounds on the number of edges and faces by using Euler's Polyhedra Formula [6]. Let $D_i(S)$ be the set³ of elements $\{e_{i1}, e_{i2}, \dots, e_{im}\}$ whose associated regions and $V(e_i)$ share an edge. For any edge of $V(e_i)$ we can find a path connecting the two elements whose regions share the edge as follows. Let z_j be a point on the edge $\bar{B}(e_i, e_{ij})$, $j = 1, 2, \dots, m$. By Lemma 1, the closed segments $\bar{z}_j, \bar{I}(z_j, e_i)$ and $\bar{z}_j, \bar{I}(z_j, e_{ij})$ lie entirely in $V(e_i)$ and $V(e_{ij})$, respectively. The path connecting e_i and e_{ij} consists of $\bar{I}(z_j, e_i)$, z_j and $\bar{z}_j, \bar{I}(z_j, e_{ij})$. Note that if e_{ij} is an endpoint of the closed segment for e_i , the two closed segments $\bar{z}_j, \bar{I}(z_j, e_i)$ and $\bar{z}_j, \bar{I}(z_j, e_{ij})$ coincide. By Theorem 1, all the closed segment $\bar{z}_j, \bar{I}(z_j, e_i)$, for $j = 1, 2, \dots, m$, do not intersect except possibly at endpoints (on e_i). We conclude that the dual graph is planar.

Now, for the bounds on the number of edges and faces observe that in the dual graph each interior face is bounded by at least three edges and the exterior face is bounded by at least two edges when $N \geq 2$. This is because two nodes n_i and n_j of $D(S)$ which correspond to elements e_i and e_j , respectively, have multiple edges only if the bisector $B(e_i, e_j)$ is broken into two or more pieces. The node n_k whose associated element e_k caused $B(e_i, e_j)$ to break into two pieces will appear on any interior face containing both n_i and n_j . Therefore our interior face must have more than two edges bounding it. If we sum over the number of edges bounding each face we will get at least $3*(f-1)+2$ edges, where f is the number of faces of $D(S)$. Since each edge appears on exactly two faces, we have $2e \geq 3f - 1$. Solving for e and plugging into Euler's Polyhedra Formula, $N - e + f = 2$, we have $N - f/2 \geq 3/2$. Therefore, $f \leq 2N - 3$ and $e \leq 3N - 5$. \square

COROLLARY 1. *Given a set S of N elements, the number of Voronoi edges and the number of Voronoi points are both $O(N)$.*

LEMMA 3. *Given a set $S = \{s_1, s_2, \dots, s_n\}$ of closed segments, let $D_i(S)$ denote the subset of closed segments of S whose Voronoi regions are adjacent to $V(s_i)$. Then there exists a closed segment $s_j \in D_i(S)$ such that $d(s_i, s_j) = \text{MIN}_{a \in S - \{s_i\}} d(s_i, a)$, i.e., $D_i(S)$ contains a nearest neighbor of s_i .*

Proof. Immediate from Lemma 2. \square

3. Construction of the Voronoi diagram for a set of line segments.

3.1. Introduction. Now let us turn to the problem of computing the Voronoi diagram for a set of line segments. Our algorithm is a generalization of the divide-and-conquer scheme first presented by Shamos and Hoey [21] for a set $S = \{p_1, p_2, \dots, p_N\}$ of N points. We therefore start by describing a version of this algorithm which includes a modification made by Lee [12]. Sort the N points lexicographically on their (x, y) coordinates. Let $L = \{p_1, p_2, \dots, p_{\lfloor N/2 \rfloor}\}$ and let $R = \{p_{\lfloor N/2 \rfloor + 1}, \dots, p_N\}$, where $\lfloor x \rfloor$ is the greatest integer in x . Then compute the Voronoi diagrams $V(L)$ and $V(R)$ recursively. If we can merge $V(L)$ and $V(R)$ in $O(N)$ time, we can compute $V(S)$ in $O(N \log N)$ time.

³The set in general is a multiset; i.e., the elements are not necessarily distinct.

The key to the merge step is the merge curve $B(L, R)$, which is the oriented bisector of the two sets L and R of points. (Fig. 6.) Every point to the left of $B(L, R)$ is closer to some point in L than to any point in R . Similarly, every point to the right of $B(L, R)$ is closer to some point in R than to any point in L . Therefore if we chop off the part of any region in $V(L)$ extending to the right of $B(L, R)$ and the part of any region in $V(R)$ extending to the left of $B(L, R)$, we end up with the diagram $V(S)$. We shall construct $B(L, R)$ in linear time as follows. Theorem 2 in the previous section suggests that we compute the convex hull $CH(S)$. The convex hulls $CH(L)$ and $CH(R)$ are disjoint and can be constructed recursively when constructing $V(L)$ and $V(R)$. $CH(S)$ can be computed by merging $CH(L)$ and $CH(R)$ in linear time using an algorithm invented by Preparata and Hong [15]. In merging $CH(L)$ and $CH(R)$ we create two new hull edges, each joining a point l in L and a point r in R . By Lemma 2 we know that $B(l, r)$ is a component bisector, called a *starting bisector*. We can imagine forming $B(L, R)$ by moving a point from infinity inward along the bisector $B(l, r)$. For example, in Fig. 6 we will move the imaginary point along $B(p_4, p_{10})$. The imaginary point follows $B(p_4, p_{10})$ until it meets an edge at $V(p_4)$. At this point it is closer to p_8 than to p_4 , so it leaves $V(p_4)$ and enters the region $V(p_8)$ by following $B(p_8, p_{10})$ until it leaves $V(p_{10})$. In this way it traces out $B(L, R)$, always following the bisector $B(p_u, p_v)$ of a point p_u in L and a point p_v in R . When the imaginary point leaves $V(p_u)$, p_u is changed to the point associated with the new region entered. p_v is updated similarly.

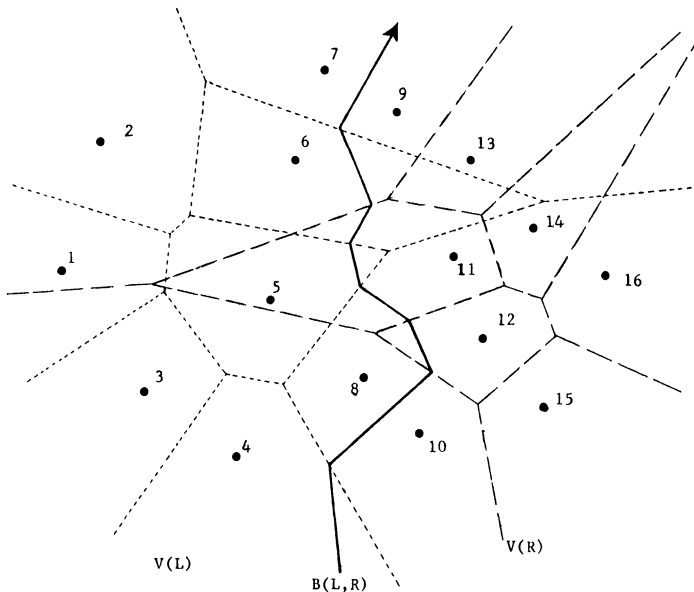


FIG. 6. $V(L)$, $V(R)$ and $B(L, R)$.

Because each $V(p_i)$ can have $O(N)$ edges on its boundary and we might examine the edges of a single region $O(N)$ times to see where a bisector first intersects the boundary of that region, the naive approach of checking each edge of $V(p_u)$ and $V(p_v)$ to find the first intersection with $B(p_u, p_v)$ could result in $O(N^2)$ work. We can avoid this by the following *scanning scheme* for edges: beginning with the point where $B(p_u, p_v)$ enters a region, scan the edges of a region in $V(L)$ in a counterclockwise (CCW) direction and the edges of a region in $V(R)$ in a clockwise (CW) direction. If an edge

does not intersect $B(p_u, p_v)$, then discard it. The following theorem shows that no backtracking is needed when using this scanning scheme.

THEOREM 4. *Let p_u be in L and p_v in R and let $B(p_u, p_v)$ be constructed during the merge process. No point of $V(p_u)$ in $V(L)$ which lies to the right of the oriented bisector $B(p_u, p_v)$ will be included in the region $V(p_u)$ in the final diagram. Similarly, no point of $V(p_v)$ in $V(R)$ which lies to the left of the oriented bisector $B(p_u, p_v)$ will be included in the region $V(p_v)$. Furthermore, scanning $V(p_u)$ in a CCW direction and $V(p_v)$ in a CW direction will find the first intersection between $B(p_u, p_v)$ and either $V(p_u)$ or $V(p_v)$.*

Proof. The fact that no point of $V(p_u)$ in $V(L)$ lying to the right of $B(p_u, p_v)$ can be included in the region $V(p_u)$ in the final diagram follows from the definition. To show that the CW-CCW scan indeed finds the first intersection point consider Fig. 7. Suppose t is the intersection point where $B(p_u, p_v)$ meets some edge of $V(p_v)$ before it meets any edge of $V(p_u)$. If t were not the first intersection, then moving a point z along the boundary of $V(p_v)$ in a CW direction we will find at least two intersection points with $B(p_u, p_v)$. Since $V(p_v)$ is generalized-star-shaped with nucleus p_v , by Theorem 1, this is impossible. Similar arguments hold if $B(p_u, p_v)$ meets an edge of $V(p_u)$ before any edge of $V(p_v)$. \square

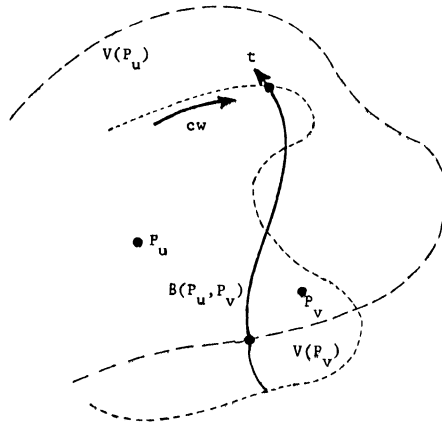


FIG. 7. Illustration of the proof of Theorem 4.

We now analyze the time taken for the merge process. After each test either an edge of $V(L)$ or an edge of $V(R)$ is discarded or a new Voronoi point is created in $V(S)$. Therefore, the time needed is proportional to the number of edges in $V(L)$ and $V(R)$ plus the number of Voronoi points on $B(L, R)$. By Corollary 1 this is $O(N)$. Therefore, the overall time needed to compute $V(S)$ is $O(N \log N)$.

What happens when we apply this algorithm to a set of closed segments? The first difficulty arises when ordering the closed segments. Lexicographical order is not defined for segments. Therefore, we arbitrarily choose the ordering of closed segments according to their left endpoints. Once we have chosen the ordering we can divide the given set S into two subsets L and R such that L and R contain the leftmost $\lfloor N/2 \rfloor$ and rightmost $\lfloor N/2 \rfloor$ closed segments respectively. Now we want to construct $V(L)$ and $V(R)$ recursively and merge them to form $V(S)$ as we did before. The first step in the merge process is to find the union of the convex hulls $CH(L)$ and $CH(R)$. But, because of the ordering that we have chosen, these two convex hulls are not necessarily disjoint ($CH(L)$ may contain $CH(R)$ in its interior). Therefore, using the idea of convex hull to

find the starting bisector may not help. Besides, the merge curve is not necessarily composed of a single piece. It may be broken into several pieces as shown in Fig. 8. That is, we have to determine the starting bisectors for all the pieces and then use the scanning scheme to construct all pieces of merge curves. The starting bisectors for the two unbounded pieces in Fig. 8 can be found by forming the union of $CH(L)$ and

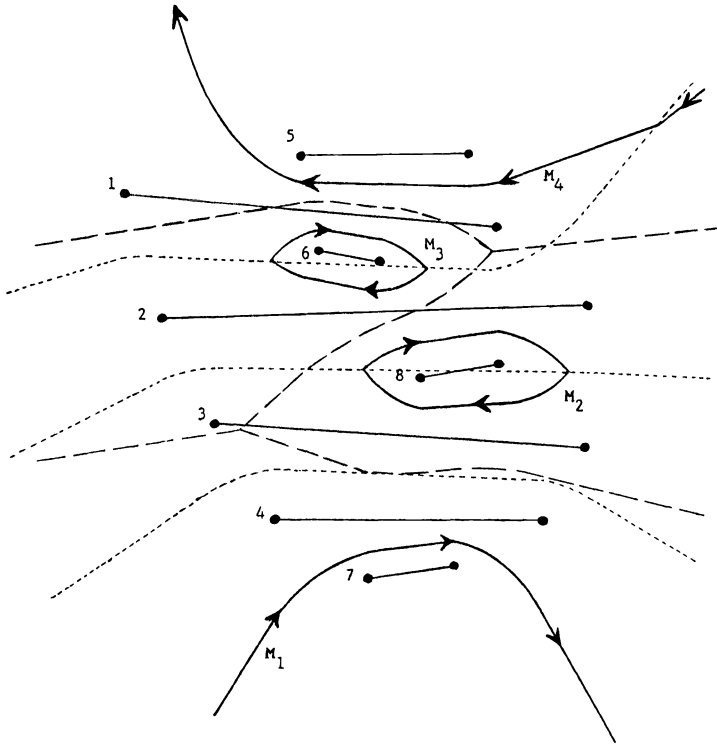


FIG. 8. Merge curve is broken into four pieces.

$CH(R)$ just as we did before. However, the convex hull gives no information about the starting bisectors for the other two pieces of $B(L, R)$ that are closed curves, called *islands*. Islands result from the closed segments in R getting “trapped” in $CH(L)$ and are a consequence of the initial ordering. The authors tried to find an ordering for the closed segments which would prevent these islands from forming. But finding this ordering seems to be as difficult as finding the Voronoi diagram itself. They also tried a number of approaches, but found no method for finding all islands in $O(N)$ time. However, the following method has been devised to determine a set of starting bisectors in $O(N \log N)$ time.

3.2. Determination of a set of starting bisectors. Suppose that we have obtained two Voronoi diagrams $V(L)$ and $V(R)$. To merge them we shall find a set, called *starter set* of L and R (denoted by $ST(L, R)$), of points (along with the information from which the corresponding starting bisectors can be derived) such that it is guaranteed that every piece of the merge curve passes through at least one of the points in $ST(L, R)$. In other words, the intersection of the set of points on the merge curve and $ST(L, R)$ is nonempty. The idea is based on the following observations. First each merge curve piece M_i encloses some subset R_i of closed segments of R . Either the subset R_i will be

completely surrounded by M_i (an island), or M_i and an edge “at infinity” will enclose R_i . A line from some segment in R_i to a segment not in R_i will cross M_i before it crosses any other piece of the merge curve. Second, M_i is the bisector $B(L_i, R_i)$ of R_i and some subset L_i of closed segments of L . It contains a bisector $B(e_j, e_k)$ for some elements $e_j \in L_i$ and $e_k \in R_i$ such that e_k is an endpoint of some closed segment of R_i . The first observation is due to the initial ordering of the set of closed segments. The second observation can be established as follows. Consider the convex hull $CH(R_i)$ of the set R_i of closed segments enclosed by M_i . One of the hull vertices, say q_k , must be an endpoint of some closed segment l_k . The half-line emanating from the endpoint q_k and perpendicular to l_k must intersect M_i at some point which is equidistant from q_k and some element e_j of L_i . Therefore, the intersection of $B(e_j, q_k)$ and M_i is nonempty and the observation follows. Based on these two observations we can determine the starter set $ST(L, R)$ as follows.

We construct the Voronoi diagram $V(Q)$ for the set Q of endpoints of R and then merge $V(Q)$ and $V(L)$ to form $V(L \cup Q)$. To merge $V(Q)$ and $V(L)$ we are confronted with the same problems as we were in merging $V(R)$ and $V(L)$. That is, the merge curve $M' = B(L, Q)$ is not necessarily composed of a single piece. We shall locate the set Q of points in the Voronoi regions of $V(L)$ to find for each point $q \in Q$ its nearest neighbor in L . If q is located in the region $V(e_i)$ then the nearest neighbor of q is e_i . Suppose $q_k \in V(e_j)$. Let q'_k be the image $I(q_k, e_j)$ of $q_k \in Q$ on some element $e_j \in L$. By Lemma 1 q_k, q'_k lies entirely in $V(e_j)$. We next find the midpoint m_k of q_k, q'_k . It is obvious that the circle K centered at m_k with radius $d(m_k, q'_k)$ will not contain any other point of L in its interior. If m_k is found to be in the Voronoi region $V(q_k)$ of $V(Q)$, then the circle K will not contain any other point of Q in its interior either. Thus, m_k will be in the starter set $ST(L, Q)$ for the merge curve M' and $B(e_j, q_k)$ is a starting bisector for a piece of M' . If m_k does not lie in $V(q_k)$, we discard it. Now we have to show that indeed each piece of M' passes through at least one point of $ST(L, Q)$. To see this consider a piece of merge curve $M'_i = B(L_i, Q_i)$, where $L_i \subseteq L$ and $Q_i \subseteq Q$. There exists a point $q_j \in Q$ and a point t of some element $e_i \in L_i$ such that $d(t, q_j) = d(L_i, Q_i)$. As a result, the point $q_j \in Q_i$ must

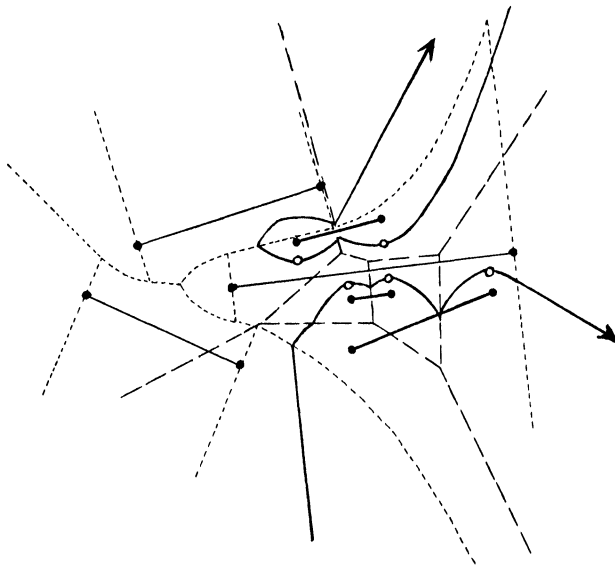


FIG. 9. Merge of two Voronoi diagrams $V(L)$ and $V(Q)$.

lie in $V(e_i)$ of $V(L)$ such that t is the image $I(q_i, e_i)$. Furthermore, t must lie in $V(q_i)$ of $V(Q)$. Consequently, the midpoint m of t, q_j must belong to both $V(e_i)$ and $V(q_j)$. Therefore, m is in $ST(L, Q)$. This proves that every piece of M' indeed passes through at least a point of $ST(L, Q)$. In other words, we shall locate the set Q of points in $V(L)$ and, after obtaining the set C of midpoints of q_k, q'_k , where $q_k \in Q$ and q'_k is the image $I(q_k, e_j)$ of q_k on $e_j \in L$, we locate the set C of midpoints in $V(Q)$. If the midpoint m_k of q_k, q'_k is found in $V(q_k)$, then we store it in $ST(L, Q)$. Otherwise, we ignore it. Fig. 9 shows the merge curve for the two Voronoi diagrams $V(L)$ and $V(Q)$, and "o" denotes the point in $ST(L, Q)$. The time needed for the determination of $ST(L, Q)$, is $O(N \log N)$. This result is due to Preparata [16]. In [16] Preparata has shown that a set of N points can be located in $O(N \log N)$ time in a planar subdivision with N vertices. After we have obtained $ST(L, Q)$, we can use the procedure to be described in the next section to construct M' in $O(N \log N)$ time.

Suppose we have constructed the Voronoi diagram $V(L \cup Q)$. The Voronoi region $V'(q_k)$ associated with $q_k \in Q$ is the intersection of the half-planes $h(q_k, e_l)$, for all $e_l \in L \cup Q - \{q_k\}$. Since the Voronoi region $V(q_k)$ associated with $q_k \in Q$ in the Voronoi diagram $V(R)$ is the intersection of the half-planes $h(q_k, e_r)$, for all $e_r \in R - \{q_k\}$, the intersection of $V'(q_k)$ and $V(q_k)$ gives rise to the Voronoi region $V''(q_k)$ in the final diagram $V(S)$. Fig. 10 shows the Voronoi regions $V''(q)$, the intersection of $V(q)$ and

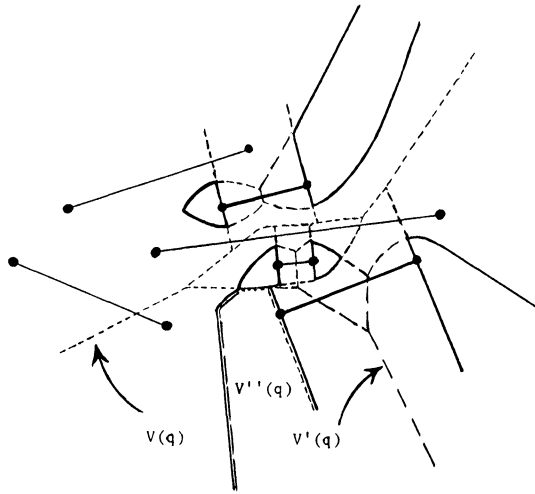


FIG. 10. Intersection of $V(q)$ and $V'(q)$.

$V'(q)$. Any point on the Voronoi edge $\bar{B}(q_k, e_l)$ of $V''(q_k)$, where $e_l \in L$, can be a point in the starter set $ST(L, R)$ and $B(q_k, e_l)$ is a possible starting bisector for a piece of the merge curve $M = B(L, R)$. The intersection of $V(q_k)$ and $V'(q_k)$, both of which are generalized-star-shaped with nucleus being a point can be found in time $O(s + t)$, where s and t are the numbers of edges of $V(q_k)$ and $V'(q_k)$, respectively, by a technique similar to one given in [20] in finding the intersection of two convex polygons. Therefore, the starter set $ST(L, R)$ can be obtained in $O(N \log N)$ time and will include at most one point for each q_k . Now we shall show that the merge curve $M = B(L, R)$ can be constructed in $O(N \log N)$ time.

3.3. Merging two Voronoi diagrams. After we have determined the starter set, we shall construct the merge curve M piece by piece. We shall take a point in $ST(L, R)$ and use the corresponding bisector as our starting bisector to construct a piece of merge curve in two passes⁴ [13]. If the piece of merge curve passes through a point of $ST(L, R)$, that point will *not* be used again. Because each point is associated with a particular q_k , it is easy to check in constant time if the current bisector piece is passing through a starter point. The merge process works in a way very similar to that given before except that we make the following modifications.

In order to start with a bisector $B(u, v)$, where $u \in L$ and $v \in R$, we need to scan the edges of $V(u)$ and $V(v)$ respectively to find the first intersection points of $B(u, v)$ and an edge of $V(u)$ and of $B(u, v)$ and an edge of $V(v)$; a simple-minded approach would result in $O(N^2)$ time to construct $M = B(L, R)$ since (i) the number of edges of $V(u)$ may be $O(N)$, (ii) there would be $O(N)$ pieces of M starting with bisectors of the form $B(u, v)$, $v \in R$, and (iii) for each piece it takes $O(N)$ time to find the first intersection point. However, since the points of $ST(L, R)$, whose corresponding bisectors involve an element $u \in L$, are on the boundary of $V(u)$ in the final Voronoi diagram, they can be “ordered” in the sense of Theorem 1. That is, if u is an endpoint, they can be ordered by polar angle with point u as the origin. If u is a segment, they can be ordered by their images on the segment. To avoid repeated examinations of the edges of $V(u)$ we shall keep track of the ordering of the points in $ST(L, R)$. For example, suppose u is a segment (a, b) and $V((a, b))$ is considered. Let m_i and m_j be two midpoints in $ST(L, R)$ such that the corresponding bisectors are $B((a, b), e_i)$ and $B((a, b), e_j)$, respectively, and m_j appears “after” m_i in the CCW direction along the boundary of $V((a, b))$. Let M_i and M_j denote the two pieces of the merge curve M using m_i and m_j as starting points, respectively. Assume that M_i has been constructed. When constructing M_j starting with $B((a, b), e_j)$ we shall scan the edges of $V((a, b))$ in the CCW direction (stipulated by the scanning scheme) starting with the edge where the piece of merge curve M_i exits from $V((a, b))$. In this manner we can eliminate backtracking. A similar situation holds if u is an endpoint.

After we have determined the starting edge to be scanned in constructing each piece of merge curve M_i , the CW-CCW scan procedure can be applied. However, instead of discarding edges when no intersection is found we replace them with a single “dummy” edge. The merge process is more complex, but not conceptually difficult. For details see [3], which includes a full description and an implementation of this merge procedure. The following theorem ensures that the modified CW-CCW scan works correctly and that no backtracking is necessary.

THEOREM 4¹. *Let e_i and e_j be elements in L and R , respectively, and let $B(e_i, e_j)$ be constructed during the merge process. No point of $V(e_i)$ in $V(L)$ which lies to the right of the oriented bisector $B(e_i, e_j)$ will be included in the region $V(e_i)$ in the final diagram. Similarly, no point of $V(e_j)$ in $V(R)$ which lies to the left of the oriented bisector $B(e_i, e_j)$ will be included in the region $V(e_j)$. Furthermore, the scanning of $V(e_i)$ in a CCW direction and $V(e_j)$ in a CW direction will find the first intersection between $B(e_i, e_j)$ and either $V(e_i)$ or $V(e_j)$.*

Proof. Similar to the proof of Theorem 4. \square

The time for constructing all pieces of merge curves is $O(N \log N)$ which is due to the ordering on the points in $ST(L, R)$ to determine the starting edge for the initial scan for each piece of merge curve. Fig. 11 shows the merge curve M of the two Voronoi

⁴ If the piece of merge curve is an island, the procedure provided in [13] will bring us back to the starting point. One pass is thus sufficient.

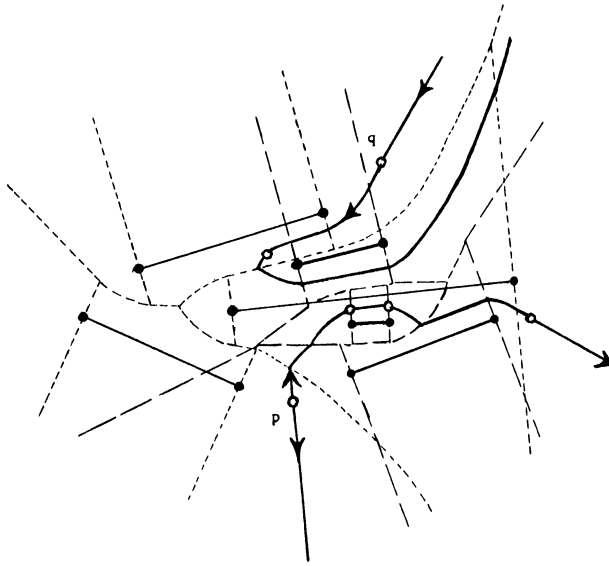


FIG. 11. Merge of two Voronoi diagrams.

diagrams $V(L)$ and $V(R)$ shown in short- and long-dashed lines respectively, starting with the points p and q ; “o” denotes the points in the starter set $ST(L, R)$.

Let us summarize the entire procedure for constructing the Voronoi diagram for a set S of N closed segments as follows.

Step 1. Order the set of N closed segments according to their left endpoints.

Step 2. Recursively construct the Voronoi diagrams $V(L)$ and $V(R)$, where L and R are the sets containing the left $\lfloor N/2 \rfloor$ and right $\lceil N/2 \rceil$ closed segments respectively.

Step 3. Construct the Voronoi diagram $V(Q)$ for the set Q of endpoints of R .

Step 4. Merge $V(Q)$ and $V(L)$ to form $V(L \cup Q)$.

Step 5. For each point $q \in Q$, find the intersection of $V(q)$ in the diagram $V(L \cup Q)$ and $V'(q)$ in the diagram $V(R)$. Let $V''(q)$ be the intersection of the two Voronoi polygons $V(q)$ and $V'(q)$.

Step 6. If the boundary of $V''(q)$ has an edge $B(q, e)$, where e is an element of L , then take any point on $B(q, e)$ and enter it into $ST(L, R)$ along with the information that $B(q, e)$ is the corresponding bisector.

Step 7. Order these points in $ST(L, R)$ in the sense of Theorem 1.

Step 8. Merge $V(L)$ and $V(R)$ by taking points of $ST(L, R)$ in a specific order and applying the CW-CCW scanning scheme. Mark those points in the starter set which are passed through by the current piece of merge curve. Repeat this step until all the points in $ST(L, R)$ are marked.

The time required for the initial sorting (Step 1) of endpoints is $O(N \log N)$. Constructing $V(Q)$ (Step 3) and $V(L \cup Q)$ (Step 4) takes $O(N \log N)$ time. Steps 6 and 8 take $O(N)$ time. Step 7 takes $O(N \log N)$ time. Since Steps 3 through 8 are executed $O(\log N)$ times due to recursion, the entire procedure takes $O(N(\log N)^2)$ time. Thus, we have the following results.

THEOREM 5. *Given a set S of N disjoint closed segments in the Euclidean plane, the Voronoi diagram $V(S)$ can be computed in $O(N(\log N)^2)$ time.*

COROLLARY 2. *Given a set of N disjoint closed segments in the Euclidean plane, the minimum weight spanning tree can be found in $O(N(\log N)^2)$ time.*

COROLLARY 3. *Given a set of N closed segments in the Euclidean plane, the all-nearest-neighbor problem can be solved in $O(N(\log N)^2)$ time.*

3.4. Extension to other figures and metrics. The technique used in constructing the Voronoi diagrams for a set of line segments in the Euclidean plane can be easily extended to the case when the given set of “objects” is a set of disjoint polygons or disjoint circles.

The procedure for constructing the Voronoi diagram for a set of line segments applies for the case of polygons, since the set of polygons can be treated as a set of non-disjoint line segments. However, for circles, the situation is different, since there is no “endpoint” that we can locate. But the following scheme works just as well. The idea is based on the fact that each piece of merge curve contains a point which is of the shortest distance from the corresponding sets of objects. The distance between two circles in the Euclidean metric is equal to the difference of the distance between the centers and the sum of their radii. Therefore as usual, we divide the set of circles into two subsets L and R of circles according to their leftmost points and recursively construct the Voronoi diagrams $V(L)$ and $V(R)$, respectively, for the two subsets L and R of circles. To merge them we need to determine the starting bisectors first. We shall locate the centers of the circles of the set R in the Voronoi diagram $V(L)$, and consequently find a set of images on the boundaries of the circles of L which are the nearest neighbors of the centers of the circles of R . Then we do another pass of point-location to locate the set of images in the Voronoi diagram $V(R)$. If the nearest neighbor of the image of c on some circle o_j in L , where c is a center of some circle o_k in R , is o_j , then $B(o_j, o_k)$ is a starting bisector. Let the image $I(c, o_j)$ of c on circle o_j in L be denoted by u , and the intersection of the straight line determined by u and c and the circle o_k in R be v . The midpoint of the line segment \overline{uv} is a point to be included in the starter set. The fact that the bisector $B(o_j, o_k)$ is a Voronoi edge can be shown in exactly the manner described earlier. Therefore, the Voronoi diagram for a set of N disjoint circles in the Euclidean plane can be found in $O(N(\log N)^2)$ time. We have the following results.

THEOREM 6. *Given a set of N disjoint objects (circles or polygons) in the Euclidean plane, the Voronoi diagram for the set of objects can be found in $O(N(\log N)^2)$ time.*

COROLLARY 4. *The minimum spanning tree for a set of N disjoint polygons or a set of disjoint circles can be constructed in $O(N(\log N)^2)$ time.*

The above results were stated in terms of the Euclidean metric. However, they can be proved using only the following assumptions about the metric in the plane.

(1) (Strict triangle inequality.) For all points x , y , and z , $d(x, y) + d(y, z) = d(x, z)$ if and only if x , y , and z are collinear in the standard Euclidean geometry in the plane.

(2) (Sides of triangles opposite nonacute angles are longest.) For any three points x , y , and z , if angle (x, y, z) has measure at least $\pi/2$ in the standard Euclidean geometry, then $d(x, z) > d(x, y)$ and $d(x, z) > d(y, z)$.

The L_p -metric, $1 < p < \infty$, satisfies these properties. But L_1 - and L_∞ -metrics do not. Finding Voronoi diagrams for line segments or other objects in these metrics is still an open problem.

4. Conclusion. We presented an $O(N(\log N)^2)$ algorithm for computing the Voronoi diagram for a set of N circles or line segments in the plane under a general metric which satisfies (i) strict triangle inequality and (ii) the property that sides of triangles opposite nonacute angles are longest. The best lower bound for this problem is $\Omega(N \log N)$, so there is still room for improvement in either the algorithm or the bound.

5. Acknowledgment. Most of this work was done when Lee was with University of Illinois at Urbana-Champaign, supported in part by the National Science Foundation under grant MCS76-17321 and in part by the Joint Services Electronics Program under Contract DAAB-07-72-0259; and when Drysdale was at Stanford University, supported by the National Science Foundation under grant MCS77-05313.

REFERENCES

- [1] M. A. BREUER, ed., *Design Automation of Digital Systems*, vol. 1, Prentice-Hall, Englewood Cliffs NJ, 1974.
- [2] J. C. DAVIS AND M. J. McCULLAGH, eds., *Display and Analysis of Spatial Data*, John Wiley, New York, 1975.
- [3] R. L. DRYSDALE, III, *Generalized Voronoi diagrams and geometric searching*, Ph.D. Thesis, Department of Computer Science, Tech. Rep. STAN-CS-79-705, Stanford University, Stanford CA, 1979.
- [4] R. L. DRYSDALE, III AND D. T. LEE, *Generalized Voronoi diagram in the plane*, Proceedings of the 16th Annual Allerton Conference on Communications, Control and Computing, Oct. 1978, pp. 833-842.
- [5] R. L. FRANCIS AND J. A. WHITE, *Facility Layout and Location*, Prentice-Hall, Englewood Cliffs NJ, 1974.
- [6] F. HARARY, *Graph Theory*, Addison-Wesley, Reading MA, 1972.
- [7] J. A. HARTIGAN, *Clustering Algorithms*, Wiley Series in Probability and Statistics, John Wiley, New York, 1975.
- [8] R. E. HORTON, *Rational study of rainfall data makes possible better estimates of water yield . . .*, Engineering News-Record (1917), pp. 211-213.
- [9] F. K. HWANG, *An $O(n \log n)$ algorithm for rectilinear minimal spanning trees*, J. Assoc. Comput. Mach., 26 (1979), pp. 177-182.
- [10] D. E. KNUTH, *The Art of Computer Programming*, vol. 3, Addison-Wesley, Reading MA, 1973.
- [11] R. J. KOPEC, *An alternative method for the construction of Thiessen polygons*, Professional Geographer, 15 (1963), pp. 24-26.
- [12] D. T. LEE, *On finding k -nearest neighbors in the plane*, M.S. Thesis, Coordinated Science Lab., Rep. R-728, University of Illinois, Urbana IL, 1976.
- [13] D. T. LEE AND C. K. WONG, *Voronoi diagrams in $L_1(L_\infty)$ metrics with 2-dimensional storage applications*, this Journal, 9 (1980), pp. 200-211.
- [14] D. T. LEE, *Proximity and reachability in the plane*, Ph.D. Thesis, Department of Computer Science, Coordinated Sci. Lab. Rep. R-831, University of Illinois, Urbana IL, 1978.
- [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] F. P. PREPARATA, *A new approach to planar point location*, submitted for publication.
- [17] D. RHYNSBURGER, *Analytic delineation of Thiessen polygons*, Geographic Analysis, 5 (1973), pp. 133-144.
- [18] C. A. ROGER, *Packing and Covering*, Cambridge University Press, Cambridge, 1964.
- [19] T. L. SAATY, *Optimization Problems in Integers and Related Extremal Problems*, McGraw-Hill, New York, 1970.
- [20] M. I. SHAMOS, *Computational Geometry*, Department of Computer Science, Yale University, 1977, to be published by Springer-Verlag.
- [21] M. I. SHAMOS AND D. HOEY, *Closest-point problems*, Proc. 16th IEEE Symposium on Foundations of Computer Science, Oct. 1975, pp. 151-162.
- [22] A. H. THIESSEN, *Precipitation averages for large areas*, Monthly Weather Review, 39 (1911), pp. 1082-1084.

FAST SORTING OF WEYL SEQUENCES USING COMPARISONS*

MARTIN H. ELLIS† AND J. MICHAEL STEELE‡

Abstract. An algorithm is given which makes only $O(\log n)$ comparisons, and which will determine the ordering of the uniformly distributed (pseudo random) Weyl sequences given by $\{(k\alpha) \bmod 1 : 1 \leq k \leq n\}$, where α is an unspecified irrational number. This result is shown to be best possible in the sense that no algorithm can perform the same task with fewer than $\Omega(\log n)$ comparisons.

Key words. sorting, Weyl sequences, information theory lower bound, alpha-sort

1. Introduction¹. Any algorithm which sorts sets of n real numbers only on the basis of comparisons will always require, in the worst case, at least $\log_2(n!) = \theta(n \log_2 n)$ comparisons. Similarly, if n reals are chosen at random from any continuous distribution, the expected number of comparisons required for sorting them is also $\theta(n \log_2 n)$. These familiar facts may make it surprising that there are sequences which share many properties with random sequences, but whose order can always be determined with fewer than $4 \log_2 n$ comparisons.

The sequences considered here are the so called Weyl sequences given by $X_k = k\alpha \bmod 1$, where α is an irrational number. These sequences share with the *independent* uniformly distributed random variables the basic property that the number of elements from X_1, X_2, \dots, X_n in (a, b) is asymptotic to $n(b-a)$, for $0 \leq a < b \leq 1$. (For a purely probabilistic proof of this property, see Feller [2, p. 268].) Since the Weyl sequences are "uniformly distributed" in the sense described, Franklin [3] has further examined the pseudo-random virtues of $\{X_k\}$ by a variety of statistical tests. This inherent randomness, together with their rich and well studied mathematical structure, makes it intriguing to see just how efficiently the Weyl sequences can be ordered.

The principal objective of this paper is to provide an algorithm which determines the order of X_1, X_2, \dots, X_n on the basis of fewer than $4 \log_2 n$ comparisons. We further show that any algorithm for sorting $\{(k\alpha) \bmod 1 : 1 \leq k \leq n\}$ by comparisons must make at least $\Omega(\log_2 n)$ comparisons, so the algorithm given here is the best possible.

One key motivation for studying the sorting of Weyl sequences is the general question: "How does one use the fact that a sequence is of a certain structure to provide a sorting algorithm which is information theoretically optimal?" This problem was explicitly posed in M. L. Friedman [4] and is implicit in Berlecamp's problem on sum set sorting (see, e.g., Harper, et al. [51]).

A second motivation for studying the sorting of Weyl sequences by comparisons is provided by recent work of Papadimitriou on efficient search for rationals. Papadimitriou [6] gives an elegant algorithm which establishes that $O(\log M)$ queries of the form "is $x \leq p/q$ ", where $p, q \leq M$, are sufficient to determine any rational $x = a/b$ with $a, b \leq M$. The present algorithm is quite distinct from Papadimitriou's in method (relative comparisons vs. absolute comparisons) and in purpose (sort vs. search). Still, there is a close connection since (as the following sections implicitly show) the ordering

* Received by the editors December 18, 1978 and in final revised form May 16, 1980. This research was supported in part by the National Science Foundation under grants MCS 77-03659 and MCS 77-16974.

† Professor Martin H. Ellis of the Department of Mathematics, Northeastern University, Boston, Massachusetts, died on February 15, 1980, after a brief illness.

‡ Department of Statistics, Stanford University, Stanford, California 94305.

¹ θ , Ω , O , o denote "order of exactly", "order of at least", "order of at most", and "order of less than", respectively.

of $\{(k\alpha) : 1 \leq k \leq n\}$ is closely connected to the location of the irrational α in the Farey dissection of the unit interval.

In the next section, we give an algorithm called Alpha-Sort, which is a very simple procedure which sorts any collection of the form $\{(k\alpha) \bmod 1 : 1 \leq k \leq n\}$ with fewer than $o(n)$ comparisons. The third section then uses the structures uncovered by Alpha-Sort to provide the required information theoretic lower bound $\Omega(\log_2 n)$. The fourth section applies a binary search speedup of Alpha-Sort which gives an explicit algorithm which performs as well as the theoretical lower bound can permit. The final section makes a brief speculation about the use of sorting as an appropriate measure of complexity of a pseudo random sequence.

2. Alpha-Sort: An $o(n)$ algorithm for sorting $(k\alpha) \bmod 1$, $1 \leq k \leq n$. For brevity, we will subsequently write $\langle k\alpha \rangle$ for the representative of $(k\alpha) \bmod 1$ in $[0, 1)$. The key idea for efficient sorting of $\{\langle k\alpha \rangle : 1 \leq k \leq n\}$ is that the order structure can be completely determined from the largest and smallest elements of the set. We define L^* and R^* to be the integers in $\{1, 2, \dots, n\}$ satisfying $\langle L^*\alpha \rangle = \min_{1 \leq k \leq n} \langle k\alpha \rangle$ and $\langle R^*\alpha \rangle = \max_{1 \leq k \leq n} \langle k\alpha \rangle$. The Alpha-Sort algorithm shows how one can compute L^* and R^* , and how these integers can be used to determine the ordering of $\langle k\alpha \rangle$, $1 \leq k \leq n$.

Alpha-Sort algorithm. Given $X_k = (k\alpha) \bmod 1$, $1 \leq k \leq n$, this algorithm returns i_1, i_2, \dots, i_n such that $X_{i_1} < X_{i_2} < \dots < X_{i_n}$.

A1. [Initialize] Set $L \leftarrow 1$, $R \leftarrow 1$, $M \leftarrow 1$.

A2. [Compute L^* and R^*] While $L + R \leq n$, set $R \leftarrow L + R$ if $X_{L+R-1} < X_{L+R}$; otherwise, set $L \leftarrow L + R$.

A3. Print $ML \bmod (L + R)$ if $ML \bmod (L + R) \leq n$.

A4. Set $M \leftarrow M + 1$. If $M < L + R$ go to A3; otherwise end program.

The fact that Alpha-Sort correctly performs the task of sorting $\{\langle k\alpha \rangle : 1 \leq k \leq n\}$ with $O(n)$ comparisons, will follow from the next two lemmas. These elementary results will form the theoretical core for the rest of the analysis.

LEMMA 1. Suppose $\min\{\langle \alpha \rangle, \langle 2\alpha \rangle, \dots, \langle j\alpha \rangle\} = \langle L\alpha \rangle$ and $\max\{\langle \alpha \rangle, \langle 2\alpha \rangle, \dots, \langle j\alpha \rangle\} = \langle R\alpha \rangle$, $L, R \in \{1, 2, \dots, j\}$. Then

(i) $\min\{\langle \alpha \rangle, \langle 2\alpha \rangle, \dots, \langle (L+R-1)\alpha \rangle\} = \langle L\alpha \rangle$,

(ii) $\max\{\langle \alpha \rangle, \langle 2\alpha \rangle, \dots, \langle (L+R-1)\alpha \rangle\} = \langle R\alpha \rangle$,

(iii) either $\langle (R+L)\alpha \rangle < \langle L\alpha \rangle$ or $\langle (R+L)\alpha \rangle > \langle R\alpha \rangle$.

Proof. (i). If $1 \leq H < R$ and $\langle (L+H)\alpha \rangle < \langle L\alpha \rangle$, then $\langle (L+H)\alpha \rangle \neq \langle L\alpha \rangle + \langle H\alpha \rangle$; hence,

$$(1) \quad \langle L\alpha \rangle + \langle H\alpha \rangle > 1.$$

The definitions of L and R and (1) imply

$$(2) \quad \begin{aligned} \langle R\alpha \rangle &= \langle R\alpha \rangle - \langle H\alpha \rangle + \langle H\alpha \rangle \\ &= \langle (R-H)\alpha \rangle + \langle H\alpha \rangle \\ &\geq \langle L\alpha \rangle + \langle H\alpha \rangle \\ &> 1. \end{aligned}$$

This contradiction establishes (i).

(ii). If $1 \leq H < R$ and $\langle (L+H)\alpha \rangle > \langle R\alpha \rangle$, then the definitions of L and R imply

$$\begin{aligned} \langle (L+H-R)\alpha \rangle &= \langle (L+H)\alpha \rangle - \langle R\alpha \rangle \\ &= \langle L\alpha \rangle + \langle H\alpha \rangle - \langle R\alpha \rangle \\ &< \langle L\alpha \rangle. \end{aligned}$$

This is a contradiction to the fact that $\langle(L+H-R)\alpha\rangle > \langle L\alpha\rangle$.

(iii). Either

$$\begin{aligned}\langle(R+L)\alpha\rangle &= \langle R\alpha\rangle + \langle L\alpha\rangle \\ &> \langle R\alpha\rangle,\end{aligned}$$

or

$$\begin{aligned}\langle(R+L)\alpha\rangle &= \langle R\alpha\rangle + \langle L\alpha\rangle - 1 \\ &< \langle L\alpha\rangle,\end{aligned}$$

so (iii) is also established. \square

LEMMA 2. For $\langle L\alpha\rangle = \min\{\langle k\alpha\rangle : 1 \leq k \leq n\}$ and $\langle R\alpha\rangle = \max\{\langle k\alpha\rangle : 1 \leq k \leq n\}$ we have $\langle L\alpha\rangle = \langle m_1\alpha\rangle < \langle m_2\alpha\rangle < \cdots < \langle m_S\alpha\rangle = \langle R\alpha\rangle$ where $m_k = kL \bmod (L+R)$ and $S = L+R-1$. Furthermore, L and R are relatively prime, and $\{m_i : 1 \leq i \leq S\}$ is a permutation of the numbers $1, 2, \dots, S$.

Proof. First we will show by induction on n that L and R are relatively prime. If $n = 1$ then $L = R = 1$, so L and R are relatively prime. Suppose the assertion is true for $n = l$. If the maximum and minimum remain unchanged when n is raised to $l+1$, they remain relatively prime. If not, then Lemma 1 implies that $l+1 = L+R$ and either L or R (but not both) must be replaced with $L+R$. Since

$$\gcd\{L, L+R\} = \gcd\{L+R, R\} = \gcd\{L, R\} = 1,$$

the assertion is established.

Since L and $R+L$ are relatively prime,

$$\{\langle m_i\alpha\rangle : 1 \leq i \leq S\} = \{\langle i\alpha\rangle : 1 \leq i \leq S\}.$$

Suppose for some $1 \leq i \leq S$

$$(1) \quad \langle m_i\alpha\rangle > \langle m_{i+1}\alpha\rangle.$$

If

$$(2) \quad \langle m_{i+1}\alpha\rangle = \langle(m_i+L)\alpha\rangle,$$

then (1) implies

$$\begin{aligned}\langle m_{i+1}\alpha\rangle &= \langle m_i\alpha\rangle + \langle L\alpha\rangle - 1 \\ &< \langle L\alpha\rangle;\end{aligned}$$

this is impossible since by Lemma 1(i), $\langle L\alpha\rangle$ must be minimal. Since (2) fails to hold,

$$(3) \quad \langle m_{i+1}\alpha\rangle = \langle(m_i-R)\alpha\rangle.$$

Since $R = m_S$ and $i < S$, Lemma 1(ii) implies $\langle m_i\alpha\rangle < \langle R\alpha\rangle$. But then,

$$\begin{aligned}\langle m_{i+1}\alpha\rangle &= \langle m_i\alpha\rangle - \langle R\alpha\rangle + 1 \\ &> \langle m_i\alpha\rangle,\end{aligned}$$

contradicting (1). Thus, $\langle m_i\alpha\rangle < \langle m_{i+1}\alpha\rangle$ for $1 \leq i < S$, and the lemma is proved. \square

The first lemma proves that step (A2) of Alpha-Sort correctly determines L^* and R^* . The second lemma shows how these two quantities completely determine the ordering of $\{\langle k\alpha\rangle : 1 \leq k \leq n\}$. We actually showed that L^* and R^* determine the ordering of the larger set $\{\langle k\alpha\rangle : 1 \leq k \leq L^* + R^* - 1\}$; step (A3) of Alpha-Sort deletes the irrelevant members from the ordering of the larger set.

Since each comparison performed by Alpha-Sort increases either L or R , and since these quantities never exceed n , the total number of comparisons performed is at most $O(n)$. One can actually show that for any fixed α , as n increases only $o(n)$ comparisons are required. In fact, one can show that for almost every α (in the sense of Lebesgue measure), for every $\varepsilon > 0$ Alpha-Sort will find L^* and R^* with $o((\log n)^{1+\varepsilon})$ comparisons. Still, by taking Liouville irrationals like $\sum_{k=1}^{\infty} 10^{-a_k}$ where $a_1 < a_2 < \dots$ is a rapidly increasing sequence, one can also show that $o(n)$ is the most precise statement which one can make about the number of comparisons required by Alpha-Sort. We do not need to elaborate on these points since the next section will show that $o(n)$ is far from theoretically optimal, and the final section will sharpen Alpha-Sort to attain that optimal rate.

3. Information theoretic bounds. In the second lemma of the preceding section, we saw that the two quantities L^* and R^* completely determine the ordering of $\{\langle k\alpha \rangle : 1 \leq k \leq n\}$. We will now show that this suggests that it may be possible to sort $\{\langle k\alpha \rangle : 1 \leq k \leq n\}$ with only $O(\log_2 n)$ comparisons, but no fewer.

Classically, the fact that a binary tree with m leaves must have height at least $\log_2(m)$, and the fact that there are $n!$ orderings of n real numbers, collectively imply that at least $O(n \log_2 n)$ comparisons are required to determine their order. This information theoretic perspective makes it interesting to determine E_n , the total number of orderings of $\{\langle k\alpha \rangle : 1 \leq k \leq n\}$ as α varies through all real values.

Explicitly, we let $|A|$ denote the cardinality of a finite set A and let σ denote any permutation of $\{1, 2, \dots, n\}$. For

$$E_n = |\{\sigma : \text{for some } \alpha \in (0, 1) \langle \sigma(1)\alpha \rangle < \langle \sigma(2)\alpha \rangle < \dots < \langle \sigma(n)\alpha \rangle\}|,$$

we have the following fact.

PROPOSITION. $n \leq E_n \leq n^2$.

Proof. For any α , we have (by Lemma 2 above) that there are integers $1 \leq L^* \leq n$, $1 \leq R^* \leq n$ which completely determine the ordering of $\{\langle k\alpha \rangle : 1 \leq k \leq n\}$. Since there are only n^2 such pairs L^* and R^* , the upper bound is established.

To see that $E_n \geq n$, we consider the n irrationals defined by $\alpha_k = 1/k - \varepsilon$ for $k = 1, 2, \dots, n$ and some very small positive irrational ε . For α_k one can see that $\langle \alpha_k \rangle < \langle 2\alpha_k \rangle < \dots < \langle k\alpha_k \rangle$ but $\langle (k+1)\alpha \rangle < \langle \alpha_k \rangle$. The $\langle \alpha_k \rangle$ thus each yield a different ordering, so $E_n \geq n$ as claimed.

Since the conclusions to be drawn from this proposition depend only on $O(\log_2 E)$, we have obtained only the simplest bounds. One can actually show that if ϕ is the Euler phi-function, we have $E_n = \sum_{k \leq n} \phi(k) = 3/\pi^2 n^2 + O(n \log n)$ (for facts on $\phi(k)$ see [1]). The proposition immediately establishes the following result.

COROLLARY. At least $\Omega(\log_2 n)$ comparisons are required in order to sort $\{\langle k\alpha \rangle : 1 \leq k \leq n\}$.

The upper bound in the preceding proposition also suggests that it might be possible to sort $\{\langle k\alpha \rangle : 1 \leq k \leq n\}$ with only $O(\log_2 n)$ comparisons. The main objective of the next section will be to show that this is in fact the case.

4. Fast Alpha-Sort: An $O(\log n)$ algorithm. In Lemma 2, we proved that the ordering of $S = \{\langle \alpha \rangle, \langle 2\alpha \rangle, \dots, \langle n\alpha \rangle\}$ is completely determined by L^* and R^* where $\langle L^*\alpha \rangle = \min_{1 \leq k \leq n} \langle k\alpha \rangle$ and $\langle R^*\alpha \rangle = \max_{1 \leq k \leq n} \langle k\alpha \rangle$. Now we will show how Alpha-Sort can be improved to compute these values with only $O(\log_2 n)$ comparisons.

The improvement over Alpha-Sort is made by replacing the linear process for computing L and R by a geometric process. The details are somewhat complicated due to the presence of several cases, but the conceptual essence of the matter is brought out

in the following Lemma 3, which shows essentially that if certain conditions are valid at times k and $2k$ they are valid at all intermediate times.

DEFINITION. A value $\langle j\alpha \rangle$ is called a *left extreme* (respectively *right extreme*) if $\langle j\alpha \rangle$ is the minimum (respectively maximum) of $\{\langle i\alpha \rangle : 1 \leq i \leq j\}$.

LEMMA 3. *Let k be a positive integer, and suppose $\langle R\alpha \rangle$ is a right extreme and each $\langle (L+iR)\alpha \rangle$, $0 \leq i \leq k$, is a left extreme. If $\langle (L+2kR)\alpha \rangle < \langle (L+kR)\alpha \rangle$, then each $\langle (L+iR)\alpha \rangle$, $k+1 \leq i \leq 2k$, is a left extreme.*

Proof. Let k be a positive integer for which each $\langle (L+jR)\alpha \rangle$, $0 \leq i \leq k$, is a left extreme. Let j be the smallest positive integer for which $\langle (L+jR)\alpha \rangle$ is not a left extreme, and suppose $j \leq 2k$. Lemma 1 implies that $\max\{\langle \alpha \rangle, \langle 2\alpha \rangle, \dots, \langle (L+(j-1)R)\alpha \rangle\} = \langle R\alpha \rangle$, and since $\langle (L+(j-1)R)\alpha \rangle$ is a left extreme, Lemma 1(iii) implies $\langle (L+jR)\alpha \rangle$ is an extreme; by choice of j it is not a left extreme, so it is a right extreme, hence,

$$(1) \quad \langle (L+jR)\alpha \rangle > \langle R\alpha \rangle.$$

Since each $\langle (L+iR)\alpha \rangle$, $0 \leq i \leq k$, is a left extreme,

$$(2) \quad \langle (L+iR)\alpha \rangle - \langle (L+(i+1)R)\alpha \rangle = 1 - \langle R\alpha \rangle, \text{ for all } 0 \leq i < k.$$

Since $\langle (L+jR)\alpha \rangle > \langle L\alpha \rangle$, (2) implies

$$(3) \quad \langle (L+(j+i)R)\alpha \rangle - \langle (L+(j+i+1)R)\alpha \rangle = 1 - \langle R\alpha \rangle, \text{ for all } 0 \leq i < k.$$

Expressing $\langle (L+2kR)\alpha \rangle$ and $\langle (L+kR)\alpha \rangle$ as telescoping sums and applying (1), (2), (3) and the fact that $j \geq k+1$, we have

$$\begin{aligned} (4) \quad \langle (L+2kR)\alpha \rangle &= \langle (L+jR)\alpha \rangle \\ &\quad + \sum_{i=0}^{2k-j-1} (\langle (L+(j+i+1)R)\alpha \rangle - \langle (L+(j+i)R)\alpha \rangle) \\ &> \langle R\alpha \rangle - (2k-j)(1 - \langle R\alpha \rangle) \\ &= 1 - (2k+1-j)(1 - \langle R\alpha \rangle) \\ &\geq 1 - k(1 - \langle R\alpha \rangle) \\ &> \langle L\alpha \rangle - k(1 - \langle R\alpha \rangle) \\ &= \langle L\alpha \rangle + \sum_{i=0}^{k-1} (\langle (L+(i+1)R)\alpha \rangle - \langle (L+iR)\alpha \rangle) \\ &= \langle (L+kR)\alpha \rangle. \end{aligned}$$

Inequality (4) shows that if there is a $j \in \{i : k+1 \leq i \leq 2k\}$ for which $\langle (L+jR)\alpha \rangle$ is not a left extreme, then $\langle (L+2kR)\alpha \rangle > \langle (L+kR)\alpha \rangle$. The lemma follows by contradiction. \square

A similar argument establishes the following result.

LEMMA 4. *Let l be a positive integer, and suppose that each $\langle (R+iL)\alpha \rangle$, $0 \leq i \leq l$, is a right extreme. If $\langle (R+2lL)\alpha \rangle > \langle (R+lL)\alpha \rangle$, then each $\langle (L+iR)\alpha \rangle$, $l+1 \leq i \leq 2l$, is a right extreme.*

Besides serving to prove the validity of the following Fast Alpha-Sort algorithms, the preceding lemmas should also serve to motivate the algorithm. As a tool for use within Fast Alpha-Sort, we will require a binary search procedure which we call $\text{SEARCH}(L, R, v, z)$. The parameters L, R, v are integers provided in the course of the Fast Alpha-Sort Algorithm, and z is either 0 or 1, depending on whether the algorithm

is looking for a new candidate for a left extreme or a right extreme. (The duality between the left and right procedures can be immediately seen, but for clarity we will not strain to unify the two.)

SEARCH(L, R, v, z).

- S1. If $v = 1$ set SEARCH(L, R, v, z) $\leftarrow 1$ and stop.
- S2. If $v = 2$ set SEARCH(L, R, v, z) $\leftarrow 3$ and stop if $z = 0$ and $L + 3R \leq n$ and $\langle(L + 2R)\alpha\rangle > \langle(L + 3R)\alpha\rangle$, or if $z = 1$ and $3L + R \leq n$ and $\langle(2L + R)\alpha\rangle < \langle(3L + R)\alpha\rangle$; otherwise if $v = 2$ set SEARCH(L, R, v, z) $\leftarrow 2$ and stop.
- S3. If $v \geq 3$ set $S \leftarrow 3 \cdot 2^{v-2}$ and set $T \leftarrow 2^{v-3}$.
- S4. If $z = 0$ and $L + SR \leq n$ and $\langle(L + \frac{1}{2}SR)\alpha\rangle > \langle(L + SR)\alpha\rangle$, or if $z = 1$ and $SL + R \leq n$ and $\langle(\frac{1}{2}SL + R)\alpha\rangle < \langle(SL + R)\alpha\rangle$, set $S \leftarrow S + T$; otherwise set $S \leftarrow S - T$.
- S5. Set $T \leftarrow \frac{1}{2}T$. If $T \geq 1$ go back to S4.
- S6. If $z = 0$ and $L + SR \leq n$ and $\langle(L + (S - 1)R)\alpha\rangle > \langle(L + SR)\alpha\rangle$, or if $z = 1$ and $SL + R \leq n$ and $\langle((S - 1)L + R)\alpha\rangle < \langle(SL + R)\alpha\rangle$, set SEARCH(L, R, v, z) $\leftarrow S$ and stop; otherwise set SEARCH(L, R, v, z) $\leftarrow S - 1$ and stop.

The SEARCH subroutine is used in the Fast Alpha-Sort algorithm, and the role it plays there is described in the proof of Lemmas 6 and 7.

Fast Alpha-Sort. Given $X_k = \langle k\alpha \rangle$, $1 \leq k \leq n$, this algorithm returns L^* and R^* such that $\langle \alpha L^* \rangle = \min_{1 \leq k \leq n} \langle k\alpha \rangle$, $\langle \alpha R^* \rangle = \max_{1 \leq k \leq n} \langle k\alpha \rangle$.

FA1. Set $L \leftarrow 0$, $R \leftarrow 1$.

FA2. Starting with $k = 1$, increment k until either $L + 2^k R > n$ or $\langle(L + 2^{k-1}R)\alpha\rangle < \langle(L + 2^k R)\alpha\rangle$.

FA3. Set $L \leftarrow L + R \cdot \text{SEARCH}(L, R, k, 0)$. If $L + R > n$ go to FA6.

FA4. Starting with $l = 1$, increment l until either $R + 2^l L > n$ or $\langle(R + 2^{l-1}L)\alpha\rangle > \langle(R + 2^l L)\alpha\rangle$.

FA5. Set $R \leftarrow R + L \cdot \text{SEARCH}(L, R, l, 1)$. If $L + R \leq n$, go to FA2.

FA6. Let $L^* = L$ and $R^* = R$, then stop.

The main result of this section is the following:

THEOREM. *The Fast Alpha-Sort Algorithm returns L^* and R^* after at most $O(\log n)$ comparisons between pairs in $\{\langle i\alpha \rangle; 1 \leq i \leq n\}$.*

Before proving the theorem we will establish three lemmas.

LEMMA 5. *Computing SEARCH(L, R, p, z) requires at most $p - 1$ comparisons between pairs in $\{\langle i\alpha \rangle; 1 \leq i \leq n\}$.*

Proof. We simply dissect the possibilities. If $p = 1$, no comparisons are made. If $p = 2$, the only comparison that may be required is between $\langle(L + 2R)\alpha\rangle$ and $\langle(L + 3R)\alpha\rangle$ if $z = 0$, between $\langle(2L + R)\alpha\rangle$ and $\langle(3L + R)\alpha\rangle$ if $z = 1$. If $p \geq 3$, then T is set to 2^{p-3} and single comparisons or no comparisons alternate with dividing T by 2, until $T < 1$. Thus, at most $p - 2$ comparisons are made before T becomes less than one. A single additional comparison may be made in (S6), for a total of at most $p - 1$ comparisons. \square

LEMMA 6. *Suppose Fast Alpha-Sort has just entered step (FA2), $L = p$, $R = q$ and $\langle q\alpha \rangle$ is a right extreme. If each $\langle(p + iq)\alpha\rangle$, $1 \leq i \leq j$, is a left extreme but $\langle(p + (j + 1)q)\alpha\rangle$ is not a left extreme, Fast Alpha-Sort will set L equal to $p + q * \min(j, \lfloor (n - p)/q \rfloor)$ after making at most $1 + 2 \min(\lfloor \log_2 j \rfloor, \lfloor \log_2 ((n - p)/q) \rfloor)$ additional comparisons between pairs in $\{\langle i\alpha \rangle; 1 \leq i \leq n\}$.*

Proof. Let $b = \lfloor \log_2 j \rfloor$ and let $c = \lfloor \log_2 ((n - p)/q) \rfloor$.

If $p + 2^{b+1}q \leq n$, Fast Alpha-Sort will sequentially compare $\langle(p + 2^{i-1}q)\alpha\rangle$ with

$\langle(p + 2^i q)\alpha\rangle$, $1 \leq i \leq b + 1$. It will then compute $\text{SEARCH}(p, q, b + 1, 0)$, which by Lemma 5 requires at most b additional comparisons, after which it will set $L \leftarrow p + jq$. The total number of comparisons made is thus at most $1 + 2b$.

If $p + j1 \leq n < p + 2^{b+1}q$, Fast Alpha-Sort will follow the same procedure as above, except the comparison of $\langle(p + 2^b q)\alpha\rangle$ with $\langle(p + 2^{b+1} q)\alpha\rangle$ will be omitted; hence at most $2b$ comparisons will be made.

If $n < p + jq$, Fast Alpha Sort will sequentially compare $\langle(p + 2^{i-1} q)\alpha\rangle$ with $\langle(p + 2^i q)\alpha\rangle$, $1 \leq i \leq c$. Upon learning that $p + 2^{c+1}q > n$, it will then compute $\text{SEARCH}(p, q, c + 1, 0)$, which by Lemma 5 requires at most c additional comparisons, after which it will set $L \leftarrow p + \lfloor(n - p)/q\rfloor q$. The total number of comparisons made is thus at most $2c$.

In any case, the total number of comparisons is at most $1 + 2 \min(b, c)$. \square

The following lemmas can be proved analogously to Lemma 6.

LEMMA 7. *Suppose Fast Alpha-Sort has just entered step (FA4), $L = p$, $R = q$, and $\langle p\alpha\rangle$ is a left extreme. If each $\langle(ip + q)\alpha\rangle$, $1 \leq i \leq j$ is a right extreme but $\langle(j + 1)p + q\rangle\alpha$ is not a right extreme, Fast Alpha-Sort will set L equal to $q + p * \min(j, \lfloor(n - q)/p\rfloor)$ after making at most $1 + 2 \min(\lfloor\log_2 j\rfloor, \lfloor\log_2((n - 1)/p)\rfloor)$ additional comparisons between pairs in $\{\langle i\alpha\rangle : 1 \leq i \leq n\}$.*

Proof of Theorem. Lemmas 6 and 7 imply that Fast Alpha-Sort correctly computes L^* and R^* . It remains to show that the number of comparisons between pairs in $\{\langle i\alpha\rangle : 1 \leq i \leq n\}$ made by Fast Alpha-Sort in computing L^* and R^* is $O(\log n)$.

Assume Fast Alpha-Sort has just computed L^* and R^* . Let

$$1 = q_0 < q_2 < \dots < q_{2V} = R^*$$

denote the values taken by R during the course of Fast Alpha-Sort, let

$$0 = p_{-1} < p_1 < p_3 < \dots < p_{2W-1} = L^*$$

denote the values taken by L during the course of Fast Alpha-Sort, and let $m = \max(2V, 2W - 1)$. (Note that $m = 2V = 2W$ if step (FA6) was entered from (FA5) and $m = 2W - 1 = 2V + 1$ if step (FA6) was entered from (FA3).) For $1 \leq i \leq m$, let $j_i = \lfloor p_i/q_{i-1}\rfloor$ if i is odd and let $j_i = \lfloor q_i/p_{i-1}\rfloor$ if i is even. Note

$$(1) \quad \prod_{i=1}^m j_i \leq \max(L^*, R^*) \leq n.$$

Lemmas 6 and 7 imply that the number of comparisons between pairs in $\{\langle i\alpha\rangle : 1 \leq i \leq n\}$ made by Fast Alpha-Sort in computing L^* and R^* is bounded above by

$$\sum_{i=1}^m (1 + 2 \lfloor \log_2 j_i \rfloor),$$

which by (1) is bounded by

$$(2) \quad m + 2 \log_2 \left(\prod_{i=1}^m j_i \right) \leq m + 2 \log_2 n.$$

The largest value m can have would occur if the p_i and q_i grew as slowly as possible (i.e., $j_i = 1$ for $1 \leq i \leq m$), in which case each p_i or q_i would be the $(i + 1)$ st number in the Fibonacci sequence. In this case,

$$(3) \quad m \leq 1 + \frac{\log_2 n}{\log_2 \phi},$$

where $\phi = (\sqrt{5} + 1)/2$. Inequalities (2) and (3) show that the number of comparisons

between pairs in $\{\langle i\alpha \rangle : 1 \leq i \leq n\}$ made by Fast Alpha-Sort in computing L^* and R^* is bounded above by

$$1 + (2 + (\log_2 \phi)^{-1}) \log_2 n,$$

which establishes the theorem. \square

5. A brief speculation. The introduction isolated two motivations for studying the sorting of Weyl sequences, and a third motivation was deferred until now. This comes from the problem of measuring the complexity of a class of sequences and using this measurement to aid one's choice of pseudo-random number generators. The Weyl sequences are not genuine candidates for pseudo-random numbers, and this is reinforced by the speed with which they are sorted. One would especially like to determine the number of comparisons needed to sort sequences generated by the widely used classes of PRN generators. This analysis has many practical and conceptual complications, but the Weyl sequences can be considered a preliminary case in this wider program.

REFERENCES

- [1] T. M. APOSTOL, *Introduction to Analytic Number Theory*, Springer-Verlag, New York, 1976.
- [2] W. FELLER, *An Introduction to Probability Theory and Its Applications*, Vol. 2, Second Ed., John Wiley, New York, 1971.
- [3] J. N. FRANKLIN, *Deterministic simulation of random processes*, Math. Comp., 17 (1963), pp. 28–59.
- [4] M. L. FRIEDMAN, *How good is the information theory bound in sorting*, Theoret. Comput. Sci., 1 (1976), pp. 355–361.
- [5] L. H. HARPER, T. H. PAYNE, J. E. SAVAGE AND E. STRAUSS, *Sorting $X + Y$* , Comm. ACM., 18 (1975), pp. 347–349.
- [6] C. H. PAPADIMITRIOU, *Efficient Search for Rationals*, Technical Report 01–78, Center for Research in Computing Technology, Harvard University, 1978.

RELATIVE TO A RANDOM ORACLE A , $P^A \neq NP^A \neq \text{co-}NP^A$ WITH PROBABILITY 1*

CHARLES H. BENNETT† AND JOHN GILL‡

Abstract. Let A be a language chosen randomly by tossing a fair coin for each string x to determine whether x belongs to A . With probability 1, each of the relativized classes LOGSPACE^A , P^A , NP^A , PP^A , and $PSPACE^A$ is properly contained in the next. Also, $NP^A \neq \text{co-}NP^A$ with probability 1. By contrast, with probability 1 the class P^A coincides with the class BPP^A of languages recognized by probabilistic oracle machines with error probability uniformly bounded below $\frac{1}{2}$. NP^A is shown, with probability 1, to contain a P^A -immune set, i.e., a set having no infinite subset in P^A . The relationship of P^A -immunity to p -sparseness and NP^A -completeness is briefly discussed: P^A -immune sets in NP^A can be sparse or moderately dense, but not co-sparse. Relativization with respect to a random length-preserving permutation π , instead of a random oracle A , yields analogous results and in addition the proper containment, with probability 1, of P^π in $NP^\pi \cap \text{co-}NP^\pi$, which we have been unable to decide for a simple random oracle. Most of these results are shown by straightforward counting arguments, applied to oracle-dependent languages designed not to be recognizable without a large number of oracle calls. It is conjectured that all P^A -invariant statements that are true with probability 1 of subrecursive language classes uniformly relativized to a random oracle are also true in the unrelativized case.

Key words. random oracle, relativized computation, probabilistic computation, computational complexity, nondeterministic computation, polynomial immunity, polynomial isomorphism, polynomial reducibility

1. Introduction. A paper by Baker, Gill and Solovay [BGS], whose notation and definitions we adopt, has indicated the subtlety of the $P = ?NP$ question by exhibiting computable sets A and B such that $P^A = NP^A$ but $P^B \neq NP^B$. Here, P^X denotes the class of languages accepted by polynomial time bounded Turing machines able to query the set X , and NP^X denotes the corresponding class for nondeterministic machines.

This paper deals not with particular oracle sets but rather with statements that hold with probability 1 when the oracle is chosen randomly. The probability measure μ on the class of oracles is defined by putting each string into a random oracle with probability $\frac{1}{2}$, independent of all other strings. (Of course, such an oracle is noncomputable with probability 1). Random oracles provide easy examples of sets such as B of [BGS], and also indicate a new sense in which $P^X \neq NP^X$ for "most" oracles X . This is a counterpart for the nondenumerable class of all oracles of Mehlhorn's result [Me] that the subset of computable oracles X that satisfy $P^X = NP^X$ is effectively meager.

Any property of oracles that is insensitive to finite changes in the oracle has probability 0 or 1, by the zero-one law for tail events [Fe2]. We determine relationships that hold with probability 1 for language classes relativized to a random oracle A . Section 2 establishes the basic results $P^A \neq NP^A \neq \text{co-}NP^A$ with probability 1, and the related results $\text{LOGSPACE}^A \neq P^A$ and $PSPACE^A \neq \text{EXPTIME}^A$ with probability 1. Section 3 relativizes the probabilistic language classes PP (languages recognizable in polynomial time by weak Monte Carlo tests, whose error probability may approach that of random guessing), and BPP (languages recognizable in polynomial time by strong Monte Carlo tests, whose error probability can be made as small as desired by iterating the test a fixed number of times). It is shown that with probability 1, the relativized class PP^A is properly contained in $PSPACE^A$ and properly contains $NP^A \cup \text{co-}NP^A$. By

* Received by the editors November 6, 1979, and in final form May 20, 1980. This research was supported in part by the National Science Foundation, under grant MCS77-07555.

† IBM Watson Research Center, Yorktown Heights NY 10598.

‡ Electrical Engineering Department, Stanford University, Stanford CA 94305.

contrast, P^A and BPP^A are shown to be equal with probability 1. Section 4 shows that with probability 1, NP^A contains a P^A -immune set, that is, a set having no infinite subset in P^A . Section 5 discusses the open question of whether, relative to a random oracle, P^A equals $NP^A \cap \text{co-}NP^A$, arguing that it will be hard to decide one way or the other. On the other hand, by relativizing with respect to a random permutation π instead of a random oracle, P^π can be shown to be properly within $NP^\pi \cap \text{co-}NP^\pi$. Indeed, $NP^\pi \cap \text{co-}NP^\pi$ contains a P^π -immune set with probability 1.

Most oracles used in recursive function theory and complexity theory contain built-in structure intended to help or frustrate a specific class of computations. A random oracle, on the other hand, is intuitively unbiased and unstructured; thus, it is plausible that theorems (for example, $P \neq NP$) that hold with probability one for computations relativized to a random oracle should also be true in the absence of an oracle. Section 6 formalizes this conjecture.

As a preview of the results to be demonstrated later, we now give heuristic arguments showing why, relative to a typical random oracle, deterministic and nondeterministic polynomial time are different ($P^A \neq NP^A$, Theorem 1), but deterministic and probabilistic time are the same ($P^A = BPP^A$, Theorem 5). Given a fixed but typical random oracle, consider the following question: do the first 2^n bits of the oracle's characteristic sequence include any run of n consecutive zeros? Such a run will be present for about half of all values of n , and if present, it could easily be detected nondeterministically by guessing the address of its beginning. On the other hand, it is fairly obvious, if not entirely straightforward to prove, that no deterministic algorithm could expect to find out whether a run exists in less than exponential time. Thus, for typical random oracles A , the language $\{0^n: \text{the first } 2^n \text{ bits of } A \text{ contain a run of } n \text{ consecutive zeros}\}$ is in $NP^A - P^A$. Similarly, the language $\{0^n: \text{the first } 2^n \text{ bits of } A \text{ contain an even number of zeros}\}$ is in $PSPACE^A - NP^A$ with probability 1.

Next consider a language, such as the set of composite numbers, that is probabilistically recognizable in the sense of BPP . Such a language could be recognized deterministically in the presence of a random oracle by: 1) iterating the original Monte Carlo test a linearly increasing number of times as a function of input size, so that the expected cumulative number of errors, summing over all inputs, remains finite; 2) simulating this more accurate Monte Carlo algorithm deterministically by using bits from the random oracle instead of coin tosses; 3) patching the errors by a finite table. A slight refinement of this argument shows that even relativized languages of the class BPP^A can be recognized in deterministic polynomial time with the help of a random oracle.

Throughout this paper, the natural number x will be identified with the x th binary string in lexicographic order ($0, 1, 2, 3, \dots \leftrightarrow \Lambda, 0, 1, 00, \dots$). The binary length of x , equal to the integral part of $\log_2(x+1)$, will be denoted $|x|$. Similarly, a set or language A will be identified with its characteristic sequence, the infinite binary sequence whose x th bit, $A(x)$, is 1 iff $x \in A$. Sets of sets (e.g., language classes or events in oracle space) will be denoted by upper case Greek letters, with Ω denoting the (nondenumerable) set of all languages. The probability measure on Ω is equivalent, via the identification of languages with infinite binary sequences, to Lebesgue measure on the unit interval.

Most of the separation results in this paper are proved by exhibiting an oracle-dependent test language L^A which belongs to the one of two relativized language classes (e.g., NP^A) for all oracles A but belongs to another narrower class (e.g., P^A) only for a set of oracles of measure zero. Results of this sort can be proven more easily by appealing to the following lemma, which depends on certain easily satisfied conditions

on the test language L^A and the (denumerable) relativized class $\mathbf{M}^A = \{M_1^A, M_2^A, M_3^A, \dots\}$ to which it is desired to prove that L^A does not belong for most A . For concreteness, M_j^A may be thought of as the language accepted by the j th machine of a given type (e.g., polynomial time bounded) when it is connected to oracle A . First the conditions will be described; then the lemma will be stated and proved.

Condition 1. The test language L^A , and each of the machine languages M_j^A , must depend on A via a total recursive operator from Ω to Ω . Each of these languages, in other words, must be recognizable by a Turing machine that halts for all oracles and inputs.

Condition 2. The family of machine languages should be finitely patchable with respect to the oracle: for each machine M_j and each finite bit string s there should exist another machine M_k such that $M_k^A = M_j^{s^*A}$ for all oracles A . Here s^*A denotes the characteristic sequence obtained by substituting the finite string s for the first $|s|$ bits of A . Machine M_k may be thought of as incorporating the bit string s in its finite control, where it intercepts and answers all sufficiently small queries.

Condition 3. The family of machine languages should be finitely patchable with respect to initial portions of any uniformly A -recursive language: For any number m , any machine M_j , and any A -recursive function φ_i^A that is total and 0-1 valued for all A , there should exist another machine M_k such that for all oracles A and inputs x ,

$$M_k^A(x) = \begin{cases} \varphi_i^A(x) & \text{if } x < m, \\ M_j^A(x) & \text{otherwise.} \end{cases}$$

In particular, when φ_i^A defines a test language L^A , machine M_k gives the “correct” answer $L^A(x)$ for inputs less than m and gives the same answer as machine M_j would for all other inputs.

Condition 4. The test language L^A (but not necessarily the machine languages M_j^A) must depend on the oracle in such a way that each bit of the oracle affects only finitely many bits of the language. (Condition 1, by König’s lemma, implies that both L^A and M_j^A already satisfy the converse of condition 4, namely, that each bit of the language depends on only finitely many bits of the oracle.) Together, conditions 1 and 4 require that the membership of x in L^A depend only on those addresses in A lying in a finite window bounded by two monotone functions of x that tend to ∞ in the limit of large x . Oracle-dependent languages of this sort have been termed “oracle properties” by Angluin [An] and Kozen and Machtey [KM].

Conditions 1 and 4 hold by definition for all the test languages used in this paper, and conditions 1–3 can readily be seen to hold for the relevant families of oracle machines, viz. logspace bounded deterministic [LL, Si], polynomial time bounded nondeterministic [BGS], and polynomial time bounded probabilistic threshold machines ([Gi], see also § 3).

LEMMA 1. *Let L^A be a test language and $\mathbf{M}^A = \{M_1^A, M_2^A, \dots\}$ a family of machine languages satisfying conditions 1–4 above. If there exists a positive constant ϵ such that each machine language differs from the test language for a class of oracles of measure $> \epsilon$, then the class of oracles for which $L^A \in \mathbf{M}^A$ has measure zero.*

Proof. The idea of the proof is to show that as a machine is fed larger and larger inputs, it keeps making fresh errors, due to bad luck at oracle addresses too large to have caused any errors earlier.

It suffices to show, for each machine M_j , that the class $\mathbf{C}_m = \{A : \forall x < m L^A(x) = M_j^A(x)\}$, of oracles for which it makes no error on the first m inputs, approaches measure zero in the limit $m \rightarrow \infty$. To prove this it suffices to show

that for each m there exists a larger n such that $\mu(C_n) \leq (1 - \epsilon)\mu(C_m)$. Because of condition 1, membership of an oracle in the class C_m depends on only a finite portion of the oracle characteristic sequence; hence C_m may be expressed as a finite disjoint union of elementary cylinders Z_s , where Z_s is the class of oracles whose characteristic sequences begin with the finite sequence s .

In view of this, the lemma would follow if one could show that ϵ is a lower bound not only for the overall error probability, $\lim_{n \rightarrow \infty} 1 - \mu(C_n)$, but also for the conditional error probability within any cylinder, $\lim_{n \rightarrow \infty} 1 - \mu(Z_s \cap C_n) / \mu(Z_s)$, even though the cylinder Z_s might consist entirely of oracles that cause no errors on small inputs.

To prove that this is indeed the case, note that the operation of M_j in any cylinder Z_s may be simulated by another, finitely patched, machine M_k , which accepts the following oracle-dependent language: **if** $[L^{s^*A}(x) \neq L^A(x)]$ **then** $L^{s^*A}(x)$ **else** $M_j^{s^*A}(x)$. Here condition 4 guarantees that $L^A(x)$ and $L^{s^*A}(x)$ differ for only finitely many x , and conditions 2 and 3 guarantee the existence of the patched machine. It is evident from the definition that the original machine's conditional error probability on cylinder Z_s is at least as great as the patched machine's unconditional error probability which in turn is at least ϵ , by the premise of the lemma. \square

Remark. Kozen and Machtey [KM] derive a result analogous to Lemma 1 but for meagerness rather than measure. Under conditions 1–4, they show that set $\{A : L^A \in M^A\}$ is either equal to all of oracle space or else is a meager subset of oracle space. Therefore, whenever Lemma 1 is used to prove a separation with probability 1 between two relativized complexity classes, the same separation holds for all but a meager subset of oracles. On the other hand, the possibility remains that two complexity classes may be equal with probability 1 even though they differ for all but a meager subset of oracles. This possibility is discussed further in connection with Theorem 5.

2. P^A, NP^A , and $LOGSPACE^A$ for random oracles A . The following definition provides a function $\xi_A(x)$ that uses the oracle A to map binary strings randomly into strings of the same length.

DEFINITION. $\xi_A(x) = A(x1)A(x10)A(x100) \cdots A(x10^{|x|-1})$, where juxtaposition indicates concatenation. In other words, $\xi_A(x)$ is a $|x|$ -bit string whose k th bit is 1 or 0 according to whether $x \times 10^{k-1}$ belongs to A .

Although it is easily computed by a machine with oracle A , the function ξ_A is ideally pseudorandom in that knowing its value for one argument tells nothing about its value for other arguments. (The same is true of the characteristic function $A(x)$, but in several of the proofs below it is convenient to have a function whose values are about the same size as its arguments.) The pseudorandomness of ξ_A is used to define languages depending on A that cannot be accepted without exponentially many queries of the oracle. The number of inverse images under ξ_A approaches a Poisson distribution for large n : for typical A the fraction of n -bit strings with exactly k inverse images under ξ_A approaches $e^{-k}/k!$ In particular, about $1/e$ of n -bit strings have no inverse image and $1/e$ have exactly one inverse image.

THEOREM 1. *If A is a random oracle, the $P^A \subseteq NP^A \neq co-NP^A$ with probability 1.*

Proof. Since P^A is closed under complementation, Theorem 1 would follow if, for all but a class of oracles A of measure zero, one could exhibit a language in NP^A whose complement is not in NP^A . Let the test language $RANGE^A$ be defined as $\{x : \exists y \xi_A(y) = x\}$, i.e., the range of ξ_A , and let $CORANGE^A$ be the complement of $RANGE^A$. Clearly, $RANGE^A$ belongs to NP^A . However, $CORANGE^A$ is not in NP^A because, intuitively, no nondeterministic oracle machine can verify for typical x and A that x is not in $RANGE^A$ without evaluating $\xi_A(y)$ for every y of length $|x|$.

In order to show that with probability 1, no polynomial time bounded nondeterministic oracle machine NP_j with random oracle A accepts exactly CORANGE^A , it suffices by Lemma 1 to show that every such machine has an input on which it errs with probability at least $\frac{1}{3}$ when A is chosen randomly.

Let an arbitrary machine NP_j be chosen and consider an input of the form $x = 0^n$, where n is sufficiently large that none of the machine's nondeterministic computation paths has time to examine more than one per cent of the 2^n n -bit strings that are potential inverse images of 0^n under the ξ function. Recalling the definition of the ξ function, an n -bit string y will be said to be *examined* when the oracle is queried about any string of the form $y10^k$, for some $k < n$.

Let $\mathbf{C}_0 = \{A: \neg \exists y \xi_A(y) = 0^n\}$ be the class of oracles for which the input 0^n is in CORANGE^A and therefore should be accepted. This class has measure between 0.36 and 0.37 for all $n \geq 5$, approaching $1/e = 0.3678 \dots$ for large n . Let α_0 be the conditional acceptance probability on \mathbf{C}_0 , i.e., the fraction of oracles in \mathbf{C}_0 for which input 0^n actually is accepted.

Consider now another class of oracles, disjoint from \mathbf{C}_0 and consisting of oracles A for which 0^n has exactly one inverse image but is not its own inverse image. This class, $\mathbf{C}_1 = \{A: \xi_A(0^n) \neq 0^n \text{ and } (\exists^{\text{uniq}} y) \xi_A(y) = 0^n\}$, has measure exactly equal to that of \mathbf{C}_0 and consists entirely of oracles for which the input 0^n does not belong to CORANGE^A and therefore should be rejected. Let α_1 be the conditional acceptance probability on \mathbf{C}_1 .

The overall error probability,

$$\varepsilon = \mu\{A: NP_j^A(0^n) \neq \text{CORANGE}^A(0^n)\},$$

is at least

$$(1 - \alpha_0)\mu(\mathbf{C}_0) + \alpha_1\mu(\mathbf{C}_1) \approx (1 + \alpha_1 - \alpha_0)/e,$$

since every rejection in \mathbf{C}_0 , and every acceptance in \mathbf{C}_1 , is an error. In order to show that $\varepsilon > \frac{1}{3}$, we exhibit a probabilistic transformation of oracles, $A \rightarrow A'$, that maps \mathbf{C}_0 onto \mathbf{C}_1 in a measure-preserving manner but changes each oracle so little that most accepting computation paths under A continue to accept under A' . Therefore, $\alpha_1 \geq \alpha_0$ and $\varepsilon \geq 1/e$.

The transformation $A \rightarrow A'$ is best described in words. To obtain A' from A , choose randomly (by coin tossing) an n -bit string z not equal to 0^n ; then delete from A all strings of the form $z \times 10^i$ for $i < n$. Recalling the definition of the ξ function, this has the effect of making $\xi_{A'}(z) = 0^n$ while preserving the equality $\xi_A(y) = \xi_{A'}(y)$ for all other arguments y . The transformation is therefore measure preserving between \mathbf{C}_0 and \mathbf{C}_1 , in the sense that the expectation of any event in \mathbf{C}_1 is equal to the expectation that a randomly chosen point in \mathbf{C}_0 will map into it under the transformation. (The probabilistic transformation may be thought of more formally as a deterministic measure-preserving mapping $(A, z) \rightarrow (A', \xi_A(z))$ from $\mathbf{C}_0 \times Y$ onto $\mathbf{C}_1 \times Y$, where Y is the probability space of n -bit strings not equal to 0^n . Hence, for any event $\mathbf{E} \subseteq \mathbf{C}_1$, $\mu(\mathbf{E}) = \mu\{(A, z) \in \mathbf{C}_0 \times Y: A' \in \mathbf{E}\}$.)

To show that $\alpha_1 \geq \alpha_0$, choose a random oracle in \mathbf{C}_0 and a random n -bit string $z \neq 0^n$ and generate the transformed oracle A' , a member of class \mathbf{C}_1 . With probability α_0 , there is at least one accepting path of $NP_j(0^n)$ under oracle A . Select the first accepting path. With conditional probability at least 0.99, the set of strings examined on this path does not include z , the one string with respect to which oracles A and A' differ, and so the path continues to accept under A' . Therefore the acceptance probability in \mathbf{C}_1 is at least 0.99 times that in \mathbf{C}_0 and $\varepsilon \geq 0.36(1 - \alpha_0 + 0.99\alpha_0) > \frac{1}{3}$.

Lemma 1 then allows us to conclude that, with probability 1, $CORANGE^A$ is not in NP^A . Since $RANGE^A$ is in NP^A , we have, with probability 1, $P^A \neq NP^A \neq co-NP^A$. \square

$LOGSPACE^A$ can be defined in various ways, depending on how the query tape is handled. We follow the conventions of Ladner and Lynch [LL]: The query tape is not charged against the space bound, but to keep it from being used as a work tape, the query tape is one-way and write-only and is erased automatically following each query. (Simon [Si] treats the query tape as one of the work tapes, a two-way read/write tape that is charged against the space bound. The Ladner-Lynch definition is less restrictive and perhaps more natural, since for a random oracle $A \in LOGSPACE^A$ holds with probability 1 for [LL] but not for [Si]. Theorem 2 holds for both definitions of $LOGSPACE^A$.)

THEOREM 2. *If A is a random oracle, then $LOGSPACE^A \neq P^A$ with probability 1.*

Proof. The language used to prove Theorem 2 is $BIGQUERY^A = \{x: \xi_A(x) \in A\}$, which is obviously in P^A for every oracle A . Every oracle machine that recognizes this language must compute and store some representation of $\xi_A(x)$ on its work tape, which costs at least $|x|$ bits. Queries of the form “ $x \times 10^i \in A?$ ” can be asked within the log space bound by simply transferring x from the input tape to the query tape, followed by the appropriate number of zeros. Such queries suffice to determine individual bits of the string $\xi_A(x)$. However, these bits cannot be accumulated on the query tape, since it is meanwhile being used for other queries, nor can they be stored on the work tape without violating the space bound. Not knowing $\xi_A(x)$, a logspace bounded machine must therefore, for every sufficiently large x , err with probability nearly $\frac{1}{2}$ in deciding whether $\xi_A(x)$ belongs to A .

More formally, let M be a logspace bounded deterministic oracle machine. A string y of length n is *queriable* by M if there is an oracle X for which y is queried by M^X on input 0^n . Initially, and just after each oracle query, the query tape is blank. Since M is logspace bounded, the total number of distinct machine states (instantaneous descriptions) with a blank query tape is at most cn^k for constants c and k depending on M but independent of n . When M is started in any one of these states, the computation proceeds deterministically, and independently of the oracle, until the next query (or until halting if no further queries were made). Therefore at most cn^k n -bit strings are queriable.

On the other hand, as the oracle A is varied, $\xi_A(0^n)$ takes on any of 2^n distinct values, all equally likely. Let $C = \{A: M^A(0^n) \text{ queries } \xi_A(0^n)\}$ be the class of oracles for which $\xi_A(0^n)$ is actually queried. C is a subclass of $\{A: \xi_A(0^n) \text{ is queriable}\}$, and so C has measure at most $cn^k/2^n$, which approaches 0 for large n . Therefore \bar{C} , the class of oracles for which $M^A(0^n)$ does *not* query $\xi_A(0^n)$, has measure 1 in the limit.

If M^A does not query $\xi_A(0^n)$, then it is obviously in a poor position to decide whether 0^n is in $BIGQUERY^A$, that is, whether $\xi_A(0^n)$ is in A . Consider the measure-preserving transformation of oracles that removes from A if it is present, or adds to A if it is absent, the string $\xi_A(0^n)$. This transformation maps \bar{C} onto itself, and for every oracle in \bar{C} changes the truth of $0^n \in BIGQUERY^A$ without changing the machine's answer $M^A(0^n)$. Therefore, for each machine M , the class of oracles on which $M^A(0^n)$ errs in determining whether 0^n belongs to $BIGQUERY^A$ has measure nearly $\frac{1}{2}$ for large n . By Lemma 1, with probability 1 $BIGQUERY^A$ is not in $LOGSPACE^A$. \square

COROLLARY. *If A is a random oracle, then $PSPACE^A \neq EXPTIME^A$ with probability 1.*

Proof. As above, using $VERYBIGQUERY^A = \{x: \xi_A(0^x) \in A\}$ as the test language. With probability 1, this test language is in $EXPTIME^A$ but not in $PSPACE^A$.

3. Probabilistic polynomial time languages. This section investigates the relativized classes of languages computable in polynomial time by probabilistic oracle machines [Gi]. Probabilistic machines are equipped with a coin toss mechanism that enables them to make fresh random choices during a computation. This randomness should be distinguished from the randomness of the oracle A , which is fixed before the computations begin. (However, some of the theorems below are proved by using the random oracle to simulate coin tosses, or vice versa.)

The language *accepted* by a probabilistic machine M with oracle A is defined as the set of inputs for which the machine halts in an accepting state with probability greater than $\frac{1}{2}$, and the characteristic function $M^A(x)$ takes on the value 1 or 0 according to this majority result (if the acceptance probability is exactly $\frac{1}{2}$, $M^A(x) = 0$). The *error probability* of M^A on input x is defined as the fraction of coin toss sequences leading to nonacceptance if $M^A(x) = 1$, or to acceptance if $M^A(x) = 0$. A probabilistic oracle machine M is polynomial time bounded if there exists a polynomial p such that, for all oracles A and inputs x , all computation paths halt within $p(|x|)$ steps.

Several classes of probabilistic polynomial time languages can be defined, depending on the allowed error probability.

DEFINITION. Let A be any oracle set.

1) \mathbf{PP}^A is the class of languages accepted by polynomial time bounded probabilistic oracle machines with oracle A . Simon [Si] has shown that the same class results if the definition, is strengthened to include only languages recognizable by machines with error probability less than $\frac{1}{2}$ on all inputs nonmembers as well as members.

2) \mathbf{BPP}^A is the class of languages accepted by polynomial time bounded probabilistic oracle machines with error probability uniformly bounded below $\frac{1}{2}$. A language L is in \mathbf{BPP}^A iff there is a polynomial time bounded probabilistic oracle machine M and a constant $\varepsilon < \frac{1}{2}$ such that $L = M^A$ and the error probability of M^A is less than ε for all inputs, members as well as nonmembers.

The difference between \mathbf{BPP} and \mathbf{PP} is that for languages in \mathbf{BPP} the error probability can be made uniformly as small as desired by repeating the probabilistic computation a uniform number of times, whereas this is not generally possible for a language in \mathbf{PP} . In particular, if a language L is recognizable with error probability uniformly below $\varepsilon < \frac{1}{2}$, then performing the computation m times and taking the majority decision (m odd) suffices to reduce the error probability uniformly below

$$\sum_{k=0}^{(m-1)/2} \binom{m}{k} \varepsilon^{m-k} (1-\varepsilon)^k,$$

which approaches zero exponentially with increasing m (this follows from the fact that for large m , the binomial distribution approximates a normal distribution of standard deviation $\sqrt{m\varepsilon(1-\varepsilon)}$ and mean $(1-\varepsilon)m$; and the fact that the area under the tail of the normal curve, from $-\infty$ to a point x standard deviations below the mean, is bounded above by $\text{const} \times \exp(-x^2/2)$ [Fe]). Thus, \mathbf{BPP}^A may be defined without loss of generality as the set of languages accepted by polynomial time bounded probabilistic oracle machines M^A with error probability uniformly below, say, $\frac{1}{4}$.

A well-known subclass of \mathbf{BPP} is the class called \mathbf{R} [AM], [Ra] or \mathbf{VPP} [Gi], consisting of languages, such as the composite numbers, that are probabilistically recognizable in polynomial time by one-sided Monte Carlo tests that never accept a nonmember of the language. \mathbf{BPP} includes such languages and their complements, as well as languages (no natural examples are known) for which only two-sided Monte Carlo tests exist.

Another subclass of **BPP**, known as **ZPP**[Gi], may be defined as $\mathbf{R} \cap \text{co-}\mathbf{R}$, or equivalently as the class of languages recognizable by probabilistic machines with zero error probability and polynomial bounded *average* run time.

It is easily shown [Gi] that $\mathbf{P} \subseteq \mathbf{ZPP} \subseteq \mathbf{BPP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$ and, perhaps more surprisingly, that $\mathbf{NP} \subseteq \mathbf{PP}$. From the definitions, it is obvious that **PP**, **BPP**, and **ZPP** are closed under complementation. All these relations continue to hold when the classes are relativized to an arbitrary oracle. In this section, we show that, relative to a random oracle A , the classes $(\mathbf{NP}^A \cup \text{co-}\mathbf{NP}^A) \subsetneq \mathbf{PP}^A \subsetneq \mathbf{PSPACE}^A$ are distinct with probability 1, whereas $\mathbf{BPP}^A = \mathbf{P}^A$ with probability 1.

THEOREM 3. *If A is a random oracle, then $\mathbf{PP}^A \subsetneq \mathbf{PSPACE}^A$ with probability 1.*

Proof. Let $\text{ODD}^A = \{x : \text{an odd number of strings of length } |x| \text{ are in } A\}$. ODD^A is computable in linear space with oracle A , and so ODD^A is in \mathbf{PSPACE}^A . On the other hand, it is intuitively clear that a probabilistic algorithm to decide whether x is in ODD^A without querying all strings of length $|x|$ must, for typical x and A , have an error probability of exactly $\frac{1}{2}$.

For any polynomial time bounded probabilistic oracle machine M , let $\varepsilon(x, A)$ be the error probability of M with oracle A and input x . The computation path of M^A on input x is determined by the random Bernoulli sequence B of coin tosses. Therefore, the error probability can be written as $\varepsilon(x, A) = \mu\{B : M^{AB}(x) \neq \text{ODD}^A(x)\}$, where $M^{AB}(x)$ is the output of $M^A(x)$ with coin toss sequence B and μ is Lebesgue measure on the set of infinite coin toss sequences.

Choose an input x so large that no computation path of $M^A(x)$ has time to query all strings of length $|x|$. Let $\mathbf{C}^+ = \{A : \varepsilon(x, A) < \frac{1}{2}\}$ and $\mathbf{C}^- = \{A : \varepsilon(x, A) > \frac{1}{2}\}$. We shall show that $\mu(\mathbf{C}^+) = \mu(\mathbf{C}^-)$.

We define a measure-preserving transformation $(A, B) \rightarrow (A', B)$, in the product space $\Omega_A \times \Omega_B$ of oracles A with Bernoulli sequences B , which maps $\mathbf{C}^+ \times \Omega_B$ onto $\mathbf{C}^- \times \Omega_B$ and vice-versa. The transformation consists of adding to A if it is absent, or removing from A if it is present, the first string of length $|x|$ not queried in the computation path $M^{AB}(x)$. The transformation thus always changes the value of $\text{ODD}^A(x)$ while never changing the machine's answer $M^{AB}(x)$. Hence, it maps $\mathbf{C}^- \times \Omega_B$ onto $\mathbf{C}^+ \times \Omega_B$ and vice versa. Therefore, $\mu(\mathbf{C}^+) = \mu(\mathbf{C}^-)$.

Since $\mathbf{C}^- \subseteq \bar{\mathbf{C}}^+$, we conclude that $\mu(\mathbf{C}^+) \leq \frac{1}{2}$. For all oracles not in \mathbf{C}^+ , the machine M^A does not correctly decide whether x is in ODD^A . Therefore, by Lemma 1, with probability 1, ODD^A is not in \mathbf{PP}^A . \square

THEOREM 4. *If A is a random oracle, then $\mathbf{NP}^A \cup \text{co-}\mathbf{NP}^A \subsetneq \mathbf{PP}^A$ with probability 1.*

Proof. By Theorem 1, RANGE^A is in $\mathbf{NP}^A\text{-co-}\mathbf{NP}^A$ and CORANGE^A is in $\text{co-}\mathbf{NP}^A\text{-}\mathbf{NP}^A$ with probability 1. Therefore, with probability 1, the combined language $\text{RANGE}^A \text{ join } \text{CORANGE}^A = \{0x : x \in \text{RANGE}^A\} \cup \{1x : x \in \text{CORANGE}^A\}$ is in neither \mathbf{NP}^A nor $\text{co-}\mathbf{NP}^A$ but is in \mathbf{PP}^A because both \mathbf{NP}^A and $\text{co-}\mathbf{NP}^A$ are subclasses of \mathbf{PP}^A . \square

Remark. This same example establishes that with probability 1 $\mathbf{NP}^A \cup \text{co-}\mathbf{NP}^A$ is properly contained in the class $\Delta_2^{\mathbf{P},A}$ of languages recognizable in polynomial time relative to an oracle in \mathbf{NP}^A . $\Delta_2^{\mathbf{P},A}$ is a member of the relativized Meyer-Stockmeyer **P**-hierarchy [MS], [BGS] and [BS], a polynomially time bounded analogue of the Kleene arithmetical hierarchy [Ro]. Like \mathbf{PP}^A , it includes \mathbf{NP}^A and is closed under complementation. Whether relativization by a random oracle separates classes higher than $\Delta_2^{\mathbf{P},A}$ in the hierarchy is currently unknown, as is the relationship between $\Delta_2^{\mathbf{P},A}$ and \mathbf{PP}^A .

THEOREM 5. *If A is a random oracle, then $\mathbf{P}^A = \mathbf{ZPP}^A = \mathbf{R}^A = \mathbf{BPP}^A$ with probability 1.*

Proof. It is sufficient to show that for every $\delta > 0$, the class of oracles A for which $\mathbf{P}^A \neq \mathbf{BPP}^A$ has measure less than δ . As noted earlier, \mathbf{BPP}^A may be defined without loss of generality as the class of languages recognizable by probabilistic polynomial time bounded oracle machines M_i^A with error probability uniformly bounded below $\frac{1}{4}$ for all inputs.

For any polynomial time bounded probabilistic oracle machine M_i we can effectively construct another, $M_{f(i)}$, that recognizes the same language as does M_i but with smaller error probability; in fact, there is a recursive function f such that if the error probability $\varepsilon_i(x, A)$ of M_i^A on input x is less than $\frac{1}{4}$, then $\varepsilon_{f(i)}(x, A)$ decreases exponentially with i and $|x|$, being bounded above by $\delta \cdot 2^{-(i+2|x|+2)}$. The machine $M_{f(i)}$ takes the majority vote of $c(i+2|x|+2)$ independent computations of $M_i^A(x)$; the constant c depends on δ , but not on i, x or A .

Next, we construct a *deterministic* polynomial time bounded oracle machine $M_{g(i)}$ that operates as follows. With input x , it first computes $p_{f(i)}(|x|)$, a polynomial upper bound on the length of queries that can be made by $M_{f(i)}$. Such a bound always exists because of the polynomial time bound on $M_{f(i)}$. Then, $M_{g(i)}^A$ simulates the probabilistic oracle machine computation $M_{f(i)}^A(x)$. Each time the simulated computation $M_{f(i)}^A(x)$ requires a coin toss, $M_{g(i)}^A(x)$ obtains a bit by querying the oracle A about the least string of length greater than $p_{f(i)}(|x|)$ that has not yet been queried.

Let \mathbf{E}_{ix} be the class of oracles A for which $M_i^A(x)$ has error probability less than $\frac{1}{4}$ but $M_{g(i)}^A(x)$ does not agree with the majority answer of $M_i^A(x)$. Since the queries made by $M_{g(i)}^A(x)$ in simulating coin tosses are larger than any queries actually made by any simulated probabilistic computation $M_{f(i)}^A(x)$, the measure of \mathbf{E}_{ix} does not exceed the error probability of $M_{f(i)}^A(x)$. That is,

$$\mu(\mathbf{E}_{ix}) \leq \max \{ \varepsilon_{f(i)}(x, A) : \varepsilon_i(x, A) < \frac{1}{4} \} < \delta \cdot 2^{-(i+2|x|+2)}.$$

Taking the union over i and x , we obtain

$$\mu\left(\bigcup_{ix} \mathbf{E}_{ix}\right) \leq \sum_{ix} \mu(\mathbf{E}_{ix}) < \delta.$$

(The convergence of this sum does not require that the events \mathbf{E}_{ix} be independent, merely that they individually be of small measure; in general the \mathbf{E}_{ix} will be strongly correlated, because the construction allows the same oracle bit to simulate a coin toss for many different machines and inputs.) To conclude the proof, we observe that $\mathbf{P}^A = \mathbf{BPP}^A$ for every oracle A not in $\bigcup_{ix} \mathbf{E}_{ix}$, since for every such oracle, if language L is accepted by M_i^A with error probability uniformly less than $\frac{1}{4}$, then L is recognized by the deterministic oracle machine $M_{g(i)}$. \square

Remark. For each δ in the above proof, the set of oracles $\mathbf{N}_\delta = \bigcap_{ix} \bar{\mathbf{E}}_{ix}$ is a nowhere-dense set in the sense of Mehlhorn [Me], and the union over δ of these sets is a meager set of measure 1 on which $\mathbf{P}^A = \mathbf{BPP}^A$. This raises the interesting possibility that the set of all oracles for which $\mathbf{P}^A = \mathbf{BPP}^A$ may be sparse in one sense (Baire category theory), but co-sparse in another, more intuitive sense (measure).

COROLLARY. *If L is a non-oracle-dependent language which belongs to \mathbf{P}^A with probability 1 for random A , then L belongs to the unrelativized class \mathbf{BPP} . Conversely, every language in \mathbf{BPP} is in \mathbf{P}^A with probability 1.*

Proof. The first part follows from the ability of a probabilistic algorithm without oracle to simulate, by coin tossing, the answers a random oracle would give to a

deterministic algorithm. The converse is a special case of the theorem just proved: **BPP** is always a subclass of \mathbf{BPP}^A , which in turn is equal to \mathbf{P}^A , with probability one.

Remark. The second part of this corollary, that any language in **BPP** is in \mathbf{P}^A with probability 1, generalizes to **BPP** Adleman's result [Ad] that any language in **R** has polynomial size circuits. A language L is in **R** iff every member of L is "witnessed" by at least half the strings of appropriate (polynomial $p(n)$) size and no nonmember is witnessed by any. Adleman showed that under these conditions, there exists for each n a specific set of $\leq n$ witnesses sufficient to witness all members of L smaller than n bits. A fixed table of $np(n)$ bits is thus enough to simulate the approximately $2^{np(n)}$ bits of witnesses that would be consulted if witnesses were generated probabilistically on each input.

In the proof of Theorem 5, if the language L accepted by probabilistic machine M_i is oracle-independent, belonging to **BPP** rather than merely to \mathbf{BPP}^A , then the bound $p_{f(i)}(|x|)$ on the size of queries by $M_{f(i)}$ can be taken to be zero. This means that the deterministic machine $M_{g(i)}$ uses the same initial bits of the random oracle, $A(1), A(2), A(3), \dots$ over and over again, to simulate the (in general different) coin toss sequences that the machines M_i and $M_{f(i)}$ would generate on different inputs. If the number of coin tosses made by the original probabilistic machine M_i is bounded by a polynomial $q(n)$ in the input length, then the number made by the more accurate machine $M_{f(i)}$ is bounded by a larger polynomial $cnq(n)$, and the number of random oracle bits needed by the deterministic machine to evaluate $L(x)$ accurately for all inputs of length $\leq n$ is also bounded by $cnq(n)$. Thus, a fixed table of $cnq(n)$ random bits suffices to compute, without error, a finite set whose probabilistic computation, with errors, would use approximately $2^n q(n)$ coin toss bits.

4. P^A -immunity. Classes such as **P** and **NP** refer to worst case performance. However, for \mathbf{RANGE}^A and the other oracle-dependent languages discussed here, *most* members are as difficult to recognize as the worst case. A particularly strong form of this property is called **P-immunity**: a set is **P-immune** if it has no infinite subset that is in **P**. For typical oracles A , \mathbf{RANGE}^A is not itself \mathbf{P}^A -immune, because, for example, it contains the \mathbf{P}^A -recognizable infinite subset $\{x: \xi_A(x) = x\}$. However, Theorem 6, proved later in this section, gives a set in \mathbf{NP}^A that is \mathbf{P}^A -immune and \mathbf{P}^A -co-immune (i.e., its complement \mathbf{P}^A -immune) with probability 1. It is of course not known whether **NP** contains a **P-immune** set in the absence of an oracle, for that would imply $\mathbf{P} \neq \mathbf{NP}$, nor is it known whether all oracles X that make $\mathbf{P}^X \neq \mathbf{NP}^X$ also imply that \mathbf{NP}^X contains a \mathbf{P}^X -immune set.

Another interesting question is whether there is an oracle X for which a set can be at once \mathbf{P}^X -immune and \mathbf{NP}^X -complete. (In order to define \mathbf{NP}^X -completeness, one must of course specify a reducibility relation. In § 6 it will be argued that, in order to be a fully relativized concept, \mathbf{NP}^X -completeness ought to be defined in terms of a relativized reducibility such as **P**, X -Turing reducibility, in which U is reducible to V iff $U \in \mathbf{P}^{X \text{ join } V}$, rather than the more customary **P**-Turing reducibility.) When X is the empty set, or a random oracle, immunity and completeness appear to be incompatible. Standard **NP**-complete sets such as $\text{SAT} = \{f: \text{the propositional formula } f \text{ is satisfiable}\}$ contain infinite easy subsets, and so are not **P-immune**. Moreover, Berman and Hartmanis [BH] have shown that all known **NP**-complete sets are p -isomorphic, and conjecture that all **NP**-complete sets are.

This conjecture would imply that no **NP**-complete set is **P-immune**, since p -isomorphism preserves **P-immunity**. [*Proof.* Let sets U and V be p -isomorphic. Then, by definition of p -isomorphism there is a 1:1 onto function f with both f and f^{-1}

computable in polynomial time such that $x \in U$ iff $f(x) \in V$. Let U be not \mathbf{P} -immune, and let $E \in \mathbf{P}$ be an infinite easy subset of U . Then, $f(E)$ is an infinite easy subset of V , making V not \mathbf{P} -immune. $f(E)$ is infinite because f is $1:1$, and $f(E) \in \mathbf{P}$ because $E \in \mathbf{P}$ and f^{-1} is computable in polynomial time. This argument also applies in a relativized form: for any A , if U and V are p -isomorphic, or even if they are only p^A -isomorphic, i.e., interconvertible by a permutation A -computable in polynomial time, then U is \mathbf{P}^A -immune iff V is \mathbf{P}^A -immune.]

The Berman–Hartmanis conjecture implies that complete sets cannot be p -sparse (a p -sparse set being one whose number of members of length $\leq n$ is bounded by a polynomial in n). Immune sets, on the other hand, may be p -sparse or not; for typical random oracles A the \mathbf{P}^A -immune set of Theorem 6 below is moderately dense, but its intersection with a p -sparse set such as 0^* is p -sparse, still \mathbf{P}^A -immune, and still in \mathbf{NP}^A . (Recently Mahaney [Ma] has shown that, unless $\mathbf{P} = \mathbf{NP}$, no \mathbf{NP} -complete set can be p -sparse).

Although \mathbf{P}^A -immune sets can be moderately dense, with probability 1 no \mathbf{P}^A -immune set in \mathbf{NP}^A can be so dense that its complement is p -sparse. [*Proof.* Let A be a typical random oracle, and let S be a co- p -sparse set accepted by the nondeterministic machine \mathbf{NP}_i^A . Since S is co- p -sparse, there exist probabilistic polynomial time algorithms (e.g., on input x , accept with probability $2^{-|x|}$) that, with probability arbitrarily close to unity, when applied to the inputs $0, 1, 2, \dots$ in sequence, accept infinitely many members of S but no nonmembers. Each such probabilistic algorithm can be simulated by a deterministic polynomial time algorithm that queries A about strings too long to have been queried by \mathbf{NP}_i^A on the same input. Thus, there is, for typical A , a deterministic algorithm to accept an infinite subset of S , rendering S not \mathbf{P}^A -immune.]

Although \mathbf{P}^A -immune sets in \mathbf{NP}^A cannot be co- p -sparse, those not in \mathbf{NP}^A can be. For example, the set $\{x: \forall j \leq |x| \varphi_j^A(0) \neq x\}$ is co- p -sparse yet has no infinite A -r.e. subset. Hence, it is certainly \mathbf{P}^A -immune.

We now show that with probability 1, \mathbf{NP}^A contains a \mathbf{P}^A -immune set.

THEOREM 6. *If A is a random oracle, the set $\mathbf{RANGE}3^A = \{x: \exists y \xi_A(y) = xxx\}$ and its complement are \mathbf{P}^A -immune with probability 1. Here, xxx denotes x thrice concatenated.*

Proof. $\mathbf{RANGE}3^A$ is infinite and co-infinite, and indeed about as dense as \mathbf{RANGE}^A , having on the average $2^n(1 - e^{-1})$ members each of length n . It is obviously in \mathbf{NP}^A . However, it is \mathbf{P}^A -immune because, intuitively, the expected cumulative number of successful guesses, on input x , of a string y that would map into xxx , approaches a finite limit as $x \rightarrow \infty$. Note that $\mathbf{RANGE}3^A$ is not \mathbf{NP}^A -complete, because it contains answers to only a few of the questions needed to recognize, say, \mathbf{RANGE}^A in polynomial time.

To prove that $\mathbf{RANGE}3^A$ is \mathbf{P}^A -immune, it suffices to prove for each deterministic polynomial-time algorithm M that \mathbf{C} , the class of oracles A for which that algorithm accepts an infinite subset of $\mathbf{RANGE}3^A$, is of measure zero.

Let M be applied to all inputs, $\Lambda, 0, 1, 00, \dots$ in sequence and consider the finite set of oracle strings first examined in the course of the computation on input w :

$$\mathbf{EXAM}(A, w) = \{y: M^A(w) \text{ examines } y\} - \{y: \exists v < w M^A(v) \text{ examines } y\}.$$

Recall that a string y is said to be examined when any of the oracle strings affecting the value of $\xi_A(y)$ is queried. In general, we have regarded the oracle as having been chosen probabilistically in the beginning, after which computations proceed deterministically relative to it; however, when considering a fixed sequence of computations, it is

permissible to regard $\xi_A(y)$ as being decided probabilistically for each argument y at the time that argument is first examined. Subsequent evaluations of $\xi_A(y)$ must of course return the same value.

In order to be useful evidence in favor of accepting a member of $RANGE3^A$, an examined string y must have $\xi_A(y) = xxx$, for some x , and must have been examined sufficiently early, $\exists_{w \leq x} y \in EXAM(A, w)$, to influence the acceptance of x . The set of strings for which this is so may be defined:

$$EVIDENCE(A) = \bigcup_w \{y : y \in EXAM(A, w) \text{ and } \exists_{x \geq w} \xi_A(y) = xxx\}.$$

It is not difficult to see that, with probability 1, $EVIDENCE(A)$ contains only finitely many members. To prove this, note that the polynomial bound on M implies that, for all but finitely many w , $EXAM(A, w)$ contains fewer than $2^{\lfloor w/2 \rfloor}$ members. Furthermore, since at the time each y in $EXAM(A, w)$ is first examined, it has by definition not been examined before, the event $\{A : \exists_{x \geq w} \xi_A(y) = xxx\}$ is independent of all previously examined parts of the oracle, and has probability $2^{-2^{\lfloor w \rfloor}}$ or less, because of the preponderance of $3n$ -bit strings not of the form xxx . Summing $2^{\lfloor w/2 \rfloor} \cdot 2^{-2^{\lfloor w \rfloor}}$ over all w , one obtains a finite expected number of strings in $EVIDENCE(A)$, and, by the Borel–Cantelli lemma, this implies that $\mu\{A : EVIDENCE(A) \text{ is infinite}\} = 0$.

We now define $x_k(A)$ as the k th input string accepted without evidence under oracle A :

$$x_k(A) = \min \{x : x > x_{k-1}(A) \text{ and } x \in M^A \text{ and } \forall_{y \in EVIDENCE(A)} \xi_A(y) \neq xxx\}.$$

$x_k(A)$ may not always be defined (e.g., when M^A , the language accepted by M with oracle A , is finite, or when, with probability zero, $EVIDENCE(A)$ is infinite); however, when infinitely many inputs are accepted, then (with conditional probability 1) all but finitely many of them are accepted without evidence.

The class $C = \{A : M^A \text{ is an infinite subset of } RANGE3^A\}$, which we seek to show has measure zero, has the same measure as $D = C \cap \{A : EVIDENCE(A) \text{ is finite}\}$. D , in turn, can be viewed as the limit of the nested sequence of classes $D_1 \supset D_2 \supset D_3 \cdots$, where

$$D_k = \{A : x_k(A) \text{ exists and } \forall_{i < k} x_i(A) \in RANGE3^A\}.$$

D can have nonzero measure only if the ratio $\mu(D_k)/\mu(D_{k-1})$ approaches unity as $k \rightarrow \infty$. However, it is easy to see that this ratio has a lim sup not exceeding $1 - e^{-1} \approx 0.632$. This is the limiting probability that, at the stage when input $x = x_k(A)$ is accepted without evidence, xxx , having no inverse image among the strings examined so far, does have an inverse image among the nearly $2^{3^{\lfloor x \rfloor}}$ strings of length $3\lfloor x \rfloor$ not examined so far. Therefore, $\mu(D) = \mu(C) = 0$, and $RANGE3^A$ is P^A -immune with probability 1.

The proof that $RANGE3^A$ is P^A -co-immune with probability 1 proceeds similarly. Here it is even clearer that if infinitely many members of the complement of $RANGE3^A$ are accepted, all but finitely many of them must be accepted without adequate evidence (no polynomial number of instances of y such that $\xi_A(y) \neq xxx$ can increase above $1/e \approx 0.368$, the asymptotic fraction of oracles for which $\forall_y \xi_A(y) \neq xxx$). \square

Remark. The set $RANGE2^A = \{x : \exists_y \xi_A(y) = xx\}$ may also be P^A -immune, inasmuch as the obvious strategy for recognizing members of it yields only finitely many. $RANGE2^A$ and $RANGE^A$ are P^A -coimmune with probability 1.

5. Relativization of the $P = ? NP \cap co-NP$ question. It is unclear whether, relative to a random oracle A , P^A is properly contained in the intersection of NP^A and $co-NP^A$.

If $\mathbf{P}^A = \mathbf{NP}^A \cap \text{co-NP}^A$, then \mathbf{P}^A includes such non-oracle-dependent, seemingly-difficult problems as factorization, known to be in $\mathbf{NP} \cap \text{co-NP}$. By the corollary to Theorem 5, this would imply that such problems are solvable probabilistically in the sense of **BPP**, making them computationally tractable in a practical sense, contrary to appearances.

On the other hand, we have not been able to find an oracle-dependent language in $\mathbf{NP}^A - \mathbf{P}^A$ whose complement is also in $\mathbf{NP}^A - \mathbf{P}^A$. The attempt to construct an oracle-dependent language analogous, say, to $\text{FACTPROJ} = \{\langle x, y \rangle : x \cong \text{prime-factorization-of}(y)\}$, which encodes factorization, is frustrated by the existence of multiple inverse images under the ξ function, in contrast with the uniqueness of factorization. Thus, FACTPROJ is in both \mathbf{NP} and co-NP , but the obvious A -dependent analogue, $\text{XIPROJ}^A = \{\langle x, y \rangle : \exists z x \cong z \text{ and } \xi_A(z) = y\}$, like RANGE^A , is in \mathbf{NP}^A but not co-NP^A .

If we replace the random function $\xi_A(x)$ by a function $\pi(x)$ which randomly maps strings of each length onto one another in a 1 : 1 fashion (i.e., a permutation), then it is easy to show that, with probability 1, \mathbf{P}^π is properly contained in $\mathbf{NP}^\pi \cap \text{co-NP}^\pi$. The probability measure is the product measure over n of an assignment of equal weight $1/(2^n!)$ to each permutation of n -bit strings. This separation can be demonstrated using the oracle-dependent language $\text{PIPROJ}^\pi = \{\langle x, y \rangle : x \cong \pi^{-1}(y)\}$, or, more simply, $\text{HALFRANGE}^\pi = \{x : \exists y \pi(0y) = x\}$. Both PIPROJ^π and HALFRANGE^π are in $(\mathbf{NP}^\pi \cap \text{co-NP}^\pi) - \mathbf{P}^\pi$.

By a proof like that of Theorem 6, the oracle-dependent set $\text{HALFRANGE3}^\pi = \{x : \exists y \pi(0y) = xxx\}$, which belongs to $\mathbf{NP}^\pi \cap \text{co-NP}^\pi$, can be shown to be \mathbf{P}^π -immune and \mathbf{P}^π -coimmune with probability 1.

All the theorems given earlier for complexity classes relativized to a random oracle A hold for the analogous complexity classes relativized to a random 1 : 1 function π . The π analogues of all but Theorem 3 are proved using the many-to-one random function $\xi_\pi(x) = [\text{the first } |x| \text{ bits of } \pi(xx)]$, which has nearly the same statistics as ξ_A . The π analogue of Theorem 3 can be proved using the language $\text{ODDPERM}^\pi = \{x : \pi \text{ performs an odd permutation on strings of length } |x|\}$. For any string length n , odd and even permutations are equiprobable, and they remain conditionally equiprobable as long as two or more arguments of the permutation remain unexamined. On the other hand, by exhaustively tracing all the permutation's cycles, its parity can be determined within a polynomial space bound. A random permutation can thus apparently substitute for a random oracle.

On the other hand, we can think of no way to use a random oracle A to construct a rapidly-evaluable random 1 : 1 function π , analogous to the construction of ξ_A from A ; for this reason, the π function is less intuitively appealing, seeming to have more built-in structure, than the many-to-one ξ function.

Oracles with even more complicated kinds of randomness can be imagined, and indeed are apparently necessary to yield an easy proof, in the relativized setting, of certain putative properties of the natural number system, viz., the ability to support classical and public-key cryptography [DH]. A secure public-key cryptosystem, for example, exists with probability 1 relative to the oracle A **join** B , where A is a random oracle of the usual sort and B contains pairs of mutually-inverse random permutations indexed by A ; e.g., for each n -bit string x , if u and v denote respectively the first and last halves of the $6n$ -bit string $\xi_A(xxxxxx)$, then the functions $\xi_B(uy) = uz$ and $\xi_B(vz) = vy$ define mutually inverse random permutations between n -bit strings y and z . Each user of such a system picks an x randomly and secretly, finds u and v from it using A , and publishes u but not v . Other users then use B in conjunction with the public key u to encrypt messages ($y \rightarrow z$) that only the original user, with private key v ,

can economically decrypt ($z \rightarrow y$) (using keys of length $3n$ rather than n insures that, despite the many-to-one nature of ξ_A , all but finitely many of the keys will be unique). The oracle A join B is a random analogue of the more complicated but recursive cryptographic oracles of Brassard [Br]. As Brassard points out, it is difficult to find an intuitively satisfactory asymptotic definition of cryptographic security. The relativized cryptosystem described above is secure in the ordinary, non-asymptotic sense that for typical message sizes (say $n = 100$), standard cyptanalytic tasks such as chosen plaintext attack could not be performed rapidly and reliably by a probabilistic query machine with a small number of internal states.

6. Discussion, random oracle hypothesis. Without oracles, the hierarchy of complexity classes includes the following known relations:

$$\text{LOGSPACE} \subseteq P \subseteq \text{ZPP} \subseteq \left\{ \begin{array}{l} R \subseteq NP \\ (R \cup \text{co-R}) \subseteq \text{BPP} \\ \text{co-R} \subseteq \text{co-NP} \end{array} \right\} \subseteq \text{PP} \subseteq \text{PSPACE}.$$

None of the inclusions is known to be proper, except that $\text{LOGSPACE} \subsetneq \text{PSPACE}$. Relativization with respect to a random oracle A yields the following greatly sharpened relations, with probability 1:

$$\text{LOGSPACE}^A \subseteq \left\{ \begin{array}{l} P^A = \text{ZPP}^A \\ = \\ R^A = \text{BPP}^A \end{array} \right\} \subseteq \left\{ \begin{array}{l} NP^A \\ \neq \\ \text{co-NP}^A \end{array} \right\} \subseteq \text{PP}^A \subseteq \text{PSPACE}^A.$$

Relativization with respect to a random permutation function π , instead of the random oracle A , yields all these results and, in addition, $P^\pi \subsetneq NP^\pi \cap \text{co-NP}^\pi$ with probability 1, which we have been unable to decide for a simple random oracle.

In view of the large number of classes that are separated by random oracle relativization, one might suppose that if there exists any oracle at all relative to which two classes are distinct, they they will be distinct relative to a random oracle. That this is not the case was shown by Hunt's [Hu] construction of an oracle X for which $P^X \subsetneq \text{ZPP}^X$, even though, by Theorem 5, these classes coincide with probability 1 relative to a random oracle. On the other hand, separations and identities that hold with probability 1 relative to a random oracle can generally also be demonstrated relative to particular recursive oracles.

Most of the random oracle results are obtained by using the oracle's randomness to force language recognition to depend on oracle queries, thereby in effect substituting number and size of queries for the more conventional (but theoretically intractable) dynamic computation resources of time and space. Thus, there is no immediate prospect of proving similarly sharp results in the absence of an oracle. On the other hand, random oracles by their very structurelessness appear more benign and less likely to distort the relations among complexity classes than the oracles traditionally used in complexity theory and recursive function theory, which are usually designed expressly to help or frustrate some class of computations. This suggests that statements that hold with probability 1 for languages relativized to a random oracle A are also true in the unrelativized case $A = \emptyset$.

To formalize this conjecture, the universe of appropriate statements needs to be defined. In particular, one wishes to include statements such as $P^A \neq NP^A$, $P^A = \text{BPP}^A$, and $\exists S (S \in NP^A \text{ and } S \text{ is } P^A\text{-immune})$, while excluding such incompletely relativized statements as $P = P^A$, or " A is recursive."

Since the languages of interest in relativized complexity theory are uniformly A -recursive (i.e., recognizable by Turing machines that halt for all oracles and inputs), they may be referred to by the Gödel numbers of their characteristic function, and a k -adic relativized relation among such languages may be represented by a denumerable set of k -tuples of Gödel numbers of languages obeying the relation.

DEFINITION: A natural number i is a *uniform index* if the function φ_i^A is total and zero-one valued for all oracles A .

In this definition, φ_i^A , as usual, denotes the function computed by the i th Turing machine with oracle A . Without loss of generality, these Turing machines may be taken to be deterministic machines with no time or space bound, since nondeterminism, probabilism, and uniform (i.e., oracle-independent) time or space bounds can be incorporated implicitly by appropriate choice of the index i . A relativized language class such as \mathbf{NP}^A (or equivalently the monadic relation $L \in \mathbf{NP}^A$) may now be formally defined as an indexed collection of A -parameterized languages invariant under appropriate group operations.

DEFINITION. Let I be a set of uniform indices. The A -parameterized class of languages $\mathbf{C}_I^A = \{\{x: \varphi_i^A(x) = 1\}: i \in I\}$ indexed by members of I is an *acceptable relativized class* iff:

1) for every oracle A , the class \mathbf{C}_I^A is invariant under p^A -isomorphism [BH]; i.e., if f is a 1:1 onto function such that both f and f^{-1} are computable in polynomial time with oracle A , and if L is any language, then $L \in \mathbf{C}_I^A$ iff $f(L) \in \mathbf{C}_I^A$;

2) the class \mathbf{C}_I^A is invariant under polynomial time Turing equivalences [La], [LLS] of the oracle set; i.e., if $B \in \mathbf{P}^A$ and $A \in \mathbf{P}^B$, then $\mathbf{C}_I^A = \mathbf{C}_I^B$.

Using this definition, it is not difficult to find index sets I for the language classes \mathbf{P}^A , \mathbf{BPP}^A , \mathbf{NP}^A , $\text{co-}\mathbf{NP}^A$, \mathbf{PP}^A , and \mathbf{PSPACE}^A . Other p^A -invariant classes generable in this manner are the class of finite languages and the class of languages with exactly k members, $k = 0, 1, 2$, etc. It is not clear, however, that more complicated classes such as $\{S: S \text{ is } \mathbf{NP}^A\text{-complete}\}$ and $\{S: S \in \mathbf{NP}^A \text{ and } S \text{ is } \mathbf{P}^A\text{-immune}\}$ can be generated by a single set of indices. To handle such cases, higher order relativized relations appear necessary.

DEFINITION: Let J be a set of ordered k -tuples of uniform indices. The A -parameterized class \mathbf{R}_J^A of k -tuples of languages indexed by the members of J is an *acceptable relativized relation* iff:

1) for every oracle A , the relation \mathbf{R}_J^A is invariant under p^A -isomorphism: i.e., if f is a p^A -isomorphism, and $\langle L, M, \dots, Q \rangle$ is a k -tuple of languages, then $\langle L, M, \dots, Q \rangle \in \mathbf{R}_J^A$ iff $\langle f(L), f(M), \dots, f(Q) \rangle \in \mathbf{R}_J^A$;

2) \mathbf{R}_J^A is invariant under polynomial time Turing equivalences of the oracle set.

Among the important dyadic relations are language equality and complementation ($L = M$ and $L = \bar{M}$) and reducibilities such as the relativized Turing reducibility \leq^A , whose index set is the union over k of pairs $\langle i, j \rangle$ such that for all A , $\varphi_i^A = \varphi_k^{L(j,A) \text{ join } A}$, where $L(j, A)$ denotes the language whose characteristic function is φ_j^A . An important refinement of \leq^A , obtained by restricting the index k to polynomial time bounded machines, is the reducibility $\leq^{\mathbf{P},A}$, which holds between two languages L and M iff they are uniformly A -recursive and L is uniformly recognizable in polynomial time with oracle M **join** A . Notice that simple Turing reducibility (or its polynomial refinement), in which the oracle for the reduction is M rather than M **join** A , is not an acceptable relativized relation, because it is not invariant under p^A -isomorphism for typical A . In the present context of full relativization, \mathbf{NP}^A -completeness should be taken to mean completeness with respect to an invariant reducibility such as $\leq^{\mathbf{P},A}$.

Intersection and union of languages may be expressed by acceptable triadic relations, e.g.,

$$R_J^A = \{(L, M, N) : L, M, \text{ and } N \text{ are uniformly } A\text{-recursive and } L = M \cap N\}.$$

The subset relation $L \subseteq M$, used for example in defining P^A -immunity, may be expressed by quantifying the above triadic relation as $\exists_N L = M \cap N$, where the bound variable N , like the free variables L and M , range over uniformly A -recursive languages.

With the notion of acceptable relativized classes and relations thus delimited, it is easy to define a broad class of statements to which the random oracle hypothesis may reasonably be expected to apply.

DEFINITION. The A -parameterized statement S^A is an *acceptable relativized statement* if it is definable in quantificational logic using

- bound variables denoting uniformly A -recursive languages;
- acceptable relativized relations on these variables;
- the logical operators AND, OR and NOT.

The oracle set A and the relations' index sets I, J , etc., appear only as parameters, and cannot be acted on by any of the quantifiers or relations. Note that acceptable relativized statements, by virtue of their invariance under P -Turing equivalence of the oracle set, can only have probability 0 or 1. The random oracle hypothesis may now be stated:

Random Oracle Hypothesis. Let S^A be any acceptable relativized statement. The corresponding unrelativized statement S^\emptyset is true if and only if S^A is true with probability 1 when A is chosen randomly.

In particular, since $NP^A \neq P^A$ and $P^A = BPP^A$ are acceptable statements that are true with probability 1, the random oracle hypothesis would imply $P = BPP \neq NP$. We believe that this hypothesis, or a similar but stronger one, captures a basic intuition of the pseudorandomness of nature from which many apparently true complexity results follow. The random oracle hypothesis could be strengthened by attempting to include non A -recursive languages, by relaxing the invariances required of acceptable relations (e.g., invariance under $\log\text{space}^A$ -isomorphism rather than p^A -isomorphism), and by asserting further that results true relative to a random *permutation* are true absolutely.

The random oracle hypothesis does not deny all differences between no oracle and a random oracle: clearly, machines equipped with a random oracle can recognize nonrecursive sets, while unaided machines cannot. Similarly, there exist sets which are immune absolutely, but, with probability 1, not immune relative to a random oracle [Ba]. However, all known differences of this sort concern partially relativized properties; in a fully relativized setting the differences disappear, since (for example) the nonrecursive sets recognized by random oracle machines are all A -recursive.

In view of the great amount of effort expended in unsuccessful attempts to prove apparently true statements such as $P \neq NP$, and $NP \neq PSPACE$, it is possible that these statements may be independent of other commonly accepted axioms of arithmetic and set theory. The random oracle hypothesis is thus a plausible candidate for a new axiom.

The random oracle hypothesis would be proved if an easily computable substitute for ξ (or A or π) could be found, e.g., a function ψ that requires little time and space to evaluate, but is pseudorandom in the sense that the inevitable correlations among $\psi(x)$ for different x cannot be exploited without large amounts of time and space. The search for this kind of pseudorandomness is related to the search (also quite unsuccessful so

far) for a provably almost-everywhere moderately-hard-to-compute function [Rb], [GB].

It is not hard to invent polynomially computable functions that *appear* pseudorandom; the difficulty arises in proving them so. For example, the function $\psi(x) =$ [the third $|x|$ bits of $\cos(x)$] appears to have the statistical properties of the ξ function, and finding inverse images appears to require an exponential search, but no one knows how to prove this.

[*Remark.* In defining ψ it is necessary to skip an increasing number of early bits of $\cos(x)$ because these early bits are more often 1 than zero, owing to the cosine's turning points at ± 1 . The bias in the k th bit is of order $2^{-k/2}$; thus, the sequence of $(2|x|+1)$ st bits of $\cos(x)$, for $x = 1, 2, 3, \dots$ should have a bias decreasing as x^{-1} , rendering it statistically indistinguishable from a random Bernoulli sequence. Other more complicated deviations from pseudorandomness, e.g., those arising from the nonuniform distribution of the difference between $\cos(x)$ and $\cos(x+1)$, would presumably be obliterated in the same way.]

By giving up the requirement that a function or set be easy to compute, one gains the ability to prove that specific sets are pseudorandom, i.e., that they have the properties of a generic random oracle with respect to bounded computation. A natural but nonrecursive example would be an *algorithmically random* set such as the bit sequence of Chaitin's real number ω [Ch], which expresses the halting probability of a universal Turing machine with random input. The set $\{x : \text{the } x\text{th digit of } \omega \text{ is a } 1\}$ is in class Δ_2 of the arithmetical hierarchy, but passes all computable tests of randomness, and could be substituted for the generic A in all the above theorems. Meyer and McCreight [MM], using a priority construction, have exhibited a recursive pseudorandom set, recognizable in quadruple exponential space but appearing random with respect to all test sets recognizable in double exponential space. Similar constructions should yield a proof of a weak analogue of the random oracle hypothesis, viz., that if an acceptable relativized statement S^A is true with probability 1 for random A , then it is also true for some recursive A . The converse, of course, does not hold, since many relativized statements [BGS] are known to be true for some recursive oracles but false for others.

Acknowledgments. The authors thank Larry Carter, Mark Wegman, George Markowsky, Dexter Kozen and Gilles Brassard for numerous helpful discussions, Gregory Chaitin for helping to formulate the method of using a random oracle to simulate probabilistic computation, and Robert Solovay for criticizing a preliminary version of the proof of Theorem 1.

REFERENCES

- [Ad] L. ADLEMAN, *Two theorems on random polynomial time*, Proceedings of the 19th IEEE Symposium on the Foundations of Computer Science, Ann Arbor, MI, 1978, pp. 75–83.
- [AM] L. ADLEMAN AND K. MANDERS, *Reducibility, Randomness, and Intractability*, Proceedings of the 9th ACM Symposium on the Theory of Computing, 1977, pp. 151–153.
- [An] D. ANGLUIN, *On counting problems and the polynomial time hierarchy*, Theoret. Comput. Sci., to appear.
- [Ba] YA. M. BARZDIN', *On computability by probabilistic machines*, Dokl. Akad. Nauk SSSR, 189 (1969), pp. 699–702, = Soviet Math. Dokl., 10 (1969), pp. 1464–1467.
- [Br] G. BRASSARD, *Relativized cryptography*, Proceedings of the 20th IEEE Symposium on the Foundations of Computer Science, San Juan, Puerto Rico, 1979, pp. 383–391.
- [BGS] T. BAKER, J. GILL AND R. SOLOVAY, *Relativizations of the $P = ? NP$ question*, this Journal, 4 (1975), pp. 431–442.

- [BH] L. BERMAN AND J. HARTMANIS, *On isomorphisms and densities of NP and other complete sets*, this Journal, 6 (1977), pp. 305–322.
- [BS] T. BAKER AND A. SELMAN, *A second step toward the polynomial hierarchy*, Proceedings of the 17th IEEE Symposium on the Foundations of Computer Science, 1976, pp. 71–75.
- [Ch] G. CHAITIN, *A theory of program size formally identical to information theory*, J. Assoc. Comput. Mach., 22 (1975), pp. 329–340.
- [DH] W. DIFFIE AND M. HELLMAN, *New directions in cryptography*, IEEE Trans. Inform. Theory IT-22 (1976), pp. 644–654.
- [Fe] W. FELLER, *An Introduction to Probability Theory and its Applications*, John Wiley, New York, 1957, Chapter 7.
- [Fe2] W. FELLER, *An Introduction to Probability Theory and its Applications, volume II*, John Wiley, New York, 1971, Chapter 4.
- [GB] J. GILL AND M. BLUM, *On almost everywhere complex recursive functions*, J. Assoc. Comput. Mach., 21 (1974), pp. 425–435.
- [Gi] J. GILL, *Computational complexity of probabilistic Turing machines*, this Journal, 6 (1977), pp. 675–695.
- [Hu] J. W. HUNT, *Topics in probabilistic complexity*, Ph.D. dissertation, Department of Electrical Engineering, Stanford University, Stanford CA, 1978, p. 58.
- [KM] D. KOZEN AND M. MACHTEY, *On relative diagonals*, J. Comput. System Sci., to appear.
- [La] R. E. LADNER, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach., 22 (1975), pp. 151–171.
- [LL] R. E. LADNER AND N. A. LYNCH, *Relativization of questions about log space computability*, Math. Systems Theory, 10 (1976), pp. 19–32.
- [LLS] R. E. LADNER, N. A. LYNCH AND A. L. SELMAN, *A comparison of polynomial time reducibilities*, Theoret. Comput. Sci., 1 (1975), pp. 103–123.
- [dLMSS] K. DE LEEUW, E. F. MOORE, C. E. SHANNON AND N. SHAPIRO, *Computability by probabilistic machines*, in Automata Studies, An. Math. Studies No. 34, Princeton University Press, Princeton, NJ, 1956, pp. 182–212.
- [Ma] STEPHEN R. MAHANEY, *Sparse complete sets for NP : solution of a conjecture by Berman and Hartmanis*, Proceedings of the 21st IEEE Symposium on the Foundations of Computer Science, Syracuse, New York, 1980, to appear.
- [Me] K. MEHLHORN, *On the size of sets of computable functions*, Proceedings of the 14th IEEE Symposium on Switching and Automata Theory, Iowa City, IO, 1973, pp. 190–196.
- [MM] A. R. MEYER AND E. M. MCCREIGHT, *Computability complex and pseudorandom zero-one valued functions*, in Theory of Machines and Computations, Z. Kohavi and Azaria Paz, eds., Academic Press, New York, 1971, pp. 19–42.
- [MS] A. MEYER AND L. STOCKMEYER, *The equivalence problem of regular expressions with squaring requires exponential time*, Proceedings of the 13th IEEE Symposium on Switching and Automata Theory, 1972, pp. 125–129.
- [Ra] C. RAČKOFF, *Relativized questions involving probabilistic algorithms*, Proceedings of 10th ACM Symposium on Theory of Computing, San Diego, CA, 1978, pp. 338–342.
- [Rb] M. O. RABIN, *Degree of Difficulty of Computing a Function and a Partial Ordering of Recursive Sets*, Tech. Rep. 2, Hebrew Univ., Jerusalem, Israel, 1960.
- [Ro] H. ROGERS, JR. *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967.
- [Si] J. SIMON, *On Some Central Problems in Computational Complexity*, Tech. Rep TR 75-224, Dept. of Computer Science, Cornell University, Ithaca, NY, 1975.

A NOTE ON THE COMPLEXITY OF GENERAL D0L MEMBERSHIP*

NEIL D. JONES† AND SVEN SKYUM‡

Abstract. In [Math. Systems Theory, 13 (1979), pp. 29–43], the authors obtained a number of upper and lower bounds for various problems concerning L -systems. In most cases the bounds were rather close; however, for general D0L membership the upper bound was \mathcal{P} , and the lower was deterministic log space. In this note we show that membership can be decided deterministically in $\log^2 n$ space, which makes it very unlikely that the problem is complete for \mathcal{P} . We also show that nonmembership is as hard as any problem solvable in nondeterministic $\log n$ space. This is an improved lower bound unless DSPACE $(\log n) =$ NSPACE $(\log n)$, which is also thought to be very unlikely.

Introduction. Let $G = (V, \delta, a)$ be a D0L system (e.g., Herman and Rozenberg [1]) so V is an alphabet, $a \in V$ a letter from V , and $\delta: V \rightarrow V^*$ a mapping. Extending δ to a homomorphism $\delta: V^* \rightarrow V^*$, we define

$$L(G) = \{\delta^r(a) \mid r = 0, 1, 2, \dots\},$$

where δ^r denotes the r -fold composition of δ with itself.

In Sudborough [4] it was shown that for each G , $L(G)$ is in DSPACE $(\log n)$ (our notation is from Jones and Skyum [3]). Each set $L(G)$ is a specific membership problem. The *general* membership problem (MEMBER^{D0L}) is: given a D0L system G and a word $v \in V^*$, to determine whether $v \in L(G)$. The problem was first addressed by Vitanyi [5]. In Jones and Skyum [3] it was shown that this problem is in \mathcal{P} , and cannot be solved deterministically in space less than $\log n$. The purpose of this note is to obtain the (apparently) tighter complexity bounds:

THEOREM. MEMBER^{D0L} is in DSPACE $(\log^2 n)$; and the nonmembership problem is hard for NSPACE $(\log n)$.

The lower bound is an improvement provided DSPACE $(\log n) \neq$ NSPACE $(\log n)$, which is widely believed. The new upper bound is not necessarily stronger; however, some of the best researchers in the field have tried and failed to show that $\mathcal{P} \subseteq$ DSPACE $(\log^2 n)$, which provides circumstantial evidence that MEMBER^{D0L} is not complete for \mathcal{P} . The new bounds are close, since DSPACE $(\log^2 n)$ is the smallest deterministic complexity class known to contain NSPACE $(\log n)$. It would seem difficult to show that MEMBER^{D0L} is complete for any class in this complexity range, since it seems quite unlikely that an NSPACE $(\log n)$ algorithm for nonmembership exists, and no natural complete problems are known for any complexity class containing NSPACE $(\log n)$ and contained in DSPACE $(\log^2 n)$.

Proof of the lower bound. Define

$$\text{AGAP} = \{\bar{\Gamma} \mid \Gamma \text{ is a digraph with node set } \{1, 2, \dots, n\} \text{ for some } n, \Gamma \text{ has a path from } 1 \text{ to } n, \text{ and } i < j \text{ for each arc } (i, j) \text{ of } \Gamma\}.$$

This problem was shown complete for NSPACE $(\log n)$ in Jones [2]. We now show how to construct from each digraph Γ a D0L system $G = (\{1, \dots, n\}, \delta, 1)$ such that $\lambda \notin L(G)$ if and only if $\bar{\Gamma}$ is in AGAP¹. Thus AGAP is reducible to nonmembership. Consequently MEMBER^{D0L} is in DSPACE $(\log n)$ only if DSPACE $(\log n) =$ NSPACE $(\log n)$.

* Received by the editors September 15, 1978, and in revised form November 13, 1979.

† Computer Science Department, University of Kansas, Lawrence, Kansas 66045. This research was partially supported by the National Science Foundation under Grant MCS76-80269.

‡ Computer Science Department, Aarhus University, DK-8000 Aarhus C, Denmark.

¹ λ is the empty string and $\bar{\Gamma}$ is a linear representation of Γ .

To build G , first add the single arc (n, n) to Γ , obtaining Γ' . Now define $\delta(i) = j_1 \cdots j_k$ to hold just in case $\{j_1, \dots, j_k\} = \{j \mid (i, j) \text{ is an arc of } \Gamma'\}$ and $j_1 < j_2 < \dots < j_k$. Clearly δ may be computed deterministically in logarithmic space.

Letting $\text{Alph}(w)$ equal the smallest alphabet $A \subseteq V$ such that $w \in A^*$, we see that for $r = 1, 2, \dots$, $\text{Alph}(\delta^r(1))$ is the set of nodes reachable from 1 by path of length exactly r . Now $\bar{\Gamma} \in \text{AGAP}$ if and only if Γ' has paths from 1 of arbitrary length if and only if $\text{Alph}(\delta^r(1)) \neq \emptyset$ for all r if and only if $\lambda \notin L(G)$.

Proof of the upper bound. For this we give an algorithm which operates in $\text{DSpace}(\log^2 n)$. Our algorithm is similar to that of Jones and Skyum [3] (which in turn is based on Vitanyi's algorithm [5]), but has several refinements to make it operate in $\log^2 n$ space. Our notation is essentially that of Vitanyi.

Define $b \in V$ to be *mortal* ($b \in M$) iff $\delta^s(b) = \lambda$ for some s , and *monorecursive* ($b \in MR$) iff $\delta^s(b) \in M^*bM^*$ for some $s > 0$. The *cycle* $\text{CYCLE}(b)$ of a monorecursive letter b is the least $s > 0$ such that $\delta^s(b) \in M^*bM^*$. Let p be the number of letters in V , and n the number of symbols required to write G and v .

It is easy to see that $L(G)$ is finite iff $\delta^p(a)$ contains only letters in $M \cup MR$, and that if $L(G)$ is infinite, then $v \in L(G)$ iff $v = \delta^r(a)$ for some $r \leq p|v|$. Our algorithm will have the form "if $L(G)$ infinite then test $v = \delta^r(a)$ for $r = 0, 1, \dots, p|v|$ else test $v \in L(G)$ by another method". Thus we first show that membership in M and MR , and " $v = \delta^r(a)$ " for $r \leq p|v|$ can be determined in $\text{DSpace}(\log^2 n)$.

Now define the function $\text{NUMBER}(b, c, s)$ for $b, c \in v$ and $s \geq 0$ as follows:

$$\text{NUMBER}(b, c, s) = \begin{cases} m & \text{if } \delta^s(b) \text{ contains } m \text{ occurrences of } c \text{ and } m \leq n, \\ \infty & \text{otherwise.} \end{cases}$$

Clearly $b \in M$ iff $\text{NUMBER}(b, c, p) = 0$ for all $c \in V$, and $b \in MR$ iff there is a $0 < i \leq p$ such that $\text{NUMBER}(b, b, i) = 1$ and if $\text{NUMBER}(b, c, i) > 0$ then $c \in M \cup \{b\}$.

Using $0 \cdot \infty = \infty \cdot 0 = 0$, NUMBER can be computed by the following deterministic algorithm, which clearly operates in space $\log^2(\max(n, s))$:

$$\begin{aligned} \text{NUMBER}(b, c, s) = & \\ & \text{if } s = 1 \text{ then the number of } c\text{'s in } \delta(b) \\ & \text{else } \sum_{d \in V} \text{NUMBER}\left(b, d, \frac{s}{2}\right) \cdot \text{NUMBER}\left(d, c, \frac{s}{2}\right). \end{aligned}$$

Thus we can compute M , MR , and $\text{CYCLE}(b)$ for all $b \in V$ in $\log^2 n$ space. Further, $L(G)$ is infinite iff $\text{NUMBER}(a, b, p) > 0$ for some $b \in M \cup MR$.

In order to test " $v = \delta^r(a)$," define the function, $\text{SYMBOL}(b, s, i)$ for $b \in V, s, i \geq 0$ and $\|w\|$ for $w \in V^*$ as follows:

$$\begin{aligned} \text{SYMBOL}(b, s, i) = & \begin{cases} \neq & \text{if } i = 0 \text{ or } i > \max(n, |\delta^s(b)|), \text{ else} \\ c & \text{if } c \text{ is the } i\text{th letter in } \delta^s(b), \end{cases} \\ \|w\| = & \begin{cases} |w| & \text{if } |w| \leq n, \\ \infty & \text{if not.} \end{cases} \end{aligned}$$

$\|\delta^s(b)\|$ can easily be computed in $\log^2(n)$ space using NUMBER if s is bounded by a polynomial in n .

Now suppose $c = \text{SYMBOL}(b, s, i)$ for some $i \leq n, s > 0$, and $\delta(b) = b_1 b_2 \cdots b_k$. Then $\delta^s(b) = \delta^{s-1}(b_1) \delta^{s-1}(b_2) \cdots \delta^{s-1}(b_k)$, so $c = \text{SYMBOL}(b, s-1, j)$ where

$$\sum_{m=1}^{r-1} \|\delta^{s-1}(b_m)\| < \sum_{m=1}^r \|\delta^{s-1}(b_m)\|$$

and

$$j = i - \sum_{m=1}^{r-1} \|\delta^{s-1}(b_m)\|.$$

Iterating, the path leading from a to $b = \text{SYMBOL}(a, s, i)$, $i \leq n$ in $\delta^s(a)$ may be traversed by the following algorithm:

```

b := a;
for  $h := s, s-1, \dots, 2$  do
  begin
    let  $\delta(b) = b_1 b_2 \dots b_k$ ;
    find  $r$  such that
      
$$\sum_{m=1}^{r-1} \|\delta^{h-1}(b_m)\| < i \leq \sum_{m=1}^r \|\delta^{h-1}(b_m)\|;$$

     $b := b_r$ ;
     $i := i - \sum_{m=1}^{r-1} \|\delta^{h-1}(b_m)\|$ 
  end

```

Using SYMBOL it is easy to test “ $v = \delta^r(a)$ for some $r \leq p \cdot |v|$ ” in DSPACE ($\log^2 n$), which finishes the test if $L(G)$ is infinite.

However, in case $L(G)$ is finite the smallest r such that $v = \delta^r(a)$ may be exponential in n . A different method is needed, and the key to this is the following observation, due to Vitanyi [5].

Observation. If $L(G)$ is finite, we can write $\delta^p(a) = v_1 a_1 v_2 a_2 \dots a_m v_{m+1}$, where each $a_i \in MR$ and $v_i \in M^*$. If $v = \delta^r(a)$ for some $r \geq 2p$, then there exist $\alpha_1, \dots, \alpha_m \in V^*$ such that

- a) $v = \alpha_1 \alpha_2 \dots \alpha_m$,
- b) for each $j = 1, 2, \dots, m$, there is an r_j such that $p \leq r_j < 2p$ and $\delta^{r_j}(a_j) = \alpha_j$,
- c) $r_j \equiv r_{j'} \pmod{\text{gcd}(\text{CYCLE}(a_j), \text{CYCLE}(a_{j'}))}$, for each pair j, j' with $1 \leq j' < j \leq m$.

Conversely, a), b) and c) together imply $v = \delta^r(a)$ for some r . In addition, $t \neq t' \pmod{\text{CYCLE}(a_j)}$ implies $\delta^t(a_j) \notin V^* \delta^{t'}(a_j) V^*$; thus, α_j if the *only* prefix of $\alpha_j \dots \alpha_m$ which is derivable from a_j .

The algorithm testing a , b , and c uses a procedure FIND (i, q, k, r), $0 \leq i, q, k, t \leq n$:

```

procedure FIND ( $i, q, k, r$ ); comment let  $v = a_1 a_2 \dots a_{|v|}$ ;
begin
   $b :=$  “the  $i$ th monorecursive letter in  $\delta^p(a)$ ” if it exists,
    otherwise reject;
   $k := \text{CYCLE}(b)$ ;
   $r :=$  “the smallest  $p \leq r < 2p$  such that  $\delta^r(b)$  is a prefix
    of  $a_{q+1} a_{q+2} \dots a_{|v|}$ ” if it exists, otherwise reject;
   $q := q + |\delta^r(b)|$ ; reject if  $q > |v|$ ;
end

```

Before giving the complete algorithm we will see that FIND can be performed in $\log^2 n$ space.

First, to find the i th monorecursive letter in $\delta^p(a)$ in $\log^2 n$ space, we can simply modify NUMBER and SYMBOL to give the number of nonmortal letters or the j th nonmortal symbol. Note that $|\delta^p(a)|$ may be exponential in n . r can be found by computing $\delta^t(b)$, for $t = p, p+1, \dots, 2p-1$, one letter at a time, and comparing it with v .

The final algorithm for MEMBER^{D0L} verifies condition a) and b) of the observation by calling FIND for $i = 1, 2, \dots, m$, and verifies condition c) by calling FIND for $i' = 1, 2, \dots, i-1$ in an inner loop for each value of i . The input is a D0L system $G = (V, \delta, a)$ and a word $v \in V^*$.

```

begin
  if  $\delta^p(a) \notin (M \cup MR)^*$  comment  $L(G)$  is infinite;
  then accept if  $v = \delta^r(a)$  for some  $r \leq p \cdot |v|$  and reject if not;
  else
    begin comment  $L(G)$  is finite;
      accept if  $v = \delta^r(a)$  for some  $r \leq 2p$ ;
       $q := 0$ ;
       $m :=$  "the number of monorecursive letters in  $v$ ";
      [if  $v = \delta^r(a)$  for some  $r > 2p$  then this equals the
       number of monorecursive letters in  $\delta^p(a)$ ]
      for  $i := 1, 2, \dots, m$  do
        begin
          FIND ( $i, q, k, r$ );
           $q' := 0$ ;
          for  $i' := 1, 2, \dots, i-1$  do
            begin
              FIND ( $i', q', k', r'$ );
              reject if  $r \not\equiv r' \pmod{\text{GCD}(k, k')}$ 
            end
          end;
          if  $q = |v|$  then accept else reject;
        end
      end
    end
  end

```

There should be no difficulty in seeing that the algorithm operates in $\log^2 n$ space.

REFERENCES

- [1] G. HERMAN AND G. ROZENBERG, *Developmental Systems and Languages*, North-Holland, Amsterdam, 1975.
- [2] N. D. JONES, *Space-bounded reducibility among combinatorial problems*, J. Comput. System Sci., 11 (1975), pp. 68–75.
- [3] N. D. JONES AND S. SKYUM, *Complexity of some problems concerning L systems*, Math. Systems Theory, 13 (1979), pp. 29–43.
- [4] I. H. SUDBOROUGH, *The time and space complexity of developmental languages*, Automata, Languages and Programming, Lecture Notes in Computer Science, v. 52, G. Goos, J. Hartmanis, eds., Springer-Verlag, New York, 1979, pp. 509–523.
- [5] P. M. B. VITANYI, *On the size of D0L languages*, in G. Rozenberg, A. Salomaa, eds., *L Systems*, Lecture Notes in Computer Science, v. 15, G. Goos, J. Hartmanis eds., Springer-Verlag, New York, 1974, pp. 78–92.

ON STRING PATTERN MATCHING: A NEW MODEL WITH A POLYNOMIAL TIME ALGORITHM*

KEN-CHIH LIU†

Abstract. A polynomial time algorithm is presented for string pattern matching. Earley's parsing algorithm is adapted for context-free patterns and is extended to allow the augmentation of the immediate assignment operation of SNOBOL4 and a powerful descriptive operator not previously implemented, set complement. Canonical pattern definition systems are defined to describe patterns for which our algorithm will perform pattern matching. The languages generated by such systems are called extended context-free languages, and are shown to properly contain the family of context-free languages and all families of k -intersection languages which have been shown to establish an infinite hierarchy between the family of context-free languages and the family of context-sensitive languages. It is also shown that for an alphabet of one character, the immediate assignment operator cannot be expressed in terms of the complement operator. Some results on the closure properties and unsolvable problems for this new family are also shown. It is shown that the worst-case time and space complexity of our algorithm is polynomial if the immediate assignment is used, and that the time (space) complexity is cubic (cubic) if the complement but not the immediate assignment operator is used, and is cubic (square) if neither operator is used. Our algorithm has no difficulty with left-recursion or null string alternatives which are problems with the SNOBOL4 pattern matching algorithm.

Key words. SNOBOL4 patterns, pattern matching, pattern matching algorithms, context-free patterns, context-free grammars, context-free parsing algorithm, formal languages, complexity

1. Introduction. String pattern matching is fundamentally a process of examining a subject string for a substring which is one of set of strings specified by a pattern. It provides a powerful facility for analyzing and manipulating strings of characters [8]. Especially, it plays a vital role in the programming language SNOBOL4 [11], which is currently strongly predominant among string-manipulation languages and has many different implementations. However, the inefficiency of the SNOBOL4 pattern matching process has been widely recognized [2], [14], [16]. For example, Liu [14] shows that for a pattern such as $P = 'B' | 'A' * P' c' | 'A' * P' D'$, the SNOBOL4 pattern matching algorithm takes at least exponential time. But the algorithm presented in this paper will only take polynomial time for this pattern.

Also the SNOBOL4 patterns have been quoted as notoriously difficult to explain and use [16], [18]. Each of these areas of difficulty relates to such things as two modes of operation (quick/full scan), problems with left recursion, heuristics in the scan, etc. Some difficulties are inherent with string patterns, but many result from the actions taken by the pattern matching algorithm which are used to define the meaning of patterns.

We take an alternative approach emphasizing that the meaning of patterns should be independent of whatever matching algorithms might be employed. Recently, Gimple [7] and Stewart [18] have discussed formal models for certain classes of string patterns. Their models were directed at the pattern matching process, particularly as it is affected by the combination of patterns under certain operations. These models do not seem well suited to extension to the operations which we consider here. Based on the analogy between context-free grammars and SNOBOL4 patterns [6], we use the formal language approach and for the operation of interest we have developed a pattern definition system together with a polynomial time pattern matching algorithm which is

* Received by the editors June 18, 1978, and in final revised form February 21, 1980.

† Sperry Univac, Roseville, Minnesota 55133. A talk based on an abbreviated version of this paper was presented at the Sixth Annual ACM Symposium on Principles of Programming Languages. This work was supported by National Science Foundation grant DCR-75-05296, carried out at the University of Iowa, Iowa City and revised at Sperry Univac.

derived from Earley's parsing algorithm [3]. The patterns to which the algorithm applies include a wide subclass of the SNOBOL4 patterns and moreover permit a powerful descriptive operator not previously implemented, set complement.

We describe the consideration of extending context-free patterns in § 2. Section 3 presents the formal model upon which we base this work. Some formal language properties of this new model is then discussed in § 4. In § 5, the informal description of the new algorithm together with illustrative examples and the formal description are given. A proof of correctness for this algorithm is sketched in § 6. Section 7 shows the polynomial time and space complexity of the algorithm.

2. Context-free pattern and its extension. Fleck [6] defines a *context-free pattern* as any SNOBOL4 pattern which can be constructed from only (1) alternation, concatenation, unevaluated expression and assignment operators, (2) the primitive pattern NULL, and (3) string constant and simple variables. Then he shows that for each context-free pattern there is an isomorphic context-free grammar (and vice versa) which defines exactly the same collection of strings.

Since the context-free languages are not closed under two set operations, complementation and intersection, we consider the augmentation of context-free patterns by these two operators for the purpose of the substantial increase in the expressive power.

Example 1. Let $\Sigma = \{A, B, C\}$ be the alphabet we are concerned with.

(a) Context-free patterns E and F ;

$$E = *E'C' \mid *Q'C',$$

$$Q = 'AB' \mid 'A'*Q'B',$$

$$F = 'A'*F \mid 'A'*T,$$

$$T = 'BC' \mid 'B'*T'C'.$$

Patterns E and F represent the context-free languages

$$L(E) = \{A^n B^n C^m \mid m, n \geq 1\}$$

and

$$L(F) = \{A^m B^n C^n \mid m, n \geq 1\}$$

respectively.

(b) Extended context-free pattern M augmented by the intersection operator. The set notation \cap is used here for this operator.

$$M = *E \cap *F.$$

Its informal interpretation is that a string matches M if and only if it matches both pattern E and pattern F . Pattern M represents $L(M) = L(E) \cap L(F) = \{A^n B^n C^n \mid n \geq 1\}$, which is not context-free.

(c) Extended context-free patterns P and R with the augmentation of the complement operator, \neg .

$$P = \neg(*R).$$

Informally, this means that a string matches P if and only if it does not match pattern R .

$$R = \neg(*E) \mid \neg(*F).$$

The informal interpretation is that a string matches R if and only if it does not match pattern E or it does not match pattern F .

Patterns $\neg E$ and $\neg F$ represent the languages $L(\neg E) = \Sigma^* - L(E)$ and $L(\neg F) = \Sigma^* - L(F)$, respectively. Pattern R represents $L(R) = L(\neg E) \cup L(\neg F)$ and pattern P represents the language

$$\begin{aligned}
 L(P) &= L(\neg R) \\
 &= \Sigma^* - L(R) \\
 &= \Sigma^* - ((\Sigma^* - L(E)) \cup (\Sigma^* - L(F))) \\
 &= L(E) \cap L(F) \\
 &\quad \text{(By De Morgan's Law)} \\
 &= \{A^n B^n C^n \mid n \geq 1\},
 \end{aligned}$$

which is not context-free.

The immediate value assignment is signified by the binary operator $\$$ in SNOBOL4 [11]. A careful definition of the semantics of this operation is difficult and will be presented in the next section. Informally, $P \$ I$ causes the substring matched by the pattern P to be assigned immediately to the variable I . This new value may be referenced later by mention of the variable. The augmentation of this operation allows the definition of sets which are not context-free.

Example 2. Extended context-free pattern T augmented by the immediate assignment operator, $\$$.

$$T = 'A' | (*T \$ I) *I.$$

Pattern T represents the language $L(T) = \{A^{2^n} \mid n \geq 0\}$ which is not context-free.

3. Canonical pattern definition systems. We now define canonical pattern definition systems which are an extension of the context-free patterns. Informally, in SNOBOL4 terms, we consider patterns which can be defined by means of the following operations: assignment, alternation, concatenation, immediate value assignment, and complementation (written as \neg). Notice that this set operator, complement, is quite different from the not operator in some implementations (e.g., [12]). The set operator, union, is implicitly included in these systems. Also the set operator, intersection, can be expressed by these systems using two levels of complement operator as previously described.

DEFINITION 1. Let C be a finite nonempty set of characters, PV a finite nonempty set of pattern variables, IAV a finite ordered set of immediate assignment variables, $IALM$ a finite ordered set of immediate assignment left markers, $IARM$ a finite ordered set of immediate assignment right markers, and \neg a symbol for complementation. Let

$$T = C \cup PV \cup IAV \cup IALM \cup IARM \cup \{\neg\}$$

and

$$Z = C \cup PV \cup IAV.$$

Assume that $|IAV| = |IALM| = |IARM|$, and that $C, PV, IAV, IALM, IARM$ and $\{\neg\}$

are mutually disjoint. A word H over T is a *canonical pattern expression* over Z if

- (1) H is one of the following primitive pattern elements:
 - (a) θ (null string)
 - (b) $u \in C^+$
 - (c) $R \in PV$
 - (d) $I \in IAV$
 - (e) $\neg R$, where $R \in PV$,

or

- (2) H is one of the following forms:
 - (a) uv (concatenation)
 - (b) $(i,u)_i$ (immediate assignment to $I_i \in IAV$),

where u and v are canonical pattern expressions over Z , $(i \in IALM$, and $)_i \in IARM$.

A canonical pattern definition system (CPDS) is a 8-tuple, $G = (C, PV, IAV, IALM, IARM, P, Q, IV)$, where

- (1) $Q \in PV$, the “start” variable,
- (2) P is a finite set of production rules of the form $R \rightarrow H$ with $R \in PV$, and H a canonical pattern expression over Z , and
- (3) IV is a finite ordered set of initial values (from C^*) of the corresponding immediate assignment variables in the ordered set IAV ; most often we will take IV to consist of a *single* string and then write $IV = x_{-|IV|+1} \cdots x_{-1}x_0$, where $|IV|$ denotes the length of that string and each $x_i \in C$, $-|IV|+1 \leq i \leq 0$, and write $IV = \theta$ if $|IV| = 0$. If $IAV = IALM = IARM = \emptyset$ then for most instances we still write $IV = \theta$ to indicate that IV is an empty set.

Note that in SNOBOL4 terms the unevaluated expression operator would be required for those pattern variables on the right side of production rules.

DEFINITION 2. A *labelled directed graph* is a triple $X = (Y, F, B)$, where Y is a finite, nonempty set of *vertices*, F a set of *edges* which are ordered triples of the form (v, w, b) with $v, w \in Y$ and $b \in B^*$, and B is an alphabet, and nonnull strings over B are used as *labels*. We say that the edges of the form (v, w, θ) are not *labelled*, and those of the form (v, w, b) with $b \neq \theta$ are *labelled* by b . A *path* in a labelled directed graph is a sequence of edges of the form $(v_1, v_2, b_1), (v_2, v_3, b_2), \dots, (v_{n-1}, v_n, b_{n-1})$. We say that the path is *from* v_1 *to* v_n and is of length $n - 1$. We also say that the path is *simple* if all v_i 's on the path, except possible v_1 and v_n , are distinct. A *cycle* is a simple path of length at least 1 which begins and ends at the same vertex.

DEFINITION 3. Given a CPDS, $G = (C, PV, IAV, IALM, IARM, P, Q, IV)$, we define the *characteristic graph* of G as a labelled directed graph $X = (PV, F, \{\neg\})$, where PV is the set of vertices and F the set of possible labelled edges, and if $U, R \in PV$, then

- (1) the ordered triple $(U, R, \theta) \in F$ (is not labelled) if $\exists U \rightarrow uRv \in P$, for some $u, v \in T^*$,
- (2) the ordered triple $(U, R, \neg) \in F$ (is labelled by \neg) if $\exists U \rightarrow u \neg Rv \in P$, for some $u, v \in T^*$. The graph $X = (PV, F, \{\neg\})$ is called \neg -acyclic if each cycle in X contains no edges labelled by \neg .

We consider the following two *basic assumptions* for the CPDS's we are going to deal with.

BA1. $IAV = \{I\}$, $IALM = \{\}$, and $IARM = \{\}$; i.e., $|IAV| = |IALM| = |IARM| = k$, where $k = 1$. This is only to ease the notations used in definitions and illustrations, and the proof of the correctness of the algorithm. But the algorithm can be slightly modified to work correctly for any positive value of k .

BA2. The characteristic graph of the CPDS is \neg -acyclic. This limits the scope of CPDS's which can be handled by the algorithm (but otherwise meaningful definitions are elusive).

The reason for making this assumption BA2 is illustrated by the problems raised in the following two examples in which the recursion in the pattern definitions involves complement (i.e., there exists a cycle which contains an edge labelled by \neg).

Firstly, a usual definition scheme for set solution of a system of equations is summarized from Fleck [6]. After defining the pattern expressions and the languages represented, he defines a pattern expression system over a character set C and a set of pattern variables $PV = \{R_1, \dots, R_m\}$ as a collection of assignments

$$R_i = H_i, \quad 1 \leq i \leq m,$$

where each H_i , $1 \leq i \leq m$, is a pattern expression over C and PV . Then he defines $L(R_i) = L(H_i)$, $1 \leq i \leq m$, and $L(R_1), \dots, L(R_m)$ to be the collection of smallest sets which satisfy this system of equations.

Example 3. Given a CPDS $G = \{C, PV, IAV, IALM, IARM, P, Q, IV = \theta\}$, where

$$\begin{aligned} PV &= \{Q\}, \\ IVA &= IALM = IARM = \emptyset \\ P &= \{Q \rightarrow A, Q \rightarrow \neg QA\}. \end{aligned}$$

The characteristic graph of G has a \neg -cycle and is given as follows.



Using the definition scheme similar to that used in Fleck [6] which we just summarized for the languages represented by patterns, we discuss this example by the following two cases:

Case 1. If the alphabet $C = \{A\}$, then $L(Q) = \{A^{2n+1} | n \geq 0\}$. Notice that $L(Q)$ is the unique smallest set which satisfies the following equation

$$(\$ \$) \quad X = \{A\} \cup (\neg X)\{A\}.$$

We sketch the proof of this statement as follows. We note without proof that $L(Q)$ is a solution to $(\$ \$)$. Let S be a proper subset of $L(Q)$. Then there exists a nonnegative integer m such that $A^{2m+1} \in L(Q) - S$. Now $A^{2m+2} = A^{2m+1}A \in (\neg S)\{A\}$ implies that $A^{2m+2} \in \{A\} \cup (\neg S)\{A\}$. But $A^{2m+2} \notin L(Q)$ implies that $A^{2m+2} \notin S$. Therefore, S does not satisfy $(\$ \$)$. Namely, $L(Q)$ is a smallest set which satisfies $(\$ \$)$. We claim that $L(Q)$ is a subset of any set which satisfies $(\$ \$)$. Let T be a set which satisfies $(\$ \$)$. Clearly, $A \in T$ and $A^2 = AA \notin T$. By induction on n , we can easily prove that $A^{2n} \notin T$ and $A^{2n+1} \in T$ for any nonnegative integers n . Therefore, $L(Q)$ is a subset of T . This completes the sketch of the proof.

Case 2. If the alphabet $C = \{A, B\}$, then the above equation does *not* have a *unique* solution. Both X_1 and X_2 are smallest sets which satisfy the equation, where

$$X_1 = \{A^{2n+1} | n \geq 0\} \cup \{wBA^{2n} | w \in C^*, n \geq 1\},$$

and

$$X_2 = \{A^{2n+1} | n \geq 0\} \cup \{wBA^{2n+1} | w \in C^*, n \geq 1\}.$$

Example 4. Given a CPDS $G = \{C, PV, IAV, IALM, IARM, P, Q, IV = \theta\}$ where

$$\begin{aligned} PV &= \{Q\}, & IAV &= IALM = IARM = \emptyset, \\ P &= \{Q \rightarrow \neg Q\}, & C &= \text{any set of characters.} \end{aligned}$$

The characteristic graph of G has a \neg -cycle is given as follows.



There does not exist any (smallest) set which satisfies the equation $X = \neg X$.

DEFINITION 4. For a characteristic graph $X = (Y, F, \{\neg\})$ of a given CPDS, we associate with each $R \in PV$ a *weight* $W(R)$ which is defined as follows.

Let $R \in PV$.

- (1) $p(R)$ denotes a noncyclic simple path p starting from R ;
- (2) $W(p(R))$ denotes the weight of the noncyclic simple path $p(R)$ and is defined as the number of labelled edges along $p(R)$;
- (3) $S(R)$ denotes the set of all noncyclic simple paths starting from R (which is clearly finite);
- (4) $W(R)$ denotes the weight of $R \in PV$ and is defined as $W(R) = \max \{W(p(R)) | p(R) \in S(R)\}$.

For those CPDS's which satisfy our basic assumptions, we shall define the languages they generate. To do so, we need the definitions of extended "direct derivation" and "derivation" which are defined below for systems with one immediate assignment variable. For the general case that there is more than one immediate assignment variable, the definitions are similar to those given.

DEFINITION 5. Let $G = (C, PV, IAV = \{I\}, IALM = \{\{\}, IARM = \{\}\}, P, Q, IV)$ be a CPDS. Let $T = C \cup PV \cup \{I, (, \neg\}$, and let SK = the set of stacks of depth up to D (assuming that is the maximum nesting level occurring for the immediate assignment markers), elements in which are nonnegative integers.

For each i , $0 \leq i \leq W(Q)$, the relation \mapsto_i called *extended direct derivation at the level i* , over $T^* \times SK \times C^*$ is defined as follows: Let $x, y, z \in C^*$, $A \in PV$, $w, v \in T^*$, $K \in SK$. Then

- D1. $\langle xAv, K, y \rangle \mapsto_i \langle xwv, K, y \rangle$ if $\exists (A \rightarrow w) \in P$;
- D2. $\langle x(v, K, y) \mapsto_i \langle xv, K \downarrow l, y \rangle$ where $l = |x|$;
- D3. $\langle x \rangle v, K \downarrow l, y \rangle \mapsto_i \langle xv, K, z \rangle$ where

$$z = \begin{cases} \text{the null string} & \text{if } l = |x|, \\ a_{l+1}a_{l+2} \cdots a_{|x|} & \text{if } x = a_1a_2 \cdots a_{|x|}; \end{cases}$$

- D4. $\langle xIv, K, y \rangle \mapsto_i \langle xyv, K, y \rangle$;

D5. $\langle x \neg Av, K, y \rangle \mapsto_i \langle xwv, K, y \rangle$ if $\exists \langle xA, K, y \rangle \xrightarrow_{i-1}^* \langle xw, K, z \rangle$ for all z , where \xrightarrow_i^* is defined below.

DEFINITION 6. For each i , $0 \leq i \leq W(Q)$, the relation \xrightarrow_i^* , called *extended derivation at the level i* , over $T^* \times SK \times C^*$ is defined as follows. Let $B, B', B'' \in T^* \times SK \times C^*$. $B \xrightarrow_i^* B'$ if and only if

- (1) $B = B'$, or
- (2) $\exists B''$ such that $B \xrightarrow_i^* B''$ and $B'' \mapsto_i B'$.

In the above definitions, $\langle Q, \theta, IV \rangle$ will be the *start symbol* of the extended derivations at the level $W(Q)$. Extended sentential forms at the level i , $0 \leq i \leq W(Q)$, are of the form $\langle v, K, y \rangle$ where v is a (conventional) sentential form, i.e., a string of

grammar symbols. K is a stack of nonnegative integers, separated by “↓”, defining the positions of previously-produced “left markers” in v , and y represents the instantaneous “contents” of the immediate assignment variable I . If $W(Q) = 0$, then only rules D1 through D4 may be applied in any extended derivation at the level 0. If $W(Q) > 0$, then D5 may also be applied in an extended direct derivation at the level $W(Q)$. In this case, a complemented pattern variable derives a substring at the level $W(Q)$ with the condition that, at the level $W(Q) - 1$, the pattern variable cannot derive that substring. And the pattern variable will be used in the first component of the new start symbol for the extended derivations at the level $W(Q) - 1$. In general, for $0 < i \leq W(Q)$ if D5 is applied at the level i , then it depends upon extended derivations at the level $i - 1$. In other words, rule D5 may be applied $W(Q)$ times to determine an extended direct derivation at the level $W(Q)$. In Definition 5, it can also be the case that $IAV = IALM = IARM = \emptyset$, namely, no immediate assignment. This simply indicates that rules D2, D3 and D4 may not be applied.

DEFINITION 7. If $G = (C, PV, IAV, IALM, IARM, P, Q, IV)$ is a CPDS then the subset of C^*

$$L(G) = \{x \in C^* \mid \langle Q, \theta, IV \rangle \xrightarrow{i}^* \langle x, \theta, y \rangle \text{ for some } y \in C^* \text{ and } i = W(Q)\}.$$

is called an *extended context-free (ecf) language*. $L(G)$ is said to be *generated by the CPDS*, or *represented by the pattern variable Q* . A language L is called *extended context-free* if there exists a CPDS G such that $L = L(G)$. $L(G)$ consists of all strings which match Q in their entirety (in SNOBOL4 terms, pattern matching would succeed for $POS(0) Q RPOS(0)$).

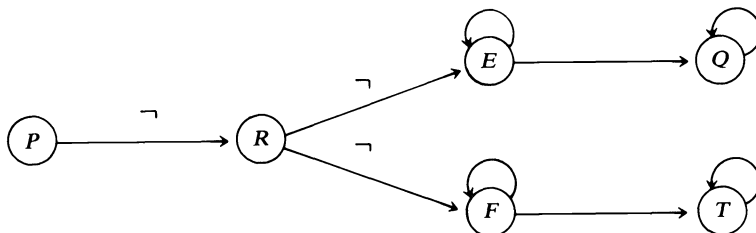
Note that CPDS's can actually generate exactly the context-free languages if $IAV = IALM = IARM = \emptyset$ and the \neg symbol does not appear on the right-hand side of any production rules. Namely, a context-free language is extended context-free.

Let us look at the examples given in § 2, and see how the SNOBOL4 or SNOBOL4-like definitions for patterns are represented by CPDS's. Also the applications of direct derivation rules are illustrated in the following examples.

Example 5. In Example 1, we have defined the patterns M and P . Now we define a CPDS G such that $L(G) = L(M) = L(P)$. Let $G = (CC, PV, IAV, IALM, IARM, PP, P, IV)$ where $CC = \{A, B, C\}$, $PV = \{E, F, P, R, Q, T\}$, $IAV = IALM = IARM = \emptyset$, $IV = \theta$, and

$$\begin{aligned} PP = \{ & E \rightarrow EC, E \rightarrow QC, Q \rightarrow AB, Q \rightarrow AQB, \\ & F \rightarrow AF, F \rightarrow AT, T \rightarrow BC, T \rightarrow BTC, \\ & R \rightarrow \neg E, R \rightarrow \neg F, P \rightarrow \neg R \}. \end{aligned}$$

The characteristic graph of G is shown below. Note that $W(P) = 2$, $W(R) = 1$, $W(E) = W(F) = W(Q) = W(T) = 0$.



A derivation is shown below to illustrate that $ABC \in L(G)$.

$$\begin{aligned} \langle P, \theta, \theta \rangle &\xrightarrow{2} \langle \neg R, \theta, \theta \rangle && \text{(D1 is applied and } (P \rightarrow \neg R) \in PP \text{ is used)} \\ &\xrightarrow{2} \langle ABC, \theta, \theta \rangle && \text{(D5 is applied),} \end{aligned}$$

since $\exists \langle R, \theta, \theta \rangle \xrightarrow{1}^* \langle ABC, \theta, z \rangle$ for all z .

This results from the following two cases:

(1)

$$\langle R, \theta, \theta \rangle \xrightarrow{1} \langle \neg E, \theta, \theta \rangle \quad \text{(D1 is applied and } (R \rightarrow \neg E) \in PP \text{ is used).}$$

$$\begin{aligned} \text{But } \langle E, \theta, \theta \rangle &\xrightarrow{0} \langle QC, \theta, \theta \rangle && \text{(D1 is applied and } (E \rightarrow QC) \in PP \text{ is used)} \\ &\xrightarrow{0} \langle ABC, \theta, \theta \rangle && \text{(D1 is applied and } (Q \rightarrow AB) \in PP \text{ is used).} \end{aligned}$$

This implies that $\exists \langle \neg E, \theta, \theta \rangle \xrightarrow{1} \langle ABC, \theta, \theta \rangle$. Also it is clear that $\exists \langle \neg E, \theta, \theta \rangle \xrightarrow{1} \langle ABC, \theta, z \rangle$ for any z .

(2)

$$\langle R, \theta, \theta \rangle \xrightarrow{1} \langle \neg F, \theta, \theta \rangle \quad \text{(D1 is applied and } (R \rightarrow \neg F) \in PP \text{ is used).}$$

$$\begin{aligned} \text{Now } \langle F, \theta, \theta \rangle &\xrightarrow{0} \langle AF, \theta, \theta \rangle && \text{(D1 is applied and } (F \rightarrow AF) \in PP \text{ is used)} \\ &\xrightarrow{0} \langle ABC, \theta, \theta \rangle && \text{(D1 is applied and } (F \rightarrow BC) \in PP \text{ is used).} \end{aligned}$$

This implies that $\exists \langle \neg F, \theta, \theta \rangle \xrightarrow{1} \langle ABC, \theta, \theta \rangle$. And it is clear that $\exists \langle \neg F, \theta, \theta \rangle \xrightarrow{1} \langle ABC, \theta, z \rangle$ for any z .

Since there are only two production rules in PP with R on the left-hand side, we conclude that $\exists \langle R, \theta, \theta \rangle \xrightarrow{1}^* \langle ABC, \theta, z \rangle$ for all z . Therefore, $\langle P, \theta, \theta \rangle \xrightarrow{2}^* \langle ABC, \theta, \theta \rangle$; namely, $ABC \in L(G)$. Note that $L(G) = \{A^n B^n C^n \mid n \geq 0\}$ is not context-free.

Example 6. Let us consider the pattern T defined in Example 2. A CPDS G is defined as follows such that $L(G) = L(T)$. Let $G = (C, PV, IAV, IALM, IARM, P, T, IV)$ where $C = \{A\}$, $PV = \{T\}$, $IAV = \{I\}$, $IALM = \{\{\}, IARM = \{\}\}$, $IV = \theta$ and $P = \{T \rightarrow A, T \rightarrow (T)I\}$. The characteristic graph of G is shown below. Note that $W(T) = 0$.



A derivation is given below to illustrate that $AAAA \in L(G)$.

$\langle T, \theta, \theta \rangle$	(The initial value of I is $IV = \theta$)
$\xrightarrow{\circ} \langle (T)I, \theta, \theta \rangle$	(D1 is applied and $(T \rightarrow (T)I) \in P$ is used.)
$\xrightarrow{\circ} \langle T)I, \downarrow 0, \theta \rangle$	(D2 is applied.)
$\xrightarrow{\circ} \langle ((T)I)I, \downarrow 0, \theta \rangle$	(D1 is applied and $(T \rightarrow (T)I) \in P$ is used.)
$\xrightarrow{\circ} \langle T)I)I, \downarrow 0 \downarrow 0, \theta \rangle$	(D2 is applied.)
$\xrightarrow{\circ} \langle A)I)I, \downarrow 0 \downarrow 0, \theta \rangle$	(D1 is applied and $(T \rightarrow A) \in P$ is used.)
$\xrightarrow{\circ} \langle AI)I, \downarrow 0, A \rangle$	(D3 is applied and the new value of I is A .)
$\xrightarrow{\circ} \langle AA)I, \downarrow 0, A \rangle$	(D4 is applied.)
$\xrightarrow{\circ} \langle AAI, \theta, AA \rangle$	(D3 is applied and the new value of I is AA .)
$\xrightarrow{\circ} \langle AAAA, \theta, AA \rangle$	(D4 is applied.)

Therefore, $\langle T, \theta, \theta \rangle \xrightarrow{\circ}^* \langle AAAA, \theta, AA \rangle$. Namely, $AAAA \in L(G)$. Note that $L(G) = \{A^{2^n} \mid n \geq 0\}$ is not context-free.

4. Formal language properties. We shall, in this section, discuss the formal language properties of this new family of extended context-free languages. The properties which have been learned previously are first summarized as follows:

(1) The family of context-free languages is contained in the family of extended context-free languages, since we have shown, in the last section, that the canonical pattern definition systems can actually generate all context-free languages if the symbol \neg , the complement operator, is not used and if $IAV = IALM = IARM = \emptyset$.

(2) The canonical pattern definition systems do generate languages which are not context-free (see Example 5 and Example 6).

(3) The family of extended context-free languages contains all families of k -intersection languages which Liu and Weiner [15] showed establish an infinite hierarchy between the family of context-free languages and the family of context-sensitive languages, since, as will be proved in Theorem 2, the family of extended context-free languages is closed under intersection. Note that a k -intersection language is a language which is expressible as an intersection of k context-free languages.

Now we might ask the question: Are the immediate assignment operator and the complement operator independent? Although the answer seems to be positive intuitively, we only obtain a partial result on this question. We shall show that, for an alphabet of one character, the immediate assignment operator cannot be expressed in terms of the complement operator. Before doing so, we define the following terms.

DEFINITION 8. An IA-ecf language (for Immediate Assignment) is a language which is generated by a CPDS in which the \neg symbol does not appear in any of the production rules. A complement-ecf language is a language which is generated by a CPDS in which there exists no immediate assignment variables.

THEOREM 1. A complement-ecf language over an alphabet of one character is regular.

Proof. Let L be a complement-ecf language over an alphabet C of one character. Let $G = (C, PV, IAV, IALM, IARM, P, Q, IV)$ be a CPDS which generates L , where $IAV = IALM = IARM = IV = \emptyset$. For each $k \leq W(Q)$, let $G_R^k = (C, PV', \emptyset, \emptyset, \emptyset, P', R, \emptyset)$ where $R \in PV'$, $W(R) = k$, PV' consists of pattern variables in PV with weight

$\leq k$, and P' consists of production rules in P such that the pattern variables on their left-hand side are of weight $\leq k$. We claim that $L(G_R^k)$ is regular for each $k \leq W(Q)$. We shall prove this claim by induction on k . If $k = 0$, then P' consists of only context-free production rules. Therefore, $L(G_R^0)$ is context-free over C and is regular since any context-free language over one character is regular. Now suppose that $L(G_R^i)$ is regular for all $i \leq m$.

To prove that $L(G_R^{m+1})$ is regular, we consider a CPDS

$$G' = (C \cup S, PV', \emptyset, \emptyset, \emptyset, P'', R, \emptyset),$$

where

(1) $S = \{b_T \mid \exists \neg T \text{ on the right-hand side of some production rules, and } W(T) \leq m\}$ is a set of new abstract symbols,

(2) P'' consist of all production rules of P with each occurrence of $\neg T$ with $W(T) \leq m$ replaced by b_T .

It is clear that $L(G')$ is a context-free language over $C \cup S$. Let f be a substitution such that $f(w) = \{w\}$, for all $w \in C$, and $f(b_T) = C^* - L(G_T^{W(T)})$, for all $b_T \in S$. Clearly, $f(L(G')) = L(G_R^{m+1})$. By the induction hypothesis, $L(G_T^{W(T)})$ is regular. Since regular sets are closed under complement, $f(b_T) = C^* - L(G_T^{W(T)})$ is regular and is therefore context-free for all $b_T \in S$. Since context-free languages are closed under substitution, $L(G_R^{m+1}) = f(L(G'))$ is context-free. Thus $L(G_R^{m+1})$ is regular, since it is a context-free language over C of one character. The claim is therefore proved. Now in particular, when $R = Q$ and $k = W(Q)$, $L(G_Q^{W(Q)})$ is regular. Since it is clear that $G = G_Q^{W(Q)}$, $L(G) = L(G_Q^{W(Q)})$ is regular. Q.E.D.

COROLLARY 1.1. *There exists an IA-ecf language which is not a complement-ecf language.*

Proof. As illustrated in Example 6, $L = \{A^{2^n} \mid n \geq 0\}$ is an IA-ecf language. But $\{A^{2^n} \mid n \geq 0\}$ is not regular. Hence L is not a complement-ecf language. Q.E.D.

Now we shall study the closure properties of this family of extended context-free languages under various operations. For definitions of the language operations such as union, complement, intersection, concatenation, closure, substitution, homomorphism, the reader can consult one of the standard references in the area of formal languages such as Hopcroft and Ullman [13] and Salomaa [17].

THEOREM 2. *The family of extended context-free languages is closed under the following operations: (1) union, (2) concatenation, (3) closure, (4) complement, and (5) intersection.*

Proof. We omit the proofs for (1), (2) and (3), since they are similar to the proof for the family of context-free languages (see [17]). To prove (4) and (5), we assume that $L(G)$ and $L(G')$ are generated by the canonical pattern definition systems $G = (C, PV, IAV, IALM, IARM, P, Q, IV)$ and $G' = (C', PV', IAV', IALM', IARM', P', Q', IV')$, respectively. We may assume that $C, C', PV, PV', IAV, IAV', IALM, IALM', IARM, IARM'$ and $\{\neg\}$ are mutually disjoint (since we can always rename elements in certain sets) except that C and C' may not be disjoint.

The language $C^* - L(G)$ is generated by the CPDS $G_1 = (C, PV \cup \{R\}, IAV, IALM, IARM, P \cup \{R \rightarrow \neg Q\}, R, IV)$, where R is a new pattern variable.

The language $L(G) \cap L(G')$ is generated by the CPDS $G_2 = (C \cup C', PV \cup PV' \cup \{T, S\}, IAV \cup IAV', IALM \cup IALM', IARM \cup IARM', P \cup P' \cup \{T \rightarrow \neg S, S \rightarrow \neg Q, S \rightarrow \neg Q'\}, T, IV \cup IV')$, where T and S are new pattern variables.

Notice that: (1) If the characteristic graphs of G and G' are \neg -acyclic so are those of G_1 and G_2 . (2) $W(R) = W(Q) + 1$ and $W(T) = 2 + \text{the maximum of } W(Q) \text{ and } W(Q')$. Q.E.D.

COROLLARY 2.1. *The Boolean closure of context-free languages is contained in the family of extended context-free languages.*

THEOREM 3. *The family of extended context-free languages is not closed under homomorphism and therefore not closed under substitution.*

Proof. It is known that a recursive enumerable set can be obtained as a homomorphic image of the intersection of two context-free languages ([9, Lemma 4.7.1, p. 125]). If the family of extended context-free languages is closed under homomorphism, then recursive enumerable sets are all in the family of extended context-free languages. This contradicts the solvability of the membership problem for any CPDS. Q.E.D.

The above theorem actually tells us that the family of extended context-free languages is not a full AFL, since a full AFL is closed under arbitrary homomorphism. The reader is advised to consult Ginsburg et al. [10], Salomma [17] or Ginsburg [9] for the definition of full AFL.

We now present the solvable and the unsolvable problems of canonical pattern definition systems which generate extended context-free languages. A problem is *solvable* if and only if there is an algorithm which outputs the answer for any given instance of the problem. The reader is assumed to be familiar with the unsolvability results of the context-free grammars. Standard references are Hopcroft and Ullman [13] and Salomma [17].

Since our pattern matching algorithm will determine whether a string is in the language generated by a CPDS, the membership problem for any CPDS is solvable.

THEOREM 4. *Each of the following problems is unsolvable for canonical pattern definition systems G and G' :*

- | | |
|---|--|
| (1) <i>Is $L(G) = \emptyset$?</i> | (7) <i>Is $L(G) \cap L(G') = \emptyset$?</i> |
| (2) <i>Is $L(G)$ finite?</i> | (8) <i>Is $L(G) \cap L(G')$ finite?</i> |
| (3) <i>Is $L(G)$ infinite?</i> | (9) <i>Is $L(G) \cap L(G')$ infinite?</i> |
| (4) <i>Is $L(G) = C^*$?</i> | (10) <i>Does $L(G) = R$, R a specific regular set?</i> |
| (5) <i>Is $L(G) = L(G')$?</i> | (11) <i>Is $L(G)$ regular?</i> |
| (6) <i>Is $L(G)$ a subset of $L(G')$?</i> | |

Proof. Any of the problems (4)–(11) is unsolvable because of the following two reasons:

- (1) The family of context-free grammars is a subset of this family of CPDS's; and
- (2) Any of the problems (4)–(11) is unsolvable for the family of context-free grammars. Since the family of extended context-free languages is (effectively) closed under intersection, the unsolvability of problems (7), (8), and (9) for context-free grammars implies the unsolvability of problems (1), (2) and (3) for CPDS's, respectively. Q.E.D.

Notice that the problems (1), (2), and (3) are solvable for context-free grammars, but are unsolvable for CPDS's. Problems of (4)–(11) are unsolvable for both context-free grammars and CPDS's.

5. The pattern matching algorithm. Since many patterns can be specified in terms of a general context-free grammar with some extensions, we adapt Earley's parsing algorithm [3] for pattern matching for context-free patterns, and extend it to handle the additional features previously mentioned.

We shall first explain our pattern matching algorithm informally. Note that throughout this section we treat only the case of systems with one immediate assignment variable. Treatment of the cases where there are more than one such variable is essentially the same, but the notation for the general case in the definitions and proof of correctness is much more cumbersome.

Given a CPDS $G = (C, PV, \{I\}, \{(), \{\}\}, P, Q, IV)$, we number the production rules in P arbitrarily $1, 2, \dots, d$, where each production rule has the form

$$D_p \rightarrow C_{p1}C_{p2} \cdots C_{pn_p}, \quad 1 \leq p \leq d,$$

with n_p indicating the number of symbols on the right-hand side of the p th production rule. We add a 0th production rule $D_0 \rightarrow Q$, where $Q \in PV$ is the starting variable.

Let x_1, x_2, \dots, x_n be a subject string. The pattern matching algorithm is to scan the subject string from left to right. When the algorithm scans each character x_i it constructs a set of states S_i and a set of complement states N_i to represent the current condition of the matching (recognition) process, where a state and complement state are defined as follows. Note that for the simplification of our description, we will write $IV = x_{-|IV|+1} \cdots x_{-1}x_0$ and use position numbers $l = -|IV|$ and $r = 0$ to represent it.

DEFINITION 9. A *state* is a 7-tuple $\langle p, j, f, l, r, K, V \rangle$, which will also be written in this section as a 6-tuple

$$\langle D(p) \rightarrow C_{p1} \cdots C_{pj} \bullet C_{p(j+1)} \cdots C_{pn(p)}, f, l, r, K, V \rangle$$

with the dot \bullet (a new symbol not in T) symbolically representing j , where

$$D(p) = \begin{cases} D_p & \text{if } 0 \leq p \leq d, \\ I & \text{if } p = -1. \end{cases}$$

$$n(p) = \begin{cases} n_p & \text{if } 0 \leq p \leq d, \\ r - l & \text{if } p = -1. \end{cases}$$

p : (1) if $0 \leq p \leq d$, then it indicates that a substring of the subject string that we are currently scanning is derived from the right-hand side of the p th production rule.

(2) if $p = -1$, then it indicates that a substring of the subject string that is currently being scanned is derived from the current value of the immediate assignment variable I . What is intended here is a “constructed” production rule $I \rightarrow x_{l+1} \cdots x_r$.

j : ($0 \leq j \leq n(p)$) indicates that an initial portion of the substring has been derived from the leftmost j symbols on the right-hand side of the p th production rule, or the “constructed” production rule.

f : ($0 \leq f \leq n$) indicates that the substring that the right-hand side of the p th production rule can possibly derive starts from the $f+1$ th character of the subject string.

l, r : ($l \leq r \leq n$) l and r , used as left and right position numbers, together specify the current value of the immediate assignment variable I which is a string $x_{l+1} \cdots x_r$ and is to be derived from I when such derivations occur. Initially, $l = -|IV|$ and $r = 0$ specify IV , which is either θ or $x_{-|IV|+1} \cdots x_{-1}x_0$, as the current value of I . When a new value is assigned to I , l and r will be updated, in which case the current value of I is the substring $x_{l+1} \cdots x_r$ of the subject string where $0 \leq l \leq r \leq n$.

K : is a push-down stack of nonnegative integers whose maximum depth is the maximum depth of nesting of immediate assignments to I and whose elements are bounded by n . $\downarrow a_1 \downarrow a_2 \cdots \downarrow a_k$ is a notation for a stack with a_1 at its bottom and a_k on its top.

V : ($0 \leq V \leq W(Q)$) represents a bound on the weight of certain pattern variables. Note that in those states with $p = -1$, the K and V components can be ignored.

DEFINITION 10. A *state set* is an ordered set of states. A state is *added* to a state set at the end of the state set after testing to avoid duplication. State sets are denoted as S_i in the following.

DEFINITION 11. A *complete state* is a state with $j = n(p) \neq 0$.

DEFINITION 12. A *complement state* is a 9-tuple $\langle ic, fc, p, j, f, l, r, K, V \rangle$, where all components except the first two are the same as described above, and

ic : ($0 \leq ic \leq n$) indicates that $C_{p(j+1)} = \neg$ and $C_{p(j+2)} \in PV$, and that a substring of the subject string starting with x_{ic+1} may be derived from $\neg C_{p(j+2)}$.

fc : ($=0$ or 1) is a flag to indicate whether $\neg C_{p(j+2)}$ derives a substring when the complement state is processed. If it is 0 when the complement state is processed, then such derivation exists. Otherwise, it does not exist.

Note that, in this section, a complement state is written as a 8-tuple using the dot notation from above

$$\langle ic, fc, D_p \rightarrow C_{p1} \cdots C_{pj} \bullet \neg C_{p(j+2)} \cdots C_{pn(p)}, f, l, r, K, V \rangle.$$

DEFINITION 13. A *complement state set* is an ordered set of complement states. Complement state sets are denoted as N_i in the following.

To start with, the algorithm constructs $S_0 = \{\langle D_0 \rightarrow \bullet Q, 0, -|IV|, 0, \theta, W(Q) \rangle\}$ and $N_0 = \emptyset$. Then for the general case, namely, for each i when $0 \leq i \leq n$, we describe how the algorithm operates on S_i and N_i as follows.

If there exists a state in S_i which has not been processed, then we apply one of the six processors described below depending on the form of the state. The states in S_i are processed *in order*. To describe these processors, we assume that the state being processed has the following form:

$$s = \langle D(p) \rightarrow C_{p1} \cdots C_{pj} \bullet C_{p(j+1)} C_{p(j+2)} \cdots C_{pn(p)}, f, l, r, K, V \rangle.$$

The three processors of Earley, (1) predictor, (2) completer and (3) scanner, process the first components of the states in the same way as in Earley's parsing algorithm ([3]) with the following extensions.

(A) The predictor is also applicable to a state with the immediate assignment variable I to the right of the dot. It causes us to add to S_i a new state in which the first component is $I \rightarrow \bullet x_{l+1} \cdots x_r$, a "constructed" production which might derive the substring $x_{l+1} \cdots x_r$, the current value of I . If the predictor processes a state in which the pattern variable to the right of the dot has an erasing production rule or the immediate assignment variable to the right of the dot has null string as its current value, then it adds immediately a new state to S_i whose components are the same as in s except that in the first component the dot is moved over that pattern variable or immediate assignment variable to indicate that the variable has derived a null string.

(B) The completer is extended to apply to a state in which the dot is at the right end of the "constructed" production. It adds to S_i the state in S_f which has the immediate assignment variable I to the right of the dot and has the same l and r values. It moves the dot over I in this state. This indicates that the string which is the current value of I has been derived from I .

When the completer applies to a complete state, it has to perform extra tasks as follows. For each complement state in N_i which (a) has not been processed, (b) has $\neg D_p$ to the right of the dot in its third component, (c) has the same value as f in its first component, and (d) has the same value as $V + 1$ in its last component, we set the flag fc in its second component to 1, indicating that a derivation from $\neg D_p$ to the substring $x_{f+1} \cdots x_i$ does not exist when the complement state is processed later.

The other three added processors are described as follows.

(4) Immediate assignment processor. This is applied to the state s for one of the following two cases.

(A) If in the first component the immediate assignment left marker (*IALM*) is to the right of the dot. It adds a new state to S_i with its components the same as in

s except that in the first component the dot is moved over that left marker and that in the fifth component a new stack is constructed by pushing the value of i on the top of the stack K in the state s to indicate that a substring beginning with x_{i+1} will be assigned as a new value to the immediate assignment variable (IAV) later.

- (B) If in the first component the immediate assignment right marker ($IARM$) is to the right of the dot. It adds a new state to S_i whose components are the same as in state s except the following changes:
 - (a) In the first component the dot is moved over that right marker.
 - (b) In the third component we put the value of the top element, say l' , of the stack K in the state s .
 - (c) In the fourth component we put the value of i .
 - (d) In the fifth component a new stack is constructed from the stack K with its top element popped.

All these mean that the substring $x_{i'+1} \cdots x_i$ has been assigned as a new value of the immediate assignment variable I , and will be used for the derivation of I until another value is assigned to I .

(5) Predictor for complement. This processor is applied to the state s if in the first component to the right of the dot is the symbol for complement \neg followed by a pattern variable (i.e., $C_{p(j+1)} = \neg$ and $C_{p(j+2)} \in PV$). It adds one new state to S_i for each production rule with that pattern variable, $C_{p(j+2)}$, on its left-hand side, and with a nonnull string on its right-hand side. Note that in each new state:

- (A) The first component is the production rule with the dot put at the left end of its right-hand side which indicates that no substring has been derived from the right-hand side.
- (B) The second component is set to i since the state is created in S_i . Again a substring beginning with x_{i+1} may be derived from the right-hand side.
- (C) The last component is set to $V - 1$. This means that a derivation from this new state is one level lower in terms of the complement, and any such derivation from the right-hand side of the production rule at the $V - 1$ level will deny a derivation from $\neg C_{p(j+2)}$ at the V level.
- (D) The rest of the components are the same as in the state s .

Then, it adds a complement state to N_i whose components are described as follows:

- (A) The first component is set to i since the complement state is created in N_i . Note that $\neg C_{p(j+2)}$ may derive a substring starting with x_{i+1} .
- (B) The second component is set to 0 if $C_{p(j+2)}$ has no erasing production rule, and is set to 1 otherwise.
- (C) The third through the seventh components are copied from the first through the fifth components in the state s .
- (D) The last component is set to $V - 1$.

The applications of the six processors which we just described may add states to S_i and/or to S_{i+1} , and may add complement states to N_i . After all states in S_i are processed, the algorithm checks if there exists a complement state in N_i which has not been processed. If there exists such complement state, then we process the most recently added complement state, applying the processor described below, and mark the complement state as processed. Let $s' = \langle ic, fc, p, j, f, l, r, K, V \rangle$ be the form of the complement state being processed. Note that $C_{p(j+1)} = \neg$ and $C_{p(j+2)} \in PV$.

(6) Completer for complement. This processor is applied to the complement state s' if fc , the flag in the second component, is zero. It adds to the state set S_i a state whose components are described as follows:

- (A) The first component is copied from the third component of s' with the dot moved over $\neg C_{p(j+2)}$, which says that the substring $x_{ic+1} \cdots x_i$ of the subject string has been derived from $\neg C_{p(j+2)}$ at the $V+1$ level. (See the description of the predictor for complement.)
- (B) The second through the fifth components are the same as the fourth through the seventh components in s' .
- (C) The last component is set to $V+1$.

Then it resets the second component fc to zero. This makes the complement state ready to be copied into N_{i+1} so that when it is processed in N_{i+1} , the substring $x_{ic+1} \cdots x_i x_{i+1}$ of the subject string may be derived from $\neg C_{p(j+2)}$.

The application of completer for complement may add a state to S_i . Now, since we may have a state in S_i which is not processed, we have to go back to S_i and repeat the process described above which operates on S_i . Thus, the algorithm processed back and forth between S_i and N_i until all elements in S_i and in N_i are processed and no elements can be added to S_i or N_i . Then it copies all complement states from N_i to N_{i+1} (with their second components reset to zero), before we start to process S_{i+1} and N_{i+1} .

Before we can scan the last character, x_n , of the subject string, if we finish processing S_i and N_i , and S_{i+1} and N_{i+1} are empty, then the pattern matching fails. After the last character x_n is scanned, if there exists a state of the form $\langle D_0 \rightarrow Q \bullet, 0, l, r, \theta, W(Q) \rangle$ in S_n for some l and r with $l \leq r \leq n$, then the pattern matching succeeds. Otherwise, the pattern matching fails.

We now show the complete runs of the algorithm on two CPDS's defined in Examples 5 and 6. Example 7 shows the case which involves complement but not the immediate assignment; the l, r, K components in the states and complement states remain unchanged, and all complement state sets are not empty. Example 8 illustrates the immediate assignment which causes the l, r and K components to be updated; all complement state sets are empty, and the V component remains unchanged.

Example 7.

Start variable: $P \rightarrow \neg R$
 $R \rightarrow \neg E | \neg F$
 $E \rightarrow EC | QC$
 $Q \rightarrow AB | AQB$
 $F \rightarrow AF | AT$
 $T \rightarrow BC | BTC$

Subject string: ABC

$S_0(x_1 = A)$		N_0	
$P \rightarrow \bullet \neg R$	0, 0, 0, θ , 2	0, \emptyset^1	$P \rightarrow \bullet \neg R$ 0, 0, 0, θ , 1
$R \rightarrow \bullet \neg E$	0, 0, 0, θ , 2	0, 0	$R \rightarrow \bullet \neg E$ 0, 0, 0, θ , 1
$R \rightarrow \bullet \neg F$	0, 0, 0, θ , 2	0, 0	$R \rightarrow \bullet \neg F$ 0, 0, 0, θ , 1
$E \rightarrow \bullet EC$	0, 0, 0, θ , 2		
$E \rightarrow \bullet QC$	0, 0, 0, θ , 2		
$F \rightarrow \bullet AF$	0, 0, 0, θ , 2		
$F \rightarrow \bullet AT$	0, 0, 0, θ , 2		
$Q \rightarrow \bullet AB$	0, 0, 0, θ , 2		
$Q \rightarrow \bullet AQB$	0, 0, 0, θ , 2		
$R \rightarrow \neg F \bullet$	0, 0, 0, θ , 2		
$R \rightarrow \neg E \bullet$	0, 0, 0, θ , 2		
$S_1(x_2 = B)$		N_1	
$F \rightarrow A \bullet F$	0, 0, 0, θ , 2	0, \emptyset^1	$P \rightarrow \bullet \neg R$ 0, 0, 0, θ , 1
$F \rightarrow A \bullet T$	0, 0, 0, θ , 2	0, 0	$R \rightarrow \bullet \neg E$ 0, 0, 0, θ , 1

$Q \rightarrow A \bullet B$	0, 0, 0, θ , 2	0, 0	$R \rightarrow \bullet \neg F$	0, 0, 0, θ , 1
$Q \rightarrow A \bullet QB$	0, 0, 0, θ , 2			
$F \rightarrow \bullet AF$	1, 0, 0, θ , 2			
$F \rightarrow \bullet AT$	1, 0, 0, θ , 2			
$T \rightarrow \bullet BC$	1, 0, 0, θ , 2			
$T \rightarrow \bullet BTC$	1, 0, 0, θ , 2			
$Q \rightarrow \bullet AB$	1, 0, 0, θ , 2			
$Q \rightarrow \bullet AQB$	1, 0, 0, θ , 2			
$R \rightarrow \neg F \bullet$	0, 0, 0, θ , 2			
$R \rightarrow \neg E \bullet$	0, 0, 0, θ , 2			
$S_2(x_3 = C)$		N_2		
$Q \rightarrow AB \bullet$	0, 0, 0, θ , 2	0, \emptyset^1	$P \rightarrow \bullet \neg R$	0, 0, 0, θ , 1
$T \rightarrow B \bullet C$	1, 0, 0, θ , 2	0, 0	$R \rightarrow \bullet \neg E$	0, 0, 0, θ , 1
$T \rightarrow B \bullet TC$	1, 0, 0, θ , 2	0, 0	$R \rightarrow \bullet \neg F$	0, 0, 0, θ , 1
$E \rightarrow Q \bullet C$	0, 0, 0, θ , 2			
$R \rightarrow \neg E \bullet$	0, 0, 0, θ , 2			
$R \rightarrow \neg F \bullet$	0, 0, 0, θ , 2			
S_3		N_3		
$T \rightarrow BC \bullet$	1, 0, 0, θ , 2	0, 0	$P \rightarrow \bullet \neg R$	0, 0, 0, θ , 1
$E \rightarrow QC \bullet$	0, 0, 0, θ , 2	0, \emptyset^1	$R \rightarrow \bullet \neg E$	0, 0, 0, θ , 1
$F \rightarrow AT \bullet$	0, 0, 0, θ , 2	0, \emptyset^1	$R \rightarrow \bullet \neg F$	0, 0, 0, θ , 1
$E \rightarrow E \bullet C$	0, 0, 0, θ , 2			
$P \rightarrow \neg R \bullet$	0, 0, 0, θ , 2			

Example 8.

Start variable: $T \rightarrow A|(T)I$

Initial value for I : null string

Subject string: AAAA

$S_0(x_1 = A)$		$N_0 = \phi$
$T \rightarrow \bullet A$	0, 0, 0, θ , 0	
$T \rightarrow \bullet (T)I$	0, 0, 0, θ , 0	
$T \rightarrow (\bullet T)I$	0, 0, 0, \downarrow 0, 0	
$S_1(x_2 = A)$		$N_1 = \phi$
$T \rightarrow A \bullet$	0, 0, 0, θ , 0	
$T \rightarrow (T \bullet)I$	0, 0, 0, \downarrow 0, 0	
$T \rightarrow (T) \bullet I$	0, 0, 1, θ , 0	
$I \rightarrow \bullet A$	1, 0, 1, θ , 0	
$S_2(x_3 = A)$		$N_2 = \phi$
$I \rightarrow A \bullet$	1, 0, 1, θ , 0	
$T \rightarrow (T)I \bullet$	0, 0, 1, θ , 0	
$T \rightarrow (T \bullet)I$	0, 0, 0, \downarrow 0, 0	
$T \rightarrow (T) \bullet I$	0, 0, 2, θ , 0	
$I \rightarrow \bullet AA$	2, 0, 2, θ , 0	
$S_3(x_4 = A)$		$N_3 = \phi$
$I \rightarrow A \bullet A$	2, 0, 2, θ , 0	
S_4		$N_4 = \phi$
$I \rightarrow AA \bullet$	2, 0, 2, θ , 0	
$T \rightarrow (T)I \bullet$	0, 0, 2, θ , 0	

The algorithm. With this background we are now prepared to proceed directly to our formal description, for which we need the following extension functions. Note that a state is of the form $s = \langle p, j, f, l, r, K, V \rangle$ and that the initial value of I is written as

$IV = x_{-|IV|+1} \cdots x_{-1}x_0$ and is specified by position numbers $l = -|IV|$ and $r = 0$.

$$C(p, j, l) = \text{if } p = -1 \text{ then } x_{l+j} \text{ else } C_{p(j)}.$$

$$D(p) = \text{if } p = -1 \text{ then } I \text{ else } D_p.$$

$$\text{COMPLETE}(s) = \text{if } p = -1 \text{ then } j \geq r - l \text{ else } j \geq n_p.$$

$$n(p) = \text{if } p = -1 \text{ then } r - 1 \text{ else } n_p.$$

The *pattern matching algorithm* (PMA) is a function of two arguments, $\text{PMA}(G, x_1x_2 \cdots x_n)$, where G is a CPDS, and $x_1x_2 \cdots x_n$ is a subject string, into the set {"SUCCESS", "FAILURE"}, which is computed as follows.

Initialization:

for $i = 0$ **step** 1 **until** n **do**

begin $S_i = \emptyset$; $N_i = \emptyset$ **end**;

add $\langle 0, 0, 0, -|IV|, 0, \theta, W(Q) \rangle$ **to** S_0 ;

Main loop:

For $i = 0$ **step** 1 **until** n **do**

begin

While (\exists a state in S_i which has not been processed or \exists a complement state in N_i which has not been processed) **do**

L1: begin

if (\exists a state in S_i which has not been processed) **then**

L2: begin

 process the states of S_i which have not been processed in some order, performing one of the following operations on each state $s = \langle p, j, f, l, r, K, V \rangle$.

(1) *Predictor:*

if not $\text{COMPLETE}(s)$, $p \neq -1$ **and** $C_{p(j+1)} \in PV \cup \{I\}$ **then**

for each q such that $C_{p(j+1)} = D(q)$ **do**

if $n(q) = 0$ **then**

add $\langle p, j+1, f, l, r, K, V \rangle$ **to** S_i

else

add $\langle q, 0, i, l, r, K, V \rangle \in S_i$;

(2) *Completer:*

if $\text{COMPLETE}(s)$ **then**

begin

C1: for each $\langle q, e, g, l', r', K, V \rangle \in S_f$ such that $C_{q(e+1)} = D(p)$ **do**

if $p \neq -1$ **or** ($p = -1$, $l = l'$ **and** $r = r'$) **then**

add $\langle q, e+1, g, l, r, K, V \rangle$ **to** S_i ;

C2: for each complement state $\langle ic, fc, q, e, g, l', r', K, V \rangle$ in N_i which has not been processed such that

$C_{q(e+2)} = D_p$ **and** $f = ic$ **do**

$fc = 1$;

end;

(3) *Scanner:*

if not $\text{COMPLETE}(s)$ **and** $C(p, j+1, l) = x_{i+1}$ **then**

add $\langle p, j+1, f, l, r, K, V \rangle$ **to** S_{i+1} ;

```

(4) Immediate Assignment Processor:
    if not COMPLETE( $s$ ),  $p = -1$  and  $C_{p(j+1)} \in \{(,)\}$  then
        if  $C_{p(j+1)} = ($  then
            add  $\langle p, j+1, f, l, r, K \downarrow i, V \rangle$  to  $S_i$ 
        else
            add  $\langle p, j+1, f, \text{top}(K), i, \text{pop}(K), V \rangle$  to  $S_i$ ;
(5) Predictor for complement:
    if not COMPLETE( $s$ ),  $C_{p(j+1)} = \neg$  and  $C_{p(j+2)} \in PV$  then
        begin
             $fc = 0$ ;
            for each  $q$  such that  $C_{p(j+2)} = D_q$  do
                if  $n_q \neq 0$  then
                    add  $\langle q, 0, i, l, r, K, V-1 \rangle$  to  $S_i$ ;
                else
                     $fc = 1$ ;
                    add  $\langle i, fc, p, j, f, l, r, K, V-1 \rangle$  to  $N_i$ ;
            end;
        end /* end L2 block */
    else
        L3: begin
            if  $\exists$  a complement state in  $N_i$  which has not been processed then
                perform the following operation on the most recently added
                complement state  $s' = \langle ic, fc, p, j, f, l, r, K, V \rangle$ , and mark  $s'$  as
                processed.
            (6) Completer for complement:
                if  $fc = 0$  then
                    add  $\langle p, j+2, f, l, r, K, V+1 \rangle$  to  $S_i$ 
                else
                     $fc = 0$ ;
                end; /* end L3 block */
            end; /* end L1 block */
             $N_{i+1} = N_i$ ;
            if  $i < n$ ,  $S_{i+1} = \phi$  and  $N_{i+1} = \phi$  then
                output "FAILURE";
            end; /* end Main Loop */
        if  $\langle 0, 1, 0, l, r, \theta, W(Q) \rangle \in S_n$  then
            output "SUCCESS"
        else
            output "FAILURE";
    /* end of the pattern matching algorithm */.

```

This pattern matching algorithm *always terminates* because of the following reasons:

(1) Since a state is added to a state set only when it has not been a member, there are no duplications of states in a state set. The addition of states does not cause infinite loop.

(2) As will be shown in Theorem 5 in which we prove the time complexity of the algorithm, the number of states in each state set S_i is $O(i^{D+3})$ where $0 \leq i \leq n$ and D is the maximum nesting level occurring for the immediate assignment operator. And the number of complement states in each complement state set N_i is $O(i^{D+4})$ where $0 \leq i \leq n$.

See Theorem 5 for details. Actually that theorem indicates that the algorithm always terminates.

6. Corrections. A proof of the correctness of this PMA has been formally presented in Liu [14], for which we outline the following.

(1) In Earley's algorithm, a state $s = \langle p, j, f \rangle$ in S_i exactly when

$$\begin{aligned} D_0 &\stackrel{*}{\Rightarrow} x_1 \cdots x_f D_p z \\ &\Rightarrow x_1 \cdots x_f C_{p1} \cdots C_{pj} C_{p(j+1)} z' \\ &\stackrel{*}{\Rightarrow} x_1 \cdots x_f x_{f+1} \cdots x_i C_{p(j+1)} z', \quad \text{for some } z \text{ and } z'. \end{aligned}$$

(2) In our PMA, a state $s = \langle p, j, f, l, r, K, V \rangle$ is placed into S_i just when

$$\begin{aligned} &\langle D_0, \theta, IV \rangle \\ &\stackrel{*}{\underset{V}{\mapsto}} \langle x_1 \cdots x_f D_p z, K', y' \rangle \\ &\mapsto \langle x_1 \cdots x_f C_{p1} \cdots C_{pj} C_{p(j+1)} z', K', y' \rangle \\ &\stackrel{*}{\underset{V}{\mapsto}} \langle x_1 \cdots x_f x_{f+1} \cdots x_i C_{p(j+1)} z', K, x_{l+1} \cdots x_r \rangle, \end{aligned}$$

for some z, z', K', y' and $IV = x_{-|IV|+1} \cdots x_0$.

(3) In this PMA, when a complement state $s' = \langle ic, fc, p, j, f, l, r, K, V \rangle$ with $fc = 0$ is processed in N_i , the following extended derivation is indicated.

$$\begin{aligned} &\langle x_1 \cdots x_{f''} D_{p''} z'', K'', y'' \rangle \\ &\stackrel{*}{\underset{V+1}{\mapsto}} \langle x_1 \cdots x_{f''} \cdots x_f D_p z, K', y' \rangle \\ &\longleftarrow \langle x_1 \cdots x_f C_{p1} \cdots C_{pj} \neg C_{p(j+2)} z', K', y' \rangle \\ &\stackrel{*}{\underset{V+1}{\mapsto}} \langle x_1 \cdots x_f x_{f+1} \cdots x_{ic} \neg C_{p(j+2)} z', K, x_{l+1} \cdots x_r \rangle \\ &\longleftarrow \langle x_1 \cdots x_{ic} x_{ic+1} \cdots x_i z', K, x_{l+1} \cdots x_r \rangle, \end{aligned}$$

for some $f'', p'', z'', z', K'', K', y''$ and y' .

Note that if $fc = 0$ when s' is processed, then the last step in the above extended derivation never occurs. Also note that if $V = W(Q) - 1$, then $\langle x_1 \cdots x_{f''} D_{p''} z'', K'', y'' \rangle = \langle D_0, \theta, IV \rangle$.

(4) The complement states in N_i are processed in the last-in-first-out order. This is due to the hierarchy of complementation in the extended derivations which requires us to handle the complementation in a bottom up manner. See Example 8 and the predictor for complement in the PMA for this justification.

(5) In the PMA we "predict" the immediate assignment variable I , by associating a new production $I \rightarrow x_{l+1} \cdots x_r$ with the generated state $\langle p, 0, i, l, r, K, V \rangle$, $p = -1$,

assuming that $s = \langle p, j, f, l, r, K, V \rangle$ is the state being “predicted” for I in S_i . The functions $C(p, j, l)$, $D(p)$, $\text{COMPLETE}(s)$ and $n(p)$ for states $s = \langle p, j, f, l, r, K, V \rangle$ are made conditional on the value of p to allow the scanner and completer to be independent of whether the production associated with a state is, or is not, this special interpretive production.

7. Time and space complexity. We shall measure the complexity in terms of operations on such random access machines (RAM) as described in Earley [3] or Aho, Hopcroft and Ullman [13]. Although the algorithm is described in a high level language called Pidgin ALGOL [1], a Pidgin ALGOL algorithm can be translated into a RAM program in a straightforward manner.

DEFINITION 14. The *time [space] complexity* of a pattern matching algorithm, written as T(PMA) [S(PMA)], is defined as the time [space] needed by the algorithm expressed as a function of the length of the subject string. If there is some constant c so that the algorithm processes each subject string of length n in time [space] not exceeding $c * f(n)$, then we say that the time [space] complexity of the algorithm is $O(f(n))$, read “order $f(n)$ ”.

THEOREM 5. *Given a canonical pattern definition system G , for a subject string of length n , the following hold true:*

(I) $T(\text{PMA}) = O(n^{2D+7})$, if immediate assignment of up to D nesting levels occurs in G ;

(II) $T(\text{PMA}) = O(n^3)$, if immediate assignment does not occur in G .

Proof.

I. (a) For $0 \leq i \leq n$, there are $O(i^{(D+2)+1})$ states in each state set S_i because the ranges of p, j , and V components of a state are bounded by some constants, while the f, l, r and K components depend on i . Note that f, l and r are bounded by i , and K , by $O(i^D)$ since the maximum depth of the stack K equals the maximum nesting level occurring for the immediate assignment operator. Actually there are i stacks of depth 1, i^2 stacks of depth 2, \dots i^D stacks of depth D , so, $i + i^2 + \dots + i^D$ stacks of maximum depth D .

(b) When we construct each state set S_i , lists of states can be constructed in such a way that each list is associated with each $\langle f, l, r, K \rangle$ where f, l, r and K depend on i , and consists of the states of the form $\langle p, j, f, l, r, K, V \rangle$ for some p, j, V . Then to test if a state $\langle p, j, l, r, K, V \rangle$ has already been added to S_i , we search through the list associated with $\langle f, l, r, K \rangle$. Therefore, the time for the testing is independent of f, l, r , and K , and is constant since the ranges of p, j , and V are bounded by some constants. Note that if $p = -1$ then there can only be one value of j associated with each $\langle f, l, r, K \rangle$. The lists can be discarded after S_i is constructed. Since duplication of states in S_i is avoided, duplication is impossible in N_i and hence no test is needed before a complement state is added to N_i .

(c) The scanner, predictor, predictor for complement and immediate assignment processor each execute a bounded number of steps per state in each state set S_i . So the total time for processing the states in S_i plus the steps executed by the scanner, predictor, predictor for complement and immediate assignment processor is $O(i^{(D+2)+1})$.

(d) The number of complement states in any complement state set N_i is $O(i^{(D+2)+2})$ because of the same reason as in (a) and that the fc component is bounded by 2, and the ic component depends on i and is bounded by i .

(e) When the completer processes a state in S_i : (1) The statement $C1$ may have to add $O(f^{(D+2)+1})$ states for S_f where $0 \leq f \leq i$, the state set pointed back to, and therefore executes $O(i^{(D+2)+1})$ steps in the worst case; (2) The statement $C2$ may have to set flages to 1 for $O(i^{(D+2)+1})$ complement states in N_i . Note that for each N_i , lists of complement

states can be constructed such that each list is associated with each ic where $0 \leq ic \leq i$ and consists of complement states of the form $\langle ic, fc, p, j, f, l, r, K, V \rangle$. Thus there are $O(i^{(D+2)+1})$ complement states in each list. Statement C2 then only searches through the list associated with $f=ic$ and sets flags. Namely, the completer executes $O(i^{(D+2)+1+(D+2)+1}) = O(i^{2(D+2)+2})$ steps in S_i , and $O(i^{(D+2)+1+(D+2)+1}) = O(i^{2(D+2)+2})$ steps in N_i .

(f) The completer for complement may have to add $O(i^{(D+2)+2})$ states to S_i from N_i . Namely, it takes $O(i^{(D+2)+2})$ steps to process complement states in N_i .

(g) Therefore, the time in the completer is dominant no matter whether complementation occurs or not. Hence summing from $i = 0, 1, \dots, n$ gives $O(n^{2(D+2)+2+1}) = O(n^{2D+7})$ steps. Note that since

$$\sum_{i=0}^n i^k \leq \sum_{i=1}^n n^k = n * n^k = n^{k+1},$$

we have

$$\sum_{i=0}^n O(i^k) \leq O\left(\sum_{i=1}^n i^k\right) \leq O(n^{k+1}).$$

II. The proof is exactly the same as that for (I) except for the following:

- (1) The immediate assignment processor will not be called.
- (2) The l, r and K components no longer depend on i . In fact $l = -|IV|$, $r = 0$ and $K = \theta$. Thus $D + 2$ becomes 0 in all exponents in the proof. Therefore, the completer executes $O(i^2)$ steps in S_i and $O(i^2)$ steps in N_i . And after summing up, the PMA takes $O(n^{2+1}) = O(n^3)$ time. Q.E.D.

THEOREM 6. *Given a canonical pattern definition system G , for a subject string of length n , the following hold true.*

- (I) $S(\text{PMA}) = O(n^{D+5})$, *if both complement and immediate assignment of up to D nesting levels occur in G ;*
- (II) $S(\text{PMA}) = O(n^{D+4})$, *if immediate assignment of up to D nesting levels but not complementation occurs in G ;*
- (III) $S(\text{PMA}) = O(n^3)$, *if complementation but not immediate assignment occurs in G ;*
- (IV) $S(\text{PMA}) = O(n^2)$, *if neither complementation nor immediate assignment occurs in G .*

Proof.

I. For $0 \leq i \leq n$, each S_i takes $O(i^{(D+2)+1})$ space, each N_i takes $O(i^{(D+2)+2})$ space. Summing from $i = 0, \dots, n$, the PMA takes $O(n^{[(D+2)+2]+1}) = O(n^{D+5})$ space.

II. As in the proof of (I), except that each N_i takes no space, we conclude that the PMA takes $O(n^{(D+2)+1+1}) = O(n^{D+4})$ space.

III. In this case, $(D + 2)$ becomes 0 in the theorem. Therefore, $O(n^{[(D+2)+2]+1}) = O(n^3)$.

IV. As in the proof of (I) except that each S_i takes $O(i)$ space, and each N_i takes no space, we therefore conclude that the PMA takes $O(n^2)$ space. Q.E.D.

THEOREM 7. *Given a canonical pattern definition system G which allows J immediate assignment variables (i.e., $|IAV| = |IALM| = |IARM| = J$) and up to D nesting levels for immediate assignment, for a subject string of length n , we have*

- (I) $T(\text{PMA}) = O(n^{2J(D+2)+3})$, and
- (II) $S(\text{PMA}) = O(n^{J(D+2)+3})$.

Proof.

I. In any state or complement state, the l, r and K components should, in this case, be replaced by a list of triples (l^b, r^b, K^b) where $1 \leq b \leq J$. The arguments are

similar to the proof for (I) of Theorem 5 except that $(D+2)$ becomes $J(D+2)$ in this case. Hence the time in the completer becomes $O(i^{J(D+2)+1+J(D+2)+1}) = O(i^{2J(D+2)+2})$ in S_i and $O(i^{J(D+2)+1+J(D+2)+1}) = O(i^{2J(D+2)+2})$ in N_i . This results in the time complexity of $O(n^{2J(D+2)+3})$.

II. In any state or complement state, the l , r and K components should, in this case, be replaced by a list of triples (l^b, r^b, K^b) where $1 \leq b \leq j$. Therefore, as we argued before, $(D+2)$ should become $J(D+2)$ in the proof for (I) of the Theorem 6. Namely, the PMA takes $O(n^{J(D+2)+3})$ space. Q.E.D.

8. Conclusions. An empirical evaluation is given in Liu [14] which consists of several case studies comparing an implementation of our algorithm using PL/I with a SNOBOL4 implementation. Despite the fact that we program our algorithm using PL/I in such a well structured manner that it is easy to understand, time comparison of the SNOBOL4 pattern matching algorithm for some patterns are favorable for the implementation of our algorithm.

In SNOBOL4, a heuristic is incorporated in the pattern matching process to prevent looping of left-recursive patterns and this introduces the possibility of producing incorrect results or interfering with intended matches. But this is not a problem for our algorithm. This polynomial time algorithm is correct and powerful.

Acknowledgment. I wish to express my deep gratitude to Arthur C. Fleck of the University of Iowa for his valuable advice in this research. I also appreciate the referees' elegant suggestions.

REFERENCES

- [1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1975.
- [2] R. B. K. DEWAR AND A. P. MCCANN, *Macro SPITBOL—a SNOBOL4 compiler*, Software-Practice and Experience, 7 (1977), pp. 95–113.
- [3] J. EARLEY, *An efficient context-free parsing algorithm*, Comm. ACM, 13 (1970), pp. 94–102.
- [4] A. C. FLECK, *Toward a theory of data structures*, J. Comput. System Sci., 5 (1971), pp. 475–488.
- [5] ———, *Formal Languages and Iterated Functions with an Application to Pattern Representations*, Dept. of Computer Science, University of Iowa, Iowa City, Iowa, 1975.
- [6] ———, *Formal models for string patterns*, in Current Trends in Programming Methodology, Vol. 4: Data Structuring, R. Yeh, ed., Prentice-Hall, Englewood Cliffs NJ, 1978, pp. 216–240.
- [7] J. F. GIMPLE, *A theory of discrete patterns and their implementation in SNOBOL4*, Comm. ACM, 16 (1973), pp. 91–100.
- [8] ———, *Algorithms in SNOBOL4*, John Wiley, New York, 1976.
- [9] S. GINSBURG, *Algebraic and Automata Theoretic Properties of Formal Languages*, American Elsevier, New York, 1975.
- [10] S. GINSBURG, S. GREIBACH AND J. HOPCROFT, *Studies in Abstract Families of Languages*, Memoirs of the American Mathematical Society, No. 87, Providence RI, 1969.
- [11] R. E. GRISWOLD, J. F. POAGE AND I. P. POLONSKY, *The SNOBOL4 Programming Language*, 2nd Ed., Prentice-Hall, Englewood Cliffs NJ, 1971.
- [12] J. C. HALLYBURTON, JR., *Advanced data structure manipulation facilities for the SNOBOL4 programming language*, S4D42, University of Arizona, Tucson AZ, 1974.
- [13] J. E. HOPCROFT AND J. D. ULLMAN, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading MA, 1969.
- [14] K. C. LIU, *An efficient algorithm for string pattern matching*, Ph.D. Thesis, Dept. of Computer Science, University of Iowa, Iowa City, Iowa, 1977.
- [15] L. Y. LIU AND P. WEINER, *An infinite hierarchy of intersections of context-free languages*, Math. System Theory, 7 (1973), pp. 185–192.

- [16] G. D. RIPLEY AND R. E. GRISWOLD, *The measurement of SNOBOL4 programs*, SIGPLAN Notices 10 (1975), pp. 36–52.
- [17] A. SALOMMA, *Formal Languages*, Academic Press, New York, 1973.
- [18] G. F. STEWART, *An algebraic model for string patterns*, ACM 2nd Symposium on Principles of Programming Languages, Association for Computing Machinery, New York, 1975, pp. 167–184.

LISTING AND COUNTING SUBTREES OF A TREE*

FRANK RUSKEY†

Abstract. Given an ordered tree T , an ordering is defined on the set of subtrees of T . Algorithms are presented for listing all subtrees in that order, and for determining the tree occupying a given position in the list. The second algorithm runs in time proportional to the number of vertices in the tree. An explicit formula is given for the total number of subtrees summed over all trees T .

Key words. ordered tree, subtree, lexicographic enumeration ranking algorithm

1. Introduction. This paper studies the problem of listing all subtrees of a given ordered tree. The problem originally arose in solving a certain optimization problem on networks [1] (in [1] subtrees were called "tree cuts"). We will also develop algorithms for ranking and unranking subtrees. The rank of a subtree is the position that it occupies in the listing and unranking is the inverse operation of taking a position and returning the subtree occupying that position.

All trees in this paper are *ordered trees*. Let T_n denote the set of all ordered trees with n vertices. Let $T \in T_n$ have root r . A r -*subtree* of T is a subtree that includes the root. By removing the root of T we get an ordered forest of subtrees T_1, T_2, \dots, T_k called the *principal subtrees* of T . If $T_i \in T_{v_i}$ then $v_1 + v_2 + \dots + v_k = n - 1$.

Let $T \in T_n$. A *labeling* of T is an assignment of the integers $\{1, 2, \dots, n\}$ to the vertices of T (distinct vertices get distinct labels). A *heap-labeling* is a labeling with the property that children receive larger labels than their parents. Thus, in a heap-labeled tree the root is labeled 1. A *left-to-right breadth first search* labeling, hereafter referred to as a BFS labeling, is the labeling one gets by labeling the vertices on a lower level before labeling those at a higher level. Vertices on the same level are labeled from left to right. A *preorder* labeling (or left-to-right depth first search) is the labeling one gets by labeling the vertices in the order that they are encountered in a preorder traversal of the tree. Fig. 1 shows the BFS and preorder labeling of a tree.

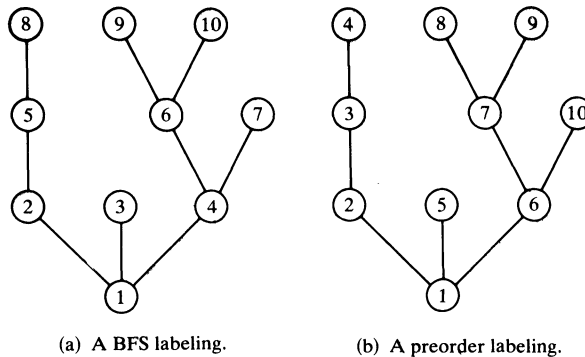


FIG. 1.

A labeled tree is often represented by an array par where $par[i]$ is the parent of vertex i (we will assume that $par[root] = 0$). For example, the parent array of the tree of Fig. 1(a) is $par = 0, 1, 1, 1, 2, 4, 4, 5, 6, 6$ and for Fig. 1(b) $par = 0, 1, 2, 3, 1, 1, 6, 7, 7, 6$. In a heap-labeled tree $par[i] < i$. In a BFS labeled tree $par[i] \leq par[i + 1]$ for $1 \leq i < n$. A

* Received by the editors February 28, 1979, and in final form April 3, 1980.

† Department of Computer Science, University of Victoria, Victoria, British Columbia, Canada V8W 2Y2.

parent array is a convenient representation when working from the leaves to the root. In working from the root to the leaves it is often more convenient to have the tree represented by adjacency lists. We will assume that the adjacency lists preserve the left-to-right ordering of the vertices. Both of these representations will be used subsequently. There are algorithms for converting one representation into the other in time proportional to the number of vertices in the tree.

We will express our algorithms in a Pascal-like notation. The adjacency list nodes will be defined as *node* = **record** *vert*: 1 · · · *n*; *next*: ↑*node* **end**. The adjacency list headers will be stored in an array *head*. Fig. 2 shows the adjacency lists for the tree of Fig. 1(a).

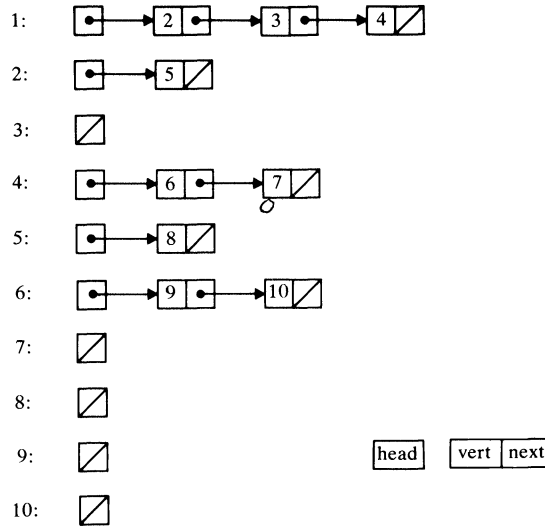


FIG. 2.

A subtree of a tree will be represented by a 0–1 array, called the *characteristic array*, *sub*, where $sub[i] = 1$ if vertex *i* is in the subtree and $sub[i] = 0$ if vertex *i* is not in the subtree. The sequence of values of the characteristic array will be referred to as the *characteristic sequence* of the subtree. A *r*-subtree is characterized by the property that $sub[i] = 1$ implies that $sub[par[i]] = 1$. For definitions of undefined terms used in this section the reader is referred to [2].

To generate all subtrees of a tree it is sufficient to be able to generate all *r*-subtrees. To generate all subtrees one simply traverses the tree and when vertex *r* is encountered generate all *r*-subtrees. We could also generate subtrees of free labeled trees by picking an arbitrary vertex to be the root.

The paper is organized as follows. Section 2 discusses generating *r*-subtrees. Section 3 develops a number of counting results for *r*-subtrees. The ranking and unranking algorithms for preorder labeled trees are given in § 4. These algorithms have running times proportional to the number of vertices in the tree. The final section contains some empirical results and mentions some further research questions.

The reader is cautioned that the suggested ordering for listing trees and the ordering used in the ranking and unranking algorithms are different.

2. The listing algorithm. Let T be a heap-labeled tree in T_n . We order the r -subtrees of T by ordering the characteristic arrays lexicographically. Thus, the first r -subtree consists of the root alone and the last r -subtree consists of all of the vertices of T . Given a characteristic array, obtaining the next one in our lexicographic order is quite simple. Scan sub from right to left changing 1's to 0's. Upon encountering a 0, say in position k , if $sub[par[k]] = 1$ then change it to a 1 and stop; if $sub[par[k]] = 0$ then continue scanning. If we scan through the entire array, then there is no next r -subtree. The procedure NEXT, given below, implements these ideas.

```

procedure NEXT;
begin
     $k := n$ ;
    while  $sub[k] = 1$  or  $sub[par[k]] = 0$  do
        begin
             $sub[k] := 0$ ;
             $k := k - 1$ ;
            if  $k = 0$  then  $last\_subtree$ 
        end;
         $sub[k] := 1$ 
    end;

```

Initializing and iterating NEXT ten times on the trees of Fig. 1 results in the sequences in Fig. 3.

1 0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0 0 0	1 0 0 0 0 1 0 0 0 0
1 0 0 1 0 0 1 0 0 0	1 0 0 0 0 1 0 0 0 1
1 0 0 1 0 1 0 0 0 0	1 0 0 0 0 1 1 0 0 0
1 0 0 1 0 1 0 0 0 1	1 0 0 0 0 1 1 0 0 1
1 0 0 1 0 1 0 0 1 0	1 0 0 0 0 1 1 0 1 0
1 0 0 1 0 1 0 0 1 1	1 0 0 0 0 1 1 0 1 1
1 0 0 1 0 1 1 0 0 0	1 0 0 0 0 1 1 1 0 0
1 0 0 1 0 1 1 0 0 1	1 0 0 0 0 1 1 1 0 1
1 0 0 1 0 1 1 0 1 0	1 0 0 0 0 1 1 1 1 0
⋮	⋮

(a) First ten subtrees of
Fig. 1(a).

(b) First ten subtrees of
Fig. 1(b).

FIG. 3

If NEXT is being used iteratively, as when generating all subtrees, then we can make use of information gained in the previous iteration. For example, let k_1 and k_2 be the final values of k in NEXT on two successive iterations. The value of k_1 may determine k_2 or at least restrict the values that k_2 can be. Let $L[k_1]$ be an upper bound on the values of k_2 . We can then replace the assignment $k := n$ of NEXT by $k := L[k_1]$. (Initially, $k = 1$).

This array $L[k]$ depends on the labeling that we are using. If the tree has the BFS labeling, then we can set $L[k]$ to be the greatest vertex j such that $par[j] \leq k$. In other words, if k is not a leaf then $L[k]$ is the rightmost child of k , and if k is a leaf then $L[k]$ is the rightmost child of the greatest vertex less than k (in the BFS labeling) that is not a leaf. Assuming that L is all 0's except $L[n+1] \neq 0$ (for proper termination), we can

compute $L[k]$ for BFS as follows.

```

for  $k := 2$  to  $n$  do  $L[\text{par}[k]] := k$ ;
  {the non-leaves now have the right  $L$  values}
 $k := 1$ ;
while  $k \leq n$  do
  begin
     $r := L[k]$ ;
     $k := k + 1$ ;
    while  $L[k] = 0$  do
      begin
         $L[k] := r$ ;
         $k := k + 1$ 
      end
    end
  end;

```

If k_1 is not a leaf, then k_2 is equal to $L[k_1]$, but when k_1 is a leaf, k_2 may or may not be equal to $L[k_1]$. For the tree of Fig. 1(a), $L = 4, 5, 5, 7, 8, 10, 10, 10, 10, 10$.

If the tree is given the preorder labeling then $L[k]$ is the greatest l such that $\text{par}[l] = j$ for some $j \leq k$. We can also calculate this L in $O(n)$ time. For the tree of Fig. 1(b), $L = 6, 6, 6, 6, 6, 10, 10, 10, 10, 10$. Note that the L in BFS uniquely determines the parent array but does not in the preorder case.

The BFS labeling produces a more efficient enumeration of r -subtrees than does the preorder labeling. At present this is not proven but the final section contains some empirical evidence. As an example of the superiority of the BFS labeling, consider a tree T with two principal subtrees, the left consisting of a single chain and the right a single vertex. This tree has $2(n-1)$ subtrees. The amount of computation required in going from one subtree to the next is essentially proportional to $L[k_1] - k_2$. Call the sum of $L[k_1] - k_2$ over all subtrees the *complexity* of that tree. The complexity of T is $\frac{1}{2}(n^2 - n + 2)$ (counting n for the last r -subtree).

Let the *average complexity* be the complexity divided by the number of r -subtrees. The average complexity of T is the worst possible, namely $\theta(n)$. This same tree with the BFS labeling has an average complexity of $\theta(1)$. No such example seems to exist with the roles of BFS and preorder reversed.

3. Counting subtrees. It is well known that the number of ordered trees with n vertices is

$$b_n = |T_n| = \frac{1}{n} \binom{2n-2}{n-1}.$$

Given $T \in T_n$ with root r , define $\rho(T)$ to be the number of r -subtrees of T . The next lemma gives us a recurrence relation for $\rho(T)$.

LEMMA 1. *If $T \in T_n$ has k principal subtrees T_1, T_2, \dots, T_k then*

$$\rho(T) = \begin{cases} 1 & \text{if } n = 1 \\ (1 + \rho(T_1))(1 + \rho(T_2)) \cdots (1 + \rho(T_k)) & \text{if } n > 1. \end{cases}$$

Proof. That $\rho(T) = 1$ for a single node tree is obvious. If $n > 1$, then there is at least one subtree when the root is removed. Let r be the root of T and r_i the root of T_i . For each subtree T_i an r -subtree either contains none of the vertices of T_i or is an r_i -subtree when restricted to T_i . This accounts for the factor $1 + \rho(T_i)$. The factors contribute multiplicatively and hence, we have the lemma. \square

For an example of the calculation of $\rho(T)$ see below:

$$\begin{aligned}
 & \rho \left(\begin{array}{c} \text{Diagram of a tree with 6 vertices} \end{array} \right) \\
 &= (1 + \rho(\text{Diagram of a 2-vertex subtree})) (1 + \rho(\text{Diagram of a 3-vertex subtree})) (1 + \rho(\text{Diagram of a 1-vertex subtree})) \\
 &= (1 + (1 + \rho(\text{Diagram of a 1-vertex subtree}))) (2) (1 + (1 + \rho(\text{Diagram of a 2-vertex subtree}))) (1 + \rho(\text{Diagram of a 1-vertex subtree})) \\
 &= (4)(2)(11) = 88.
 \end{aligned}$$

Let a_n be the number of r -subtrees summed over all trees with n vertices. In other words,

$$a_n = \sum_{T \in T_n} \rho(T).$$

The next lemma provides us with a recurrence relation for a_n .

LEMMA 2. *If $n = 1$ then $a_n = 1$, and if $n > 1$ then*

$$a_n = \sum_{j=1}^{n-1} (a_j + b_j) a_{n-j}.$$

Proof. Consider all trees in T_n that have j vertices in the leftmost principal subtree. Summing over all trees in T_n gives $a_j a_{n-j}$ r -subtrees that include the root of the leftmost subtree and $b_j a_{n-j}$ that do not include the root of the leftmost subtree. \square

Using Lemma 2, we can compute a table of the a_n 's; see Table 1.

TABLE 1

n	1	2	3	4	5	6	7	8	9	10
b_n	1	1	2	5	14	42	132	429	1430	4862
a_n	1	2	7	29	131	625	3099	15818	82595	943259

Let $A(x)$ and $B(x)$ be the ordinary generating functions of the sequences a_n and b_n , respectively. The function $B(x)$ is known to be (see [2])

$$(1) \quad B(x) = \frac{1}{2}(1 - \sqrt{1 - 4x})$$

and to satisfy the relation

$$(2) \quad B(x)^2 = B(x) - x.$$

By Lemma 2,

$$\begin{aligned}
 A(x) &= x + \sum_{n \geq 2} \sum_{j=1}^{n-1} (a_j + b_j) a_{n-j} x^n \\
 &= x + A(x)^2 + B(x)A(x).
 \end{aligned}$$

Therefore,

$$(3) \quad A(x)^2 + (B(x) - 1)A(x) + x = 0.$$

Thus,

$$(4) \quad \begin{aligned} A(x) &= \frac{1}{2}[1 - B(x) - \sqrt{(1 - B(x))^2 - 4x}] \\ &= \frac{1}{2}[1 - B(x) - \sqrt{1 - 5x - B(x)}] \\ &= \frac{1}{2}[\frac{1}{2}(1 + \sqrt{1 - 4x}) - \sqrt{\frac{1}{2}(1 + \sqrt{1 - 4x}) - 5x}]. \end{aligned}$$

By using (4), we can derive an explicit (nonrecursive) expression for a_n . First,

$$\begin{aligned} \sqrt{1 - (5x + B(x))} &= 1 + \sum_{m \geq 1} \binom{\frac{1}{2}}{m} (-1)^m (5x + B(x))^m \\ &= 1 - \sum_{m \geq 1} \frac{1}{m 2^{2m-1}} \binom{2m-2}{m-1} (5x + B(x))^m. \end{aligned}$$

Also,

$$\begin{aligned} (5x + B(x))^m &= \sum_{k \geq 0} \binom{m}{k} 5^k x^k B(x)^{m-k} \\ &= 5^m x^m + \sum_{k=0}^{m-1} \binom{m}{k} 5^k x^k \sum_{j \geq m-k} t(j, m-k) x^j, \end{aligned}$$

where

$$t(n, k) = \sum_{\substack{v_1 + v_2 + \dots + v_k = n \\ v_i \geq 1}} b_{v_1} b_{v_2} \dots b_{v_k}$$

are the numbers defined and studied in [3], [4]. There it is shown that

$$t(n, k) = \frac{k}{2n - k} \binom{2n - k}{n - k}.$$

Note that $t(n, k)$ is the number of trees with $n + 1$ vertices and root of degree k . We now know that $2A(x)$ is equal to

$$(5) \quad \begin{aligned} & - \sum_{m \geq 1} b_m x^m + \sum_{m \geq 1} \frac{b_m}{2^{2m-1}} 5^m x^m \\ & + \sum_{m \geq 1} \frac{b_m}{2^{2m-1}} \sum_{k=0}^{m-1} \binom{m}{k} 5^k x^k \sum_{j \geq m-k} t(j, m-k) x^j. \end{aligned}$$

From (5) it can be deduced that a_n is

$$b_n \left(\left(\frac{5}{4} \right)^n - \frac{1}{2} \right) + \sum_{m=1}^n \frac{b_m}{4^m} \sum_{k=0}^{m-1} \binom{m}{k} 5^k t(n-k, m-k).$$

Note that the average number of r -subtrees of an ordered tree is exponential in n if each tree is regarded as being equally likely.

4. Ranking and unranking subtrees. In this section, we derive ranking and unranking algorithms for r -subtrees of an ordered tree, where the subtrees are given the preorder ordering.

The rank of an r -subtree is the position that it occupies in the list of r -subtrees. To calculate the rank of an r -subtree we only need to know how many r -subtrees precede it in the list. Let $r(S, T)$ denote the rank of an r -subtree S of an ordered tree T . Let T_1, T_2, \dots, T_k be the principal subtrees of T , and let S_i be that portion of S that is in T_i . If r_1, r_2, \dots, r_k are the roots of the principal subtrees then the characteristic sequence of S can be written as

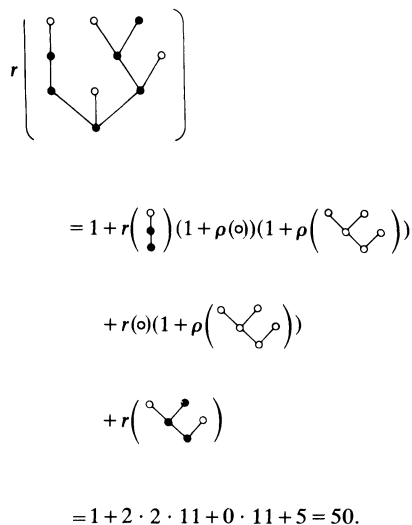
$$1 \quad \underbrace{s_{r_1} \dots}_{T_1} \quad \underbrace{s_{r_2} \dots}_{T_2} \quad \dots \quad \underbrace{s_{r_k} \dots}_{T_k}$$

The number of r -subtrees such that $s_{r_1} = s_{r_2} = \dots = s_{r_j} = 0$ and $s_{r_{j+1}} = 1$ will be denoted by $\tau(T_j)$ and is equal to $(1 + \rho(T_{j+1}))(1 + \rho(T_{j+2})) \dots (1 + \rho(T_k))$. Also, $\tau(T_k) = 1$. Before changing any bit in T_j all r_{j+1} -subtrees of T_{j+1} are generated, plus the empty subtree. We can therefore express the rank as the following recurrence:

$$(6) \quad r(S, T) = 1 + \sum_{j=1}^k \tau(T_j) r(S_j, T_j),$$

where $r(\phi, T) = 0$.

For example (blackened vertices denote S),



This calculation is top-down (root to leaves). The procedure RANK, given below, performs a bottom-up calculation using the parent array. RANK also makes use of four other arrays *rho*, *tau*, *rank* and *sib*. If vertex j has a left sibling i then $sib[j] = i$, otherwise $sib[j] = none$. The array *sib* is used in calculating the *tau* values. The other arrays *rho*, *tau*, and *rank* have the values of the functions ρ, τ, r , except that we specify the root instead of the entire tree. Fig. 4 shows the data structure being used. The wavy lines represent edges that are in the subtree being ranked, and dotted lines the sibling pointers. The rank is computed along with the values of ρ, τ, r in one pass over the tree.

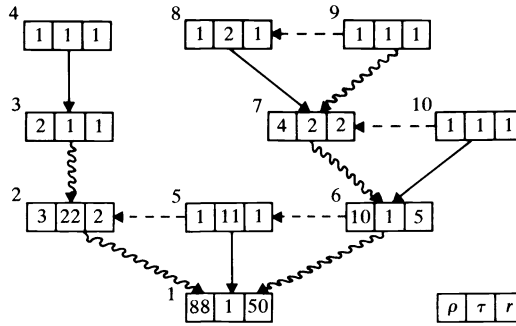


FIG. 4. The data structure used for computing ranks.

```

function RANK (sub: characteristic__array): integer;
begin
  for j := 1 to n do rho[j] := tau[j] := rank[j] := 1;
  for j := n down to 2 do
    begin
      rho[par[j]] := rho[par[j]] * (1 + rho[j]);
      if sub[j] ≠ none then
        tau[sib[j]] := tau[j] * (1 + rho[j]);
      if sub[j] = 1 then
        rank[par[j]] := rank[par[j]] + rank[j] * tau[j]
      end;
      RANK := rank[1]
    end;
end;

```

Clearly, RANK is $O(n)$ where n is the number of vertices in T . If the values of ρ, τ are already calculated and the r -subtree is given as a list of vertices (rather than a characteristic array) then RANK runs in time proportional to the number of vertices in the subtree.

The unranking algorithm runs best from root to leaves and so we will assume that T is represented by adjacency lists (as outlined in the introduction). The r -subtree will be returned as a characteristic array. The procedure UNRANK does a preorder traversal of that portion of T that is in the subtree. It is assumed that the array *sub* is initialized to all 0's

```

procedure UNRANK (rank: integer; v: vertex);
begin
  sub[v] := 1;
  rank := rank - 1;
  p := head[v];
  while p ≠ nil do
    begin
      w := p↑.vert;
      if rank ≧ tau[w] then
        begin
          UNRANK (rank div tau[w], w);
          rank := rank mod tau[w]
        end;
      p := p↑.next
    end
  end;

```


The running time of $\text{UNRANK}(\text{rank}, 1)$ is essentially proportional to the sum of the sizes of all adjacency lists of vertices of the r -subtree that occupies position rank in the lexicographic order. This is at most $O(n)$.








We can use UNRANK to produce random r -subtrees of a tree by first selecting an integer r in the range 1 to $\rho(T)$ at random and then calling $\text{UNRANK}(r, 1)$. By random, it is meant that every r -subtree is equally likely. One can also produce random subtrees by first computing ρ for each vertex in T , then choosing a vertex v at random (the vertices being weighted by their rho values), and finally calling $\text{UNRANK}(r, v)$ where r is an integer chosen at random from $\{1, 2, \dots, \rho[v]\}$.

5. Final remarks. The results presented here do not seem to fit into the general settings for listing and ranking that are presented in [5], [6] and [7]. This is because our algorithms are not based on any fixed simple recursion.

The worst case average complexity of the BFS labeling has not been determined (for preorder it is $\theta(n)$). However, considering the sequence of trees with $m^2 + 1$ vertices, where the root has degree m and each principal subtree consists of a chain with m vertices, we can deduce that the worst case average complexity is not bounded by a constant. It might be hoped that the average complexity averaged over all trees would be bounded by a constant, but the following evidence indicates otherwise. By exhaustively listing all trees with $n = 4, \dots, 10$ vertices and all of their r -subtrees the results shown in Table 2 were obtained.

In generating the trees for Table 2 it would have been possible to use the results of [3], [4]. However, it was found easiest to proceed directly from one parent array to the next in a lexicographic fashion. The parent array of a BFS labeled tree is characterized

TABLE 2

n	Average average complexity	Worst-case average complexity	Worst-case tree
4		1.6667	
5		1.2500	
6	1.1278	1.3889	
7	1.1754	1.4762	
8	1.2201	1.6111	
9	1.2611	1.7333	
10	1.2982	1.8375	

by the following two properties: (i) $0 \leq \text{par}[i] < i$ and (ii) $\text{par}[i] \leq \text{par}[i+1]$ for $1 \leq i \leq n$. Thus, we can produce the lexicographically next parent array as follows.

```

procedure NEXT__TREE;
  begin
     $i := n$ ;
    while  $\text{par}[i] = i - 1$  do  $i := i - 1$ ;
    if  $i = 0$  then last__tree;
     $\text{par}[i] := \text{par}[i] + 1$ ;
    for  $k := i + 1$  to  $n$  do  $\text{par}[k] := \text{par}[i]$ 
  end;

```

It would be interesting to have a nontrivial upper bound on the worst case average complexity. Another subject for future research is the finding of listing and ranking algorithms for r -subtrees with a fixed number of vertices.

REFERENCES

- [1] T. C. HU AND F. RUSKEY, *Circular cut in a network*, Math. Oper. Res., to appear.
- [2] D. E. KNUTH, *The Art of Computer Programming*, vol. 1, Addison-Wesley, Reading, MA, 1973.
- [3] F. RUSKEY AND T. C. HU, *Generating binary trees lexicographically*, this Journal, 6 (1977), pp. 745–758.
- [4] F. RUSKEY, *Algorithmic solution of two combinatorial problems*, Ph.D. Thesis, University of California, San Diego, 1978.
- [5] H. S. WILF, *A unified setting for sequencing, ranking and selection algorithms for combinatorial objects*, Adv. in Math., 24 (1977), pp. 281–291.
- [6] H. S. WILF, *A Unified setting for selection algorithms II*, Algorithmic Aspects of Combinatorics, Alspach et al., eds., North-Holland, Amsterdam, 1978.
- [7] S. G. WILLIAMSON, *On the ordering, ranking, and random selection of basic combinatorial sets*, Proceedings of the Table Ronde, Strasbourg, France, April 1976.

NONPREEMPTIVE LP-SCHEDULING ON HOMOGENEOUS MULTIPROCESSOR SYSTEMS*

MANFRED KUNDE†

Abstract. Unequal execution time task systems are nonpreemptively scheduled on $m \geq 2$ identical processors without additional resource constraints. Worst-case bounds for the ratio of the length of an LP-schedule (longest path) and an optimal schedule are given for two classes of dependency structures—chains and trees. Moreover, the asymptotic bounds, which are independent of the number of processors, are given for these classes and for anti-tree-systems.

Key words. nonpreemptive scheduling, list scheduling, LP-schedules, worst-case analysis, performance bounds, critical-path algorithm, multiprocessing

1. Introduction. In this paper we consider special classes of task systems to be nonpreemptively scheduled on $m \geq 2$ identical processors. The tasks are thought to have no additional resource requirements like memory space, etc. To find an optimal schedule with minimal schedule length, or equivalently with maximal processor utilization, is known to be polynomially complete for most of these problems [17]. Thus a lot of work has been done to investigate heuristic algorithms producing suboptimal schedules. One of these heuristics is the LP-algorithm (longest path), which has also been called CP-algorithm (critical path), or LPT-algorithm (largest processing time).

There are several results concerning LP-scheduling, if all tasks have equal execution times. In this case the LP-schedule turns out to be optimal if the dependency structure is a tree [10] or an anti-tree [16]. For so called n -free task dependency structure the ratio of length of the LP-schedule and an optimal schedule is bounded by $\frac{3}{2}$ [15]. For arbitrary dependency structure the bound is $\frac{4}{3}$ for two processors [2] and $2 - 1/(m - 1)$ for $m > 2$ processors [1], [12]. In this case a special LP-algorithm, due to Coffman and Graham, generates optimal schedules for two processors [4], and the worst case is bounded by $2 - 2/m$, for $m > 2$ processors [13]. The difference between these bounds and the general bound for list scheduling $2 - 1/m$ [7] is significant only in those cases where m is small.

If unequal execution time task systems are considered, then for arbitrary dependency structure LP-scheduling does not improve the general worst-case behavior [8], [9]. However, it turns out to be rather good if the tasks are independent [5], [7], [9], even in the case of uniform processor systems [6], [14]. For m identical processors the bound is $\frac{4}{3} - 1/3m$ [7].

In this paper we examine the worst-case behavior of unequal execution time task systems with three kinds of dependency structures—trees, anti-trees, and chains. We prove that for tree-systems the bound is $2 - 2/(m + 1)$, as conjectured in [9]. For anti-tree-systems we cannot give the exact bound, but we show that this bound is generally worse than the bound for tree-systems. That means, both bounds converge to the factor 2 with a growing number of processors. However, for the class of chain-systems, the intersection of the classes of tree- and anti-tree systems, we can determine the worst-case-bound which is asymptotically limited by $\frac{5}{3}$. That is, in a certain sense

* Received by the editors August 31, 1978, and in final revised form May 7, 1980.

† Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, Olshausenstr. 40-60, D-2300 Kiel 1, West Germany.

chain systems form a maximal class of task systems which are good for nonpreemptive LP-scheduling.

2. The model. The scheduling model we want to use is a restricted version of the one which is described comprehensively in [3]. A *task system* $G = (\mathcal{T}, <, \mu)$ for a given set $P = \{P_1, \dots, P_m\}$ of $m \geq 2$ identical processors is defined as follows:

1. $\mathcal{T} = \{T_1, \dots, T_n\}$ is a set of tasks to be executed.
2. $<$ is an irreflexive partial order defined on \mathcal{T} which specifies operational precedence constraints. $T_i < T_j$ means that T_i must be completed before T_j can start.
3. $\mu : \mathcal{T} \rightarrow \mathbb{R}_+ = \{x/x \text{ real}, x > 0\}$ is a map where $\mu(T)$ denotes the *execution time* of a task T on a processor P_i .

For a task system G let $IP(T) = \{T'/T' \in \mathcal{T}, T' < T, \text{ there is no } T'' \in \mathcal{T} \text{ with } T' < T'' < T\}$ be the *set of all immediate predecessors* of a task T , and $IS(T) = \{T'/T' \in \mathcal{T}, T \in IP(T')\}$ the *set of all immediate successors*. For a set M the cardinality is denoted by $|M|$.

Generally a schedule $S_{G,m}$ for m processors and a given task system G is a description of the work to be done by each processor at each moment of time. In our case we say: A nonpreemptive *schedule* for m identical processors and a task system G is a map $S = S_{G,m} : \mathbb{R} \rightarrow \mathcal{P}(\mathcal{T})$ with

1. $S(a) = \emptyset$, for $a < 0$.
2. $|S(a)| \leq m$, for all $a \in \mathbb{R}$.
3. There exists a minimal real number $\omega(S)$ such that for all $a \geq \omega(S)$ we have $S(a) \neq \emptyset$. $\omega(S)$ is the *total execution time* of S or the *schedule length* of S .
4. $\bigcup_{0 \leq a < \omega(S)} S(a) = \mathcal{T}$.
5. Let $t_S(T) = \inf_{T \in S(a)} a$ be the *starting time* of a task T . Then $T \in S(a)$ iff $t_S(T) \leq a < t_S(T) + \mu(T)$. That is, if a task T has started, then its execution is not allowed to be interrupted.
6. $t_S(T) + \mu(T) \leq t_S(T')$, for all $T', T < T'$.

Point 2 means that at most m tasks are allowed to be executed in parallel. Nothing is said about whether a task T is worked at processor P_i or P_j . This is not important, because all processors are identical.

For brevity we write $t(T)$ instead of $t_S(T)$, if it is clear which schedule S is meant. A task T' is *available* or *ready for execution* at time a , if $t(T) + \mu(T) \leq a$, for all $T \in IP(T')$.

A simple way to generate schedules is by using lists. A list is a permutation of the tasks in \mathcal{T} , $L = (T_{i_1}, \dots, T_{i_n})$. Each time a processor P_i becomes free for assignment, the list is scanned from left-to-right, until the first task T is found which is available. The task T is then assigned. If there is no such task, the processor becomes idle. If more than one processor becomes free at the same time, we assume that the task T is assigned to that processor with the smallest index.

A schedule $S_{G,m}$ is called *optimal* (with respect to the total execution time) if $\omega(S_{G,m}) \leq \omega(S'_{G,m})$ for all possible schedules $S'_{G,m}$. An optimal schedule is denoted by $SOP = SOP_{G,m}$. It is well known that there may be no optimal schedule among all schedules generated by lists. This is caused by the fact that list scheduling does not allow processors to be idle while tasks are available. But normally quite good results are derived by list scheduling. A special kind is the LP-scheduling discipline where the tasks are ordered according to their length or level.

The *length* of a task T is given by

$$l(T) = \mu(T) \quad \text{if } IS(T) = \emptyset, \quad \text{and}$$

$$l(T) = \mu(T) + \max_{T' \in IS(T)} l(T') \quad \text{otherwise.}$$

The length of a set $N \subseteq \mathcal{T}$ is defined as $l(\emptyset) = 0$, and

$$l(N) = \max_{T \in N} l(T) \quad \text{if } N \neq \emptyset.$$

A list of tasks $L = (T_{i_1}, \dots, T_{i_n})$ is called an LP-list, if $l(T_{i_j}) \geq l(T_{i_k})$, for all j and k , with $1 \leq j < k \leq n$. A list-schedule S generated by a LP-list is called a LP-schedule and denoted by SL.

Now let us classify special sets of tasks. A task system G is called

- a) an *independent system* ($G \in \text{IND}$) iff
for all $T \in \mathcal{T}$, $\text{IP}(T) = \text{IS}(T) = \emptyset$.
- b) a *chain-system* ($G \in \text{CHAIN}$) iff
for all $T \in \mathcal{T}$ $|\text{IP}(T)| \leq 1$ and $|\text{IS}(T)| \leq 1$.
- c) a *tree-system* ($G \in \text{TREE}$) iff
for all $T \in \mathcal{T}$ $|\text{IS}(T)| \leq 1$.
- d) a *anti-tree-system* ($G \in \text{ATREE}$) iff
for all $T \in \mathcal{T}$ $|\text{IP}(T)| \leq 1$.

A *chain* is a subset of tasks $C = \{T_{i_1}, \dots, T_{i_k}\}$ with $\text{IP}(T_{i_1}) = \emptyset$, $\text{IS}(T_{i_k}) = \emptyset$, and $T_{i_j} \in \text{IS}(T_{i_{j-1}})$, for $j = 2, \dots, k$.

3. Results. If F is a set of task systems, then the *worst-case behavior* of F with respect to LP-scheduling on m homogeneous processors is defined by

$$\text{WCB}_m(F) = \sup_{G \in F} \left(\frac{\omega(\text{SL}_{G,m})}{\omega(\text{SOP}_{G,m})} \right).$$

It is known that there are sets F for which we get $\text{WCB}_m(F) = 2 - 1/m$ [8]. That is, for those sets there is no difference in worst-case behavior between LP-scheduling and arbitrary list scheduling [7]. But the situation changes if independent tasks are considered.

THEOREM 1 [7].

$$\text{WCB}_m(\text{IND}) = \frac{4}{3} - \frac{1}{3m}, \quad m \geq 2.$$

This result shows that LP-scheduling is quite good for independent systems. Moreover, a generalization of this result [5] demonstrates that better bounds can be obtained in many cases.

In [9] an example of a tree-system, due to G. S. Graham, is given showing that the ratio $\omega(\text{SL})/\omega(\text{SOP})$ can be arbitrarily close to $2 - 2/(m+1)$. In the next section we prove this to be a general bound for tree-systems, and therefore we can state the following.

THEOREM 2.

$$\text{WCB}_m(\text{TREE}) = 2 - \frac{2}{(m+1)}.$$

The proof of this theorem is essentially based on a result for tree systems given in [11]:

$$\frac{\omega(\text{SL})}{\omega(\text{SOP})} \leq 1 + (m-1) \frac{\max \{\mu(T) / T \in \mathcal{T}\}}{\sum_{T \in \mathcal{T}} \mu(T)}.$$

One can easily check that this bound is worse than $2 - 2/(m+1)$ for certain tree-systems. Nevertheless the above result shows that the LP-algorithm is quite a good

heuristic, if the execution times of single tasks are small compared with the sum of all execution times in the tree-system.

Reversing all precedence constraints (arc directions) in a tree-system we get an anti-tree-system. Though we cannot give the exact worst-case behavior of anti-tree-systems, the next theorem states some lower bounds for $WCB_m(\text{ATREE})$. For some m it turns out that worst-case behavior of anti-tree-systems is slightly worse than that of tree-systems.

THEOREM 3.

a) If $m = 2^k - 1$ for an integer $k \geq 2$, then

$$WCB_m(\text{ATREE}) \geq 2 - \frac{2}{m+1}.$$

b) There are integers m' with

$$WCB_{m'}(\text{ATREE}) > 2 - \frac{2}{m'+1}.$$

The proof is given by the general examples of § 7. The bounds given in Theorems 2 and 3 are both near to 2 if the number of processors m is large. A fundamentally better bound can be achieved for chain-systems, the intersection of tree- and anti-tree-systems.

THEOREM 4.

$$WCB_m(\text{CHAIN}) = 1 + \max_{\substack{1 \leq k \leq (m+1)/3, \\ k \text{ an integer}}} \frac{(m-2k+1)(2k-1)}{(m-2k+1)(3k-1) + k^2}.$$

It will be shown in § 5 that the right side of the above equation is a bound for chain-systems. In § 6 a general example shows that this bound is tight.

Obviously we get the bound $1 + (m-1)/(2m-1)$ for $m = 2, 3, 4$. Moreover, we can get the following Theorem 4', allowing an easier computation of $WCB_m(\text{CHAIN})$ than Theorem 4. Because of its purely algebraic nature the proof is omitted here. The interested reader can find some hints in the appendix. Let x be a real number, then $\lfloor x \rfloor = \max \{i/i \text{ an integer}, i \leq x\}$ and $\lceil x \rceil = \min \{i/i \text{ an integer}, i \geq x\}$.

THEOREM 4'.

a) For $m = 2, 3, 4$ we have $WCB_m(\text{CHAIN}) = 1 + (m-1)/(2m-1)$.

b) For $m \geq 5$ let $x_m = (m+1/2m)(\sqrt{1+2m-1})$, and $A_m = \{\lfloor x_m \rfloor, \lceil x_m \rceil\}$; then

$$WCB_m(\text{CHAIN}) = 1 + \max_{k \in A_m} \frac{(m-2k+1)(2k-1)}{(m-2k+1)(3k-1) + k^2}.$$

The next theorem gives upper bounds for various classes of task systems considered in this paper. These bounds are independent of the number of processors and might especially illustrate the result of Theorem 4. Let $WCB(F) = \lim_{m \rightarrow \infty} WCB_m(F)$, define the *asymptotic worst-case behavior* of a given set F of task systems.

THEOREM 5.

a) $WCB(\text{IND}) = \frac{4}{3}$,

b) $WCB(\text{CHAIN}) = \frac{5}{3}$,

c) $WCB(\text{TREE}) = 2$,

d) $WCB(\text{ATREE}) = 2$.

a), c), and d) are obvious by Theorem 1, Theorem 2 and Theorem 3. That $\{WCB_m(\text{CHAIN})\}$ converges to $\frac{5}{3}$ is derived from Theorem 4' and shown in the

appendix. The result of Theorem 5 is of special interest. Note that the following relations hold:

$$\text{IND} \subsetneq \text{CHAIN} \begin{matrix} \subsetneq \text{TREE} \\ \subsetneq \text{ATREE} \end{matrix}$$

and

$$\text{CHAIN} = \text{TREE} \cap \text{ATREE}.$$

It is easy to observe that the asymptotic worst-case behavior is still 2, if binary tree-systems or binary anti-tree-systems, that is, $|\text{IP}(T)| \leq 2$ or $|\text{IS}(T)| \leq 2$, respectively, are scheduled. But 2 is the asymptotic worst-case behavior for an algorithm which produces an arbitrary list schedule [7], [8], [9]. That means that there is no difference with respect to the asymptotic bound if we consider arbitrary list schedules or LP-schedules for tree- and anti-tree-systems. In this sense a chain-system is a maximal structure for which LP-scheduling is suboptimal.

4. Bounds for tree-systems. Let $G = (\mathcal{T}, <, \mu)$ be a tree-system and $S = S_{G,m}$ a list schedule. In the following we will write $t(T)$ instead of $t_S(T)$, if it is clear which schedule S is meant.

Let $d(T) = d_S(T)$ denote the minimal time at which T is available. That is, $d(T) = 0$, if T has no predecessor, and $d(T) = \max\{t(T') + \mu(T') \mid T' \in \text{IP}(T)\}$ otherwise. Now define $R = \{T \mid T \in \mathcal{T}, t(T) > d(T)\}$ to be the set of all those tasks which are ready for execution at a certain point of time, but are not assigned because other tasks are preferred. If $R = \emptyset$, then the number of tasks without any predecessor is less than or equal to the number of processors, because G is a tree-system. In this case the schedule is optimal, of course. If there is a T in R , then note that all processors must be busy during $[0, t(T))$, since G is a tree-system.

Before proving Proposition 1 we give two preliminary lemmas which are used in the next section, too.

LEMMA 1. *Let S be a LP-schedule for a tree-system G . If $T \in R$, then for all T'' with $t(T'') < t(T)$ we have $l(T'') \geq l(T)$.*

Proof. Assume that there are tasks in R for which the lemma is wrong. Let Q be the subset of all those tasks, and let $T \in Q$ be a task with minimal starting time:

(1) $t(T) \leq t(T^+)$, for all $T^+ \in Q$.

(2) Let T'' be a task with $t(T'') < t(T)$, and $l(T'') < l(T)$.

T is ready for execution during $[d(T), t(T)]$. That means, for all T^+ with $d(T) \leq t(T^+) < t(T)$ we get $l(T^+) \geq l(T)$ because of LP-scheduling. Therefore we have

(3) $0 \leq t(T'') < d(T)$ and $\text{IP}(T) \neq \emptyset$.

The set $M = \{T^+ \mid t(T^+) = d(T)\}$ is not empty, because at least one $T' \in \text{IP}(T)$ finishes execution and there are tasks ready for execution. As G is a tree-system, all $T' \in \text{IP}(T)$ have at most one immediate successor, namely T , and hence all tasks in M are independent of all tasks in $\text{IP}(T)$. At time $d(T)$, $|M|$ tasks start their executions and for $|M|$ tasks execution is finished. Therefore there must be a task $T^* \in M$ with $d(T^*) < t(T^*)$, hence $T^* \in R$. From (3) and (2) follows $t(T'') < d(T) = t(T^*)$ and $l(T'') < l(T) \leq l(T^*)$. But $T^* \in Q$ and $t(T^*) < t(T)$ contradict (1).

For an arbitrary schedule S define $N = \{T \mid T \in \mathcal{T}, t(T) + l(T) = \omega(S)\}$. Obviously $N \neq \emptyset$, because there is at least one task T with $t(T) + \mu(T) = t(T) + l(T) = \omega(S)$. Let $R_0 = N \cap R$. If $R_0 = \emptyset$, then for all T in N we have $d(T) = l(T)$. That means, in the case $\text{IP}(T) \neq \emptyset$ there is a $T' \in \text{IP}(T)$ with $t(T') + \mu(T') = t(T)$, and thus $T' \in N$. As there is

only a finite number of tasks in the system, there must be a $T \in N$ with $IP(T) = \emptyset$. From $T \notin R$ we get $t(T) = 0$, hence $\omega(S) = l(T) = \omega(SOP)$.

Summarizing the above discussion we can state:

LEMMA 2. *If S is a nonpreemptive schedule with $R_0 = \emptyset$, then S is optimal.*

Now we are going to prove Theorem 2. Some basic ideas are similar to those which are used by Kaufman in [11].

PROPOSITION 1.

$$\frac{\omega(SL)}{\omega(SOP)} \leq 1 + \frac{(m-1)}{(m+1)}.$$

Proof. Let G be a tree-system and SL be a LP-schedule for G on m identical processors. Because of Lemma 2 $R_0 \neq \emptyset$ is assumed. From the definition of R and R_0 we know that $r = \max\{t(T)/T \in R_0\} \neq 0$ and $l(SL(x))/m = r$, for all $x, 0 \leq x < r$. With $q = \omega(SL) - r$ we get $\omega(SOP) \geq r + q/m$. If $r \geq q$, then

$$\frac{\omega(SL)}{\omega(SOP)} \leq 1 + \frac{\left(1 - \frac{1}{m}\right)q}{r + \frac{q}{m}} \leq 1 + \frac{m-1}{m+1}.$$

Now suppose $r < q$. By definition of r there exists a subchain C with length $l(C) = q$, starting time r and finishing time $\omega(SL) = r + q$. Lemma 1 implies that each task T of set $A_1 = \{T/t(T) + \mu(T) \leq r\}$ or of set $A_2 = \{T/t(T) < r < t(T) + \mu(T)\}$ has length $l(T) \geq l(C) = q$.

To get a lower bound for $\omega(SOP)$ we transform G into a system G' in the following way.

Split up each $T \in A_2$ into a subchain $T' < T''$ with $\mu(T') = r - t(T)$ and $\mu(T'') = \mu(T) - \mu(T')$. Let B consist of all those T' and of all tasks in A_1 . Note that for each $T \in B$ we have $\mu(T) \leq r$ and $l(T) \geq q$. Thus each $T \in B$ has a successor T'' with $l(T'') = l(T) - \mu(T) \geq q - r$. Moreover, by construction we have $\sum_{T \in B} \mu(T) = rm$.

Furthermore, if there is a task T in the subchain C with $l(T) > q - r > l(T) - \mu(T)$, then split up T into $T' < T''$ with $\mu(T') = l(T) - (q - r)$ and $\mu(T'') = \mu(T) - \mu(T')$. Note that $l(T'') = q - r$. If there is no such task, then let T'' denote that task in C with length $l(T'') = q - r$. Let D consist of all those tasks T of this new chain having length $l(T) > q - r$. Obviously T'' is a successor for each $T \in D$ and $\sum_{T \in D} \mu(T) = r$.

Thus each task $T \in B \cup D$ has at least one successor T'' with $l(T'') \geq q - r$. Since in an optimal schedule SOP' for G' there is at least one such T'' after executing all tasks in $B \cup D$, we conclude

$$\begin{aligned} \omega(SOP) &\geq \omega(SOP') \geq \left(\frac{1}{m}\right) \sum_{T \in B \cup D} \mu(T) + q - r \\ &= \frac{rm + r}{m} + q - r = q + \frac{r}{m} > r + \frac{r}{m}. \end{aligned}$$

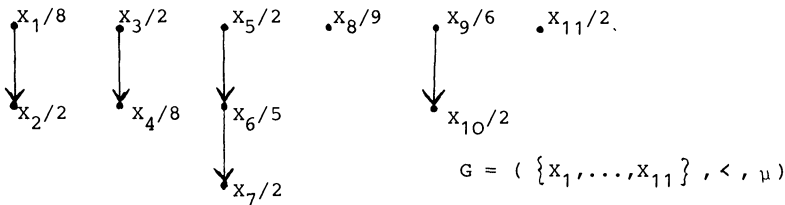
Hence

$$\frac{\omega(SL)}{\omega(SOP)} \leq \frac{r + q}{q + \frac{r}{m}} < 1 + \frac{m-1}{m+1}.$$

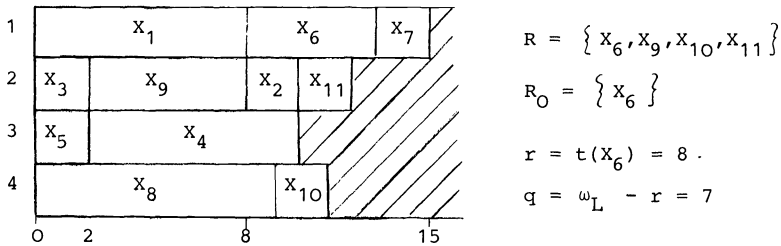
5. Bounds for chain-systems. In this section we give a proof for Theorem 4. Since every chain-system $G = (\mathcal{T}, <, \mu)$ is a tree-system, we are allowed to use some results of § 4. For a LP-schedule SL let R_0 be defined as before. From Lemma 2 we know that $R_0 = \emptyset$ yields $\omega(\text{SL})/\omega(\text{SOP}) = 1$. Therefore $R_0 \neq \emptyset$ is assumed for the remaining part of this section. As above let r and q be real numbers defined by $r = \max \{t_{\text{SL}}(T)/T \in R_0\}$ and $q = \omega_L - r$, where ω_L is an abbreviation for $\omega(\text{SL})$. Moreover, we shall use ω_{op} instead of $\omega(\text{SOP})$.

To have a good estimation for a minimal schedule length for a task system G we now construct a new task system G' in the following way (compare Fig. 1):

1. Remove all chains C with $l(C) < q$. Note that Lemma 1 states that such a chain is not started in SL before the time point r .



$(X_1, X_3, X_5, X_8, X_9, X_4, X_6, X_2, X_7, X_{10}, X_{11})$ generates the LP-schedule:



i	C_i	$l(C_i)$	l_i	T_i^*	$l(T_i^*)$	$\mu(T_i^*)$	$t(T_i^*)$	a_i	$U_i < V_i$	$\mu(V_i)$	t_i
1	X_1, X_2	10	3	X_1	10	8	0	3	yes	5	3
2	X_3, X_4	10	3	X_4	8	8	2	1	yes	7	3
3	X_5, X_6, X_7	9	2	X_6	7	5	8	0	no	5	8
4	X_8	9	2	X_8	9	9	0	2	yes	7	2
5	X_9, X_{10}	8	1	X_9	8	6	2	1	yes	5	3

$$H = \{3\}, \quad I = \{1, \dots, 5\}, \quad J = \{1, 2, 4, 5\}$$

$i \in J$	$t(T_i^*) + \mu(T_i^*)$	$T_i < W_i$	$\mu_i = \mu(T_i)$
1	8	no	5
2	10	yes	5
4	9	yes	6
5	8	no	5

FIG. 1. Example for a chain-system.

2. Let $m+k$, $k \geq 1$, be the number of chains with length greater than or equal to q . These chains are called *main chains*. (Note that $k \leq 0$ yields $R_0 = \emptyset$.) In each of these chains C_i , $1 \leq i \leq m+k$, there is a task T_i^* with

$$t_{\text{SL}}(T_i^*) \leq r \quad \text{and} \quad l(T_i^*) \geq q > l(T_i^*) - \mu(T_i^*).$$

For $i = 1, \dots, m+k$ let

$$a_i = l(T_i^*) - q \quad \text{and} \quad t_i = t_{\text{SL}}(T_i^*) + a_i.$$

If $a_i = 0$, then rename $V_i = T_i^*$.

If $a_i \neq 0$, then split up T_i^* into a subchain of two tasks $U_i < V_i$ with

$$\mu(U_i) = a \quad \text{and} \quad \mu(V_i) = \mu(T_i^*) - a_i.$$

Let $H = \{i/t_i = r\}$ be the set of indices of those V_i which "start in SL" at time r . Rename $T_i = V_i$ for all $i \in H$. Let $J = \{1, \dots, m+k\} - H = \{i/t_i < r\}$ be the set of indices of those V_i which "start" before r . For $i \in J$ do the following:

If $t_{\text{SL}}(T_i^*) + \mu(T_i^*) \leq r$, then $T_i = V_i$ and $\mu_i = \mu(T_i) = \mu(V_i)$.

If $t_{\text{SL}}(T_i^*) + \mu(T_i^*) > r$, then split up V_i into a subchain $T_i < W_i$ with

$$\mu_i = \mu(T_i) = r - t_i = r - t_{\text{SL}}(T_i^*) - a_i,$$

and

$$\mu(W_i) = -r + t_{\text{SL}}(T_i^*) + \mu(T_i^*).$$

Note that for all T_i , $i = 1, \dots, m+k$, we get $l(T_i) = q$.

For brevity we write $I = \{1, \dots, m+k\}$. For all $i \in I$ let

$$l_i = l(C_i) - l(T_i) = l(C_i) - q.$$

The following lemmas are evident.

LEMMA 3. If ω_{op} is the schedule length of an optimal schedule for G , and ω'_{op} that for G' , then $\omega_{\text{op}} \geq \omega'_{\text{op}}$.

LEMMA 4.

$$\sum_{i \in I} l_i + \sum_{i \in J} \mu_i = rm.$$

We now state some technical results which will be used frequently in the sequel. Lemmas 5 and 6 give some insight into the dependence of the various quantities given by the construction of G' . Both results will be used in the proofs of Lemma 7 and 8 which give lower bounds for ω_{op} . These bounds are needed to state Proposition 2 containing the fundamental part of Theorem 4. Moreover, Proposition 3 will make use of Lemma 7. Let $h = /H/$ be the number of those subchains with length q starting at time r .

LEMMA 5. Let A and B be disjoint subsets of J . Then

$$\sum_{i \in A} (l_i + \mu_i) \geq r(/A/ + /B/ + h - k) - \sum_{i \in B} (l_i + \mu_i) - \sum_{i \in H} l_i.$$

Proof. From $l_i + \mu_i \leq r$ for all $i \in J$ and Lemma 4 we get

$$\begin{aligned} rm &= \sum_{i \in I} l_i + \sum_{i \in J} \mu_i \\ &= \sum_{i \in H} l_i + \sum_{i \in A} (l_i + \mu_i) + \sum_{i \in B} (l_i + \mu_i) + \sum_{i \in J - (A \cup B)} (l_i + \mu_i) \\ &\leq \sum_{i \in H} l_i + \sum_{i \in A} (l_i + \mu_i) + \sum_{i \in B} (l_i + \mu_i) + r(/J/ - /A/ - /B/). \end{aligned}$$

Thus, with $/J/ = m + k - h$,

$$\begin{aligned} \sum_{i \in H} l_i + \sum_{i \in A} (l_i + \mu_i) + \sum_{i \in B} (l_i + \mu_i) &\geq r(m - /J/ + /A/ + /B/) \\ &= r(/A/ + /B/ + h - k). \end{aligned}$$

LEMMA 6. *If $m \geq k - h > 0$, then there exists a set $B \subseteq J$ with $/B/ = 2(k - h)$ and $\sum_{i \in B} \mu_i \leq (k - h)r$.*

Proof. Let s_0, s_1 , and s_2 be respectively the number of processors to which 0, 1, and at least 2 tasks T_i^* , $i \in J$, are assigned. Let F be the set of indices of those tasks T_i^* , $i \in J$, assigned to the s_2 processors.

Suppose $f = /F/ \geq 2(k - h)$. If $s_2 \leq k - h$, then pick any subset B of F , $/B/ = 2(k - h)$. Then $\sum_{i \in B} \mu_i \leq s_2 r \leq (k - h)r$. If $s_2 > k - h$ and B' is a set of indices of tasks assigned to $k - h$ of the s_2 processors, then $/B'/ \geq 2(k - h)$. In this case pick a subset B of B' to fulfill the lemma.

On the other hand, if $f < 2(k - h)$, we can take B to consist of F and any $2(k - h) - f$ indices of the rest of J . Then $\sum_{i \in B} \mu_i \leq (s_2 + 2(k - h) - f)r$. But $s_0 + s_1 + s_2 = m$ and $f + s_1 = m + k - h$ imply $f - s_2 \geq k - h$, thus $\sum_{i \in B} \mu_i \leq (k - h)r$.

Now let $x = \omega_L - \omega_{op}$, that is, $\omega_{op} = r + q - x$. The following two lemmas give lower bounds for ω_{op} depending on this difference x .

LEMMA 7. *If $m > k$ and $r \geq q$, then $m(q - x) \geq (2k - 1)q - (k - 1)r$.*

Proof. Obviously from Lemma 4

$$\begin{aligned} (1) \quad m(r + q - x) &\geq \sum_{i \in I} (l_i + q) = \sum_{i \in I} l_i + \sum_{i \in J} (\mu_i + q - \mu_i) + hq \\ &= rm + \sum_{i \in J} (q - \mu_i) + hq. \end{aligned}$$

In the case $k - h > 0$ let B be a subset of J satisfying Lemma 6. Then we get $\sum_{i \in J} (q - \mu_i) \geq \sum_{i \in B} (q - \mu_i) \geq 2(k - h)q - (k - h)r$, and with (1) and $r \geq q$,

$$\begin{aligned} (2) \quad m(q - x) &\geq 2(k - h)q - (k - h)r + hq = (2k - h)q - (k - h)r \\ &= (2k - 1)q - (k - 1)r + (h - 1)(r - q) \\ &\geq (2k - 1)q - (k - 1)r. \end{aligned}$$

On the other side, if $h \geq k \geq 1$, then (1) and $r \geq q$ immediately imply

$$\begin{aligned} (3) \quad m(q - x) &\geq hq \geq kq = (2k - 1)q - (k - 1)q \\ &\geq (2k - 1)q - (k - 1)r. \end{aligned}$$

LEMMA 8. *If $m \geq 3k - 1$, then $\omega_{op} \geq x \cdot k / (2k - 1) + q$.*

Proof. Assume that $\omega_{op} = r + q - x < xk / (2k - 1) + q$ or equivalently

$$(1) \quad r < \left(\frac{k}{2k - 1} + 1 \right) x.$$

In a given optimal schedule SOP let t_i denote the starting time of each task T_i , $i \in I$. If there is an $i \in I$ with $t_i \geq xk/(2k-1)$, then $\omega_{\text{op}} \geq t_i + l(T_i) \geq xk/(2k-1) + q > \omega_{\text{op}}$, which is a contradiction. We may therefore assume that

$$(2) \quad t_0 = \max_{i \in I} t_i < \frac{xk}{2k-1}.$$

If $k-h > 0$, then let $B \subseteq J$ be a set with $|B| = 2(k-h)$ and $\sum_{i \in B} \mu_i \leq (k-h)r$. The existence of such a set is guaranteed by Lemma 6.

Three sets of indices are defined in the following way:

$$(3) \quad D = \begin{cases} I - (H \cup B) & \text{if } k-h > 0, \\ I - H = J & \text{otherwise;} \end{cases}$$

$$E = \{i/i \in D, l_i + \mu_i \leq t_0\};$$

$$F = \{i/i \in D, l_i + \mu_i > t_0\}.$$

Let be $d = |D|$, $e = |E|$, and $f = |F|$. Note that

$$(4) \quad |E'| \cdot t_0 - \sum_{i \in E'} (l_i + \mu_i) \geq 0,$$

for each subset E' of E . Now set $r_i = l_i + \mu_i$ for each $i \in F$ and $r_0 = t_0$. Furthermore assume that $F = \{1, \dots, f\}$, provided $F \neq \emptyset$, and

$$r_1 \leq r_2 \leq \dots \leq r_f.$$

This is always possible by rearranging the indices. Now we prove the following statement.

(5) If $k > h$, then there is a subset A of D with $|A| = k$ and

$$\sum_{i \in H \cup B} (l_i + q) \leq (2k-h)\omega_{\text{op}} - \sum_{i \in A} (l_i + \mu_i) + kt_0.$$

If $e \geq k$, then take any subset A of $E \subseteq D$ with $|A| = k$, and by (4) we immediately get (5).

On the other side consider $e < k$. Then we have

$$\begin{aligned} f &= d - e = m + k - (2k-h) - e \\ &= m - k + h - e > 3k-1 - k + h - k \geq k. \end{aligned}$$

We know that all tasks T_i , $i \in F$, start on execution not later than t_0 and are ready with execution not sooner than r_i . Thus, in the time interval $[r_{i-1}, r_i)$, $1 \leq i \leq f$, all tasks T_j , with $i \leq j \leq f$, are assigned to some processor. Therefore at most $m - (f - i + 1)$ processors are available during $[r_{i-1}, r_i)$ for the tasks of the main chains C_j , $j \in B \cup H$. Set $a = k - e$, and note that for $i \leq a$

$$\begin{aligned} m - (f - i + 1) &= m - (d - e - i + 1) \\ &= m - (m - k + h - e - i + 1) \\ &= k - h + e + i - 1 < 2k - h. \end{aligned}$$

Generally the $2k-h$ main chains C_j , $j \in B \cup H$, may be assigned to at most $2k-h$ processors. But this is only possible during the time intervals $[0, t_0)$ and $[r_a, \omega_{\text{op}})$ as

shown above. Thus we conclude that the following inequality must hold:

$$\begin{aligned}
\sum_{i \in BUH} l(C_i) &\leq (2k-h)t_0 + (2k-h)(\omega_{op} - r_a) + \sum_{i=1}^a (k-h+e+i-1)(r_i - r_{i-1}) \\
&= (2k-h)(\omega_{op} + t_0 - r_a) + \sum_{i=1}^a i(r_i - r_{i-1}) + (k-h+e-1) \sum_{i=1}^a (r_i - r_{i-1}) \\
&= (2k-h)(\omega_{op} + t_0 - r_a) + ar_a - \sum_{i=1}^{a-1} r_i - r_0 + (k-h+e-1)(r_a - r_0) \\
(6) \quad &= (2k-h)\omega_{op} + (k-(e-1))(t_0 - r_a) + ar_a - t_0 - \sum_{i=1}^{a-1} r_i \\
&= (2k-h)\omega_{op} + (a+1)t_0 - (a+1)r_a + ar_a - t_0 - \sum_{i=1}^{a-1} r_i \\
&= (2k-h)\omega_{op} + at_0 - \sum_{i=1}^a r_i.
\end{aligned}$$

Set $A = E \cup \{1, \dots, a\}$. Then from (4) and (6) follows

$$\begin{aligned}
\sum_{i \in BUH} (l_i + q) &\leq (2k-h)\omega_{op} + at_0 - \sum_{i=1}^a r_i + et_0 - \sum_{i \in E} (l_i + \mu_i) \\
&= (2k-h)\omega_{op} + kt_0 - \sum_{i \in A} (l_i + \mu_i),
\end{aligned}$$

and (5) is proved. In view of Lemma 5 and (5) we conclude

$$\begin{aligned}
\sum_{i \in BUH} l_i + (2k-h)q &= \sum_{i \in BUH} l(C_i) \\
&\leq (2k-h)\omega_{op} + kt_0 - \sum_{i \in A} (l_i + \mu_i) \\
&\leq (2k-h)\omega_{op} + kt_0 - r(A) + r(B) + r(H) - k + \sum_{i \in B} (l_i + \mu_i) + \sum_{i \in H} l_i.
\end{aligned}$$

By Lemma 6 follows

$$\begin{aligned}
(2k-h)q &\leq (2k-h)\omega_{op} + kt_0 - r(2k-h) + \sum_{i \in B} \mu_i \\
&\leq (2k-h)(r+q-x) + kt_0 - r(2k-h) + (k-h)r,
\end{aligned}$$

and with (1) and (2),

$$\begin{aligned}
 0 &\leq -(2k-h)x + kt_0 + (k-h)r \\
 &< -(2k-h)x + k\left(\frac{xk}{2k-1}\right) + (k-h)\left(1 + \frac{k}{2k-1}\right)x \\
 &= -(2k-h)x + (k-h)x + \frac{k}{2k-1}(k+k-h)x \\
 &= -kx + k\frac{2k-h}{2k-1}x \\
 &\leq -kx + kx = 0,
 \end{aligned}$$

which is a contradiction. Thus for this case the assumption (1) cannot be true.

If $h \geq k$, then there is a subset A of D with $|A| = k$ and

$$(7) \quad \sum_{i \in H} l(C_i) \leq h\omega_{\text{op}} - \sum_{i \in A} (l_i + \mu_i) + kt_0.$$

The proof technique is the same as that for (5). If $e \geq k$, then take any subset A of E with $|A| = k$, and (7) follows immediately from (4).

If $e < k$, then define $a = k - e$. Note that (3) and $m \geq h$ yield $f = m + k - h - e \geq k - e > 0$. During the time interval $[r_{i-1}, r_i]$, $1 \leq i \leq f$, all tasks T_j with $i \leq j \leq f$ are assigned to some processor. Thus for the h main chains C_i , $i \in H$, only $m - (f - i + 1)$ processors are available. Analogously to (6) we conclude;

$$\begin{aligned}
 \sum_{i \in H} l(C_i) &\leq ht_0 + h(\omega_{\text{op}} - r_a) + \sum_{i=1}^a (h - k + e - 1 + i)(r_i - r_{i-1}) \\
 &= h(\omega_{\text{op}} - r_a + t_0) + (h - k + e - 1)(r_a - r_0) + ar_a - \sum_{i=1}^{a-1} r_i - r_0 \\
 (8) \quad &= h\omega_{\text{op}} - (a + 1)(r_a - t_0) + ar_a - \sum_{i=1}^{a-1} r_i - t_0 \\
 &= h\omega_{\text{op}} - \sum_{i=1}^a r_i + at_0.
 \end{aligned}$$

With (4) and (8) it is easily checked on that $A = E \cup \{1, \dots, a\}$ fulfills (7).

From (2), (7) and Lemma 5 we derive

$$\begin{aligned}
 \sum_{i \in H} l_i + hq &= \sum_{i \in H} l(C_i) \\
 &\leq h(r + q - x) - r(k + h - k) + \sum_{i \in H} l_i + kt_0,
 \end{aligned}$$

and thus $0 < -hx + k(k/(2k-1))x \leq 0$, and this is impossible. Therefore the assumption (1) cannot be true, and the lemma is proved.

Now we can state the main part of the proof for Theorem 4.

PROPOSITION 2. *Let $G = (\mathcal{J}, <, \mu)$ be a chain-system to be scheduled on $m \geq 2$ processors. If $m + k$, $1 \leq k \leq (m + 1)/3$, is the number of main chains in G , then*

$$\frac{\omega_L}{\omega_{\text{op}}} \leq 1 + \frac{(m - 2k + 1)(2k - 1)}{(m - 2k + 1)(3k - 1) + k^2}.$$

Proof. Set $\omega_{\text{op}} = r + q - x$. If $x = 0$, then there is nothing to prove. So if $x > 0$, then there is a real number $b > 0$ with

$$(1) \quad \omega_{\text{op}} = bx + q,$$

or equivalently

$$r = (1 + b)x.$$

Thus we can write

$$(2) \quad \frac{\omega_L}{\omega_{\text{op}}} = \frac{r + q}{r + q - x} = 1 + \frac{x}{bx + q}.$$

Lemma 8 states that

$$(3) \quad b \geq \frac{k}{2k - 1}.$$

If $q > r$, then $q > (1 + b)x$, and with (2) and (3) we get

$$(4) \quad \begin{aligned} \frac{\omega_L}{\omega_{\text{op}}} &< 1 + \frac{x}{2bx + x} \leq 1 + \frac{2k - 1}{4k - 1} \\ &= 1 + \frac{(m - 2k + 1)(2k - 1)}{(m - 2k + 1)((3k - 1) + k)} \\ &\leq 1 + \frac{(m - 2k + 1)(2k - 1)}{(m - 2k + 1)(3k - 1) + k^2}. \end{aligned}$$

In the case $r \geq q$, Lemma 7 gives

$$(5) \quad (q - x)m \geq (2k - 1)q - (k - 1)r.$$

With (1) follows $q(m - 2k + 1)/m \geq x - (k - 1)(1 + b)x/m$, and thus

$$(6) \quad q \geq \frac{m}{m - 2k + 1} \left(1 - \frac{k - 1}{m} (1 + b) \right) x.$$

Therefore from (2) and (6) we conclude

$$(7) \quad \begin{aligned} \frac{\omega_L}{\omega_{\text{op}}} - 1 &= \frac{x}{bx + q} \leq \frac{x}{bx + \frac{m}{m - 2k + 1} \left(1 - \frac{k - 1}{m} (1 + b) \right) x} \\ &= \frac{m - 2k + 1}{b(m - 2k + 1) + m - (k - 1)(1 + b)} \\ &= \frac{m - 2k + 1}{b(m - 3k + 2) + m - k + 1}. \end{aligned}$$

Since $m - 3k + 2 \geq 0$ and $b \geq k/(2k - 1)$, we have

$$\begin{aligned}
 \frac{\omega_L}{\omega_{op}} - 1 &\leq \frac{(m - 2k + 1)(2k - 1)}{k(m - 3k + 2) + (2k - 1)m - (2k - 1)(k - 1)} \\
 (8) \qquad &= \frac{(m - 2k + 1)(2k - 1)}{m(3k - 1) - 5k^2 + 5k - 1} \\
 &= \frac{(m - 2k + 1)(2k - 1)}{(m - 2k + 1)(3k - 1) + k^2}.
 \end{aligned}$$

PROPOSITION 3. *If there are $m + k$ main chains and $m \leq 3k - 2$, then*

$$\frac{\omega_L}{\omega_{op}} \leq 1 + \frac{m - 1}{2m - 1}.$$

Proof. As above let $\omega_L = r + q$, $\omega_{op} = r + q - x$, and $x > 0$. Note that

$$(1) \qquad \frac{m - k}{m + k} \leq \frac{3m - m - 2}{3m + m - 2} = \frac{m - 1}{2m - 1}.$$

First assume that $\omega_L \leq 2q$. Then $\omega_{op} \geq (m + k)q/m$ and (1) imply

$$\frac{\omega_L}{\omega_{op}} \leq \frac{2mq}{(m + k)q} = 1 + \frac{m - k}{m + k} \leq 1 + \frac{m - 1}{2m - 1}.$$

In the case $\omega_L > 2q$, let be $r = q + y$, $y > 0$. By $r + q/m \leq \omega_{op} = r + q - x$ we get $mx \leq (m - 1)q$. Thus for $y - x \geq -q/m$ we may conclude

$$\frac{\omega_L}{\omega_{op}} = 1 + \frac{x}{2q + y - x} \leq 1 + \frac{(m - 1)q}{2mq - q} = 1 + \frac{m - 1}{2m - 1}.$$

If $y - x < -q/m$, then $(m + k)q \leq \omega_{op} \cdot m \leq 2mq - q$, and thus $k < m$. Therefore we are allowed to use Lemma 7 claiming $m(q - x) \geq (2k - 1)q - (k - 1)r = kq - (k - 1)y$, or equivalently

$$(2) \qquad (m - k)q \geq mx - (k - 1)y.$$

Thus from (2), $3k - 2 - m \geq 0$, and $y < x$, we get

$$\begin{aligned}
 (3) \qquad (m - k)\omega_{op} &= (m - k)(2q + y - x) \\
 &\geq 2mx - 2(k - 1)y + (m - k)y - (m - k)x \\
 &= (m + k)x - (3k - 2 - m)y \geq 2(m - k + 1)x.
 \end{aligned}$$

Therefore from $\omega_{op} \geq 2x + 2x/(m - k) \geq (2m - 1)x/(m - 1)$ we get

$$\frac{\omega_L}{\omega_{op}} = 1 + \frac{x}{\omega_{op}} \leq 1 + \frac{m - 1}{2m - 1}.$$

Note that the bound given in Proposition 3, $1 + (m - 1)/(2m - 1)$, is the same as in Proposition 2, if k is restricted to $k = 1$. That is, to describe worst-case behavior we only have to consider chain-systems with $m + k$ main chains, where $1 \leq k \leq (m + 1)/3$. This will be done in the next section.

Remark. In [5] there is given a generalized bound on LP-scheduling for independent systems. A more detailed result for chain-systems may be formulated in the following way:

If there is a chain-system with $m + k$ main chains, then

$$\frac{\omega_L}{\omega_{op}} \leq 1 + \frac{(m-2k+1)(2k-1)}{(m-2k+1)(3k-1)+k^2} \quad \text{for } 1 \leq k \leq (m+1)/3;$$

$$\frac{\omega_L}{\omega_{op}} \leq 1 + \frac{m-1}{2m-1} \quad \text{for } (m+1)/3 < k \leq m-1;$$

$$\frac{\omega_L}{\omega_{op}} \leq 1 + \frac{m-1}{m+k} \quad \text{for } k > m-1.$$

6. Examples for chain-systems. In this section we show that the bound given by Proposition 2 is the best possible one. First of all note that $1 \leq k \leq (m+1)/3$ implies $2k \leq m$, hence $-2k - 2m \leq -m - 4k$, and

$$4km - 2k - 2m + 1 \leq 4km - m - 4k + 1,$$

or

$$(2k-1)(2m-1) \leq (4k-1)(m-1),$$

and thus

$$\frac{2k-1}{4k-1} \leq \frac{m-1}{2m-1}.$$

Therefore $m = 3k - 1$ yields

$$\begin{aligned} \frac{\omega_L}{\omega_{op}} &\leq 1 + \frac{(m-2k+1)(2k-1)}{(m-2k+1)(3k-1)+k^2} = 1 + \frac{k(2k-1)}{k((3k-1)+k)} \\ &= 1 + \frac{2k-1}{4k-1} \leq 1 + \frac{m-1}{2m-1}. \end{aligned}$$

That is, to find the best bound we only have to consider the cases where

$$(1) \quad 1 \leq k < \frac{(m+1)}{3}.$$

But it is easy to find a chain-system which describes the worst-case behavior for the case $k = (m+1)/3$. Now define

$$\begin{aligned} (2) \quad x &= (2k-1)(m-2k+1), \\ r &= (3k-1)(m-2k+1), \\ q &= (2k-1)(m-k+1) - (k-1)k = (2k-1)m - (k-1)(3k-1), \\ y &= (k-1)(m-2k+1) = \left(\frac{k-1}{2k-1}\right)x, \\ z &= r - x = k(m-2k+1) = \left(\frac{k}{2k-1}\right)x. \end{aligned}$$

Note that $m > 3k - 1 \geq 1$ implies $x > 0$, $r > 0$, and $q > 0$. It is easy to see that

$$(3) \quad \begin{aligned} r - q &= k(m - 3k + 1) > 0, \\ q - x &= k^2 > 0, \\ z + y &= x \quad \text{and} \quad z + q = r + q - x. \end{aligned}$$

For brevity we set $p = m - 2k + 1$. Now let $G = (\mathcal{T}, <, \mu)$ be a chain-system which is defined as follows:

$$1. \quad \mathcal{T} = \{T_i / i = 1, \dots, k\} \cup \{U_{i,j} / i = 1, \dots, p, j = 1, \dots, r - q + 1\} \\ \cup \{V_{i,j} / i = 1, \dots, k, j = 0, \dots, y + 1\} \\ \cup \{W_{i,j} / i = 1, \dots, k - 1, j = 1, 2\}.$$

2. For $i = 1, \dots, p$ let

$$U_{i,1} < U_{i,2} < \dots < U_{i,r-q+1} = U'_i, \\ \mu(U_{i,j}) = 1 \quad \text{for } j = 1, \dots, r - q, \quad \mu(U'_i) = q.$$

3. For $i = 1, \dots, k$ let

$$\mu(T_i) = r$$

and

$$V_i^* = V_{i,0} < V_{i,1} < \dots < V_{i,y} < V_{i,y+1} = V'_i, \quad \text{with} \\ \mu(V_i^*) = z, \quad \mu(V_{i,j}) = 1 \quad \text{for } j = 1, \dots, y, \quad \mu(V'_i) = q - x.$$

Note that $l(V_i^*) = z + y + q - x = q$.

4. For $i = 1, \dots, k - 1$ let

$$W_{i,1} < W_{i,2}, \quad \text{with} \\ \mu(W_{i,1}) = x \quad \text{and} \quad \mu(W_{i,2}) = q - x.$$

Note that $l(W_{i,1}) = q$.

If $k = 1$, then the set $\{W_{i,j}\}$ is empty, $y = 0$, and $z = x$. The only thing to do is to assign the tasks V_1^* and V'_1 to processor P_1 (compare Fig. 2).

Obviously the schedule shown in Fig. 2 is a LP-schedule, and so we have

$$\omega_L = r + q.$$

Now we show that the schedule given by Fig. 3 is optimal. There are only two little problems to be understood.

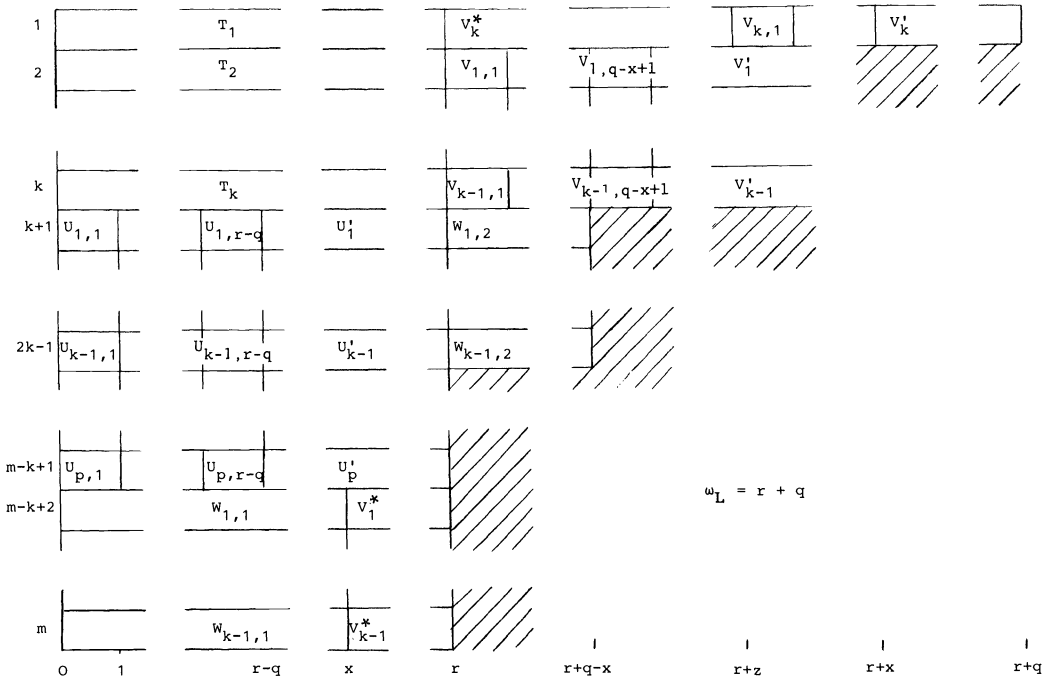


FIG. 2. LP-schedule for a chain-system.

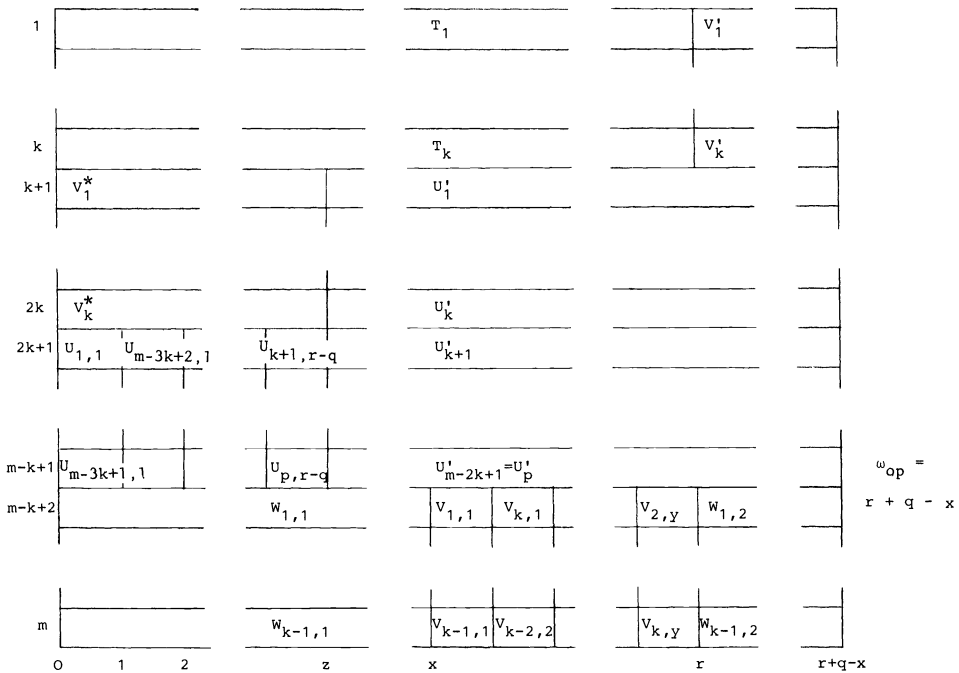


FIG. 3. Optimal schedule for a chain-system.

- For $\mu(U_{i,j}) = 1$, for all $i = 1, \dots, p$ and $j = 1, \dots, r - q$ and

$$\begin{aligned} \sum_{i=1}^p \sum_{j=1}^{r-q} \mu(U_{i,j}) &= p(r - q) \\ &= (m - 2k + 1)k(m - 3k + 1) \\ &= (m - 3k + 1)z, \end{aligned}$$

it is possible to execute these $p(r - q)$ tasks in z time units on $m - 3k + 1$ processors.

- Similarly, if $k > 1$, then ky tasks $V_{i,j}$ can be executed on the $k - 1$ processors P_{m-k+2}, \dots, P_m in the time interval $[x, r)$. This is an immediate consequence of the following equation:

$$\sum_{i=1}^k \sum_{j=1}^y \mu(V_{i,j}) = ky = k(k - 1)p = (k - 1)(r - x).$$

Thus we have by (2) and (3)

$$\frac{\omega_L}{\omega_{op}} = 1 + \frac{x}{r + q - x} = 1 + \frac{(2k - 1)(m - 2k + 1)}{(3k - 1)(m - 2k + 1) + k^2}.$$

6. Examples for anti-tree-systems. The general structure for all examples is shown in Fig. 4. For an integer $k \geq 2$ an anti-tree-system $G = (\mathcal{T}, <, \mu)$ is defined by

- $\mathcal{T} = \{V_i / i = 1, \dots, k - 1\} \cup \{U_i / i = 1, \dots, k\}$
 $\cup \{T_{i,j} / i = 0, \dots, k - 2, j = 1, \dots, m - 1 - i\}$
 $\cup \{F_i / i = 1, \dots, m - 1\}.$

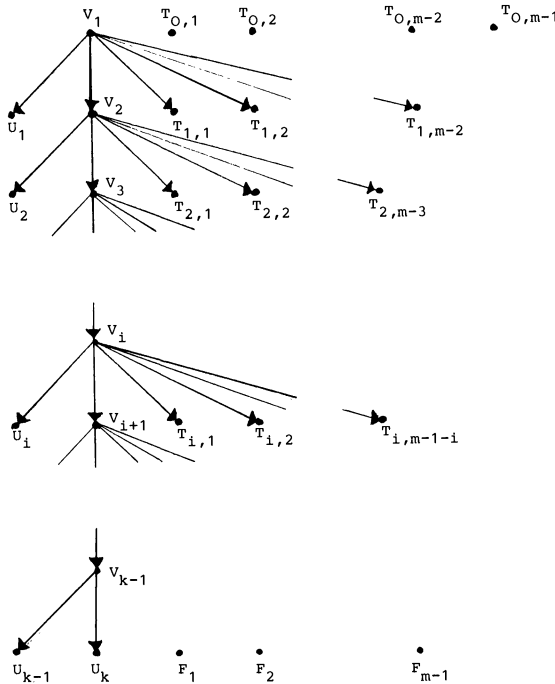


FIG. 4. Structure of an anti-tree-system.

2. For $i = 1, \dots, k-2$, let

$$\begin{aligned} V_i &< U_i, \\ V_i &< T_{i,j} \quad \text{for } j = 1, \dots, m-1-i, \\ V_{k-1} &< U_{k-1} \quad \text{and} \quad V_{k-1} < U_k. \end{aligned}$$

3. Let $m = 2^k - 1$. Then the execution times of the tasks are defined as follows:

$$\begin{aligned} \mu(V_i) &= \frac{\varepsilon}{k-1} && \text{for } i = 1, \dots, k-1, \\ \mu(U_i) &= 2^{k-1} - \varepsilon \cdot \frac{i-1}{k-1} && \text{for } i = 1, \dots, k, \\ \mu(T_{i,j}) &= 2^i && \text{for } i = 0, \dots, k-2; j = 1, \dots, m-1-i, \\ \mu(F_i) &= \varepsilon && \text{for } i = 1, \dots, m-1. \end{aligned}$$

Note that for $i = 1, \dots, k-2$

$$l(U_i) = \mu(U_i) = \frac{\varepsilon}{k-1} + \mu(U_{i+1}) = l(V_{i+1}).$$

The ε is thought to be real and less than $1/2k$. The LP-list $L = (V_1, U_1, V_2, U_2, \dots, U_{k-1}, U_k, T_{k-2,1}, \dots, T_{0,m-1}, F_1, \dots, F_{m-1})$ generates a LP-schedule SL with

$$\omega_L = \omega(\text{SL}) = \sum_{i=0}^{k-2} 2^i + 2^{k-1} - \varepsilon = 2^k - 1 - \varepsilon.$$

SL is shown in Fig. 5.

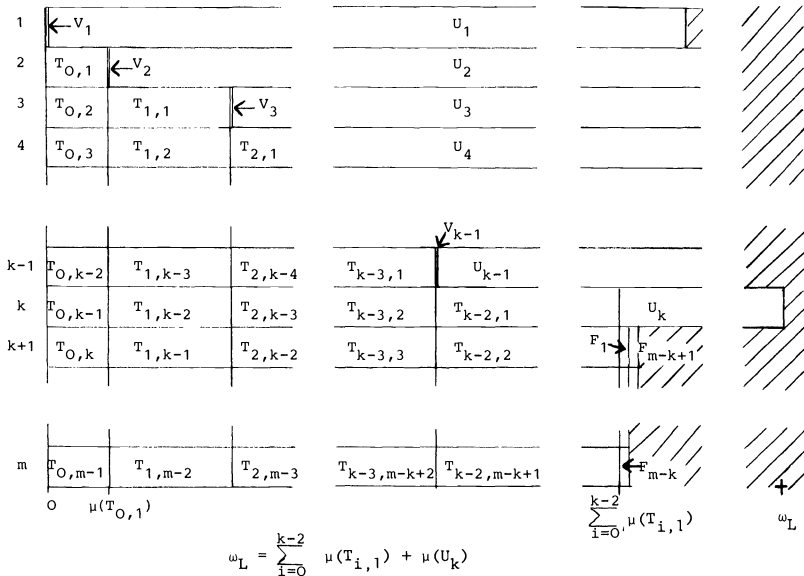


FIG. 5. LP-schedule of an anti-tree-system.

A list which generates an optimal schedule is, for example,

$$L_{\text{op}} = (V_1, \dots, V_{k-1}, F_1, \dots, F_{m-1}, U_1, \dots, U_{k-1}, T_{k-2,1}, T_{k-2,2}, \dots, T_{0,m-1}).$$

The optimal schedule is shown in Fig. 6. For the schedule length we get

$$\omega_{\text{op}} = 2^{k-1} + \varepsilon.$$

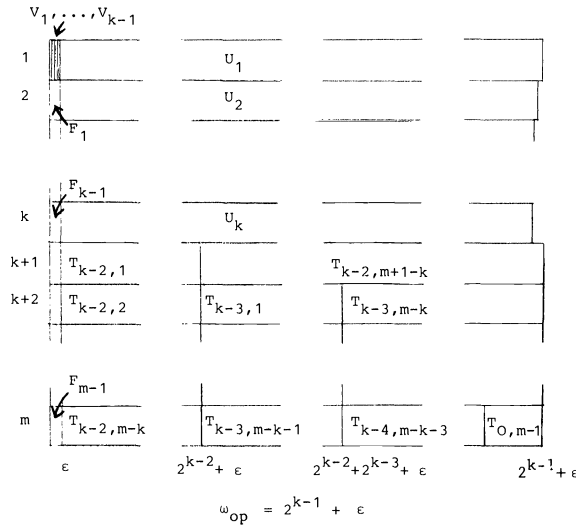


FIG. 6. Optimal schedule for an anti-tree-system.

The problem is to show that it is possible to execute the set of tasks $\{T_{i,j} / i = 0, \dots, k-2, j = 1, \dots, m-1-i\}$ on $m-k$ processors in 2^{k-1} time units. Note that

$$\begin{aligned} \sum_{i=0}^{k-2} (k-1-i)2^i &= \sum_{i=0}^{k-2} \sum_{j=0}^i 2^j = \sum_{i=0}^{k-2} (2^{i+1} - 1) \\ &= 2(2^{k-1} - 1) - (k-1) \\ &= 2^k - 1 - k \\ &= m - k. \end{aligned}$$

Thus we can conclude

$$\begin{aligned} \sum_{i=0}^{k-2} \sum_{j=1}^{m-1-i} \mu(T_{i,j}) &= \sum_{i=0}^{k-2} (m-1-i)2^i = (m-k) \sum_{i=0}^{k-2} 2^i + \sum_{i=0}^{k-2} (k-1-i)2^i \\ &= (m-k)(2^{k-1} - 1) + m - k \\ &= (m-k)2^{k-1}. \end{aligned}$$

Therefore we get

$$\begin{aligned} \frac{\omega_L}{\omega_{op}} &= \frac{2 \cdot 2^{k-1} - 1 - \varepsilon}{2^{k-1} + \varepsilon} = 2 - \frac{1 + 3\varepsilon}{2^{k-1} + \varepsilon} = 2 - \frac{2 + 6\varepsilon}{2^k + 2\varepsilon} \\ &\xrightarrow{\varepsilon \rightarrow 0} 2 - \frac{2}{2^k} = 2 - \frac{2}{m+1} = 1 + \frac{m-1}{m+1}. \end{aligned}$$

But there are anti-tree-systems and certain m -processor-systems such that $\omega_L/\omega_{op} > 2 - 2/(m+1)$. For example consider the case where $m = 10$ and $k = 3$. For the task system $G = (\mathcal{T}, <, \mu)$ let be the task set \mathcal{T} , the structure, and the indices the same as in the example above. The execution times of the tasks are defined by

$$\begin{aligned} \mu(U_i) &= 6 - \frac{i-1}{2} \cdot \varepsilon && \text{for } i = 1, 2, 3, \\ \mu(T_{0,j}) &= 2 && \text{for } j = 1, \dots, 9, \\ \mu(T_{1,j}) &= 3 && \text{for } j = 1, \dots, 8, \\ \mu(V_i) &= \frac{\varepsilon}{2} && \text{for } i = 1, 2, \\ \mu(F_i) &= \varepsilon && \text{for } i = 1, \dots, 9. \end{aligned}$$

For the LP-schedule as shown in Fig. 5 we then have

$$\omega_L = \mu(T_{0,1}) + \mu(T_{1,1}) + \mu(U_3) = 11 - \varepsilon.$$

It is easily shown that an optimal schedule has schedule length

$$\omega_{op} = 6 + \varepsilon.$$

Thus we have for $\varepsilon \rightarrow 0$

$$\frac{\omega_L}{\omega_{op}} \rightarrow \frac{11}{6} = 2 - \frac{1}{6} > 2 - \frac{2}{11} = 2 - \frac{2}{m+1}.$$

8. Conclusions. In this paper we have analyzed worst-case behavior of longest path schedules for three classes of dependency structures—trees, anti-trees and chains. The exact bounds for tree- and chain-systems and the asymptotic bounds for all of the classes were determined. However, determining the exact bound for anti-tree-systems is still an open question.

Appendix. For an integer $m \geq 5$ let $g_m(x) = (m - 2x + 1)(2x - 1)$ and $h_m(x) = (m - 2x + 1)(3x - 1) + x^2$ be two functions defined on $E = \{x/.5 < x < (m + 1)/3 + .5\}$. Then for all $x \in E$ we have $h_m(x) > g_m(x) > 0$, and thus $f_m(x) = g_m(x)/h_m(x) > 0$ is differentiable on E . Let x_m defined by $x_m = (\sqrt{1 + 2m - 1})((m + 1)/2m)$. We want to show that

$$(1) \quad a_m = \max_{\substack{1 \leq k \leq (m+1)/3 \\ k \text{ an integer}}} f_m(k) = \max \{f_m(\lfloor x_m \rfloor), f_m(\lceil x_m \rceil)\}, \quad \text{and}$$

$$(2) \quad \lim_{m \rightarrow \infty} a_m = \frac{2}{3}.$$

We get the first derivative of f_m by the well-known formula $f'_m = (g'_m \cdot h_m - g_m \cdot h'_m)/(h_m)^2$. It is easily shown that $(h_m(x))^2 \cdot f'_m(x) = -2mx^2 - 2(m+1)x + (m+1)^2$, and thus $f'_m(x) \geq 0$ iff $x^2 + ((m+1)/m)x - (m+1)^2/(2m) \leq 0$. That is, $f_m(x)$ has only one local maximum on E in x_m . A short computation gives $1 \leq \lfloor x_m \rfloor \leq x_m \leq \lceil x_m \rceil \leq (m+1)/3$. Moreover, note that $f'_m(x) > 0$ iff $.5 < x < x_m$, $f'_m(x) = 0$ iff $x = x_m$, and $f'_m(x) < 0$ iff $x_m < x < (m+1)/3 + .5$. Thus to get the maximum value for $f_m(k)$, if only integers are allowed, it must be those nearest to the real maximum. But these are $\lfloor x_m \rfloor$ and $\lceil x_m \rceil$, and (1) is proved.

Now let $k_m, \lfloor x_m \rfloor \leq k_m \leq \lceil x_m \rceil$, be that integer satisfying $a_m = f_m(k_m)$. Evidently we have

$$(3) \quad a_m = \frac{(m-2k_m+1)(2k_m-1)}{(m-2k_m+1)(3k_m-1)+k_m^2} \leq \frac{2k_m-1}{3k_m-1} \leq \frac{2}{3}.$$

Moreover, we get $x_m \leq \sqrt{1+2m}-1 \leq 2\sqrt{m}-1$ and thus $k_m \leq 2\sqrt{m}$. If $m \geq 64$, then $\sqrt{m}-4 \geq \sqrt{m}/2$ and therefore

$$\frac{k_m^2}{m-2k_m+1} \leq \frac{4m}{m-4\sqrt{m}} \leq \frac{4\sqrt{m}}{\sqrt{m}-4} \leq 8.$$

This yields

$$a_m = \frac{2k_m-1}{3k_m-1 + \frac{k_m^2}{m-2k_m+1}} \geq \frac{2k_m-1}{3k_m+7} \quad \text{for } m \geq 64.$$

Since $\{k_m\}$ is unbounded, we get $\lim_{m \rightarrow \infty} a_m \geq \frac{2}{3}$; hence with (3) we have (2).

Acknowledgment. The author wishes to thank the referees for their helpful comments. In particular, the proof of Lemma 7 is due to one of the referees.

REFERENCES

- [1] N. F. CHEN, *An Analysis of Scheduling Algorithms in Multiprocessor Computing Systems*, Dept. of Computer Science, University of Illinois, Urbana, 1975.
- [2] N. F. CHEN AND C. L. LIU, *On a class of scheduling algorithms for multiprocessing computing systems*, Proc. 1974 Sagamore Computer Conference on Parallel Processing, T. Feng, ed., Springer, Berlin, 1975, pp. 1-16.
- [3] E. G. COFFMAN, JR., *Introduction to deterministic scheduling theory*, Computer and Job/Shop Scheduling Theory, E. G. Coffman, ed., John Wiley, New York, 1976, pp. 1-50.
- [4] E. G. COFFMAN, JR. AND R. L. GRAHAM, *Optimal scheduling for two processor systems*, Acta Informatica, 1 (1972), pp. 200-213.
- [5] E. G. COFFMAN, JR. AND R. SETHI, *A generalized bound on LPT sequencing*, Proc. Int. Symp. Comptr., Performance Modelling, Measurement and Evaluation, 1976, pp. 306-310.
- [6] T. GONZALEZ, O. H. IBARRA AND S. SAHNI, *Bounds for LPT schedules on uniform processors*, SIAM J. Comput., 6 (1977), pp. 155-166.
- [7] R. L. GRAHAM, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416-429.
- [8] ———, *Bounds on multiprocessing anomalies and related packing algorithms*, AFIPS Conference Proceedings Vol. 40, 1972, pp. 205-217.
- [9] R. L. GRAHAM, *Bounds on the performance of scheduling algorithms*, Computer and Job/Shop Scheduling Theory, E. G. Coffman, ed., John Wiley, New York, 1976, pp. 165-227.
- [10] T. C. HU, *Parallel sequencing and assembly line problems*, Operations Res., 9 (1961), pp. 841-848.
- [11] M. T. KAUFMAN, *An almost-optimal algorithm for the assembly line scheduling problem*, IEEE Trans. Comp., C-23 (1974), pp. 1169-1174.

- [12] M. KUNDE, *Beste Schranken beim LP-scheduling*, Institut für Informatik und Prakt. Math., Bericht 7603, Universität Kiel, 1976.
- [13] S. LAM AND R. SETHI, *Worst case analysis of two scheduling algorithms*, this Journal, 6 (1977), pp. 518–536.
- [14] J. W. S. AND C. L. LIU, *Bounds on scheduling algorithms for heterogeneous computing systems*, Proc. IFIP 74, North-Holland, Amsterdam, 1974, pp. 349–353.
- [15] L. NETT, *On scheduling algorithms for n-free task dependency structures*, Proceedings of the 1977 International Conference on Parallel Processing, J.-L. Baer, ed., 1977, pp. 100–107.
- [16] S. SCHINDLER, *On optimal schedules for multiprocessor systems*, Proc. of Princeton Conf. on Information, Sciences and Systems, 1972, pp. 219–223.
- [17] J. D. ULLMAN, *NP-complete scheduling problems*, J. Comput. System Sci., 10 (1975), pp. 384–393.

OPERATOR PRECEDENCE GRAMMARS AND THE NONCOUNTING PROPERTY*

STEFANO CRESPI-REGHIZZI†, GIOVANNI GUIDA† AND DINO MANDRIOLI†

Abstract. The notion of noncounting language, initially introduced for regular languages recognized by counter-free finite machines, and recently extended to parenthesized context-free languages, is here further studied for general (i.e., nonparenthesized) context-free languages.

While weakly equivalent context-free grammars do not, in general, fall in the same class with respect to the noncounting property, it is shown by a complex proof that weakly equivalent operator precedence grammars are all counting or all noncounting (a property which distinguishes the operator precedence languages from classical deterministically parsable families).

Key Words. context-free, noncounting, operator precedence, parsing, derivation

1. Introduction. Counter-free machines and their languages (noncounting languages) have been widely investigated in the literature of formal language theory [1], [8], [9], [10], [14], [18]. Intuitively, a counter-free machine is a finite-state automaton without the capacity of counting modulo $n > 1$; i.e., which cannot distinguish two strings which differ only in the number of repetitions of some substring.

Noncounting regular languages (or events in [9]) have been proved to have several interesting algebraic properties and relations with other classes of languages; e.g., they are characterized by a syntactic monoid having only trivial subgroups, they are the closure with respect to Boolean operations and concatenation of locally testable events, they can be characterized by star-free regular expressions [9], [15], [16], [17].

More recently [1], it has been observed that the noncounting property can be found even in languages which are not described by a regular grammar but by a context-free grammar; e.g., in no natural language is a noun phrase required to have an odd number of adjectives; Algol does not impose writing programs with a number of blocks which multiplies, say, 3 [1], [23]. It seems therefore a reasonable goal to extend the notion of noncounting languages to the context-free model.

In [1], we have formalized the notion of noncounting for parenthesis context-free languages [2], [4], [5] or, equivalently, sets of trees recognized by tree-automata [11], [12], [13], as a natural extension of the notion of noncounting events stated in [9].

It was realized that some properties of noncounting events naturally extend to noncounting context-free parenthesis languages and grammars, notably the characterization in terms of permutation-free graphs [1], [2], [9], while for other properties several difficulties, if not altogether impossibilities, arise [1].

In this paper, we raise the question whether it is well-posed to define noncounting general (i.e., not parenthesized) context-free languages; the problem is shown to be a nontrivial one. The answer is shown to be positive for operator precedence languages. The main result states that the noncounting property for an operator precedence language does not depend on the particular structure (parenthesization) of the language.

As a by-product, new algebraic properties of operator-precedence languages are added to other previously studied ones [3], [6], [7].

* Received by the editors April 17, 1979 and in revised form June 25, 1980. This work was supported by CNR.

† Istituto di Elettrotecnica ed Elettronica del Politecnico di Milano-Politecnico di Milano, Piazza Leonardo da Vinci, 32, 1-20133 Milan, Italy.

The paper is organized as follows. In § 2, basic formal definitions are given; in § 3, the noncounting property is briefly discussed and the main result is introduced; § 4 is devoted to the proof of the main theorem, and § 5 to the discussion of the result and of possible extensions.

2. Basic definitions. Let $G = (N, T, P, S)$ be a *context-free* (CF) grammar, where N is the nonterminal alphabet, T the terminal alphabet (not containing “[” or “]”), P the production set and $S \subseteq N$ the set of the axioms of G . The *parenthesis (P) grammar associated* to G is $\tilde{G} = (N, \tilde{T}, \tilde{P}, S)$, where $\tilde{T} = T \cup \{[,]\}$, $\tilde{P} = \{A \rightarrow [\alpha] \mid A \rightarrow \alpha \in P\}$ [1], [2]. G is the CF grammar *underlying* \tilde{G} .

We adhere to the following conventions, unless otherwise stated. Nonterminal characters are denoted by upper case italic letters, terminal characters by lower case italic letters from the beginning of the alphabet, terminal strings by lower case italic letters from the end of the alphabet, and strings over $N \cup T$ by lower case Greek letters. ε denotes the empty string. Furthermore, $*x$ will denote any cyclic permutation of x : different occurrences of $*x$ will not denote in general the same permutation of x , unless the contrary is explicitly stated.

Let $|\alpha|$ denote the length of the string α . For a CF(P) grammar G and a nonterminal A , $A \xrightarrow{i}_G \alpha$ denotes a derivation of G of length i and $A \xrightarrow{*}_G \alpha$ (resp. $A \xrightarrow{+}_G \alpha$) the reflexive and transitive (resp. transitive) closure of $\xrightarrow{1}_G$.

Let $L(A) = \{x \mid A \xrightarrow{*}_G x, x \in T^*\}$, and $L(G) = \bigcup_{S_i \in S} L(S_i)$, the *language* (resp. *P language*) generated by a CF grammar (resp. P grammar) G . Assuming that for a CF grammar $G = (N, T, P, S)$, each production of P is labeled by an identification number in \mathbb{N} (where \mathbb{N} is the set of natural numbers), $A \xrightarrow{\pi}_G \alpha$, with $\pi \in \mathbb{N}^+$, denotes a parsing [4] corresponding to the derivation $A \xrightarrow{*}_G \alpha$. Let T be a terminal alphabet not containing “[” or “]” and $\tilde{T} = T \cup \{[,]\}$. We define the homomorphism $h: \tilde{T} \rightarrow T$, such that

$$h(a) = a \quad \text{for any } a \in T,$$

$$h("[") = h("]") = \varepsilon.$$

The *stencil* [2] of a string $\alpha \in (T \cup N)^+$ is the string $u = f(\alpha)$ in $(T \cup \{-\})^+$, where “-” is a new character and f is the homomorphism $f(a) = a$ for any $a \in T$, $f(A) = -$ for any $A \in N$. For a string α , $^{(1)}\alpha$ (resp. $\alpha^{(1)}$) denotes the leftmost (resp. rightmost) terminal character of α .

Two grammars G_1 and G_2 are *weakly equivalent* (resp. *strongly equivalent*) iff $L(G_1) = L(G_2)$ (resp. $L(\tilde{G}_1) = L(\tilde{G}_2)$).

A string $\tilde{z} \in \tilde{T}^*$ is *well-parenthesized* (w.p.) iff there exists a P grammar \tilde{G} such that

$$S_i \xrightarrow{*}_{\tilde{G}} \tilde{z}, \quad S_i \in S.$$

A CF (P) grammar is *backwards-deterministic* (BD) or *invertible* iff no two productions have the same right part [2]. A CF(P) grammar is *reduced* iff no two distinct nonterminals A_1 and A_2 are equivalent (A_1 and A_2 are equivalent [2] if for every context $\alpha - \beta$ such that $\alpha\beta$ is w.p., either both $\alpha A_1 \beta$ and $\alpha A_2 \beta$ are derivable from some $S_i \in S$ in the grammar or neither one is derivable) and it has no useless nonterminal [2]. It is known from [2] that:

STATEMENT 1. *Every P grammar has a strongly equivalent BD reduced P grammar effectively obtainable from it.* \square

We refer to [3], [4], [6] for the definition of *operator precedence* (OP) grammar and we recall from [6] the definition of *Fischer Normal Form* (FNF) and of *structural equivalence*.

An OP Grammar $G = (V_N, V_T, P, S_0)$, where $S_0 \in V_N$, is in FNF iff

- (1) G is invertible;
- (2) S_0 does not occur in the right part of any production of P ;
- (3) there are no renaming rules except those with left part S_0 (if any).

Structural equivalence is the same as strong equivalence (v.s.), except that it refers to the following slightly different definition of parenthesization of an OP grammar,

$$\hat{G} = (N, \hat{T}, \hat{P}, S_0),$$

where

$$\hat{T} = T \cup \{[,]\},$$

$$\hat{P} = \{A \rightarrow [\alpha] \mid A \rightarrow \alpha \in P \text{ and } \alpha \notin N\},$$

(that is, renaming rules are not parenthesized).

It is known [6] that:

STATEMENT 2. *For each OP grammar a structurally equivalent OP FNF can be effectively constructed.* \square

In this paper, we assume a slightly different definition of OP grammar; namely, consistently with the above definition of CF grammar, we allow an OP grammar to have a set of axioms $S \subseteq N$, instead of exactly one axiom as in [3], [4]. As a consequence, the FNF, now devoid of renaming rules [6], will be called a *modified FNF* (mFNF).

Statement 2 holds with the clause “FNF” substituted by “mFNF” and, therefore, *without any loss in generality we shall consider in the sequel only OP grammars in mFNF.*

Note that for OP grammars in mFNF the two concepts of strong and structural [6] equivalence exactly match.

An *operator precedence parenthesis* (OPP) grammar is a P grammar such that its underlying CF grammar is OP (in mFNF).

STATEMENT 3. *Every OPP grammar G (in mFNF) has a strongly equivalent BD reduced OPP grammar (in mFNF) effectively obtainable from it.* \square

Proof. The hypothesis G is in mFNF implies that G is BD and does not contain useless nonterminals. An algorithm is well known [2] for obtaining from G a strongly equivalent BD grammar G' without equivalent [2] nonterminals. Since G and G' are strongly equivalent, their precedence relations on T are identical by virtue of an obvious extension to mFNF-grammars of a result given in [7, Statement 2.3] for FNF-grammars. \square

Let $\tilde{G} = (N, \tilde{T}, \tilde{P}, S)$ be a P grammar. The language $L(\tilde{G})$, or, equivalently, \tilde{G} itself, is *noncounting* (NC) iff there exists an integer $n > 0$, such that for any $\tilde{x}, \tilde{v}, \tilde{w}, \tilde{u}, \tilde{y} \in \tilde{T}^*$, such that $\tilde{w}, \tilde{v}\tilde{w}\tilde{u}$ are w.p., and for any integer $m > 0$, $\tilde{x}\tilde{v}^n\tilde{w}\tilde{u}^n\tilde{y} \in L(\tilde{G})$ iff $\tilde{x}\tilde{v}^{n+m}\tilde{w}\tilde{u}^{n+m}\tilde{y} \in L(\tilde{G})$ [1].

It is a straightforward consequence of the properties proved in [1] that we have the following:

STATEMENT 4. *$L(\tilde{G})$ is NC iff there exists an integer $\tilde{n} > 0$ such that, for any $\tilde{x}, \tilde{v}, \tilde{w}, \tilde{u}, \tilde{y} \in \tilde{T}^*$ such that $\tilde{w}, \tilde{v}\tilde{w}\tilde{u}$ are w.p., and for any integer $n \geq \tilde{n}$, $\tilde{x}\tilde{v}^n\tilde{w}\tilde{u}^n\tilde{y} \in L(\tilde{G})$ implies $\tilde{x}\tilde{v}^{n+1}\tilde{w}\tilde{u}^{n+1}\tilde{y} \in L(\tilde{G})$.* \square

Let \tilde{G} be a P grammar.

A derivation $A \xrightarrow{\tilde{G}}^+ \tilde{\alpha}$ is cyclic iff $\tilde{\alpha} = \tilde{\beta}A\tilde{\gamma}$.

A cyclic derivation is counting iff it is of the type $A \xrightarrow{\tilde{G}}^+ \tilde{\alpha}^mA\tilde{\beta}^m$ with $m > 1$ integer, and no derivation exists $A \xrightarrow{\tilde{G}}^+ \tilde{\alpha}A\tilde{\beta}$ [1].

It is known [1]:

STATEMENT 5. Let \tilde{G} be a BD reduced P grammar. $L(\tilde{G})$ is NC iff no derivation of G is counting. \square

LEMMA 1. Let \tilde{G} be a BD reduced P grammar. There exists an integer $Q > 0$ such that $L(\tilde{G})$ is NC iff there exists an integer $n > 0$ such that, for any $\tilde{x}, \tilde{v}, \tilde{w}, \tilde{u}, \tilde{y} \in \tilde{T}^*$ such that $\tilde{w}, \tilde{v}\tilde{w}\tilde{u}$ are w.p. and $|\tilde{x}|, |\tilde{v}|, |\tilde{w}|, |\tilde{u}|, |\tilde{y}| \leq Q$, for any integer $m > 0$, $\tilde{x}\tilde{v}^{n+m}\tilde{w}\tilde{u}^{n+m}\tilde{y} \in L(\tilde{G})$ iff $\tilde{x}\tilde{v}^{n+m}\tilde{w}\tilde{u}^{n+m}\tilde{y} \in L(\tilde{G})$. \square

We omit the simple proof of this lemma, based on eliminating useless cyclic derivations in $S_i \xrightarrow{\tilde{G}}^* \tilde{x}\tilde{v}^n\tilde{w}\tilde{u}^n\tilde{y}$ and in $S_i \xrightarrow{\tilde{G}}^* \tilde{x}\tilde{v}^{n+m}\tilde{w}\tilde{u}^{n+m}\tilde{y}$. Details are in the homonymous report [24]. For other usual concepts and definitions occurring in the following pages but not explicitly recalled here, we refer to [4].

3. Discussion of the NC topic and introduction to the main result. The notion of NC we introduced is suitable for context-free parenthesis (i.e., structured) languages; it is a natural generalization of NC regular languages in the sense that the parenthesis language $L(\tilde{G})$, with G a regular grammar, is NC according to our definition iff the underlying regular language $L(G) = h(\tilde{L})$ is NC according to McNaughton and Papert [9].

Furthermore, the investigation of NC parenthesis languages can be relevant from the point of view of grammar inference [1], [20].

Incidentally, we notice that according to our definition, for a derivation $A \xrightarrow{\tilde{G}}^* \tilde{t}A\tilde{r}$ to be counting, both \tilde{t} and \tilde{r} must be the repetition of some strings \tilde{v} and \tilde{u} , $n > 1$ times: to illustrate the point, a language such as $L(G_0)$, where $\tilde{G}_0 = \{S \rightarrow [aA_1c], A_1 \rightarrow [bA_2c], A_2 \rightarrow [aA_1c] \mid [d]\}$, which counts an even number of c but only at the right end of the string, is NC. It would be entirely reasonable to give other definitions leading to the notions of left (or right) NC languages (not to be confused with the concepts of [22]).

On the other hand, the question naturally arises whether the NC concept can be applied to general (i.e., unstructured) context-free languages. A first remark is that the definition of NC languages cannot be applied to general CF languages (removing the condition that \tilde{w} , and $\tilde{v}\tilde{w}\tilde{u}$ be w.p.) since all nontrivial languages would result counting.

Consider, e.g., the language $L_1 = \{a^m b^m \mid m \geq 1\}$. For no n would the requirement for NC be verified, since for any n , we can choose $x = \varepsilon, v = a^3, w = \varepsilon, u = b^2, y = b^n$, and find $xv^n w u^n y \in L_1$ while $xv^{n+1} w u^{n+1} y \notin L_1$.¹ On the other hand, we feel that every reader would intuitively consider L_1 as a noncounting language; this is due, perhaps to the fact that its “natural” grammar $G'_1 = \{S \rightarrow aSb, S \rightarrow ab\}$ generates a NC parenthesis language \tilde{L}'_1 while the weakly equivalent grammar $G''_1 = \{S \rightarrow aBb, B \rightarrow aSb, S \rightarrow ab, S \rightarrow aaAbb, A \rightarrow aaAbb, S \rightarrow aabb\}$, which generates a counting parenthesis language \tilde{L}''_1 , appears to be far less “natural”.

On the other hand, the intuitive tendency to consider $L_2 = \{a^{2n} cb^{2n} \mid n \geq 0\}$ as a counting language has to be resisted since L_2 is generated by two equally “natural” grammars, \tilde{G}'_2 counting and \tilde{G}''_2 NC, with $G'_2 = \{S \rightarrow aAb, A \rightarrow aSb, S \rightarrow c\}$ and $\tilde{G}''_2 = \{S \rightarrow aaSbb, S \rightarrow c\}$.

The goal of exactly defining the “natural” grammar of a language as a foundation for extending NC concepts to unparenthesized languages seems elusive. However, in

¹ In the conclusion, however, we present an interesting alternative definition suggested by a referee.

case the language is operator precedence, a clear solution of the problem can be given thanks to the main result of this paper: *If two operator precedence grammars are weakly equivalent, then either both are noncounting or both are counting.*

As a consequence, a structural property of languages, to be NC or not, is in this case independent of the particular structure given to the language. Intuitively, the main result is due to the fact that the parsing of an operator precedence string does not depend on the nonterminal alphabet of the grammar (Lemma 2), but only on the precedence relations on the terminal alphabet; as a consequence, the parsing of n repetitions of a string must be identical in any context. Furthermore, operator precedence grammars allow the deterministic construction of any bottom-up parsing, not necessarily of a reverse-rightmost one as happens for LR and simple precedence grammars.

This result does not hold for other classes of classical deterministically parsable grammars such as LR, LL or simple precedence grammars [4]. For example, the above grammars G'_2 and G''_2 are at the same time LR and LL, while G'_2 is simple precedence and counting, and the next grammar G_3 generating the same L_2 is simple precedence and NC,

$$G_3 = \{S \rightarrow A_1B, X \rightarrow AB_1, A_1 \rightarrow AX, B_1 \rightarrow SB, A \rightarrow a, B \rightarrow b, X \rightarrow c\}.$$

Unfortunately, we have not been able to give a simpler proof of our result than the one we propose; we feel this is a typical situation which requires the exhaustive examination of several cases to characterize the behavior of the iterative pairs [21] of a context-free language.

4. Main result. Let us first give two preliminary lemmas.

Let $\tilde{G} = (N, \tilde{T}, \tilde{P}, S)$ be an OPP grammar and $G = (N, T, P, S)$ the OP grammar underlying \tilde{G} .

Assume that corresponding productions in \tilde{P} and P have the same identification number.

LEMMA 2. *If*

$$S_i \xrightarrow{\pi}^* \alpha_1 a_1 A a_2 \alpha_2 \xrightarrow{G} \pi \alpha_1 a_1 \alpha a_2 \alpha_2,$$

$$S_j \xrightarrow{\rho}^* \beta_1 a_1 \beta a_2 \beta_2,$$

and $f(\alpha) = f(\beta)$, i.e., α and β have the same stencils, with $S_i, S_j \in S$, $\alpha_1, \alpha_2, \alpha, \beta_1, \beta_2, \beta \in (N \cup T)^*$, then there exists $B \in N$ such that

$$S_j \xrightarrow{\rho}^* \beta_1 a_1 B a_2 \beta_2 \xrightarrow{G} \rho \beta_1 a_1 \beta a_2 \beta_2,$$

and, for the two derivations

$$A \xrightarrow{\pi}^* \tilde{\alpha}, \quad B \xrightarrow{\rho}^* \tilde{\beta},$$

$f(\tilde{\alpha}) = f(\tilde{\beta})$. \square

Intuitively, Lemma 2 states that the parsings of two strings identical up to the names of nonterminals, within the same terminal context, must be identical up to the names of nonterminals.

A proof by induction on the length of π is given in [24], but is not reported here since it is a straightforward generalization of [7, Statement 2.4], which formally states

that parenthesization of OP languages depends only on the OP matrix, which in turn is not affected by the nonterminal alphabet.

LEMMA 3. (Fundamental lemma). *Let \tilde{G} be an OPP grammar and G the OP grammar underlying \tilde{G} . If $A \xrightarrow{*}_G \alpha^k A \beta^k$ where $k > 1$, $\alpha, \beta \in (T \cup N)^+$ and α (resp. β) displays only $< \cdot$ and \doteq (resp. $> \cdot$ and \doteq) precedence relations, then there exist $\tilde{\alpha}, \tilde{\beta}$, such that $h(\tilde{\alpha}) = \alpha$, $h(\tilde{\beta}) = \beta$ and $A \xrightarrow{*}_G \tilde{\alpha}^k A \tilde{\beta}^k$.*

Proof. Since $A \xrightarrow{*}_G \alpha^k A \beta^k \xrightarrow{*}_G \alpha^k \alpha^k A \beta^k \beta^k$, it follows that $\alpha^{(1)} < \cdot^{(1)} \alpha$ and $\beta^{(1)} > \cdot^{(1)} \beta$. In order to prove the assertion, we must show, that $\alpha A \beta$ is a phrase [3], [6] of G ; then by Lemma 2, the thesis will follow. To this purpose, we shall utilize a geometrical construction.

$$\text{Let} \quad \alpha = \alpha_1 \alpha_2 \cdots \alpha_p, \quad \alpha_i < \cdot \alpha_{i+1},^2$$

where for any $i = 1, \dots, p$, $\alpha_i \in (T \cup N)^*$, and if α_i contains the terminal characters $a_{i1}, a_{i2}, \dots, a_{in_i}$ in this order, then $a_{i1} \doteq a_{i2}, a_{i2} \doteq a_{i3}, \dots, a_{in_i-1} \doteq a_{in_i}$.

Similarly, let

$$\beta = \beta_q \beta_{q-1} \cdots \beta_1, \beta_{j+1} > \cdot \beta_j,^2$$

where for any $j = 1, \dots, q$, $\beta_j \in (T \cup N)^*$, and if β_j contains the terminal characters $b_{j1}, b_{j2}, \dots, b_{jn_j}$ in this order, then $b_{j1} \doteq b_{j2}, b_{j2} \doteq b_{j3}, \dots, b_{jn_j-1} \doteq b_{jn_j}$.

It is convenient to rename the substrings α_i in order to locate their positions within α^k :

$$\alpha^k = \alpha_{1/1} \alpha_{2/1} \cdots \alpha_{p/1} \alpha_{1/2} \alpha_{2/2} \cdots \alpha_{p/2} \cdots \alpha_{p/k},$$

where, for $i, j = 1, \dots, k$ and $n = 1, \dots, p$, $\alpha_{n/i} = \alpha_{n/j} = \alpha_n$; and

$$\beta^k = \beta_{q/k} \beta_{q-1/k} \cdots \beta_{1/k} \beta_{q/k-1} \beta_{q-1/k-1} \cdots \beta_{1/k-1} \cdots \beta_{1/1},$$

where, for $i, j = 1, \dots, k$ and $n = 1, \dots, q$, $\beta_{n/i} = \beta_{n/j} = \beta_n$.

We can graphically represent the parsing [3], [4] of the sentential form $\alpha^k A \beta^k$ in the following way. Let us consider the first quadrant of an integer Cartesian plane β, α ; we associate in an orderly way to the abscissas $0, 1, 2, \dots$, of the β -axis the strings $\beta_{q/k}, \beta_{q-1/k}, \beta_{q-2/k}, \dots, \beta_{1/1}$, and to the ordinates $0, 1, 2, \dots$, of the α -axis the strings $\alpha_{p/k}, \alpha_{p-1/k}, \alpha_{p-2/k}, \dots, \alpha_{1/1}$. We denote by $\beta_{q/0}$ (resp. $\alpha_{p/0}$) the point on the β - (resp. α -) axis immediately following $\beta_{1/1}$ (resp. $\alpha_{1/1}$). Since G is OP, each α_i in α (resp. β_j in β) entirely belongs to a prime phrase, and the parsing [3], [6] of the phrase $\alpha^k A \beta^k$ can be graphically represented in the β, α -plane by a continuous polygonal called a *parsing line* (see Fig. 2), which is a path starting in the origin, i.e., $(\beta_{q/k}, \alpha_{p/k})$, and ending in $(\beta_{q/0}, \alpha_{p/0})$, made of three sorts of segments: diagonal (of length $\sqrt{2}$), horizontal (of length 1), and vertical (of length 1).

Precisely, if the point $(\beta_{i/j}, \alpha_{m/n})$ is on the parsing line:

the diagonal segment $(\beta_{i/j}, \alpha_{m/n})(\beta_{i-1/j}, \alpha_{m-1/n})^3$ belongs to the parsing line iff $\alpha_{m/n} \doteq \beta_{i/j}$;

the horizontal segment $(\beta_{i/j}, \alpha_{m/n})(\beta_{i-1/j}, \alpha_{m/n})$ belongs to the parsing line iff $\alpha_{m/n} < \cdot \beta_{i/j}$;

the vertical segment $(\beta_{i/j}, \alpha_{m/n})(\beta_{i/j}, \alpha_{m-1/n})$ belongs to the parsing line iff $\alpha_{m/n} > \cdot \beta_{i/j}$.

² For $\alpha_1, \alpha_2 \in (T \cup N)^*$, $\alpha_1 < \cdot \alpha_2$ means $\alpha_1^{(1)} < \cdot^{(1)} \alpha_2$.

³ We assume that if $i = 1$ (resp. $m = 1$), $\beta_{i-1/j}$ (resp. $\alpha_{m-1/n}$) denotes $\beta_{q/j-1}$ (resp. $\alpha_{p/n-1}$).

Let us consider, as a clarifying example, the sentential form $\alpha^3 A \beta^3$, with $\alpha = abcd$, $\beta = cad$ generated by an OP grammar G having the precedence matrix [3] in Fig. 1. In this case, we have

$$\alpha_1 = a, \quad \alpha_2 = bb, \quad \alpha_3 = c, \quad \alpha_4 = d, \quad \beta_3 = c, \quad \beta_2 = a, \quad \beta_1 = d$$

a	$\langle \cdot$	\langle		\rangle
b	\rangle	\equiv	\langle	
c	\rangle		\equiv	\langle
d	\langle		\rangle	\langle
	a	b	c	d

FIG. 1

and the parsing line of $\alpha^3 A \beta^3$ is graphically represented in Fig. 2. Note that the parsing line must start in $(\beta_{q/k}, \alpha_{p/k})$ and, since $\alpha^k A \beta^k$ is a phrase of G , it must terminate exactly in $(\beta_{q/0}, \alpha_{p/0})$.

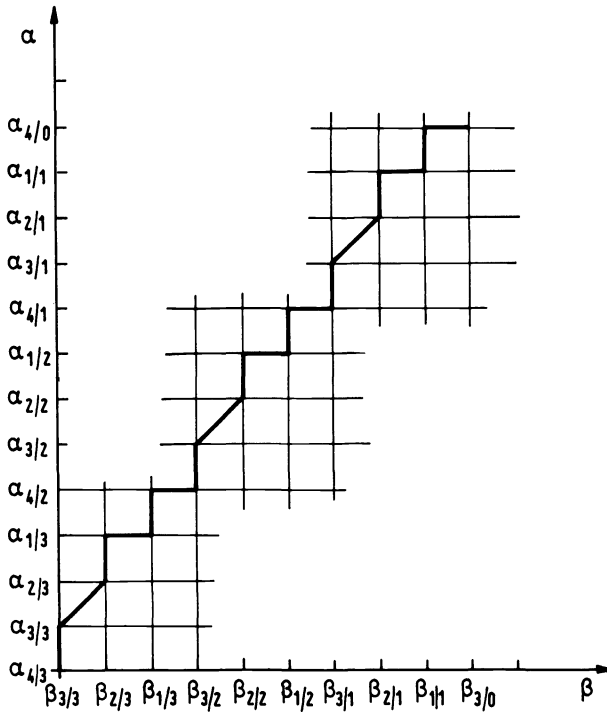


FIG. 2

In order to prove that $\alpha A \beta$ is a phrase of G , we must show that the parsing line reaches $(\beta_{q/k-1}, \alpha_{p/k-1})$.

To prove this fact by contradiction, suppose that the parsing line does not reach $(\beta_{q/k-1}, \alpha_{p/k-1})$. Since the parsing line reaches the point $(\beta_{q/0}, \alpha_{p/0})$, there must exist an

integer l , with $1 < l \leq k$, such that the parsing line reaches the point $(\beta_{q/k-l}, \alpha_{p/k-l})$, while it does not reach any $(\beta_{q,k-i}, \alpha_{p,k-i})$ with $i = 1, \dots, l-1$.

Now denote by X the abscissas and by Y the ordinates of the β, α -plane; for a point $P \equiv (\beta_{i/j}, \alpha_{m/n})$, the X, Y coordinates are

$$X = (k - j)q + (q - i), \quad Y = (k - n)p + (p - m).$$

Decompose now the parsing line L into disjoint portions L_j , $1 \leq j \leq k$, such that if $P \equiv (X, Y)$ is in L_j , then for $j > 1$, $(j - 1)q < X \leq jq$, and for $j = 1$, $0 \leq X \leq q$; denote by $P_j \equiv (X_j, Y_j)$ the point of L_j with $X_j = jq$ and $Y_j = \max \{Y \mid (X_j, Y) \in L_j\}$.

For each L_j , define the new lines L'_j by the following transformation of coordinates:

$$X' = X - (j - 1)q, \quad Y' = Y - (j - 1)p.$$

Observe that L_j , and therefore L'_j , are still continuous polygonal since, by construction, from any point $P \equiv (X, Y)$ of L onward, the remaining part of the path is completely contained in the first quadrant of the Cartesian plane assuming P as origin; in other words, L is never directed downward nor leftward.

The hypothesis assumed *per absurdum* entails $P_1 \neq (q, p)$. Let us consider only the case $P_1 = (q, Y_1)$, $Y_1 < p$ since the case $Y_1 > p$ is similar; on the other hand, L_l contains (lq, lp) ; i.e., for L'_l , $P'_l \equiv (q, Y'_l)$, $Y'_l \geq p$. Thus, there exists \bar{j} such that $1 < \bar{j} \leq l$, $Y'_{\bar{j}-1} < p$, $Y'_{\bar{j}} \geq p$, but this implies, by the continuity of any L'_j , that there exists $\bar{P}' \equiv (\bar{X}', \bar{Y}')$ of intersection between $L_1 \equiv L'_1$ and $L'_{\bar{j}}$. Furthermore, since L consists only of the three elementary segments described above (in particular no segment of the type $(X, \bar{Y})(\bar{X} + 1, \bar{Y} - 1)$ may belong to it), \bar{X}' and \bar{Y}' must be integers. But this means that $(\beta_{\bar{t}/\bar{j}}, \alpha_{\bar{m}/\bar{j}})$ and $(\beta_{\bar{t}/k}, \alpha_{\bar{m}/k})$, with $\bar{t} = q - \bar{X}'$, $\bar{m} = p - \bar{Y}'$, belong respectively to $L_{\bar{j}}$ and L_1 .

Fig. 3 gives a pictorial explanation of the above reasoning: in the absurd hypothesis that the parsing line exhibits a path as the one shown there, L_1 and L'_2 would intersect in

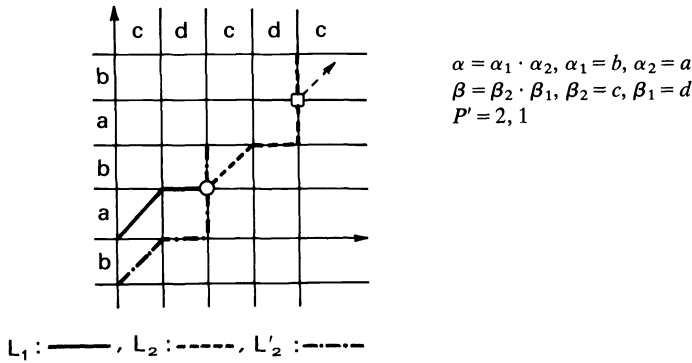


FIG. 3

the circled point to which corresponds in L_2 the one denoted by a small square. On the other hand, since the two points $(\beta_{\bar{t}/\bar{j}}, \alpha_{\bar{m}/\bar{j}})$, $(\beta_{\bar{t}/k}, \alpha_{\bar{m}/k})$ belong to the parsing line, it follows that the same parsing action must be performed in both of them since at both points the precedence relation between $\alpha_{\bar{m}}$ and $\beta_{\bar{t}}$ holds (in fact, the parsing line of Fig. 3 would imply a precedence conflict $b \doteq c, b \cdot > c$); thus, the parsing line must display from $(\beta_{\bar{t}/\bar{j}}, \alpha_{\bar{m}/\bar{j}})$ onward exactly the same shape as from $(\beta_{\bar{t}/k}, \alpha_{\bar{m}/k})$ on (i.e., it is periodic). This prevents the parsing line from reaching the point $(\beta_{q/k-l}, \alpha_{p/k-l})$ since P_l would be $(lq, Y_1 + (l - 1)p)$ and concludes the proof. \square

THEOREM 1. (Main theorem). *Let G_1 and G_2 be OP grammars (in mFNF) and \tilde{G}_1, \tilde{G}_2 the associated OPP grammars. If $L(G_1) = L(G_2)$, then either both $L(\tilde{G}_1)$ and $L(\tilde{G}_2)$ are NC or neither one is NC.*

Proof. By virtue of Statement 3, there exist two BD reduced OPP grammars \tilde{G}'_1 and \tilde{G}'_2 such that $L(\tilde{G}'_1) = L(\tilde{G}_1)$ and $L(\tilde{G}'_2) = L(\tilde{G}_2)$. Therefore, it suffices to prove that if $L(G_1) = L(G_2)$, then either both $L(\tilde{G}'_1)$ and $L(\tilde{G}'_2)$ are NC or neither one is NC.

If we show that $L(\tilde{G}'_2)$ NC implies $L(\tilde{G}'_1)$ NC, then similarly $L(\tilde{G}'_1)$ NC will imply $L(\tilde{G}'_2)$ NC and the thesis will follow from the tautology $(A \wedge B) \vee (\neg A \wedge \neg B) \equiv (A \supset B) \wedge (B \supset A)$. Suppose $L(\tilde{G}'_2)$ is NC, and let \bar{n}_2 be an integer satisfying the conditions for \bar{n} of Statement 4 for \tilde{G}'_2 .

Let Q_1 be an integer satisfying the definition of Q in Lemma 1 for \tilde{G}'_1 . Then let $\tilde{v}_1, \tilde{w}_1, \tilde{u}_1, \tilde{x}_1, \tilde{y}_1 \in \tilde{T}^*$, such that $\tilde{v}_1\tilde{w}_1\tilde{u}_1, \tilde{w}_1$ are w.p. strings and

$$|\tilde{v}_1|, |\tilde{w}_1|, |\tilde{u}_1|, |\tilde{x}_1|, |\tilde{y}_1| \leq Q_1.$$

By virtue of Statement 4 and Lemma 1, we must prove that there exists a sufficiently large integer \bar{n}_1 such that for any integer $n \geq \bar{n}_1$,

$$\tilde{z}_1 = \tilde{x}_1\tilde{v}_1^n\tilde{w}_1\tilde{u}_1^n\tilde{y}_1 \in L(\tilde{G}'_1) \text{ implies}$$

$$\tilde{z}'_1 = \tilde{x}_1\tilde{v}_1^{n+1}\tilde{w}_1\tilde{u}_1^{n+1}\tilde{y}_1 \in L(\tilde{G}'_1).$$

Let us assume first $\bar{n}_1 \geq \bar{n}_2 + 2Q_1 + 1$.

Let G'_1 and G'_2 be the OP grammars (in mFNF) underlying \tilde{G}'_1 and \tilde{G}'_2 , respectively. Let $z = xv^nu^n y$ with $x = h(\tilde{x}_1)$, $v = h(\tilde{v}_1)$, $w = h(\tilde{w}_1)$, $u = h(\tilde{u}_1)$, $y = h(\tilde{y}_1)$, and $z' = xv^{n+1}wu^{n+1}y$. Our target is to prove that $z' \in L(G'_1)$ and then to show $z' = h(\tilde{z}'_1)$, $\tilde{z}'_1 \in L(\tilde{G}'_1)$. Since, by hypothesis, $L(G_1) = L(G_2)$, hence $L(G'_1) = L(G'_2)$, and, in order to prove that $z' \in L(G'_1)$, it suffices to prove $z' \in L(G'_2)$.

As a preliminary step, let us parse the substrings v^n and u^n in $z \in L(G'_2)$. Assume first $v \neq \varepsilon$ and let $v^n = vv^{n-2}v$, so that we can parse v^{n-2} in the context $v \cdot v$. Let us consider the following construction which parses all that can be parsed inside v^n by first parsing inside v , then parsing all factors vv standing within v^n .

Step 1. Let $\alpha' = v$.

Step 2. If α' does not contain any prime phrase [3], [6], then let

$$(1) \quad \alpha'^{n-2} \xrightarrow[\tilde{G}'_2]{*} v^{n-2},$$

and go to step 6.

Step 3. Let $\alpha' = \alpha'_1 \langle \cdot \alpha'_2 \cdot \rangle \alpha'_3$,⁴ where α'_2 contains no $\cdot \rangle$ and $\langle \cdot$ precedence relations and the rightmost and leftmost characters of α'_1 and α'_3 , respectively, are terminal ones.

Step 4. Reduce all the prime phrases $\langle \cdot \alpha'_2 \cdot \rangle$ in α'^{n-2} ; let

$$A' \xrightarrow[\tilde{G}'_2]{1} \alpha'_2.$$

Step 5. Let $\alpha' = \alpha'_1 A' \alpha'_3$ and go back to step 2.

Step 6. Let

$$\xi = \zeta = \varepsilon,$$

$$s = 2.$$

⁴From now on, the notation $\alpha = \alpha_1 \odot \alpha_2 \cdots \odot \alpha_n$, where \odot stands either for $\cdot \rangle$ or for $\langle \cdot$, will abbreviate $\alpha = \alpha_1, \alpha_2, \cdots, \alpha_n, \alpha_1 \odot \alpha_2, \cdots, \alpha_{n-1} \odot \alpha_n$.

Step 7. If α'^{n-s} does not contain any prime phrase, then let

$$(2) \quad \xi \alpha'^{n-s} \zeta \xrightarrow[G_2^*]{*} v^{n-2}$$

and exit.

Step 8. Let $\alpha' = \alpha'_1 \cdot > \alpha'_2 < \cdot \alpha'_3$, where α'_1 and α'_3 contain no $\cdot >$ and $< \cdot$ precedence relations respectively, and the leftmost and rightmost characters of α'_2 are terminal ones.

Step 9. Reduce all the prime phrases $< \cdot \alpha'_3 \alpha'_1 \cdot >$ in α'^{n-s} ; let

$$A' \xrightarrow[G_2^*]{1} \alpha'_3 \alpha'_2.$$

Step 10. Let

$$\begin{aligned} \alpha' &= \alpha'_2 A', \\ \xi &= \xi \alpha'_1, \\ \zeta &= \alpha'_2 \alpha'_3 \zeta, \\ s &= s + 2 \end{aligned}$$

and go back to step 7.

Observe that the predicate

$$(\alpha' \xrightarrow[G_2^*]{h} *v) \wedge (\alpha' \in ((T'_2 \cup N'_2)^+ - N'_2) \wedge (|\xi| < |\alpha'| \cdot Q_1)),$$

where $*v'$ is a cyclic permutation of v and $h \leq Q_1$ (since $|v| \leq Q_1$), is a loop invariant [25] of the two loops of the above construction, because of Lemma 2, since G_2' is an OP grammar. Therefore, since this algorithm halts in a finite number of steps, derivation (2) is obtained with $s \leq 2Q_1$ and $\alpha \in (T'_2 \cup N'_2)^+ - N'_2$. Moreover, by construction, α'^{n-s} may not contain both $\cdot >$ and $< \cdot$ precedence relations.

Similarly, if $u \neq \varepsilon$, from $u^n = uu^{n-2}u$ it follows that

$$(3) \quad \varphi \beta'^{n-t} \psi \xrightarrow[G_2^*]{*} u^{n-2},$$

where $t \leq 2Q_1$, $\beta' \xrightarrow[G_2^*]{k} *u$, with $k \leq Q_1$, $\beta' \in (T'_2 \cup N'_2)^+ - N'_2$, $*u$ is a cyclic permutation of u , and β'^{n-t} does not contain both $< \cdot$ and $\cdot >$ precedence relations.

We thus obtain

$$(4) \quad S_i \xrightarrow[G_2^*]{*} x \xi \alpha'^{n-s} \zeta w \varphi \beta'^{n-t} \psi y \xrightarrow[G_2^*]{*} x v^n w u^n y.$$

Notice that we have been able to state these derivations thanks to the fact that, G_2' being an OP grammar, the parsing of v^n is not affected by the parsing of x, w, u^n, y . Furthermore, the same relations hold, possibly with a different S_i , for any $n', n'' > n$ such that $xv^{n'}wu^{n'}y \in L(G_2)$.

Let us now discuss the following three cases.

A. $v \neq \varepsilon, u = \varepsilon$. In this case, v^{n-2} is obtained by means of a derivation of the type of derivation (2). If $\alpha' = \alpha'_1 \cdot > \alpha'_2$ then we can factorize α'^{n-s} as

$$\alpha'^{n-s} = \alpha'_1 \alpha \cdot > \alpha \cdot > \dots \cdot > \alpha \alpha'_2 \quad n-s-1 \text{ times}$$

with $\alpha = \alpha'_2 \alpha'_1 = {}^* \alpha'$, where ${}^* \alpha'$ denotes a cyclic permutation of α' , and therefore, $\alpha \xrightarrow{*}_{G'_2} {}^* v$. Since $n \geq \bar{n}_2 + 2Q_1 + 1$ and $s \leq 2Q_1$ from well-known properties of CF grammars, it follows that $z \in L(G'_2)$ must be obtained by means of a cyclic derivation of the type $A \xrightarrow{*}_{G'_2} \theta A \alpha^k$ with $|N'_2| \geq k \geq 1$.

Since $|x|, |v| \leq Q_1$, from the hypothesis $|\alpha'_1| < |\alpha'| \leq |v| \leq Q_1$, because G'_2 does not contain any ε -rule [4], [6], $|\xi| < |\alpha'| \cdot Q_1 \leq Q_1^2$, and because of the above construction, it follows that $|xv\xi\alpha'_1| \leq 3Q_1 + Q_1^2$. Thus, if \bar{n}_1 has been chosen sufficiently large⁵, it must be the case that $\theta = \varepsilon$.

By virtue of Lemma 3, there exists one derivation $A \xrightarrow{*}_{\tilde{G}'_2} [{}^{k,r} A \tilde{\alpha}^k]$, with $h(\tilde{\alpha}) = \alpha$ and $r \in \{0, 1, 2, \dots, Q_1\}$. But since \tilde{G}'_2 is BD reduced and $L(\tilde{G}'_2)$ is NC by virtue of Statement 5, it follows that $k = 1$.

In conclusion, $z \in L(G'_2)$ is obtained by means of a *unilateral derivation* of the type

$$(5) \quad A \xrightarrow{*}_{G'_2} A\alpha,$$

with $\alpha \xrightarrow{*}_{G'_2} {}^* v$, in the following way:

$$\begin{aligned} S_i &\xrightarrow{*}_{G'_2} \check{x}A\hat{y}, \\ A &\xrightarrow{*}_{G'_2} A^*v \quad m \text{ times}, \\ A &\xrightarrow{*}_{G'_2} w'', \end{aligned}$$

where $S_i \in S'_2$, $w'' \in T'^+_2$, and $\check{x}w''^*v^m\hat{y} = xv^nwy = z$. Therefore, by applying the derivation $A \xrightarrow{*}_{G'_2} A^*v$ $m + 1$ times, z' is obtained, and thus $z' \in L(G'_2)$.

Similarly, if $\alpha' = \alpha'_1 < \alpha'_2$, it follows that $z \in L(G'_2)$ is obtained by means of a unilateral derivation of the type

$$(6) \quad A \xrightarrow{*}_{G'_2} \alpha A,$$

with $\alpha \xrightarrow{*}_{G'_2} {}^* v$, and then $z' \in L(G'_2)$.

B. $v = \varepsilon, u \neq \varepsilon$. In this case, u^{n-2} is obtained by means of a derivation of the type of derivation (3). Similar to the preceding case A, if $\beta' = \beta'_1 < \beta'_2$ (resp. $\beta' = \beta'_1 > \beta'_2$), it follows that $z \in L(G'_2)$ is obtained by means of a unilateral derivation of the type

$$(7) \quad B \xrightarrow{*}_{G'_2} \beta B,$$

$$(8) \quad (\text{resp. } B \xrightarrow{*}_{G'_2} B\beta),$$

with $\beta \xrightarrow{*}_{G'_2} {}^* u$, and then $z' \in L(G'_2)$.

⁵ Precisely, $\bar{n}_1 \geq (\bar{n}_2 + 2Q_1 + 1) + (3Q_1 + Q_1^2) \cdot |N'_2|$.

C. $v \neq \varepsilon$, $u \neq \varepsilon$. This case is split into two subcases.

C1. The string $z = xv^n wu^n y \in L(G'_2)$ is obtained by means of two unilateral derivations of the type of derivations (5) or (6) and (7) or (8). Similar to the above cases A and B, it follows that $z' \in L(G'_2)$.

C2. In the derivation of $z \in L(G'_2)$, there exists a *bilateral cyclic derivation* of the following type⁶:

$$(9) \quad A \xrightarrow[G'_2]{*} \alpha^{k_1} A \beta^{k_2},$$

with $k_1, k_2 \geq 1$, and

$$\alpha \xrightarrow[G'_2]{*} *v, \quad \beta \xrightarrow[G'_2]{*} *u.$$

If $k_1 = k_2$, by virtue of Lemma 3 and Statement 5, it follows that $k_1 = k_2 = 1$ and therefore, $z' \in L(G'_2)$.

If $k \neq k_2$, let us discuss the case $k_1 < k_2$ (the case $k_1 > k_2$ can be treated in a similar way).

Let us consider the language

$$L' = \{z \mid z = xv^{n'} wu^{n''} y, n'' \geq n' \geq \bar{n}_1, z \in L(G'_2)\}.$$

Notice that because of derivation (9), since $n'' - n'$ can be arbitrarily large, $L' \neq \emptyset$. From $L' \subseteq L(G'_2)$ and $L(G'_2) = L(G'_1)$, it ensues that $L' \subseteq L(G'_1)$. But this implies that there exists in G'_1 a cyclic derivation of the type

$$(10) \quad H_1 \xrightarrow[G'_1]{*} H_1 * u^h \quad \text{with } h \geq 1.^7$$

By virtue of derivation (10), for any integer $t > 0$, there exists in L' a string $z = xv^{n'} wu^{n''} y$ such that $n''/n' > t$. But, since $L' \subseteq L(G'_2)$, this requires that there exists in G'_2 a cyclic derivation of the type

$$A \xrightarrow[G'_2]{*} A \beta^k \xrightarrow[G'_2]{*} A^* u^k.$$

Since $L(G'_2)$ is NC, by virtue of Lemma 3 and Statement 5, it follows that $k = 1$, and then

$$(11) \quad A \xrightarrow[G'_2]{*} A^* u.$$

Moreover, since $L' \subseteq L(G'_1)$, there exist in L' strings $z = xv^{n'} wu^{n''} y$ such that $n'' = n'$. This implies, since $L' \subseteq L(G'_2)$, that there exists in G'_2 a cyclic derivation of the type

$$E \xrightarrow[G'_2]{*} (*v)^h E,$$

with $h \geq 1$. Again, by virtue of Lemma 3 and Statement 5 it follows, with $L(G'_2)$ NC, $h = 1$, that

$$(12) \quad E \xrightarrow[G'_2]{*} *vE.$$

⁶ Obviously "tertium non datur;" i.e., the exclusion of the preceding cases necessarily implies the existence of a derivation of type (9) if \bar{n}_1 is sufficiently large.

⁷ This can be obtained by reasoning similar to that used above for G'_2 .

Now, having demonstrated the existence of derivations of type (9)–(12), we can analyze how the strings $z = xv^{n'}wu^{n''}y \in L'$ are derived in $L(G'_2)$, recalling that the relations (4) hold. We consider two cases.

If $n'' \gg (k_2/k_1)n'^8$, then z is obtained by means of derivations of the type (9) and (11); precisely, we can state:

STATEMENT 6. *For any $z \in L'$, if $n'' \gg (k_2/k_1)n'$, z is obtained by means of a derivation of the following type:*

- (i) $S_i \xRightarrow{*} \hat{x}A_i\check{u}y$,
- (ii) $A_i \xRightarrow{*} A_i\beta \xRightarrow{*} A_i^*u$ j times,
- (iii) $A_i \xRightarrow{*} \check{x}\hat{v}B_i\beta^{q_2} \xRightarrow{*} \check{x}\hat{v}B_i(*u)^{q_2}$,
- (13) (iv) $B_i \xRightarrow{*} \alpha^{q_{i_1}}B_i\beta^{q_{i_2}} \xRightarrow{*} (*v)^{q_{i_1}}B_i(*u)^{q_{i_2}}$,
- (v) $B_i \xRightarrow{*} \alpha^{k_i}B_i\beta^{k_2} \xRightarrow{*} (*v)^{k_i}B_i(*u)^{k_2}$ i times,
- (vi) $B_i \xRightarrow{*} \check{v}v^{q'_1}wu^{q'_2}\hat{u}$,

where:

$$S_i \in S'_2,$$

$$\hat{x}\check{x} = x,$$

$$\hat{v}\check{v} = v, \quad \check{v}\hat{v} = *v,$$

$$\hat{u}\check{u} = u, \quad \check{u}\hat{u} = *u;$$

j, i can take any value in \mathbb{N} ;

$$q_{i_1} \in \{0, 1, \dots, k_1 - 1\}, \quad q_{i_2} \in \{0, 1, \dots, k_2 - 1\};$$

q'_1, q'_2 are unique natural numbers, $q'_1, q'_2, q_2 < r(|N'_2|)$, where r is a recursive function of $|N'_2|$;

$B_i \in \{B_1, B_2, \dots, B_{\bar{k}}\}$, where $B_1, B_2, \dots, B_{\bar{k}} \in N'_2$ and

$$(vii) \quad B_1 \Rightarrow^1 \alpha B_2^1 \beta, \quad B_2 \Rightarrow^2 \alpha B_3^2 \beta, \quad \dots, \quad B_{\bar{k}} \Rightarrow^{\bar{k}} \alpha B_1^{\bar{k}} \beta$$

with ${}^1\alpha^2\alpha \dots \bar{k}\alpha = \alpha^{k_1}$ and $\bar{k}\beta \dots {}^2\beta^1\beta = \beta^{k_2}$. \square

Proof of Statement 6. Consider a bottom-up (not left-to-right!) parsing [3], [4] of z . Since by hypothesis we have assumed the existence in the derivation of z of a derivation of type (v), a parsing corresponding to (vi) must be first performed. (vi) cannot contain any cyclic derivations of the type

$$(viii) \quad C \xRightarrow{*} \alpha^{h_1}C\beta^{h_2},$$

not containing (v), since this would imply the repetition of the parsing corresponding to (viii) while C is in the context $(*v)^{h_1} - (*u)^{h_2}$, thus preventing the parsing (v).

⁸ In the rest of the paper $m \gg n$ stands for the expression $m > n + r$, where r is a positive integer which depends only on N'_2 (or N'_1); we avoid its cumbersome computation, since it is sufficient to show that r is recursively bounded by $|N'_2|$ (or $|N'_1|$).

Consequently q'_1, q'_2 are bounded by a recursive function $r(|N'_2|)$ and are the same for any derivation of type (13). Successively, the parsing corresponding to (v) is repeated i times while B_1 has as left context $\hat{v}(*v)^{k_1}$ (by hypothesis $(*u)^{k_2}$ will be a prefix of its right context).

Consider now the derivation $B_1 \xrightarrow{*} \alpha^{k_1} B_1 \beta^{k_2}$ and expand it as (vii) (note that if $\bar{k}\beta = \dots^2\beta = {}^1\beta = \beta$, then $\bar{k} = k_2$, in any other case $k_2 < \bar{k}$). Since, after the parsing of (v) i times, B_1 will be in the context $x\hat{v}(*v)^{q_{i_1}} - (*u)^m \hat{u}y$, $m \gg k_2$, $q_{i_1} < k_1$, derivation (iv) will be constructed, which is a portion of (v) .

Afterwards, a number q_2 of $*u$ will be reduced to β and then to an A_i , possibly involving a portion of x , until a cyclic derivation of type (ii) is identified: x, v being of bounded length, q_2 is bounded and a derivation of type (ii) will be ultimately parsed and repeated j times, as far as a right context $*u\hat{u}$ will occur at the right of A_i . Finally (i) will be parsed.

Note that in general, q_2 and A_i will depend on B_i . \square

If $n' \leq n'' \ll (k_2/k_1)n'$, z is obtained by means of derivations of the type (9) and (12). Similarly to Statement 6, it can be proved that:

STATEMENT 7. For any $z \in L'$, if $n' \leq n'' \ll (k_2/k_1)n'$, z is obtained by means of a derivation of the following type:

$$\begin{aligned}
 (14) \quad S_i &\xrightarrow{*} x\hat{v}E_s\hat{y}, \\
 E_s &\xrightarrow{*} *vE_s \quad g \text{ times}, \\
 E_s &\xrightarrow{*} (*v)^{q_1} B_s \hat{u}\hat{y}, \\
 B_s &\xrightarrow{*} (*v)^{q_{i_1}} B_1 (*u)^{q_{i_2}}, \\
 B_1 &\xrightarrow{*} (*v)^{k_1} B_1 (*u)^{k_2} \quad i \text{ times}, \\
 B_1 &\xrightarrow{*} \check{v}v^{q'_1} wu^{q'_2} \hat{u},
 \end{aligned}$$

where:

$$\begin{aligned}
 S_i &\in S'_2, \\
 \hat{y}\check{y} &= y, \\
 \hat{v}\check{v} &= v, & \check{v}\hat{v} &= *v, \\
 \hat{u}\check{u} &= u, & \check{u}\hat{u} &= *u;
 \end{aligned}$$

g can take any value in \mathbb{N} ;

$$q_{i_1} \in \{0, 1, \dots, k_1 - 1\}, \quad q_{i_2} \in \{0, 1, \dots, k_2 - 1\};$$

q'_1, q'_2 are exactly the same as in derivation (13); $q'_1, q'_2, q_1 < r(|N'_2|)$ as in Statement 6; and $B_s \in \{B_1, B_2, \dots, B_{\bar{k}}\}$.

Let us now consider how the same strings $z = xv^{n'}wu^{n''}y \in L'$ are derived in $L(G'_1)$. For $n'' \gg n'$, recalling that by hypothesis

$$\tilde{z}_1 = \tilde{x}_1 \tilde{v}_1^n \tilde{w}_1 \tilde{u}_1^n \tilde{y}_1,$$

we obtain z by means of a derivation of the following type:

$$\begin{aligned}
 S_i &\xrightarrow{*} \hat{x}H_s y, \\
 H_s &\xrightarrow{*} H_1 \check{u} u^{p_s}, \\
 H_1 &\xrightarrow{*} H_1^* u^h \quad f \text{ times}, \\
 H_1 &\xrightarrow{*} \check{x} K_t u^{p'} \hat{u}, \\
 K_t &\xrightarrow{*} v^{p_t} K_1 u^{p_t}, \\
 K_1 &\xrightarrow{*} v^k K_1 u^k \quad l \text{ times}, \\
 K_1 &\xrightarrow{*} v^{p''} w u^{p''},
 \end{aligned}
 \tag{15}$$

where:

$$\begin{aligned}
 S_i &\in S'_1, \\
 \hat{x}\check{x} &= x, \\
 \hat{u}\check{u} &= u, \quad \check{u}\hat{u} = {}^*u;
 \end{aligned}$$

f, l can take any value in \mathbb{N} ;

$$\begin{aligned}
 p_s &\in \{0, 1, \dots, h-1\}, \quad h \geq 1, \\
 p_t &\in \{0, 1, \dots, k-1\}, \quad k \geq 1;
 \end{aligned}$$

p'' is a unique natural number, $p'', p' < r(|N'_1|)$; $H_s \in \{H_1, H_2, \dots, H_h\}$, where $H_1, H_2, \dots, H_h \in N'_1$ and

$$H_1 \xrightarrow{*} H_2^* u, \quad H_2 \xrightarrow{*} H_3^* u, \quad \dots, \quad H_h \xrightarrow{*} H_1^* u;$$

$K_t \in \{K_1, K_2, \dots, K_k\}$, where $K_1, K_2, \dots, K_k \in N'_1$ and

$$K_1 \xrightarrow{*} v K_2 u, \quad K_2 \xrightarrow{*} v K_3 u, \quad \dots, \quad K_k \xrightarrow{*} v K_1 u.$$

Note that, while K_1 and p'' are the same in any derivation of type (15), H_1, p' , and possibly h itself depend on K_t (but the parenthesization of the $n'' - n'$ occurrences of u at the left of y does not depend on it).

More simply, for $n'' = n'$, z is obtained by means of a derivation of the type:

$$\begin{aligned}
 S_i &\xrightarrow{*} x K_t y, \\
 K_t &\xrightarrow{*} v^{p_t} K_1 u^{p_t}, \\
 K_1 &\xrightarrow{*} v^k K_1 u^k \quad l \text{ times}, \\
 K_1 &\xrightarrow{*} v^{p''} w u^{p''},
 \end{aligned}
 \tag{16}$$

where:

$$\begin{aligned}
 S_i &\in S'_1, \\
 p_t &\in \{0, 1, \dots, k-1\};
 \end{aligned}$$

p'' is a unique natural number, exactly the same as in derivation (15);

$$K_t \in \{K_1, K_2, \dots, K_k\}.$$

We can now derive some relations between indices.

Let us now consider a *fixed* n' and choose n'' such that

$$n'' \gg (k_2/k_1)n';$$

then $z \in L'$ is obtained by means of derivations of type (13) and (15) in G'_2 and G'_1 , respectively. We thus have,

$$(17) \quad \begin{aligned} q_{t_1} + ik_1 + q'_1 + 1 &= p'' + p_t + lk, \\ j + q_2 + q_{t_2} + ik_2 + q'_2 + 1 &= p_s + fh + p' + p'' + p_t + lk + 1. \end{aligned}$$

Since the first member of the second equation can take any value $n'' \cong q_2 + q_{t_2} + q'_2 + 1$ in \mathbb{N} such that $n'' \gg (k_2/k_1)n'$, because of the arbitrary number j of repetitions of $A_t \xrightarrow{*} A_t^*u$, the same must be true also for the second member of the same equation; i.e., it must be possible, by means of a derivation of type (15), to produce in $z \in L(G'_1)$ any integer number (n'') of occurrences of u in order to satisfy (17).

In particular, for a given value of n' , p_t also, and therefore, p' , H_1 , and h in (15) are fixed. As a consequence, for any fixed n' , p_s must take any value in $\{0 \dots h - 1\}$.

Let us now decrease the value of f in derivation (15) in such a way that (without changing the value of n') $n' \ll n'' \ll (k_2/k_1)n'$. In this case, z is obtained in G'_2 by means of a derivation of type (14). Therefore, we have

$$(18) \quad \begin{aligned} g + q_1 + q_{t_1} + ik_1 + q'_1 + 1 &= p'' + p_t + lk, \\ q_{t_2} + ik_2 + q'_2 + 1 &= p_s + fh + p' + p'' + p_t + lk + 1, \end{aligned}$$

with the same h , p' , p_t , l , k , p'' as in (17).

Since the second member of the second equation can take any value $n'' \cong p' + p'' + p_t + lk + 1$ in \mathbb{N} , it follows that q_{t_2} must take any value in $\{0, 1, \dots, k_2 - 1\}$ (since q'_2 is fixed).

Finally, set $n'' = n'$; $z \in L$ is thus obtained by means of derivations of types (14) and (16) in G'_2 and G'_1 respectively. Therefore, we have

$$\begin{aligned} g + q_1 + q_{t_1} + ik_1 + q'_1 + 1 &= p_t + lk + p'', \\ q_{t_2} + ik_2 + q'_2 + 1 &= p_t + lk + p''. \end{aligned}$$

Since q_{t_2} can take any value in $\{0, 1, \dots, k_2 - 1\}$, we can choose i , q_{t_2} , in such a way that $n'' = n$. Successively increasing q_{t_2} by one (or, if $q_{t_2} = k_2 - 1$, letting $q_{t_2} = 0$, and increasing i by one), we obtain $n'' = n + 1$. In the first member of the first equation, the values of q_1 , q_{t_1} , q'_1 , and i corresponding to $n'' = n$ change accordingly by only a bounded amount as a consequence of the change by one in q_{t_2} (or in i). But, since g can take any value in \mathbb{N} , we can choose it in such a way that $n' = n + 1$. Therefore, we can conclude that, also in this last case, $z' = xv^{n+1}wu^{n+1}y \in L(G'_2)$.

To conclude the proof, we have only to show that $\tilde{z}_1 \in L(G'_1)$ and $z' \in L(G'_1)$ imply $\tilde{z}'_1 \in L(\tilde{G}'_1)$. But this is an immediate consequence of the fact that z' is obtained by means of a derivation of type (16) and of Lemma 2. \square

5. Concluding remarks. As a consequence of Theorem 1, we are now able to give the following.

DEFINITION. An operator precedence language L is *noncounting* iff for any operator precedence grammar G such that $L(G) = L$, $L(\tilde{G})$ is noncounting.

This definition, of course, does not mean that all (nonoperator precedence) grammars generating L are noncounting, and conversely; e.g., the language $L = \{a^{2^n} \mid n \geq 1\}$ is generated by a counting, regular, and hence operator precedence, grammar, but also by the noncounting grammar $\{S \rightarrow aSa, S \rightarrow aa\}$ which is not operator precedence.

We found in all examples considered that the NC property associated in this way to OP languages coincides with the intuitive notion of the concept.

Our result raises the problem of attaching the noncounting property to a wider class of (deterministic?) languages. For example, we would like to have some formal tool to claim the language $L = \{a^n b a^n \mid n \geq 0\}$ to be noncounting, in accordance with the fact that its "natural" grammar is noncounting. But, which is, in general, the "natural" grammar of a language from the NC point of view?

We wish to mention here the following possible definition suggested by a referee.

A context-free language L is counting if there exist x, v, w, u, y such that $xv^m w u^m y \in L$ for infinitely many m and $xv^m w u^m y \notin L$ for infinitely many m .

This definition seems, at first glance, to be well-posed, i.e., to extend previous definitions in a natural way, and to open several interesting questions (e.g., decidability).

Finally, we notice that the strict interdependency of syntax structures which we have discovered for equivalent operator precedence grammars hints at the following conjecture: if G and G' are equivalent operator precedence grammars and G counts modulo k , G' also counts modulo k . Were this conjecture true, one could start working on the equivalence problem for operator precedence grammars.

Note added in proof. An example, however, has recently been discovered concerning the definition suggested above which seems worth consideration.

The language $L_1 = \{(ab)^n c^{2^n} \mid n \geq 1\}$ is NC according both to the above definition and to ours. On the other hand, the language $L_2 = \{(ab)^*(ab)^n c^{2^n}\}$ turns out to be counting according to the referee's definition ($(ab)^{2^n} c^{2^n} \in L_2 \forall n \geq 1$, $(ab)^{2^{n+1}} c^{2^{n+1}} \notin L_2 \forall n \geq 1$). This is in contrast with our intuition, which likes to think the concatenation of two NC languages to be NC, and with the fact that any OP grammar generating L_2 is NC.

This example reopens the debate about what does "noncounting" mean in the framework of context-free languages. Our intuition is that the term noncounting is really meaningful if applied to the machine which generates, or recognizes, a language rather than to the language itself.

Nevertheless, we agree with our referee that there is much space for subjective feeling in such a topic.

REFERENCES

- [1] S. CRESPI-REGHIZZI, G. GUIDA AND D. MANDRIOLI, *Noncounting context-free languages*, J. Assoc. Comput. Mach., 25 (1978), pp. 571–580.
- [2] R. MCNAUGHTON, *Parenthesis grammars*, J. Assoc. Comput. Mach., 14 (1967), pp. 460–500.
- [3] R. W. FLOYD, *Syntactic analysis and operator precedence*, J. Assoc. Comput. Mach., 10 (1963), pp. 316–333.
- [4] A. AHO AND J. ULLMAN, *The Theory of Parsing, Translation and Compiling—Volume 1: Parsing*, Prentice-Hall, Englewood Cliffs, NJ, 1972.
- [5] J. HOPCROFT AND J. ULLMAN, *Formal Languages and their Relations to Automata*, Addison-Wesley, Reading, MA, 1969.

- [6] M. FISCHER, *Some properties of precedence languages*, Proc. ACM Symp. on Theory of Computing, Marina del Rey, Cal., 1969, pp. 181–190.
- [7] S. CRESPI-REGHIZZI, D. MANDRIOLI AND D. F. MARTIN, *Algebraic properties of operator precedence languages*, Inform. and Control, 37 (1978), pp. 115–133.
- [8] S. CRESPI-REGHIZZI AND D. MANDRIOLI, *A class of grammars generating noncounting languages*, Inform. Process. Lett., 7 (1978), pp. 24–26.
- [9] R. MCNAUGHTON AND S. PAPERT, *Counter-Free Automata*, MIT Press, Cambridge, MA, 1971.
- [10] J. A. BRZOZOWSKI, *Hierarchies of aperiodic languages*. Revue Française. Aut. Inf. Rech. Op., 10 (1976), pp. 33–49.
- [11] J. W. THATCHER, *Generalized sequential machine maps*, J. Comp. System. Sci., 4 (1970), pp. 339–367.
- [12] S. EILENBERG AND J. B. WRIGHT, *Automata in general algebras*, Inform. and Control, 11 (1967), pp. 452–470.
- [13] J. MEZEI AND J. B. WRIGHT, *Algebraic automata and context-free sets*, Inform. and Control, 11 (1967), pp. 3–29.
- [14] J. A. BRZOZOWSKI, K. ČULÍK, II AND A. GABRIELIAN, *Classification of noncounting events*, J. Comput. System. Sci., 5 (1971), pp. 41–53.
- [15] R. S. COHEN AND J. A. BRZOZOWSKI, *Dot-depth of star-free events*, J. Comput. System Sci., 5 (1971), pp. 1–16.
- [16] A. R. MEYER, *A Note on star-free events*, J. Assoc. Comput. Mach., 16 (1969), pp. 220–225.
- [17] M. P. SCHÜTZENBERGER, *On finite monoids having only trivial subgroups*, Inform. and Control, 8 (1965), pp. 190–194.
- [18] S. CRESPI-REGHIZZI AND D. MANDRIOLI, *Abstract Profiles for Context-Free Languages*, Int. Rep. 77-18, Istituto di Elettrotecnica ed Elettronica, Politecnico di Milano, Milan, Italy, December, 1977.
- [19] W. J. LEVELT, *Formal Grammars in Linguistics and Psycholinguistics, Vol I*, Mouton, The Hague, 1974.
- [20] S. CRESPI-REGHIZZI, *Noncounting languages and learning*, Computer Oriented Learning Processes, J. C. Simon, ed., Noordhoff, Groningen, 1976.
- [21] J. SAKAROVITCH, *Sur les monoides syntactiques des langages algébriques déterministes*, Proc. 3rd Int. Colloquium on Automata, Languages and Programming, Edinburgh, 1976, pp. 52–65.
- [22] H. J. SHYR AND G. THIERRIN, *Left-noncounting languages*, Internat. J. Comput. Inform. Sci., 1 (1975), pp. 95–102.
- [23] G. LAKOFF, *Irregularity in Syntax*, Holt, Reinhart and Winston, New York, 1970.
- [24] S. CRESPI-REGHIZZI, G. GUIDA AND D. MANDRIOLI, *Operator-Precedence Grammars and the Noncounting Property*, Int. Rep. 79-6, Istituto di Elettrotecnica ed Elettronica, Politecnico di Milano, Milan, Italy, 1979.
- [25] Z. MANNA, *The Mathematical Theory of Computation*, McGraw-Hill, New York, 1974.

SEQUENCING JOBS WITH UNEQUAL READY TIMES TO MINIMIZE MEAN FLOW TIME*

M. I. DESSOUKY† AND J. S. DEOGUN‡

Abstract. This paper presents a procedure for sequencing a set of jobs on a single processor (machine) with the objective of minimizing the mean flow time, when the jobs may have unequal ready times. The procedure involves implicit enumeration with the branch and bound technique, coupled with some devices to improve the efficiency of the search. The devices include a sufficient optimality condition, a simple but tight lower bound and rules for tree pruning. The approach has potential applications to other sequencing problems.

Key words. sequencing, scheduling, branch and bound

1. Introduction. The problem of sequencing n jobs on one processor or machine has been studied extensively under different assumptions and objective functions. In the simple problem of sequencing n jobs with equal ready times and no imposed due dates with the objective of minimizing the total flow time, it has been shown [2] that the shortest processing time (SPT) rule provides an optimal solution. According to this rule, jobs are sequenced from beginning to end on the basis of an ascending order of their processing times. One of the most constraining assumptions in this simple problem is the equality of the ready times of the jobs. The ready time of a job on a machine is the earliest time that the job can be made available for the machine to start processing it. This time is dictated by the completion time of the last operation conducted on the job at other machines, the arrival time of materials to the machine, the dispatching time of job instructions, and other relevant factors. It is conceivable that in the general case, the job ready times may not be identical.

The inequality of the ready times has been recognized in research on other single machine problems, where due dates were also taken into consideration [1], [3]. The objective of this paper is to develop an optimal scheduling procedure for the case when the ready times are not equal, and the total flow time is to be minimized. No due dates will be considered. Lenstra et al. [4] have shown that the problem is NP-complete; accordingly, an enumerative procedure for solving the problem will be needed.

2. Problem statement. A set N of n jobs, $N = \{i \mid i = 1, 2, \dots, n\}$, is to be processed, one job at a time, on a single processor (machine). For each job i , the ready time, r_i and the processing time, p_i , are given. Completion of all jobs requires establishing a sequence $S = (s_1, s_2, \dots, s_n)$, where s_y is the index number of the job in position y . When a job parameter or variable is identified by the job's position in a given sequence rather than its index number, the position is indicated as an underlined subscript to the parameter or variable. Thus r_4 means the ready time of the job in position 4 in the sequence considered.

Suppose that a sequence is constructed by adding one job at a time, starting from position 1. At any point, we have a partial sequence S_K of a job set $K \subseteq N$, $S_K = (s_1, \dots, s_k)$. The earliest start time of a job $i \in N$, R_i , and its completion time, C_i , are

* Received by the editors January 11, 1979, and in revised form January 23, 1980.

† Department of Mechanical and Industrial Engineering, University of Illinois, Urbana, Illinois 61807.

‡ Department of Computer Science, University of South Carolina, Columbia, South Carolina 29208.

given by

$$(1a-c) \quad R_i = \begin{cases} r_i & \text{if } i = s_1, \\ \max(r_i, C_{y-1}) & \text{if } i = s_y \in K, i \neq 1, \\ \max(r_i, C_k) & \text{if } i \in \bar{K} = N - K; \end{cases}$$

$$(2) \quad C_i = R_i + p_i.$$

For any job i , the flow time F_i and the waiting time W_i are defined as follows: $F_i = C_i - r_i$ and $W_i = R_i - r_i$. For a sequence S , the total completion time $C(S) = \sum_{i=1}^n C_i(S)$, the total flow time $F(S) = \sum_{i=1}^n F_i(S)$, and the total waiting time $W(S) = \sum_{i=1}^n W_i(S)$. Conway, Maxwell and Miller [2] show that a sequence S^0 which minimizes $C(S)$ will also minimize $F(S)$ and $W(S)$. In addition, the mean values \bar{C} , \bar{F} , and \bar{W} are minimized. The purpose of this paper is to present a procedure for determining S^0 such that $C(S^0) = \min_S C(S)$. This problem, with equal or unequal ready times, is commonly called the $n/1/\bar{F}$ problem [2].

3. Two sequencing rules. First, we present two rules for sequencing the n jobs which are utilized in the derivation of a procedure for solving the $n/1/\bar{F}$ problem addressed. Each rule specifies priority criteria for adding a job to an existing partial sequence, $S_K, K \subseteq N$, starting with position 1.

A. The earliest completion time (ECT) rule: Select job i with $\min_{i \in \bar{K}} C_i$. Break ties by choosing i with $\min R_i$, and further ties by choosing i with $\min i$. Update R_i and $C_i, i \in \bar{K}$, after each addition using (1c) and (3).

B. The earliest start time (EST) rule: Select job i with $\min_{i \in \bar{K}} R_i$. Break ties by choosing i with $\min C_i$, and further ties by choosing i with $\min i$. Update R_i and $C_i, i \in \bar{K}$ after each addition using (1c) and (2).

It can be easily shown that for any job set $K \subseteq N$, an EST sequence guarantees the minimum completion time for the last job in the set.

4. The approach. The proposed procedure involves a branch and bound search conducted along the branches of a tree in which a node at level k represents a partial sequence S_K of a set K of k jobs. For each node S_K , we compute a lower bound $\underline{C}(S^*|S_K)$ and an upper bound $\bar{C}(S^*|S_K)$ on $C(S^*|S_K)$, the minimal total completion time of any sequence starting with S_K , that is, conditional on S_K .

A node at level $k + 1$ is formed by selecting a job $i \in \bar{K}$ and adding it to S_K in position $k + 1$ to form (S_K, i) . At any iteration, the node being expanded is called the *current node* and has the current lowest lower bound. A closed (fathomed) *node* is one whose corresponding partial sequence has been found dominated, and, hence, is eliminated from consideration. Dominance is tested between nodes generated from the same parent. A partial sequence (S_K, j) is dominated if another partial sequence (S_K, i) exists and $C(S^*|(S_K, i)) \geq C(S^*|(S_K, j))$; it is *strictly dominated* if $C(S^*|(S_K, i)) > C(S^*|(S_K, j))$. We apply a number of tests (pruning rules or elimination criteria) to identify dominated nodes. A set V contains all active nodes ordered by nondecreasing lower bounds with the current node placed at its beginning. An *active node* is one which has not been found dominated.

5. Optimality and dominance properties. In this section, we present relevant properties of the $n/1/\bar{F}$ problem which are useful in testing a sequence for optimality, in defining lower bounds and in devising elimination criteria. We state the properties as theorems with proof given in the Appendix. We will identify R_i and C_i by the sequence in question whenever it is not clear from the context.

THEOREM 1. *Given a job set N and a partial sequence S_K , $K \subset N$, if a job $i \in \bar{K} = N - K$ has $p_i \leq p_j$ all $j \in \bar{K}$, and a job $h \in \bar{K}$ has $R_h(S_K) \geq R_i(S_K)$, then i dominates h in position $k + 1$. If $R_h(S_K) > R_i(S_K)$, then (S_K, i) strictly dominates (S_K, h) .*

THEOREM 2. *Consider a job set N of n jobs, and a partial sequence S_K of k jobs, $K \subset N$, and let job $i \in \bar{K}$ have $C_i(S_K) \leq C_j(S_K)$ all $j \in \bar{K}$. A partial sequence (S_K, j) is strictly dominated if $r_j \geq C_i(S_K)$.*

THEOREM 3. *Given S_K , $K \subset N$, and two jobs $i, j \in \bar{K}$, if $p_i \leq p_j$ and $C_j(S_K) \geq C_i(S_K)$, then (S_K, i) dominates (S_K, j) .*

In a sequence S , define a block $b \subseteq S$ as a set of consecutive jobs with the first job s_u having $r_u > C_{u-1}(S)$ and all other jobs $s_w \in b$ having $r_w \leq C_{w-1}(S)$.

THEOREM 4. *Let S^c be an ECT sequence of the set N consisting of one block with job i at the beginning and S^a be another sequence of the same set starting with job j . If $r_i = r_j$, then $C(S^c) \leq C(S^a)$; and if $r_i < r_j$, then $C(S^c) < C(S^a)$.*

COROLLARY 4. *An ECT sequence of a job set is optimal if it consists of a single block and the first job has the smallest ready time.*

THEOREM 5. *A sufficient condition for the optimality of a sequence is that every block is ordered according to ECT and starts with the job having the smallest ready time.*

6. The algorithm. The algorithm consists of three basic phases: initialization, branching and termination. Initialization involves defining the initial values of the variables, constructing an initial sequence and testing the sequence for optimality. Branching is an iterative procedure; in each iteration it generates the descendants of the current node, eliminates dominated nodes, computes lower and upper bounds for active nodes, updates the file of active nodes and identifies the next current node, the one with the least lower bound. The upper bound at a node is the value of a complete sequence starting with the node's partial sequence. A noncurrent node is dominated if its lower bound is not smaller than the current lowest upper bound, which is the best solution reached thus far. The termination phase consists of identification of the optimal sequence S^0 and computation of $C(S^0)$ and $F(S^0)$.

Since the procedure for determining the lower and upper bounds at a node S_K is used repetitively in the algorithm, we list it separately below as procedure BOUND. To compute the upper bound $\bar{C}(S^* | S_K)$, we construct a complete sequence S . We define S as $S = (S_K, S_{\bar{K}})$, where $S_{\bar{K}}$ is an ECT sequence of \bar{K} following S_K . We later demonstrate that S is generally a tight upper bound. To determine the lower bound $\underline{C}(S^* | S_K)$, we construct a sequence $S' = (S_K, S'_{\bar{K}})$ which is similar to S above, in the order of jobs and their processing times, but with the ready times partially relaxed. To construct S' given S_K , schedule each job s'_{k+h} , $1 \leq h \leq n - k$ (s'_{k+h} is identical to s_{k+h} in S) to start at the completion time of the preceding job s'_{k+h-1} , or at the minimum ready time of unscheduled jobs, including s'_{k+h} itself, whichever is greater. The sum of completion times of the jobs in S' , $C(S')$, is a lower bound at S_K , $\underline{C}(S^* | S_K)$. This statement is proven in Theorem 6.

I. Initialization.

1. Let N be the set of n jobs $\{i | i = 1, 2, \dots, n\}$. Define the parameters r_i and p_i for all $i \in N$.
2. Set $k = 0$, $V = K = \phi$, $S_K = \phi$, $\bar{K} = N - K$ and $C_k(S_K) = C(S_K) = 0$.
3. To construct an ECT sequence S^c of N and compute upper and lower bounds, define $y = k$, $Y = K$, $\bar{Y} = \bar{K}$, $S_Y = S_K$, $C(S_Y) = C(S_K)$ and $C'_y(S_Y) = C_y(S_Y) = 0$. Go to procedure BOUND. The lower bound $\underline{C}(S^0) = \text{LB}$, the upper bound $\bar{C}(S^0) = \text{UB}$ and $S^c = S_y$. Set the lowest upper bound $\text{LUB} = \text{UB}$.
4. Apply the optimality test of Theorem 5 to S^c . If S^c passes the test, go to step 12; otherwise, proceed to step 5.

II. *Branching.*

5. Define the set of jobs eligible for adding to S_K as $E = \bar{K}$. Compute $R_j(S_K) = \max(r_j, C_k(C_K))$, $C_j(S_K) = R_j(S_K) + P_j$, and $C_m(S_K) = \min_{j \in \bar{K}} C_j(S_K)$. Eliminate all j from E for which $r_j \geq C_m(S_K)$ (Theorem 2).

6. Identify v satisfying $p_v = \min_{j \in \bar{K}} p_j$. If $v \in E$, eliminate all $j \in E$ for which $R_j(S_K) \geq R_v(S_K)$ (Theorem 1).

7. Eliminate j from E if some $i \in E$ exists for which $p_i \geq p_j$ and $C_i(S_K) \leq C_j(S_K)$ (Theorem 3).

8. For each $i \in E$, determine the lower and upper bounds at (S_K, i) and the conditional ECT sequence S^i as follows: set $y = k + 1$, $Y = \{K \cup i\}$, $\bar{Y} = \{\bar{K} - i\}$, $S_Y = (S_K, i)$, $C(S_Y) = C(S_K) + C_i(S_K)$ and $C'_y(S_Y) = C_y(S_Y) = C_i(S_K)$. Go to procedure BOUND. We obtain $\underline{C}(S^* | (S_K, i)) = \text{LB}$, $\bar{C}(S^* | (S_K, i)) = \text{UB}$, and $S^i = S_Y$. If $\text{LB} \geq \text{UB}$, eliminate i from E ; otherwise, set $\text{LUB} = \min(\text{LUB}, \text{UB})$.

9. Remove node S_K from V , and add nodes (S_K, i) , all $i \in E$. Store $C_i(S_K)$, $\underline{C}(S^* | (S_K, i))$, $\bar{C}(S^* | (S_K, i))$.

10. Identify the node S_Q in V with the least lower bound. Let the number of jobs in the set Q be q . Set $k = q$, $K = Q$, $\bar{K} = N - K$, $S_K = S_Q$, $C(S_K) = C(S_Q)$ and $C_k(S_K) = C_q(S_Q)$. Let the last job in S_K be i .

11. To recreate the conditional ECT sequence S^i , set $y = k$, $Y = K$, $\bar{Y} = \bar{K}$, $S_Y = S_K$, $C(S_Y) = C(S_K)$ and $C_y(S_Y) = C_i(S_Y)$. Go to procedure BOUND. If $\bar{K} = \text{or } \underline{C}(S^* | S_K) = \text{LB} = \text{UB} = \bar{C}(S^* | S_K)$ proceed to step 12; otherwise, go to step 5.

III. *Termination.*

12. Let $S^0 = S^i$, $C(S^0) = \bar{C}(S^* | S_K)$, and $F(S^0) = C(S^0) - \sum_{i=1}^n r_i$.

Terminate.

Now to determine LB , UB and S^i at a node representing a partial sequence S_Y of y jobs ending with job i , follow procedure BOUND outlined below. The variables $C(S_Y)$, $C'_y(S_Y)$ and $C_y(S_Y)$ have already been defined in the calling step of the main algorithm.

Procedure BOUND.

B1. Initialize: $x = y$, $X = Y$, $S_X = S_Y$, $\text{LB} = \text{UB} = C(S_Y)$ and $C'_x = C_x = C_y(S_Y)$.

B2. Let $m_{x+1} = \min_{j \in \bar{X}} r_j$ and compute $R_j = \max(C_x, r_j)$, and $C_j = R_j + p_j$, all $j \in \bar{X}$.

B3. Select $h \in \bar{X}$ such that $C_h = \min_{j \in \bar{X}} C_j$. Break ties by choosing h to minimize R_h .

B4. Place h in position $x + 1$ and set $R'_{x+1} = R'_h = \max(C_x, m_{x+1})$, $C'_{x+1} = C'_h = R'_{x+1} + P_h$, $P_{x+1} = R_h$; $C_{x+1} = C_h$, $\text{LB} = \text{LB} + C'_{x+1}$, and $\text{UB} = \text{UB} + C_{x+1}$.

B5. Set $x = x + 1$, $X = \{X \cup h\}$, $\bar{X} = \{\bar{X} - h\}$, and $S_X = (S_x, h)$.

B6. If $x = n$, set $S = S_X$; $\underline{C}(S^* | S_Y) = \text{LB}$; and $\bar{C}(S^* | S_Y) = \text{UB}$ and return to calling step; otherwise, go to step B2.

The sequence defining the upper bound UB is $S = (S_Y, S_{\bar{Y}})$, an ECT ordering of \bar{Y} . Now let $S' = (S_Y, S_{\bar{Y}'})$ be the sequence defining the lower-bound LB ; that is, $\text{LB} = C(S')$. It may be seen that S and S' are identical in their job order but that their job start and completion times may be different. Theorem 6 establishes that $C(S')$ is a lower bound on the optimal solution conditional on S_Y .

THEOREM 6. $C(S') = C(S_Y, S_{\bar{Y}'}) \leq C(S^* | S_Y)$.

7. Example. This example is especially constructed to demonstrate most of the features of the procedure. Table 1 exhibits the job set parameters. For simplicity, we have ordered the jobs by ECT. Detailed computations for the first iteration are given below, where the step numbers correspond to those in the algorithm.

1. $n = 5$, and values of r_i and p_i for $i = 1, 2, \dots, 5$ are given in Table 1.
2. Set $k = 0$, $K = \phi$, $S_K = \phi$ and $\bar{K} = (1, 2, 3, 4, 5)$. Set $C_k = C(S_K) = 0$.

TABLE 1. Job set parameters

Job Indices	Ready Times	Processing Times
1	35	3
2	22	18
3	34	17
4	37	21
5	66	25

3. $S^c = (1, 2, 3, 4, 5)$. From procedure BOUND, $C(S^*|S_K) = LB = 205$ and $\bar{C}(S^*|S_K) = UB = 386$.

4. The optimality test of Theorem 6 fails because the first job does not have the minimum ready time.

5. $E = (1, 2, 3, 4, 5)$. $C_1 = 38$ is the minimum among all $C_j, j \in E$. $R_5 = 66 > C_1 = 38$. Therefore, removing job 5 from E (Theorem 2), $E = (1, 2, 3, 4)$.

6. Job 1 has the shortest $p_i, p_1 = 4$. Since $r_4 = 37 > r_1 = 3$, eliminate 4 from E (Theorem 1). $E = (1, 2, 3)$.

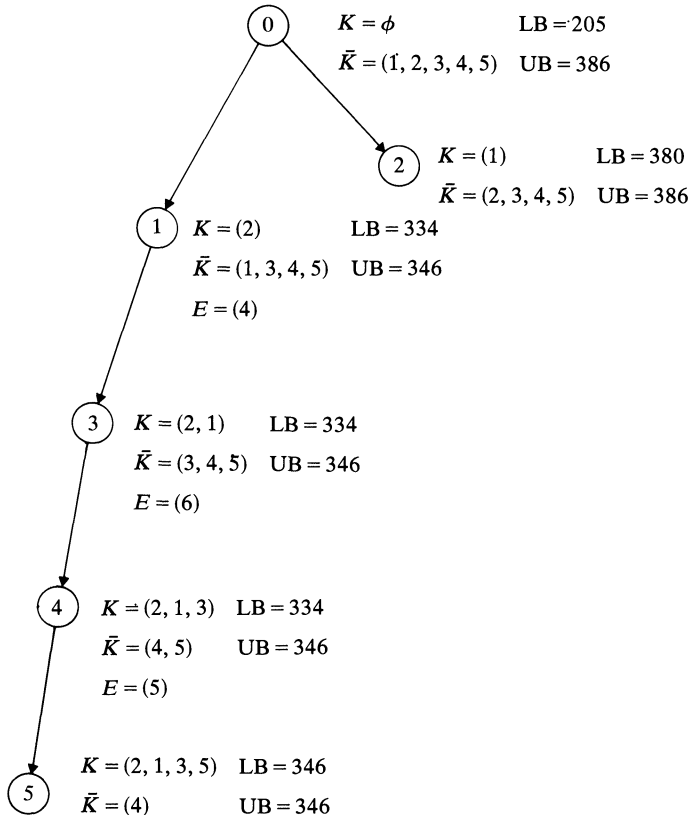


FIG. 1

7. $C_3 = 54$, $>C_2 = 40$ and $p_3 = 17 < p_2 = 18$, hence job 2 dominates job 3 (Theorem 3). Remove job 3 from E , $E = (1, 2)$.

8. Compute lower and upper bounds by procedure BOUND. $LB(1) = C(S^*|(1)) = 380$ and $UB(1) = \bar{C}(S^*|(1)) = 386$. Similarly, $LB(2) = C(S^*|(2)) = 334$ and $UB(2) = \bar{C}(S^*|(2)) = 346$. The lowest upper bound LUB is updated as follows: $LUB = \min(386, 346) = 346$. As $C(S^*|(S_{\bar{K}}, 1)) = 380$ $LUB = 346$. Remove job 1 from E . $E = (2)$.

9. Node (2) is added to set V .

10. Locate the node in V with the lowest lower bound. Set $y = 1$, $Y = K = \{2\}$, $\bar{Y} = \bar{K} = \{1, 3, 4, 5\}$.

11. Using procedure BOUND, reconstruct $S_{\bar{K}} = (1, 3, 4, 5)$ and $LB(2) = C(S^*|(2)) = 334$ and $UB(2) = \bar{C}(S^*|(2)) = 346$. Neither $\bar{K} = \phi$, nor $LB(2) = UB(2)$ therefore, proceed to step 5.

This completes the first iteration in the branch and bound algorithm. The complete tree of optimal solutions generated is shown in Fig. 1. The optimal sequence is (2, 1, 3, 5, 4) and the minimum value of the total completion time is 346. The minimum total flow time is 152.

8. Experimental evaluation. The algorithm was programmed and tested on a Cyber 175, using a FORTRAN G compiler. Two tests were conducted, the first test consisting of 220 job sets, each generated randomly from two independent probability distributions of r_i and p_i . Four distributions for r_i and eleven for p_i combined to produce 44 job set distributions, each generating five problems (job sets of 20 jobs each). The second test consisted of 44 job sets, 50 jobs each, with each job set from each of the forty-four distributions above. The probability distributions were chosen to be uniform to avoid biasing particular values within their ranges, and to eliminate the need for truncation. They are specified by the upper and lower limits of their ranges. We selected the distributions to include the ranges which are most challenging to the algorithm.

Table 2 shows the experimental results for all problems in the first test. Out of the 220 problems, 214, 215 and 217 were solved within $\frac{1}{2}$, 1 and 2 seconds respectively. Within the two-second limit, the mean and maximum computation times were .05 and 1.88 seconds, the mean and maximum number of nodes generated were 13 and 514, and the mean and maximum number of iterations were 9 and 497. The three problems which were not solved within two seconds were later run to completion. The maximum computation time for any problem was 4.61 seconds. The mean time to solve all 220 problems was 0.08 seconds. Comparing the optimal solution for each of the three problems with that obtained within two seconds, the mean and maximum per cent deviation from optimality were 17.6% and 31%. Averaged over all problems, the mean per cent deviation was 0.24%.

Apart from these summaries, Table 2 also indicates that the performance of the algorithm deteriorates as the ratio of the range of r_i to the mean of p_i exceeds 2.5.

In the second test all problems were run to completion. Table 3 shows the experimental results for the second test. Out of 44 problems, 30, 39 and 40 problems were solved within $\frac{1}{2}$, 1, and 2 seconds. The mean and maximum computation times were .96 and 8.47 seconds, the mean and maximum number of nodes generated were 47 and 640, and the mean and maximum number of iterations were 42 and 484. These results demonstrate that the algorithm can be effectively used to solve large problems.

The effectiveness of ECT sequencing may be demonstrated by the fact that in 173 out of the 220 problems the initial sequence was optimal, although it was recognized as optimal in only 53 problems. The mean deviation from optimality of the initial ECT

TABLE 2
Summary of results: Branch and bound—ECT procedure for $N/1/\bar{F}$ (20 jobs in each set)

Range of ready times	Range of processing times											
	1-25	26-50	1-75	51-75	1-125	76-100	1-175	101-125	1-225	126-150	1-275	
0-200	***	2 1	8 8	1 0	25 19	1 1	2 2	0 0	1 2	1 1	0 0	Min 0 Max 43 Mean 3
	***	3 2	17 14	3 4	17 14	1 1	8 10	0 0	3 4	1 1	0 0	
	64 24	2 1	13 9	0 1	7 9	0 0	11 13	1 1	3 4	0 0	3 4	
	178 43	1 0	7 3	1 0	7 7	0 0	7 10	1 1	1 1	0 0	1 1	
	3 1	1 0	9 8	0 1	1 1	1 0	2 4	0 0	1 2	0 1	1 3	
	Min 1	Min 0	Min 3	Min 0	Min 1	Min 0	Min 2	Min 0	Min 1	Min 0	Min 0	
	Max 43	Max 2	Max 14	Max 4	Max 19	Max 1	Max 13	Max 1	Max 4	Max 1	Max 4	
Mean 13	Mean 0	Mean 8	Mean 1	Mean 10	Mean 0	Mean 7	Mean 0	Mean 2	Mean 0	Mean 1		
25-175	13 5	4 4	12 8	1 1	30 23	0 0	9 14	1 1	4 4	0 1	10 10	Min 0 Max 188 Mean 10
	71 43	9 6	25 17	1 1	5 3	1 0	7 6	0 0	8 12	0 0	4 6	
	25 5	5 6	***	1 0	23 23	1 1	4 5	0 0	3 4	0 0	1 1	
	395 150	0 0	2 1	1 1	1 1	0 0	1 2	1 2	3 4	0 0	1 4	
	497 188	0 0	11 5	1 2	1 2	1 2	2 3	2 3	3 3	0 0	3 4	
	Min 5	Min 0	Min 1	Min 0	Min 1	Min 0	Min 2	Min 0	Min 3	Min 0	Min 1	
	Max 188	Max 6	Max 17	Max 2	Max 23	Max 2	Max 14	Max 3	Max 12	Max 1	Max 10	
Mean 13	Mean 3	Mean 6	Mean 1	Mean 10	Mean 0	Mean 6	Mean 1	Mean 5	Mean 0	Mean 4		
50-150	102 50	1 1	7 9	0 0	3 4	1 1	1 2	0 0	3 3	0 0	3 5	Min 0 Max 50 Mean 3
	18 8	2 3	14 10	0 0	8 10	0 1	2 2	0 0	6 10	0 1	3 5	
	6 3	1 1	10 9	0 1	2 2	1 1	1 2	0 0	1 1	0 0	2 3	
	3 1	1 1	13 12	1 1	12 17	1 3	2 1	0 0	2 4	0 0	1 1	
	9 7	1 1	6 6	0 0	1 1	1 1	0 1	0 0	1 1	0 1	1 1	
	Min 1	Min 1	Min 6	Min 0	Min 1	Min 1	Min 1	Min 0	Min 1	Min 0	Min 1	
	Max 50	Max 3	Max 12	Max 1	Max 17	Max 3	Max 2	Max 0	Max 10	Max 1	Max 5	
Mean 13	Mean 1	Mean 9	Mean 0	Mean 6	Mean 1	Mean 1	Mean 0	Mean 3	Mean 0	Mean 3		
75-175	90 47	1 1	7 11	0 0	6 7	1 2	0 0	1 2	1 1	0 1	0 0	Min 0 Max 47 Mean 3
	58 21	0 0	2 1	1 0	27 36	0 0	2 2	1 2	1 2	0 0	2 2	
	16 9	1 1	4 4	1 0	4 4	0 1	1 1	1 2	2 2	1 2	0 0	
	3 2	1 1	19 21	1 2	0 0	0 0	1 2	1 3	1 2	0 0	2 1	
	25 14	0 0	2 1	1 1	1 2	1 1	2 3	1 2	3 5	0 1	3 6	
	Min 2	Min 0	Min 1	Min 0	Min 0	Min 0	Min 0	Min 2	Min 1	Min 0	Min 0	
	Max 47	Max 1	Max 21	Max 2	Max 36	Max 2	Max 3	Max 3	Max 5	Max 2	Max 6	
Mean 18	Mean 0	Mean 7	Mean 0	Mean 9	Mean 0	Mean 1	Mean 2	Mean 2	Mean 0	Mean 1		
Min 1	Min 0	Min 1	Min 0	Min 0	Min 0	Min 0	Min 0	Min 1	Min 0	Min 0		
Max 188	Max 6	Max 21	Max 4	Max 36	Max 3	Max 14	Max 3	Max 12	Max 2	Max 10		
Mean 30	Mean 1	Mean 7	Mean 0	Mean 8	Mean 0	Mean 3	Mean 0	Mean 3	Mean 0	Mean 2		

Number of jobs = 20

Minimum computation time = 0 Minimum number of iterations = 0
 Maximum computation time = 188 Maximum number of iterations = 497
 Mean computation time = 5 Average number of iterations = 9
 Number exceeding 1/2 second = 6 Minimum number of nodes generated = 1
 Number exceeding one second = 5 Maximum number of nodes generated = 514
 Average number of nodes generated = 13

Note: For each box, the first column is the number of iterations and the second column is computer time in centi-seconds. The *'s in a column show that the corresponding problem was not solved in 2 seconds.

sequence was 3% for all problems. Thus, the use of ECT sequencing provides fairly tight upper bounds which can help to eliminate many dominated nodes.

9. Conclusions and recommendations. This paper presents an analysis of optimality and dominance conditions for the $n/1/\bar{F}$ problem with unequal ready times and a procedure for solving the problem. Although the proofs of the theorems are rather lengthy, the procedure itself is simple. Experimentation with the algorithm on a Cyber 175 computer showed in the first test that for 220 job sets with 20 jobs each, the

TABLE 3
 Summary of results: Branch and bound—ECT procedure for $N/1/\bar{F}$ (50 jobs in each set)

Range of ready times	Range of processing times											
	1-25	26-50	1-75	51-75	1-125	76-100	1-175	101-125	1-125	126-150	1-275	
0-200	308 633	3 72	22 162	1 2	38 81	1 41	5 55	1 3	5 11	1 42	11 67	Min 2 Max 633 Mean 106
25-175	455 773	10 12	28 76	1 3	37 92	2 4	5 48	0 2	31 87	0 39	11 18	Min 2 Max 773 Mean 104
50-150	313 847	1 3	14 100	1 5	12 50	3 7	19 11	0 1	6 52	0 41	8 21	Min 1 Max 847 Mean 103
75-175	484 553	1 5	7 50	1 2	4 48	1 3	6 49	1 2	4 47	1 5	2 7	Min 2 Max 653 Mean 70
Min	553	3	50	2	48	3	11	1	11	5	7	
Max	847	72	162	5	92	41	55	3	87	42	67	
Mean	701	23	97	3	67	13	40	2	49	31	28	

Number of jobs = 50

- Minimum computation time = 1
- Maximum computation time = 847
- Mean computation time = 96
- Number exceeding 1/2 second = 14
- Number exceeding 1 second = 5
- Minimum number of iterations = 0
- Maximum number of iterations = 484
- Average number of iterations = 42
- Minimum number of nodes generated = 1
- Maximum number of nodes generated = 540
- Average number of nodes generated = 47

Note: For each range of processing time, the first column is the number of iterations and the second column is computer time in centi-seconds. The statistics Min, Max and Mean are shown for computer time only.

algorithm was capable of solving any problem within a maximum of 4.61 seconds of computer time and that all but 3 out of 220 problems were solved within two seconds, with an overall average of 0.08 seconds.

The second test demonstrated the effectiveness of the algorithm to solve large problems. For 44 job sets with 50 jobs each, in the second test, the algorithm solved any problem within a maximum of 8.47 seconds and an overall average of .96 seconds.

The algorithm can be used to test alternative heuristics for closeness to optimality. It can also be applied to two and multiple machine problems in the process of computing lower bounds on the optimal solution to those problems.

Minimizing the sum of completion times is only one of many interesting functions that could be considered. However, only this function has been addressed in this paper because it gives a simple measure of the effectiveness of a schedule and it yields an algorithm which may be extended to solve more complex functions. Natural extensions are to minimize the sum of weighted completion times, and to minimize the maximum flow time.

Appendix. Proofs of theorems.

Proof of Theorem 1. Any sequence $S^h = (S_K, S_K^h)$ having h in position $k + 1$ and i in position y , $y > k + 1$, will have $r_i \leq R_i(S_K) \leq R_h(S_K) = R_{k+1}(S_K) \leq R_x(S^h)$, $k < x < y$.

Interchanging i with its immediate predecessor in position $y - 1$ will not increase the completion time of the jobs in positions $y - 1$ and y , nor that of any preceding or succeeding job in S^h . Repeating the interchange backward to position $k + 1$ following S_K will produce a sequence S^i having $C_x(S^i) \leq C_x(S^h)$, $1 \leq x \leq n$, and $C(S^i) \leq C(S^h)$; that is, (S_K, i) dominates (S_K, h) . Similarly, we can show that if $R_h(S_K) > R_i(S_K)$, then (S_K, i) strictly dominates (S_K, h) . Q.E.D.

Proof of Theorem 2. Let S^j be the optimal sequence of N given that j is in position $k + 1$ following S_K . An idle time gap equal to $r_j - C_k(S^j)$ will exist on the machine between the jobs in positions k and $k + 1$. If job i is shifted from its position, say g , to position $k + 1$ without changing the order of any other job, it will occupy no more of the machine time than the gap and will gain a reduction in its completion time. Now form S^i starting with S_K , followed by i , then j , then the remaining jobs in the same order of S^j . Shifting the jobs in positions $g + 1$ to n as early as possible may reduce their completion times without influencing the completion times of the jobs in positions $k + 2$ to g , which remain the same. Therefore, $C(S^i) < C(S^j)$ and (S_K, j) is strictly dominated. Q.E.D.

Proof of Theorem 3. Let S^j be an optimal sequence of N given that j is in position $k + 1$ following S_K . Consider removing jobs i and j from the schedule. Without affecting the completion times of any other jobs, job i may be scheduled in position $k + 1$ as $C_i(S_K) \leq C_j(S_K)$; and job j may be scheduled in i 's former position as $p_j \leq p_i$. The total completion time of jobs i and j is not increased as neither is completed later in the new schedule S^i than the other was in S^j . Thus $C(S^j) \geq C(S^i)$. Q.E.D.

Proof of Theorem 4. Assume that the ready time of each job $y \in N$ is modified to $r'_y = \max(r_y, r_i)$. This assumption will not affect the start or completion time of any job in S^c or S^a , as no job in either sequence starts before r_i .

Consider the case when $r_i = r_j$. First, we show that $C_k(S^c) \leq C_k(S^a)$, $1 \leq k \leq n$. For a given k , reorder the first k jobs in S^a according to their earliest start times (EST), assuming modified ready times, and breaking ties by giving priority to the job with the smallest completion time among tied jobs. Call the resulting sequence S'_K ; as mentioned in § 3, $C_k(S^t) \leq C_k(S^a)$. Note that S^c is both an ECT and an EST sequence of the modified set N . Therefore, at the first position x , $1 \leq x \leq k$ in which $C_x(S^c) \neq C_x(S^t)$, it must be true $C_x(S^c) < C_x(S^t)$ and $s_x^c \notin S'_K$. Therefore, replacing s_k^t with s_k^c will not result in an increase in any $C_y(S^t)$, $1 \leq y \leq k$. This replacement may cause the set of jobs in S'_K following the new s_k^t to lose its EST sequencing property. In that case, reordering the set by EST will not increase $C_k(S^t)$ since $C_x(S^t)$ has been reduced. Redefine S'_K to describe the new sequence. Repeating this procedure for all y , $x < y \leq k$, comparing C_y^c and C_y^t and making necessary changes as shown above will produce a sequence identical to the partial sequence of the first k jobs in S^c , without increasing C_k^t . Thus $C_k^c \leq C_k^t \leq C_k^a$, $1 \leq k \leq n$ and $C(S^c) = \sum_{k=1}^n C_k^c \leq \sum_{k=1}^n C_k^a = C(S^a)$, proving the first part of the theorem.

For $r_i < r_j$, applying the same procedure yields $C_k^c \leq C_k^a$, $1 \leq k \leq n$. From Theorem 3, $C_i \leq C_j$ implies that if $p_i > p_j$ then $C(S^c) < C(S^a)$ and the theorem is proved. If $p_i \leq p_j$ then $C_1^c = C_1^c < C_1^a = C_1^a$, and $C(S^c) < C(S^a)$. Q.E.D.

Proof of Theorem 5. Let the number of blocks be B . If $B = 1$, the theorem is true by Corollary 4. Suppose that the theorem is true for the first b blocks where $1 \leq b < B$. From the definition of a block and the conditions of the theorem, $r_j > C_i(S)$ for all j in block $b + 1$, where i is the last job in the optimal sequence of the first b blocks. By Theorem 2, all jobs in block b are dominated by job i ; therefore, for optimality, no jobs in block $b + 1$ should be scheduled earlier and block $b + 1$ can be optimized separately and then combined to form a complete optimal sequence for $b + 1$ blocks. By induction, all B blocks can be optimized separately. Since each block satisfies the optimality conditions of Corollary 4, then the sequence is optimal. Q.E.D.

Proof of Theorem 6. First, we define the parameters of the set of jobs \bar{Y}' in $S' = (S_Y, S_{\bar{Y}'})$ to satisfy the following conditions: (1) for each $s'_x \in \bar{Y}$, $y < x \leq n$, $r'_x \leq r_x$, $p'_x = p_x$; and (2) $S_{\bar{Y}'}$ is an ECT sequence of \bar{Y}' following S_Y . Therefore, if we show that $S_{\bar{Y}'}$ passes the test of optimality of Theorem 5, then $C(S') = C(S_Y, S_{\bar{Y}'}) \leq C(S_Y, S_{\bar{Y}}^*) = C(S^* | S_Y)$, where $S_{\bar{Y}}^*$ is the optimal sequence of \bar{Y} following S_Y since the ready time constraints of \bar{Y}' are either equal to or more relaxed than those of \bar{Y} . If we define $r'_1 = r_1$ and $r'_x, y \leq x \leq n$, iteratively by

$$(3) \quad r'_x = \min \{r_x, \max (C'_{x-1}, m_x)\}, \quad y < x \leq n,$$

where $m_x = \min_{y < x < n} r'_y$, then $m_x \leq r'_x \leq r_x$, satisfying condition (1) for \bar{Y}' .

Furthermore, the earliest start time of s'_x ,

$$(4) \quad R'_x = \max(C'_{x-1}, r'_x) = \max [C'_{x-1}, \min \{r_x, \max (C'_{x-1}, m_x)\}].$$

Since $m_x \leq r_x$, this expression reduces to

$$(5) \quad R'_x = \max (C'_{x-1}, m_x)$$

and

$$(6) \quad C'_x = R'_x + p'_x = R'_x + p_x.$$

Therefore, (5) and (6) define R'_x and C'_x , as given by step B4 on procedure BOUND. We now show that the resulting sequence $S_{\bar{Y}'}$ follows the ECT rule. For this to be true,

$$(7) \quad C'_x = R'_x + p_x = \max (C'_{x-1}, r'_x) + p_x \leq \max (C'_{x-1}, r'_w) + p_w, \quad x < w \leq n.$$

Consider any w , $x < w \leq n$. Since $S_{\bar{Y}'}$ has the same order of $S_{\bar{Y}}$, which is an ECT sequence of \bar{Y} , then

$$(8) \quad C_x = \max (C_{x-1}, r_x) + p_x \leq \max (C_{x-1}, r_w) + p_w, \quad x < w \leq n.$$

Before we proceed, we observe that $R'_x \leq R_x$, $C'_x \leq C_x$ and for $x < w$, $C'_x \leq R'_w$ and $m_x \leq m_w$. From (3) and (5),

$$(9) \quad r'_x = \min (r_x, R'_x).$$

Two cases exist:

Case 1. $r_x \geq r_w$. From (8), $p_x \leq p_w$. Since $m_x \leq m_w$ and $C'_{x-1} < C'_{w-1}$ then $R'_x = \max (m_x, C'_{x-1}) \leq \max (m_w, C'_{w-1})$ and $C'_x = R'_x + p_x \leq \max (C'_{x-1}, r'_w) + p_w$.

Case 2. $r_x < r_w$. From (9) and $R'_x < C'_x \leq R'_w$, $r'_x = \min (r_x, R'_x) \leq \min (r_w, R'_w) = r'_w$. If $r_w \geq R'_x$, then $r'_w = R'_w$, implying that $\max (C'_{x-1}, r'_w) + p_w \geq R'_w + p_w = R'_w + p_w > R'_w \geq C'_x$. If $r_w < R'_w$ then $r'_w = r_w$, while $r'_x \leq r_x$. Noting that $C_{x-1} \geq C'_{x-1}$, and applying (8) we again have $\max (C'_{x-1}, r'_x) + p_x \leq \max (C'_{x-1}, r'_w) + p_w$.

Therefore, (7) is satisfied in all cases, proving that $S_{\bar{Y}'}$ orders \bar{Y}' by ECT following S_Y and satisfying condition (2) for \bar{Y}' .

By definition, any $s'_x, y < x \leq n$, starting a block has $r'_x > C'_{x-1}$. From (3) this condition implies that $r'_x = m_x \leq m_w \leq r_x, x < w \leq n$, and $S_{\bar{Y}'}$ satisfies the optimality condition of Theorem 5. Therefore, $C(S') = C(S_Y, S_{\bar{Y}'}) \leq C(S_Y, S_{\bar{Y}}^*) = C(S^* | S_Y)$. Q.E.D.

REFERENCES

[1] P. BRATLEY, M. FLORIAN AND P. ROBILLARD, *On scheduling with earliest start and due dates with application to computing bounds for the (n/m/G/F max) problem*, Naval Research Logistics Quarterly, 20 (1973), pp. 57-67.

- [2] R. CONWAY, W. MAXWELL AND L. W. MILLER, *Theory of Scheduling*, Addison-Wesley, Reading, MA, 1967.
- [3] M. I. DESSOUKY AND C. R. MARGENTHALER, *The one-machine sequencing problem with early starts and due dates*, AIIE Trans., 4 (1972), pp. 214-222.
- [4] J. K. LENSTRA, A. H. R. RINNOOY KAN AND P. BRUCKER, *Complexity of machine scheduling problems*, Ann. Discrete Math., (1977), pp. 343-362.

LINEAR TIME AUTOMORPHISM ALGORITHMS FOR TREES, INTERVAL GRAPHS, AND PLANAR GRAPHS*

CHARLES J. COLBOURN† AND KELLOGG S. BOOTH‡

Abstract. An algorithm based upon Edmonds's procedure for testing isomorphism of trees is extended to answer various questions concerning automorphisms of a labeled forest. This and linear pattern matching techniques are used to build efficient algorithms which find the automorphism partition and a set of generators for the automorphism group, determine the order of the automorphism group, and compute a coding for forests, interval graphs, outerplanar graphs, and planar graphs.

Key words. automorphism partition, graph automorphism, interval graph, outerplanar graph

1. Preliminaries. Starting from an algorithm for tree isomorphism we show how to build efficient algorithms which can answer a number of questions concerning the automorphisms of labeled forests. These algorithms operate in time which is linear with the size of their input. They can be applied to other types of graphs whenever those graphs can be conveniently represented by labeled forests. This is the case for interval graphs and for planar graphs, as will be demonstrated in later sections. By applying linear pattern matching techniques and a knowledge of their additional structure we can produce an even faster algorithm for outerplanar graphs, a special case of planar graphs which has been examined in the literature.

We assume the usual definitions used in graph theory [3], [11]. Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. A bijection $\phi: V \rightarrow V'$ which maps pairs of adjacent vertices onto pairs of adjacent vertices and pairs of nonadjacent vertices onto pairs of nonadjacent vertices is an *isomorphism* of G with G' . An *automorphism* of a graph G is an isomorphism of G with itself. The *automorphism group* of G is the group whose elements are the automorphisms of G . A *set of generators* for the automorphism group of G is any set of automorphisms whose closure under functional composition and inverse is the entire automorphism group. Two vertices x and y are *similar* in G whenever there exists an automorphism of G which carries x onto y . Similarity is an equivalence relation on V whose equivalence classes form the *automorphism partition* of G . A *coding* is a function on graphs such that the values assigned to G and G' are identical if and only if G and G' are isomorphic.

In the following sections we will first review the tree isomorphism test and then extend it to answer other questions about automorphisms of a labeled forest. Using these results as a foundation we will go on to construct efficient automorphism algorithms for interval graphs and outerplanar graphs. Finally, we will sketch how algorithms for planar graphs might be built using these same ideas.

Weinberg previously presented algorithms for computing the automorphism partitions of triconnected planar graphs and of trees. His algorithms require

*Received by the editors January 31, 1979 and in final form April 11, 1980. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant No. A4307. This paper was typeset at the University of Waterloo using the Troff software developed for the Unix operating system. Final copy was produced on a Photon Econosetter.

†Department of Computational Science, University of Saskatchewan, Saskatoon, Saskatchewan, Canada S7N 0W0.

‡Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

$O(n^2)$ steps [36], [37]. Corneil, and later James and Riha, have produced substantially simpler algorithms for the tree problem. Their algorithms also have an $O(n^2)$ time complexity [8], [21]. Most recently Fontet has given an $O(n)$ algorithm for arbitrary planar graphs which is very similar to the approach taken here [9]. To our knowledge there are no linear time algorithms in the literature for finding a set of generators for the automorphism group of a planar graph or for computing the order of its automorphism group. All of the results on automorphisms of interval graphs and outerplanar graphs are new. Additional references to previous work are contained within subsequent sections.

2. Trees. A *tree* is a connected graph having no cycles. Edmonds presented an efficient procedure for testing isomorphism of trees which is based on a canonical numbering of the vertices [6]. This method has been rediscovered a number of times by different researchers [22], [30], [34]. A description of the algorithm and a proof that it runs in linear time is given by Aho, Hopcroft, and Ullman [1, § 3.2]. Their version of the algorithm allows the trees to be labeled but assumes that the trees are rooted. This is no real drawback because unrooted trees can be rooted in a unique fashion by finding the center or bicenter of the tree and using it as a root. This can easily be accomplished in linear time. For our purposes we will assume that all trees have been rooted.

These techniques can be applied to forests of trees having labels which consist of integers or strings of integers in a range which is $O(n)$ where n is the number of vertices in the forest. All of our algorithms will be linear in the size of the forest plus the sum of the lengths of the labels. The forest algorithms will then become the building blocks for the algorithms in later sections.

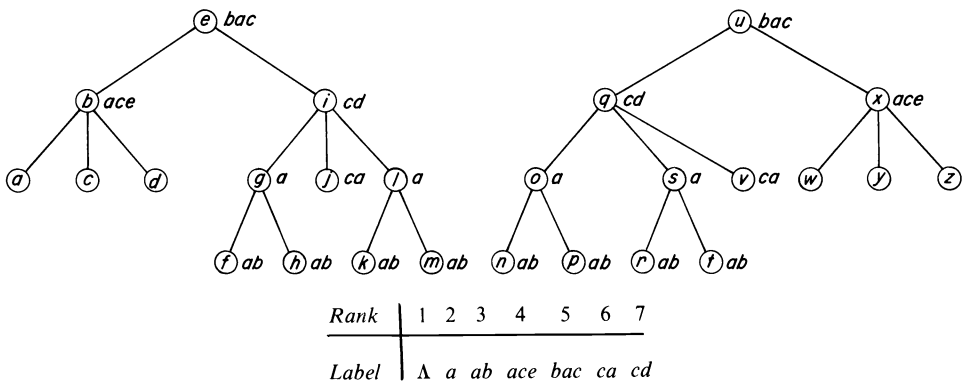


Fig. 2.1. An example of a labeled forest. Vertex names (symbols within vertices) are used to identify vertices and should not be confused with vertex labels or with any of the numbers which will later be assigned to vertices. Rank information for the bucket sorted labels is displayed below the forest. These ranks will be used to generate working labels for the *i*-numbering which follows.

We briefly sketch our version of Edmonds's procedure for testing isomorphism of two labeled forests. All of the labels are first bucket sorted and assigned an integer rank within the sorted list. Then each vertex in the forest receives an integer *i*-number according to the following scheme. Beginning at the vertices of maximum depth in the forest a *working label* is assigned which consists

of the rank of the original label followed by the i -numbers, in increasing order, of all the children. The working labels of all of the vertices at the current depth are then sorted and the i -number of each vertex becomes the rank of its working label within the sorted list of working labels. This process proceeds upwards until the roots of all of the trees have received an i -number. Using a bucket sort it is possible to perform all of these operations so that the total work is linear in the size of the forest plus the sum of the lengths of the original labels [1].

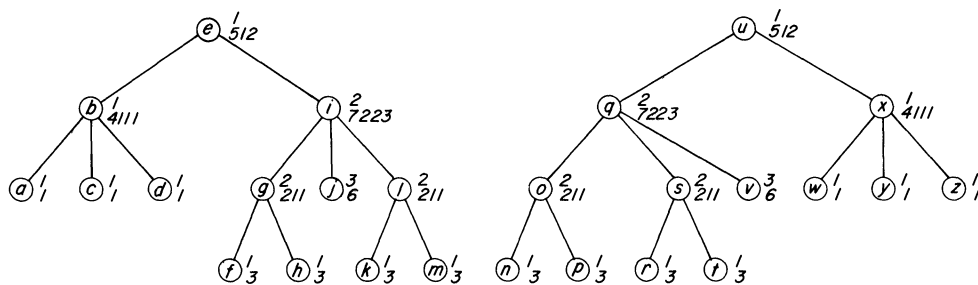


Fig. 2.2. The i -numbering for the forest in Fig. 2.1. Shown to the right of each vertex are its i -number (above) and its working label (below). A working label consists of the rank of the original vertex label followed by the i -numbers (in increasing order) of the children. Note that because the i -numbers of the roots are identical the two trees must be isomorphic.

A complete isomorphism test consists of performing the i -numbering on two labeled forests in parallel, checking at each level that the sets of working labels are the same in both forests. If a mismatch occurs at any level the forests cannot be isomorphic and the algorithm terminates announcing non-isomorphism. Otherwise the algorithm succeeds in finding an isomorphism when the sets of i -numbers assigned to the roots are identical in the two forests.

We can use a similar procedure to get other information concerning the automorphisms of a forest. Only a little more work is necessary to find the automorphism partition. Suppose that the forest has been i -numbered. The vertices are partitioned into equivalence classes by their i -numbers together with their depths. This is usually not the automorphism partition but the automorphism partition is always a refinement of this partition and can be determined from a second pass over the forest, this time top down.

LEMMA 2.1. *Let G be an i -numbered labeled forest. Two vertices x and y are similar if and only if they have the same depth and i -number and their parents (if present) are similar.*

Proof. Only If: Let x and y be two similar vertices in G . If $x = y$ then the conditions hold trivially. So assume that $x \neq y$. Any automorphism must preserve depths and ancestors within a forest. Thus x and y are at the same depth and either have similar parents or no parents at all. It remains only to show that their i -numbers are the same.

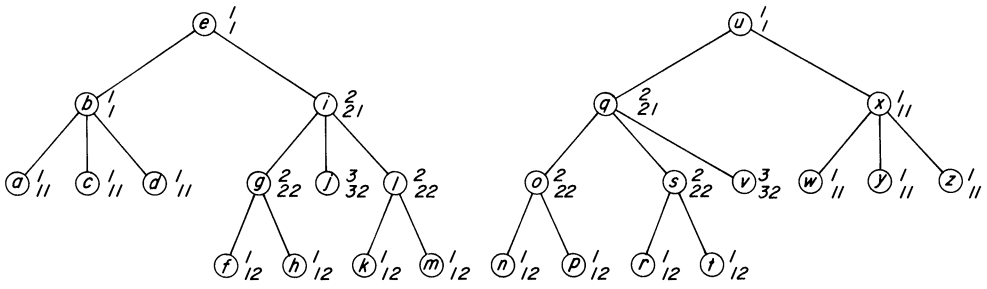
From the correctness of the isomorphism algorithm which generates the i -numbers it follows that two vertices at the same depth have the same i -number if and only if the subtrees rooted at those vertices are isomorphic. Since x is similar to y their subtrees are isomorphic and hence their i -numbers are indeed identical.

If: Again we assume that $x \neq y$ but that both are at the same depth, have

similar parents, and the same i -numbers. If x and y are at depth 0 they are roots in the forest and by the observation cited before their trees are isomorphic and hence they must be similar vertices. Otherwise x and y are at a positive depth and both have parents which are similar within the forest. We show that there is an automorphism carrying x onto y and thus complete the proof.

Any automorphism mapping the parent of x to the parent of y must map x to some sibling of y . Such an automorphism exists (the parents are similar) so we can choose some sibling of y . Call this sibling z . Clearly x and z are similar. Using the only-if part of this lemma x , y , and z all have the same i -number and thus the subtrees rooted at y and z must be isomorphic. We produce the desired mapping by composing the automorphism which carries x to z with the automorphism which interchanges the two siblings y and z . This mapping is an automorphism of G and it carries x onto y . \square

We now explain how to compute the automorphism partition of a labeled forest. After rooting each tree at its center or bicenter we assign i -numbers as before. A second set of integer j -numbers is then assigned in a top down fashion. At each level beginning with the roots, vertices are assigned working labels which consist of the i -number of the vertex followed by the j -number of the parent. These working labels are then sorted on each level and the j -number for the vertex is the rank of the working label within the sorted list of working labels. From the lemma it follows that the j -numbers plus depth information partition the vertices into similarity classes.



The blocks of the automorphism partition for the forest

$$\{a,c,d,w,y,z\} \{b,x\} \{e,u\} \{f,h,k,m,n,p,r,t\} \{g,l,o,s\} \{i,q\} \{j,v\}$$

Fig 2.3. The forest from Fig. 2.1. Shown to the right of each vertex are its j -number (above) and its working label (below). A working label consists of the i -number of the vertex followed by the j -number of its parent. The working label for a root is its i -number. The seven blocks of the automorphism partition are shown below the forest. These are found by making all vertices which have the same depth and j -number equivalent.

This j -numbering algorithm is easily implemented to run in linear time. It is a straightforward extension of Edmonds’s algorithm. We have now shown that the automorphism partition of a labeled forest can be found in linear time. So far our work duplicates results of Fontet who also realized that Edmonds’s approach could be used to compute the automorphism partition of a tree [9]. We go further, however, and produce a set of generators for the automorphism group and also determine the order of the automorphism group.

Let us first consider the problem of computing the automorphism group of a forest. We cannot hope to achieve an algorithm whose running time is linear in the size of the forest. The size of the output alone prohibits this. The star $K_{1,n}$, for example, has $n!$ automorphisms. Even computing a set of generators for such a group is costly. Weinberg has shown how to compute a set of generators in $O(n^2)$ time and space. The time bound cannot be improved because all sets of generators could contain $\Omega(n)$ automorphisms and each generator has $\Omega(n)$ ordered pairs.

Since there is no way to circumvent this problem we can instead agree to a *compact representation* for the automorphisms. If we represent an automorphism ϕ by its set of nonfixed points then ϕ becomes the set

$$\{(x, \phi(x)) \mid x \neq \phi(x)\},$$

and we can achieve an $O(n)$ algorithm which produces a set of generators for the automorphism group of a labeled forest.

LEMMA 2.2. *The compact representation of a set of generators for a labeled forest can be computed in linear time.*

Proof. First compute the automorphism partition of the forest. From each similarity class of roots choose a single representative r . For each of the other roots r' which are similar to r output the compact representation of the automorphism which maps the tree rooted at r onto the tree rooted at r' , all other vertices within the forest remaining fixed. This can be accomplished by walking the two trees in parallel. Discard all of the trees rooted at the various r' keeping only the tree rooted at r .

After all of the similarity classes of roots have been processed delete the roots from the remaining forest and repeat this process level-by-level until the forest is empty. The time and space used is $O(n)$ because each vertex appears at most once in the range of an automorphism. A detailed proof that these mappings are actually automorphisms and that they generate the entire group is given in the first author's masters thesis [7]. \square

The final automorphism problem is handled in a like manner. Suppose that we want to determine the order of the automorphism group. Mathon [24] has shown that in general this problem is polynomially equivalent to testing graph isomorphism, but his reduction involves a factor of n^2 increase in running time. We will show that for forests (and hence trees) the order of the automorphism group can be computed in linear time.

Given a j -numbered forest the order of the automorphism group is found by a third walk of the forest, this time bottom up as in the original i -numbering. This pass generates k -numbers which indicate the order of the automorphism group for the subtree rooted at each vertex.

LEMMA 2.3. *Let G be a rooted j -numbered forest in which x_0 is a vertex and x_1, x_2, \dots, x_p are its children. If the p children are partitioned into q classes according to their j -numbers and c_t is the size of class t then the number of automorphisms for the subtree rooted at x_0 is given by*

$$k_0 = \left(\prod_{s=1}^p k_s \right) \cdot \left(\prod_{t=1}^q c_t! \right),$$

where k_s is the number of automorphisms for the subtree rooted at x_s , for $0 \leq s \leq p$.

Proof. Any automorphism of the subtree rooted at x_0 must carry each child of x_0 to a similar child of x_0 . For each of the q classes there are exactly $c_i!$ ways of permuting the c_i children in that class. In addition the subtree rooted at each x_s has k_s automorphisms. All of the choices are independent so the total number of possibilities is the product of these numbers. \square

An algorithm to compute the order of the automorphism group of a forest is now easy. Working from the bottom up each vertex receives its k -number by counting the number of children within each similarity class (using the j -numbers) and then multiplying together the appropriate factorials and the k -numbers of the children. If an imaginary root is added acting as a parent for all of the real roots in the forest then its k -number is precisely the order of the automorphism group for the forest.

The entire calculation is linear with the exception of the factorial computation. If factorial is not a primitive operation in the model of computation a table of factorials for the integers from zero to n can be precomputed and stored in a table of size $O(n)$. The cost of building the table is $O(n)$ if a uniform cost measure is used. Under the more realistic logarithmic cost criterion, in which the number of bits involved in each operation is counted, the cost becomes $O(n^2 \log n)$ since $n!$ requires $\Theta(n \log n)$ bits [1].

This last observation effectively kills any hope of a linear algorithm since a tree whose automorphism group has anywhere near $n!$ elements will require output whose representation is too large to print in linear time. Our point here is that the number of “data” operations need only be linear in the size of trees being considered; we ignore the fact that pointers and indices require $\Theta(\log n)$ bits for a graph on n vertices and that arithmetic operations on large numbers will require more than unit cost. Even for problems of practical size $n!$ is large and multiple precision arithmetic will have to be used, although if only an order of magnitude result is required we could use a floating-point calculation (or its equivalent, a fixed-point approximation of the logarithm). Asymptotically this may be an unrealistic assumption but we ignore this point and maintain the fiction that arbitrary integers can be held in a single register, adopt the uniform cost criterion, but keep in mind the warning made by a referee that “[this] assumption—if abused—leads to all sorts of unexpected consequences.”

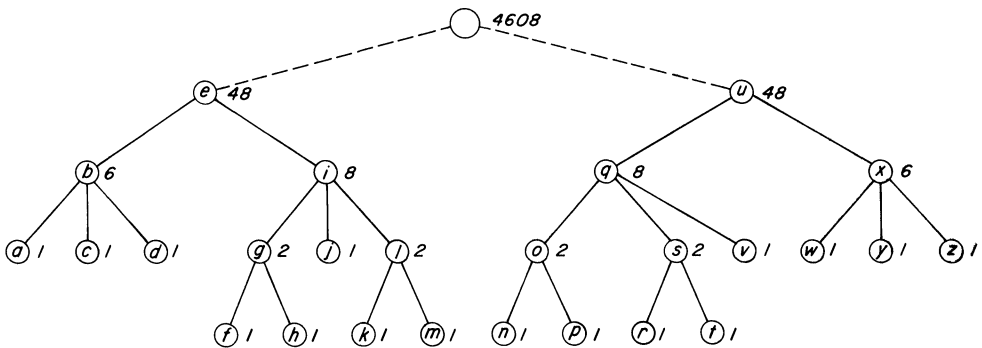


Fig 2.4. The k -numbers for the forest in Fig. 2.1. An imaginary root vertex has been added to the forest. Its k -number is the order of the automorphism group, in this example 4608.

Being able to compute the order of the automorphism group enables us to answer additional questions about isomorphisms. In particular the number of isomorphisms between two forests is easy to compute. Mathon has noted that any two graphs either are nonisomorphic or else their number of isomorphisms between them is the same as the number of automorphisms of one of the graphs [24]. Our algorithm is easily adapted to this purpose. We omit the details here.

We recall in passing that Edmonds's algorithm is already known to provide an efficient method for coding trees. The modification for forests is trivial. Starting with a sorted list of the i -numbers for the roots we recursively substitute the working label, in parentheses, for each vertex's i -number (also substituting original labels for ranks) until we have a single string consisting of the original labels suitably parenthesized to indicate the structure of the forest. This is a canonical representation. The code for each tree in our example forest is the string

$$(bac(ace())(cd(a(ab)(ab))(a(ab)(ab))(ca))),$$

and the code for the entire forest is the tree code repeated twice.

In summary we can state the following result about the automorphisms of a labeled forest. As a special case these same results apply to trees.

THEOREM 2.4. *It is possible to test isomorphism, to find the automorphism partition or a set of generators for the automorphism group, and to count the number of automorphisms or isomorphisms for labeled forests in time and space which is linear in the size of the forests plus the sum of the lengths of the labels.*

We will use these results in the following sections where we represent more general graphs by labeled forests. The labels will always be integers or strings of integers chosen from a range which is linear in the size of the original graphs so the results of this section will apply. We will thus be able to obtain linear automorphism algorithms for a class of graphs larger than just forests.

3. Interval graphs. An *interval graph* is a graph $G = (V, E)$ in which there is a one-to-one correspondence between the vertices V and some family of intervals on the real line. The correspondence must have the property that two vertices are adjacent in G if and only if their two corresponding intervals have a non-empty intersection. We will briefly review some facts about interval graphs here. Further background and additional references are given by Booth and Lueker [5], [23]. They have shown how to reduce the problem of testing interval graph isomorphism to the problem of testing isomorphism between special labeled trees. Here we explain how to extend those methods to obtain linear algorithms for computing the automorphism partition, a set of generators for the automorphism group, and the order of the automorphism group.

The basis of their isomorphism algorithm is a procedure for representing an interval graph by a *PQ-tree*. The leaves of the *PQ-tree* are the *dominant (maximal) cliques* of the interval graph. Internal nodes are either *P-nodes* which, like nodes in normal trees, have an unordered set of children, or else they are *Q-nodes* which have three or more children whose left-to-right order is rigid up to a complete reversal. Figure 3.1 provides an example of an interval graph and a *PQ-tree* which represents it.

Throughout the remainder of this section we will assume that G is an interval graph. To avoid confusion we will depart somewhat from our previous notation

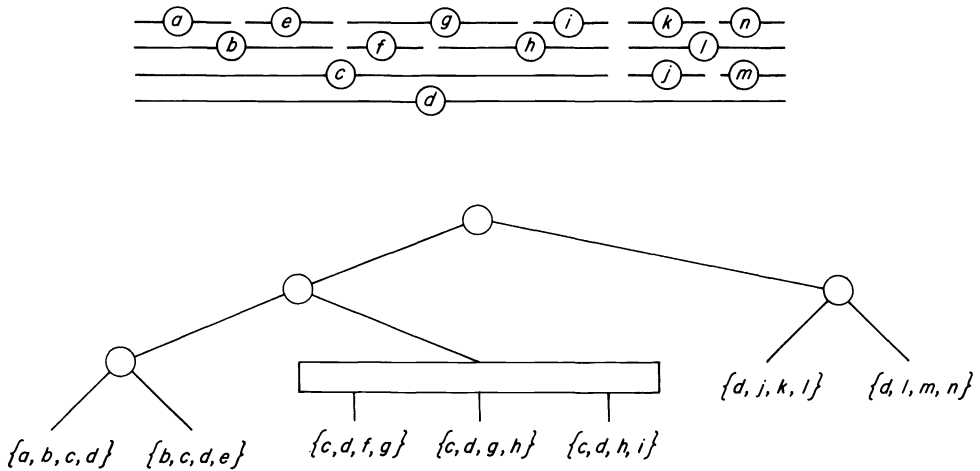


Fig. 3.1. A set of intervals (above) and the PQ -tree (below) which represents the associated interval graph. Each of the seven leaves in the tree is a dominant clique from the interval graph. P -nodes are drawn as circles and Q -nodes are drawn as rectangles.

and refer to the vertices of a PQ -tree as nodes so as to distinguish them from the vertices of G . We will further assume that all PQ -trees are drawn in the normal fashion with their roots at the top and their leaves at the bottom and that the leaves have a specific left-to-right order in this presentation. Every automorphism of a PQ -tree will thus correspond to an implicit re-drawing of the tree in which similar nodes occupy similar positions.

It has been shown [23] that the PQ -tree for an interval graph is unique up to isomorphism. An even stronger result can be proven which will enable us to solve the automorphism problems.

LEMMA 3.1. *Every automorphism of the PQ -tree for an interval graph G induces a distinct automorphism on G .*

Proof. Consider a particular embedding of the PQ -tree as discussed above. The leaves taken in left-to-right order form the *frontier* of the PQ -tree. Any automorphism of the PQ -tree will produce a different frontier and hence a different ordering of the dominant cliques.

Suppose that we have a particular frontier and that each clique is labeled with the sequence of degrees for its vertices, the degrees being specified in increasing order. We claim that the interval graph can be reconstructed from this information. The key observation is the fact, proven elsewhere [23], that the set of cliques to which each individual vertex belongs is always a consecutive set of leaves along the frontier. This is a basic property of the PQ -tree for an interval graph and forms the foundation for all of the PQ -tree algorithms.

Figure 3.2 contains a Pidgin Algol procedure which reconstructs an interval graph from the frontier of its PQ -tree. The reconstruction proceeds as follows. Starting with the leftmost clique a set of consecutively numbered vertices (beginning with one) is created for each degree of a vertex in the clique, the degrees being processing in increasing order. Edges are added between all created vertices to form a clique.

```

procedure RECONSTRUCT( { $C_i$ },  $k$ ):
  begin
     $n \leftarrow 0$ ;
     $E \leftarrow \emptyset$ ;
     $A \leftarrow \emptyset$ ;
    for  $i \leftarrow 1$  until  $k$  do
      begin
        for  $j \in A$  do
           $C_i \leftarrow C_i - \{degree(j)\}$ ;
        while  $C_i \neq \emptyset$  do
          begin
             $n \leftarrow n + 1$ ;
             $degree(n) \leftarrow \min(C_i)$ ;
             $C_i \leftarrow C_i - \{degree(n)\}$ ;
             $edges(n) \leftarrow 0$ ;
            for  $j \in A$  do
              begin
                 $E \leftarrow E \cup \{j, n\}$ ;
                 $edges(j) \leftarrow edges(j) + 1$ ;
                 $edges(n) \leftarrow edges(n) + 1$ 
              end;
             $A \leftarrow A \cup \{n\}$ ;
            for  $j \in A$  do
              if  $edges(j) = degree(j)$  then  $A \leftarrow A - \{j\}$ 
            end
          end
        end
      end
    end
  
```

Fig. 3.2. A Pidgin Algol procedure [1] which reconstructs an interval graph from its PQ-tree. The first input is the sequence of degree sets for the cliques along the frontier of the PQ-tree. The second input is the number of cliques. The procedure produces the set of edges E for the graph and the number of vertices n. The vertices which are missing edges at any step are "active" and are placed in the set A. They continue to have edges added as new vertices are created until finally their number of edges is equal to their degree, at which point they become "inactive" and are removed from A.

At a general step, having reached a new clique, the set of degrees is first modified by removing an instance of each degree for which there is a previously created vertex with the same original degree but which still has fewer edges than indicated by its degree. The requirement that vertices appear in consecutive cliques guarantees that there is no loss of generality in this step because all vertices which still have missing edges must be in this new clique. The process then proceeds as before, continuing the creation of consecutively numbered vertices in increasing order of degree and adding edges among all new vertices and also between all new vertices and all old vertices which are still missing edges.

An easy induction on the number of cliques verifies that the graph which is constructed is isomorphic to the original interval graph from which the PQ-tree was built. Every automorphism of the tree thus induces a permutation of the vertices which is an automorphism of the interval graph. Each of these interval graph automorphisms is distinct.

To be convinced of this, observe that every automorphism of the PQ-tree has

a distinct frontier. If we consider two distinct automorphisms of the PQ -tree, at least one clique must be in a different position within the frontier. No two dominant cliques have identical sets of vertices. Thus the clique which differs in the two PQ -tree automorphism must have a vertex which is treated differently by the reconstruction procedure. It then follows that the two interval graph automorphisms differ on that vertex, implying that they are in fact distinct automorphisms. \square

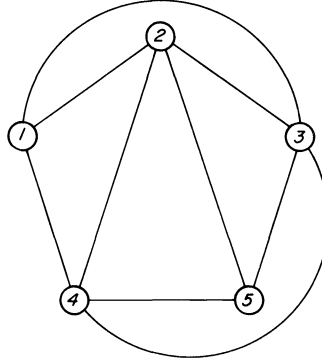


Fig. 3.3. The interval graph produced by the reconstruction procedure of Fig. 3.2 using the frontier of the PQ -tree from Fig. 3.1. The procedure has been run for two iterations ($i \leftarrow 1, 2$) of the outermost for-loop. At this point in the execution $n=5$, $A=\{3,4\}$, and the degrees for vertices 1 through 5 are respectively 3, 4, 8, 13, and 3. If the algorithm is run to completion it will construct a graph having fourteen vertices. It is easy to check that this is an interval graph for which the intervals in Fig. 3.1 form an intersection model which represents the graph [5].

Having discovered that every automorphism of the PQ -tree leads naturally to a different automorphism of G we might ask whether the converse is true. Does every automorphism of G induce a different automorphism of the PQ -tree? The answer is no. The reason can be traced to the way in which the interval graph was reconstructed from the tree. At various stages vertices were created to correspond to degrees of vertices within cliques. It can easily turn out that the same degree is used for more than one vertex being created for a particular clique. In the algorithm of Fig. 3.2 this follows from the fact that the selection of the minimum of C_i is nondeterministic since C_i is in general a multiset of degrees. The reconstruction algorithm does not distinguish these vertices. They are similar but distinct, so some automorphism should interchange them. It is these additional automorphisms which we must characterize.

We need a bit more information concerning the PQ -tree for this task. Every vertex in G has a *characteristic node* in the PQ -tree. This is the unique node (there always is one) which roots the subtree whose leaves are exactly the cliques to which the vertex belongs [5], [23]. Strictly speaking the term “characteristic node” is a bit imprecise because some vertices have characteristic nodes which are actually only part of a Q -node. The rigid left-to-right order of a Q -node’s children means that only some of them (always a consecutive subset) have leaves which contain the vertex in question. Nevertheless we will use the term “characteristic node” to mean the leaf, P -node, or portion of a Q -node which contains those cliques. Figure 3.4 illustrates the characteristic nodes for the interval graph shown earlier.

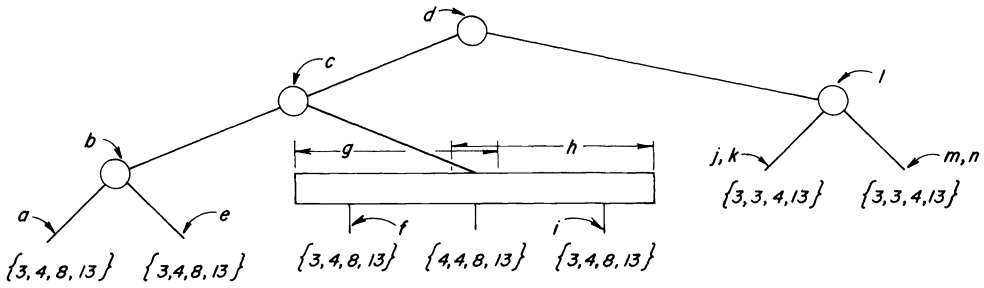


Fig. 3.4. The characteristic nodes for the interval graph in Fig. 3.1. Note that intervals g and h have characteristic nodes which are portions of the Q -node. The characteristic node for g is the two leftmost children of the Q -node whereas the characteristic node for h is the two rightmost children.

LEMMA 3.2. Every automorphism of G is completely determined by an automorphism of the PQ -tree for G together with a permutation of the vertices which preserves characteristic nodes.

Proof. Consider an automorphism of the interval graph G . It can be decomposed into two automorphisms, one on the PQ -tree and one which preserves characteristic nodes, in the following manner.

The automorphism induces an automorphism on the PQ -tree. This in turn induces an automorphism of G , which is not necessarily the same as the original automorphism because because the reconstruction has lost some information distinguishing vertices belonging to the same set of cliques. Vertices which are not distinguished by the reconstruction procedure are precisely those vertices which have identical characteristic nodes. They are always similar. Moreover, they may be arbitrarily permuted within the cliques with no change to the reconstruction. Thus the particular automorphism can be obtained from the automorphism induced by the reconstruction by a reordering of each set of vertices having the same characteristic node.

It follows immediately from these remarks that the additional shuffling of these vertices accounts for all possible automorphisms of G . \square

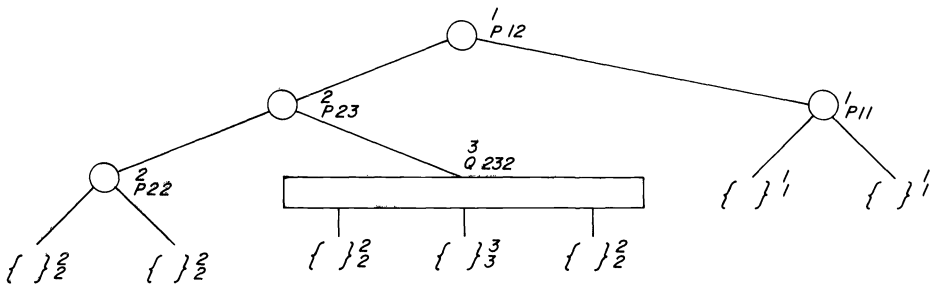


Fig. 3.5. The i -numbering for the PQ -tree of Fig. 3.1. Internal nodes have working labels which begin with a P or a Q to differentiate the two types of nodes. The remainder of a P -node's working label consists of the i -numbers, in increasing order, of the children. The i -numbers in the working label of a Q -node are in left-to-right order of the children and thus not always in increasing order.

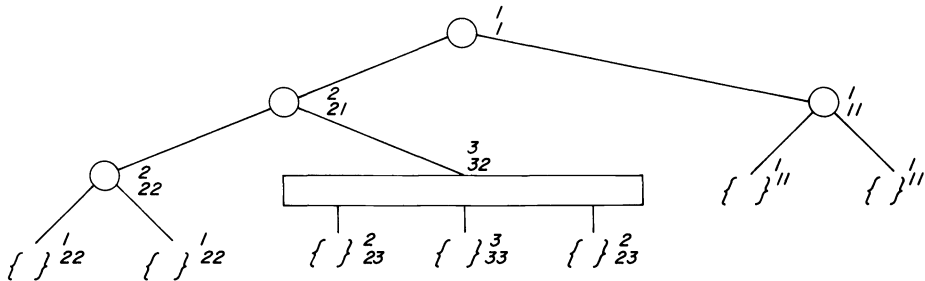
We are now ready to state the algorithm for computing the automorphism partition of an interval graph. We first label each leaf with the degrees of its vertices in increasing order. The i -numbering then proceeds as before followed by j -numbering. Two vertices are similar if and only if their characteristic nodes are similar in the PQ -tree. The j -numbers and depth information give this information.

LEMMA 3.3. *The automorphism partition of an interval graph can be found in linear time.*

Proof. Booth and Leuker prove that the PQ -tree of an interval graph can be constructed in linear time and that the characteristic nodes can be found for each vertex within the same time bound [5], [23]. The rest of the calculation is linear since it is a variation on the computation of the automorphism partition of a labeled tree.

Working labels for P -nodes consist of the letter P followed by the i -numbers of the children in increasing order. Working labels for Q -nodes consist of the letter Q followed by the i -numbers of the children in left-to-right or right-to-left order, whichever gives the lexicographically smaller label. The j -numbering is identical to the earlier calculation for trees.

Vertices of the interval graph are partitioned by assigning each one a label which consists of the similarity class for the characteristic node. If the characteristic node is a P -node this is trivial. If it is a Q -node the label must distinguish the portion of the Q -node (a consecutive set of children) comprising the characteristic node. This is accomplished by numbering the children from left-to-right (or right-to-left, depending upon how the working label was generated during the i -numbering) and appending the index of the left-most and right-most children to the label for the vertex. A final bucket sort accomplishes the partitioning into similarity classes. \square



The blocks of the automorphism partition of the interval graph

{a,e} {b} {c} {d} {f,i} {g,h} {j,k,m,n} {l}

Fig. 3.6. The j -numbering for the PQ -tree of Fig. 3.1. The eight blocks of the automorphism partition for the original interval graph are shown below the PQ -tree.

Finding a set of generators for the automorphism group is an easy generalization of the automorphism algorithm for a forest. There are two minor variations to the basic forest algorithm which must be made. The first is that a set of generators must be added which permute each family of vertices having the

same characteristic node. A bucket sort produces the families and the compact representation for a set of generators is then easily produced. This can be done before the i -numbering so we will not worry about this contribution to the complexity since it is obviously linear.

The second variation is that as the generators are produced there can arise a situation which does not occur in the normal algorithm. Some Q -nodes can have their children reversed left-to-right and still have the resulting subtree isomorphic to the original subtree. When this happens it is necessary to output the compact representation of an automorphism which performs this reversal. This is easy to do by walking the tree but we need to verify that the algorithm is still linear. The earlier argument that each vertex is in the range of an automorphism at most once no longer holds. But careful analysis shows that each time a Q -node is reversible we are guaranteed that one half of its children will disappear at the next level because they must be similar to the other half of the children. If there are an odd number of children the middle child does not have to participate in the reversal because it maps to itself. This is enough to ensure linearity because each vertex is now in the range at most twice.

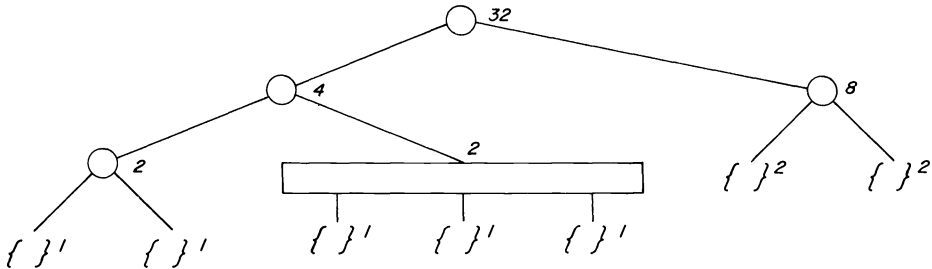


Fig. 3.7. The k -numbering of the PQ -tree of Figure 3.1. The order of the automorphism group of the interval graph is the k -number of the root. In this example there are 32 distinct automorphisms.

Counting the number of automorphisms for an interval graph is equally straightforward. A k -numbering of the PQ -tree determines the order of its automorphism group and the remaining automorphisms can be counted by multiplying by the total number of permutations which preserve characteristic nodes. This is simply the product of the factorials of all families of vertices having a common characteristic node and is again a linear time computation, modulo the entire discussion of § 2 regarding the computation of factorial. In practice it is simpler to include this in the k -numbering. The k -number of each node is multiplied by the factorial of the number of vertices for which it is a characteristic node. Q -nodes will have a slightly more complicated k -number because they can correspond to more than a single characteristic node in which case the product of the factorials is used. This modified k -numbering is illustrated in Fig. 3.7.

These PQ -tree algorithms can be extended to handle labeled interval graphs in an obvious manner. Instead of each clique receiving a label which consists only of degrees we first bucket sort all of the vertices' labels and assign ranks which are then added to the degree information labeling the cliques. The remainder of the algorithm is the same except that two vertices are similar if and only if they have the same label and similar characteristic nodes. We have proven the following.

THEOREM 3.4. *It is possible to test isomorphism, to find the automorphism partition or a set of generators for the automorphism group, and to count the number of automorphisms or isomorphisms for labeled interval graphs in time and space which is linear in the size of the interval graphs plus the sum of the lengths of the labels.*

4. Outerplanar graphs. An *outerplanar graph* is a graph having a planar embedding in which all of the vertices lie on the exterior face [11]. These graphs are obviously planar so we know (jumping ahead to § 5) that they have linear time automorphism algorithms. But algorithms to handle the general case are fairly complicated and any implementation is likely to be quite intricate. However, outerplanar graphs are a greatly restricted subset of the planar graphs and we can take advantage of their additional structure to produce algorithms which are simpler than the corresponding algorithms for labeled planar graphs.

Our outerplanar algorithms have the same linear asymptotic running time as do the planar algorithms but the constants of proportionality are much lower compared with the more general algorithms. In this respect we offer quite practical solutions to the isomorphism and automorphism problems for outerplanar graphs. These new algorithms, like much of the previous work on outerplanar graphs, are based on a result of Tang [32]. The following lemma is also given by Harary [11, Problem 11.26].

LEMMA 4.1. *If G is a graph in which there are at most two vertex-disjoint paths of length greater than one between each pair of vertices then*

- (1) G is planar,
- (2) $e \leq 2n - 2$, and
- (3) if G is biconnected and $n \geq 5$ then G has a unique Hamilton cycle.

It is easy to see that outerplanar graphs satisfy the hypotheses of the lemma. Any two vertex-disjoint paths must lie on different portions of the exterior face and thus there are at most two such paths joining any pair of vertices. We can state the following result.

COROLLARY 4.2. *Every biconnected outerplanar graph having at least three vertices possesses a unique Hamilton cycle.*

Proof. For five or more vertices Tang's lemma guarantees that there is a unique Hamilton cycle. The remaining cases are verified exhaustively. The only biconnected outerplanar graphs having at least three but no more than four vertices are the graphs K_3 , $K_{2,2}$, and $K_4 - x$. Each of these has a unique Hamilton cycle as shown in Fig. 4.1. \square

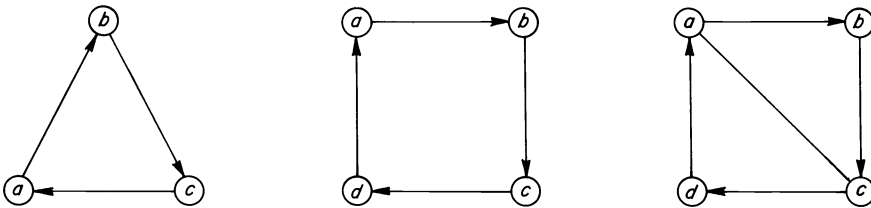


Fig. 4.1. The three biconnected outerplanar graphs on three or four vertices with their unique Hamilton cycles. Edges within the cycles are indicated by the directed arrows.

For an arbitrary outerplanar graph standard depth-first search algorithms will produce the biconnected components in linear time [1], [33]. Using the unique Hamilton cycles we will show that each of these biconnected components can be placed into a canonical form from which information about the automorphisms can be easily extracted. These in turn can be used to build a canonical tree representation for each of the connected components so that the resulting forest is a canonical representation for the entire graph. From this forest, automorphism information can be easily obtained using algorithms similar to those developed in §§ 2 and 3. We thus examine the problem of placing the biconnected components into canonical form, putting off until later the task of piecing this information back together again for the whole graph.

Given a biconnected outerplanar graph the unique Hamilton cycle can be found in linear time by successively removing vertices of degree two. Beyer, Jones, and Mitchell use this approach to test isomorphism of maximal outerplanar graphs [2]. They rely upon the fact that the sequence of vertex degrees encountered along the Hamilton cycle, the *Hamilton degree sequence*, completely characterizes the graph. Unfortunately this characterization does not extend to the more general case of biconnected outerplanar graphs. Figure 4.2 illustrates this situation, showing two nonisomorphic biconnected outerplanar graphs which have the same Hamilton degree sequence.

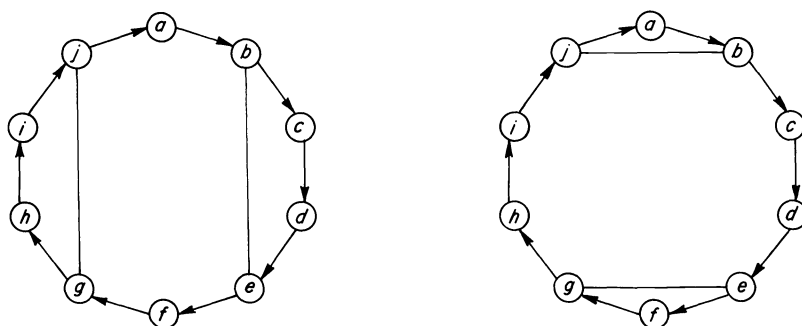


Fig. 4.2. Two non-isomorphic biconnected outerplanar graphs having the same Hamilton degree sequence 2322323223.

Syslo [31] gave a similar example in his description of a linear time coding algorithm (and hence a linear time isomorphism test) for biconnected outerplanar graphs. He again used the unique Hamilton cycle, but he proposed a more elaborate scheme which uses additional information concerning the structure of the graph. After identifying the unique Hamilton cycle and numbering the vertices accordingly his algorithm performs a series of reduction and labeling steps to finally obtain a string representation which is then minimized and used as a canonical representation for the graph.

We think that our approach is somewhat simpler and leads more directly to efficient automorphism algorithms. We observe that although the Hamilton degree sequence is not sufficient to completely determine a biconnected outerplanar graph we certainly can reconstruct the graph if we have a list of the

adjacencies for each vertex along the cycle. The problem of course is to represent the adjacencies in a manner which is independent of the exact starting position chosen for the cycle since otherwise there might be as many as n different representations.

To this end we represent the adjacencies at each vertex by a list which consists of the distances along the Hamilton cycle between the vertex and each of its neighbors. We further stipulate that these distances be in increasing order. Notice that the lists depend upon the orientation chosen for the Hamilton cycle but not upon its starting point. Figure 4.3 shows the adjacency lists for the two graphs of Fig. 4.2. Each vertex is assigned two lists, the first for a clockwise traversal of the Hamilton cycle and the second for a counter-clockwise traversal.

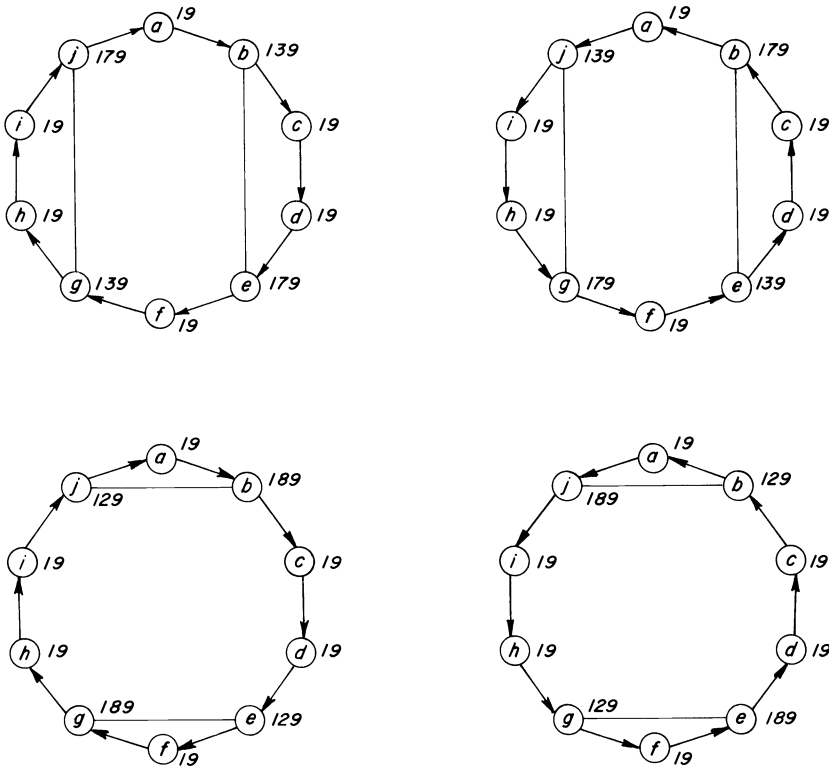


Fig. 4.3. Clockwise (on the left) and counter-clockwise (on the right) Hamilton adjacency lists for the two graphs of Fig. 4.2. The fact that the two sequences are different implies that the top graph is not isomorphic with the bottom graph.

In general we will let S be the sequence obtained by following the Hamilton cycle in one direction, concatenating adjacency lists, and we will let R be the sequence obtained by concatenating in the opposite direction, using the reversed adjacency lists. These are the *Hamilton adjacency sequences* for the graph. Isomorphism of biconnected outerplanar graphs can be characterized directly in terms of these sequences.

LEMMA 4.3. *Let G_1 and G_2 be biconnected outerplanar graphs and suppose*

that S_1 and S_2 are the Hamilton adjacency sequences for their unique Hamilton cycles and that R_1 and R_2 are the Hamilton adjacency sequences for the reversed Hamilton cycles. Then G_1 is isomorphic to G_2 if and only if S_1 is either a cyclic shift of S_2 or a cyclic shift of R_2 .

Proof. Clearly two isomorphic graphs must have Hamilton adjacency sequences which are either cyclic shifts or reversals of each other. But as we have already remarked, given the Hamilton adjacency sequence of a graph, the graph can be reconstructed up to isomorphism. \square

The unique Hamilton cycle in a biconnected outerplanar graph can be found in linear time by successively removing vertices of degree two, being careful to keep track of vertices which must be adjacent in the cycle [2], [27], [31]. The sequences $S_1, S_2, R_1,$ and R_2 can be constructed at the same time by numbering the vertices along the cycle and bucket sorting the distances for each adjacency, building the lists for each of the vertices by adding the smallest distances first.

Restating the last lemma as a pattern matching problem we see that two graphs are isomorphic if and only if S_1 is a substring of $S_2S_2R_2R_2$. This condition can be tested using any one of many algorithms for linear time pattern matching [1, Chapter 9]. Noting that all of the objects are linear in the size of the graphs (the strings S and R each have length $2e$) and that the graphs themselves are linear in n (by Lemma 4.1) we have proven the following result.

COROLLARY 4.4. *Two biconnected outerplanar graphs can be tested for isomorphism in $O(n)$ steps.*

Syslo's coding algorithm achieves the same asymptotic running time but we claim that our algorithm is easier to implement and that it will have a smaller multiplicative constant. But what is more important, our algorithm is easily extended to solve the automorphism problems discussed in earlier sections. All that is required is a little more information from the pattern matching algorithms. Let p_i be the position within S in which the adjacency list for vertex i begins and let q_i be the position in R in which the reversed adjacency list for vertex i begins.

LEMMA 4.5. *Let G be a biconnected outerplanar graph, let S be its Hamilton adjacency sequence, and let R be its reversed Hamilton adjacency sequence. Two vertices i and j are similar if and only if the substring of length $2e$ which begins in position p_i of SS is identical with the substring of length $2e$ which begins either in position p_j of SS or in position q_j of RR .*

One way to compute the similarity classes is to find the position tree for the string $SSRR$ using Weiner's algorithm [1], [37]. Two vertices are in the same similarity class whenever they have a common ancestor at depth $2e$ in the position tree. The standard algorithm for finding position trees requires time proportional both to the length of the input string and to the alphabet size. Since our strings have symbols ranging from 1 to n this would imply an $\Omega(n^2)$ running time.

But this is more machinery than we need bring to bear on this problem. Any automorphism must map a Hamilton cycle onto a Hamilton cycle. Thus for biconnected outerplanar graphs the only possible automorphisms are cyclic shifts and reversals of the Hamilton cycle. It is sufficient to compute just the similarity class of vertex 1; the class for vertex j is found by adding $j-1$ (modulo n) to all of the vertices in the first class if the automorphism corresponds to shifting the Hamilton cycle; if the automorphism corresponds to reversing the Hamilton cycle the vertices in each class must be "reversed" in the obvious manner.

It follows that the automorphism group is always a subgroup of a dihedral group, the $2n$ permutations which cyclically shift and/or reverse a cycle of length n . The number of automorphisms for a biconnected outerplanar graph is the

number of distinct cyclic shifts and reversals which map the Hamilton cycle onto itself. We can find the order of the group by counting the number of times that S occurs as a substring in $SS'SRR'$ where S' and R' are the respective strings S and R with their final symbols removed to avoid counting the identity and reversal automorphisms twice.

Guy [10] gave similar counting results for automorphisms of maximal outerplanar graphs, showing that there were at most six automorphisms. This is a very special case because the more general biconnected outerplanar graphs can realize the full dihedral group, as evidenced by the n -cycle which is biconnected, outerplanar, and has $2n$ automorphisms.

For the automorphism groups of biconnected outerplanar graphs a set of generators consists of a single cyclic shift of the minimum distance and (if appropriate) a reversal of the cycle. Again, these two automorphisms are easily found using linear time pattern matching.

LEMMA 4.6. *There exist linear time algorithms to compute the automorphism partition, to determine the order of the automorphism group, and to find a set of generators for the automorphism group of a biconnected outerplanar graph.*

We turn our attention next to the problem originally solved by Syslo, that of producing a coding for biconnected outerplanar graphs in linear time. The Hamilton adjacency sequence is almost a canonical form for a biconnected outerplanar graph. If we encode the graph by finding the starting vertex and direction for the Hamilton cycle which produces the lexicographically smallest adjacency sequence we will produce a coding. It is easy to modify the Knuth-Morris-Pratt pattern matching algorithm to have it find the lexicographically smallest cyclic shift of a string in linear time and space; there are other algorithms, the best known to us is due to Shiloach [4], [28], [29]. This yields a linear time coding algorithm for biconnected outerplanar graphs.

We conclude this section with a sketch of the automorphism algorithms for general outerplanar graphs. The biconnected components can be used to build a tree (a variant of the *block cut-vertex tree*) for each connected component. These trees form a forest representing the entire graph.

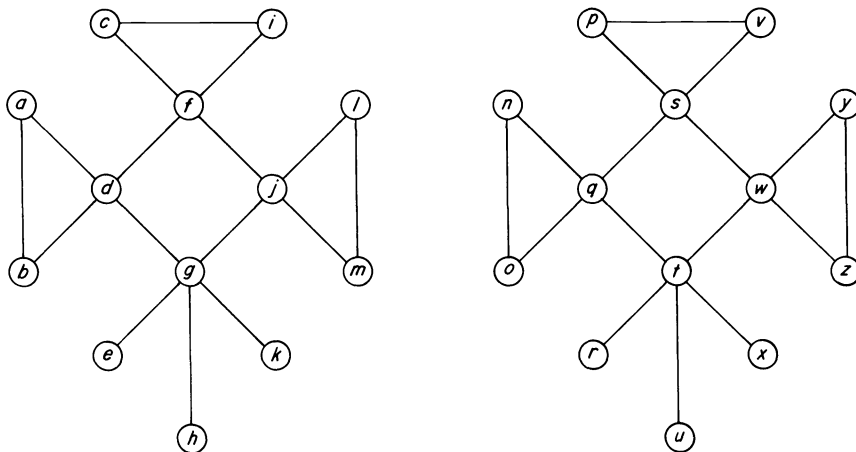


Fig. 4.4. A general outerplanar graph having two connected components and fourteen biconnected components.

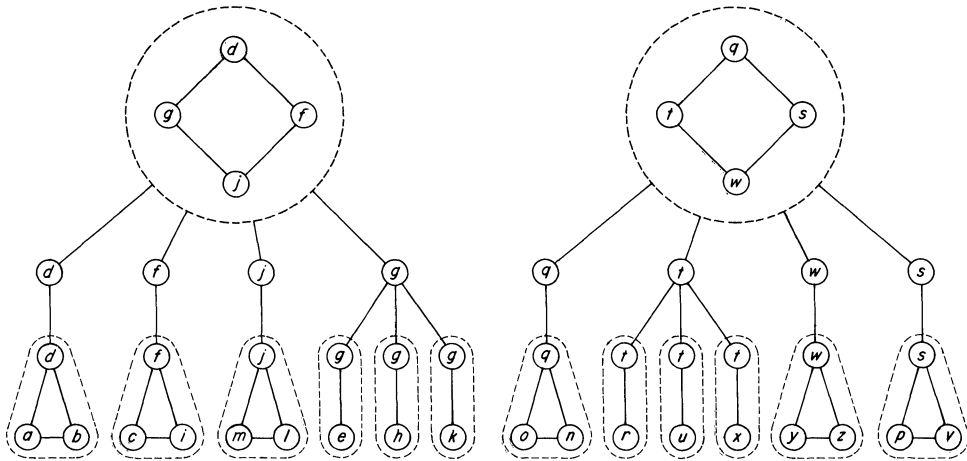


Fig. 4.5. The block cut-vertex forest for the outerplanar graph of Fig. 4.4. Each node within the forest is either a block of the graph or a cut vertex of the graph. Blocks (biconnected components) are indicated by dotted lines.

Biconnected components are labeled with their Hamilton adjacency sequence which has been shifted so it begins with the unique cut vertex which is the parent in the tree. The Hamilton adjacency sequence is reversed if necessary to give the smallest possible lexicographic value. The exception to this rule is a biconnected component which has no parent in its tree (there is at most one in each tree). These are represented by the shift or reversal of the sequence which gives a lexicographically least value after the i -numbers for the children have been inserted into the Hamilton adjacency sequence.

An isomorphism test is a straightforward modification of the earlier labeled tree isomorphism algorithm. At each level in the forest, beginning at the bottom, the labels are bucket sorted and assigned ranks. A check is made that the multiset of labels is identical for the two forests. The i -numbers are then passed up the

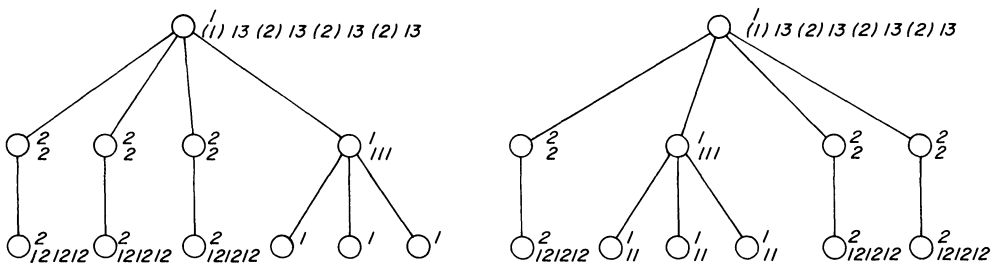


Fig. 4.6. The i -numbering for the block cut-vertex forest of Fig. 4.5. The code for this forest is a canonical representation of the original graph. The working label of a block is its Hamilton adjacency sequence. For non-root blocks, the sequence begins with the unique cut vertex which is the parent of the block; a root block is labeled with its lexicographically least sequence. In nonleaf blocks the adjacency list of a cut vertex is preceded by the i -number of the cut vertex, contained within parentheses. Working labels for cut vertices consist of the i -numbers of the children in increasing order.

tree, inserted appropriately into the parents' labels (being careful to distinguish ranks from distances) and the next level is bucket sorted. This continues until the roots are the only unprocessed nodes. At the top level the labels with rank information inserted are each placed into a canonical form (minimum lexicographic order) and then bucket sorted. Two outerplanar graphs are isomorphic if and only if their forests generate the same multiset of labels for the roots.

Algorithms for computing the automorphism partition, counting the number of automorphisms, and producing a set of generators for the automorphism group are easy to build using the forests and the automorphism algorithms from § 2. Labeled outerplanar graphs are handled by incorporating the ranks of the bucket sorted labels into the Hamilton adjacency sequences in an obvious manner. The modifications are quite similar to those used for interval graphs. We can state the following result.

THEOREM 4.7. *It is possible to test isomorphism, to find the automorphism partition or a set of generators for the automorphism group, and to count the number of automorphisms or isomorphisms for labeled outerplanar graphs in time and space which is linear in the size of the outerplanar graphs plus the sum of the lengths of the labels.*

5. Planar graphs. In this section we will briefly sketch the techniques which might be used to build linear time automorphism algorithms for general planar graphs. The algorithms we propose are admittedly quite complicated. We have not implemented them; to do so will require that many details be worked out which we have not fully thought out. Nevertheless, we maintain that the basic ideas sketched here form a convincing argument that the automorphism problems for the general case of planar graphs can also be solved in linear time.

Our methods build upon those developed by Hopcroft, Tarjan, and Wong [16], [20] which were used to produce linear time algorithms for testing isomorphism of planar graphs. Fontet [9] has also used their results as the basis of a linear algorithm for finding the automorphism partition of a planar graph. His method is quite similar to ours, but he bases his construction on the Hopcroft-Tarjan algorithm whereas we favor the Hopcroft-Wong solution. We indicate algorithms for solving all of the automorphism problems.

To review, here is a short survey of the development of planar graph isomorphism algorithms. In 1970 Hopcroft [13] demonstrated that the states of a finite automaton could be minimized in $O(n \log n)$ time. The application of this technique to planar graph isomorphism was pursued in a series of papers by Hopcroft and Tarjan [14]-[17]. They reduced the isomorphism problem for planar graphs to the problem for triconnected planar graphs by using a tree representation in which each leaf corresponds to a maximal triconnected subgraph. The connected, biconnected, and triconnected components are all found in linear time [18], [33]. The problem was further reduced to isomorphism of triconnected planar embeddings using Whitney's theorem that a triconnected planar graph has a unique embedding [19], [38].

Hopcroft and Tarjan were only able to show that planar graphs can be tested for isomorphism in $O(n \log n)$ time [17] but Hopcroft and Wong were able to obtain linear algorithms using the planar embeddings [20], [39]. The Hopcroft-Tarjan algorithm produces the automorphism partition of the triconnected

embedding as part of the isomorphism test. Fontet [9] later improved the time bound for Hopcroft and Tarjan's method to $O(n)$ by establishing some interesting properties of planar embeddings. He was the first to show that the automorphism partition of a planar graph can be found in linear time.

We take a different approach than Fontet and base our automorphism algorithms on the Hopcroft-Wong algorithm. Necessarily we will only be able to sketch our procedure. A more complete description and a proof of correctness would require a more thorough analysis of the Hopcroft-Wong algorithm than has yet appeared in the literature. Our algorithms will use the linear time forest automorphism algorithms along with linear pattern matching.

The overall algorithm follows Hopcroft and Wong. The connected, biconnected, and triconnected components are found and used to construct a forest, similar to the forest used for outerplanar graphs, which is then i -numbered, j -numbered, and k -numbered. The difficult part is the way in which the triconnected planar embeddings are handled.

Hopcroft and Wong's algorithm successively reduces the planar embeddings to smaller and smaller embeddings by applying two operations which remove loops or multiple edges and vertices of low degree. The information from these reductions is kept so that the graph can be reconstructed. A second critical step is the application of a *circle isomorphism* procedure which determines whether two labeled cycles are isomorphic. The overall algorithm eventually tests the entire graph by winding back through the reductions, propagating information obtained at lower levels in the algorithm.

Wong was able to show that every triconnected planar embedding must either have an applicable reduction or else it must be one of three simple graphs: a labeled vertex, a labeled cycle, or a Platonic solid [39]. This guarantees that the test is linear since all of these cases, with the exception of the labeled cycles, involve only graphs of bounded size.

We propose using linear pattern matching to place all of the circles into a canonical form by finding their lexicographically smallest shift and using that shift to represent the cycle. Isomorphism and automorphism questions for circles are then easily answered as discussed in § 4. The result is a canonical i -numbering of the tree which represents the planar graph. After j -numbering and k -numbering we can easily find all of the other automorphism information for the graph. We state without proof our main result concerning planar graphs.

PROPOSITION 5.1. *It is possible to test isomorphism, to find the automorphism partition or a set of generators for the automorphism group, and to count the number of automorphisms or isomorphisms for labeled planar graphs in time and space which is linear in the size of the planar graphs plus the sum of the lengths of the labels.*

This section has been only a sketch of the methods we propose. The details are similar to those given in previous sections but are significantly more complicated due to the extra complexity of finding triconnected components and other complexities inherent in the Hopcroft-Wong algorithm. We claim that our version of circle isomorphism may be an improvement over the original version and that our algorithms in principle can be used to solve all of the automorphism problems, in contrast with Fontet's solution which finds only the automorphism partition. A verification of these claims awaits further work to clarify the many details omitted here.

6. Concluding remarks. We have shown that four classes of graphs having linear time isomorphism tests have linear time algorithms to compute the automorphism partition, to count the number of automorphisms, to produce a set of generators for the automorphism group, and also to determine codings for the graphs. The algorithms for outerplanar graphs, particularly for the biconnected case, are very easy to implement and should be much faster than the algorithms for the general planar case.

We conclude with a few remarks suggesting possible areas for future investigation. Polya [26] showed that the automorphism group of a tree can always be expressed as the direct or wreath products of symmetric groups. Weinberg, Harary, and Tutte [12], [36] showed that triconnected planar graphs have automorphism groups which are linear in size. Biconnected outerplanar graphs have automorphism groups which are always subgroups of a dihedral group and maximal outerplanar graphs have at most six automorphisms [10].

These examples suggest that there is a relationship between the “complexity” of the automorphism group and the “complexity” of testing for isomorphism. In particular we conjecture that for classes of graphs in which the isomorphism problem is equivalent to the general graph isomorphism problem it will turn out that in some vague sense almost every group will be the automorphism group of some graph within the class. A precise statement of this proposition awaits further research.

7. Acknowledgments. We would like to express our appreciation to Marlene Colbourn, Ian Munro, Brian Plante, and Ron Read for assistance with various aspects of this research. Leo Guibas first told us of the work of Shiloach. Brian Finch supervised the typesetting.

REFERENCES

- [1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [2] T. Beyer, W. Jones, and S. Mitchell, *A linear time algorithm for isomorphisms of maximal outerplanar graphs*, J. Assoc. Comput. Mach., 26(1979), pp. 603-610.
- [3] J. A. Bondy and J. S. R. Murty, *Graph Theory With Applications*, MacMillan, London, 1976.
- [4] K. S. Booth. *Finding a lexicographic least shift of a string*, Information Processing Letters, 10(1980), pp. 240-242.
- [5] K. S. Booth and G. S. Leuker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13(1976), pp. 335-379.
- [6] G. Busacker and T. L. Saaty, *Finite Graphs and Networks*, McGraw Hill, New York, 1965.
- [7] C. J. Colbourn, *Graph Generation*, M.Math. Thesis, University of Waterloo, Ontario, 1977.
- [8] D. G. Corneil, *Graph Isomorphism*, Ph.D. Thesis, University of Toronto, Ontario, 1968.
- [9] M. Fontet, *Linear algorithms for testing isomorphism of planar graphs*, in *Proceedings Third Colloquium on Automata, Languages, and Programming*, 1976, pp. 411-423.
- [10] R. Guy, *Dissecting a polygon into triangles*, Bull. Malayan Math. Soc., 5(1958), pp. 56-60.
- [11] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
- [12] F. Harary and W. T. Tutte, *On the order of the group of a planar map*, J. Combin. Theory, 1(1966), pp. 394-395.
- [13] J. E. Hopcroft, *An $n \log n$ algorithm for minimizing states in a finite automaton*, in *Theory of Machines and Computations*, Z. Kohavland, and A. Paz, eds., Academic Press, New York, 1971.

- [14] ———, *An $n \log n$ algorithm for isomorphism of planar triply connected graphs*, Computer Science Technical Report STAN-CS-71-192, Stanford University, Stanford, CA, 1971.
- [15] J. E. Hopcroft and R. E. Tarjan, *A V^2 algorithm for determining isomorphism of planar graphs*, Information Processing Letters, 1(1971), pp. 32-34.
- [16] ———, *Isomorphism of planar graphs*, in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, pp. 131-151, 1972.
- [17] ———, *A $V \log V$ algorithm for isomorphism of triconnected planar graphs*, J. Comp. System Sci., 7(1971), pp. 323-331.
- [18] ———, *Dividing a graph into triconnected components*, this Journal, 2(1973), pp. 135-158.
- [19] ———, *Efficient planarity testing*, J. Assoc. Comput. Mach., 21(1974), pp. 549-568.
- [20] J. E. Hopcroft and J. K. Wong, *Linear time algorithm for isomorphism of planar graphs*, in *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, 1974, pp. 172-184.
- [21] K. R. James and W. Riha, *Algorithm for description and ordering of trees*, Technical Report 54, University of Leeds, Leeds, England, 1974.
- [22] J. Lederberg, *Notational algorithms for tree structures*, NASA Technical Report CR57029, 1964.
- [23] G. S. Leuker and K. S. Booth, *A linear time algorithm for deciding interval graph isomorphism*, J. Assoc. Comput. Mach., 26(1979), pp. 183-195.
- [24] R. A. Mathon, *A note on the graph isomorphism counting problem*, Information Processing Letters 8(1979), pp. 131-132.
- [25] S. Mitchell, *Algorithms on Trees and Maximal Outerplanar Graphs*, Ph.D. Thesis, University of Virginia, 1977.
- [26] G. Polya, *Kombinatorische Anzahlbestimmungen für Gruppen, Graphen, und Chemische Verbindungen*, Acta Mathematica, 68(1937), pp. 154-245.
- [27] R. C. Read, and D. G. Corneil, *The graph isomorphism disease*, J. Graph Theory, 1(1977), pp. 339-363.
- [28] Y. Shiloach, *A fast equivalence-checking algorithm for circular lists*, Information Processing Letters, 8(1979), pp. 236-238.
- [29] ———, *Fast canonization of circular strings*, submitted for publication.
- [30] H. I. Scoins, *Placing trees in lexicographic order*, in Machine Intelligence, D. Michie, ed., 3(1968), pp. 43-60.
- [31] M. M. Syslo, *Linear time algorithm for coding outerplanar graphs*, in *Beiträge zur Graphentheorie und deren Anwendungen*, Oberhof, GDR, 1977, Ilmenau, East Germany, 1978, pp. 259-269.
- [32] D. T. Tang, *Bi-path networks and multicommodity flows*, IEEE Transactions on Circuit Theory, 11(1964), pp. 468-474.
- [33] R. E. Tarjan, *Depth-first search and linear graph algorithms*, this Journal, 1(1972), pp. 146-160.
- [34] L. Weinberg, *Algorithms for determining automorphism groups of planar graphs*, in *Proceedings of the Third Annual Allerton Conference on Circuit and System Theory*, 1965, pp. 913-929.
- [35] ———, *Alternative algorithms for determining isomorphisms and automorphism groups of planar graphs*, in *Proceedings of the Fourth Annual Allerton Conference on Circuit and System Theory*, 1966, pp. 444-453.
- [36] ———, *On the maximum order of the automorphism group of a planar triply-connected graph*, SIAM J. Appl. Math., 14(1966), pp. 729-738.
- [37] P. Weiner, *Linear pattern matching algorithms*, in *Proceedings of the Fourteenth Annual Symposium on Switching and Automata Theory*, 1973, pp. 1-11.
- [38] H. Whitney, *A set of topological invariants for graphs*, Amer. J. Math., 55(1933), pp. 231-235.
- [39] J. K. Wong, *Isomorphism Problems Involving Planar Graphs*, Ph.D. Thesis, Cornell University, 1975.

THE TOTAL CORRECTNESS OF PARALLEL PROGRAMS*

LAWRENCE FLON[†] AND NORIHISA SUZUKI[‡]

Abstract. We describe a formal theory of the total correctness of parallel programs, including such heretofore theoretically incomplete properties as safety from deadlock and starvation under fair-scheduling. We present a sound and complete set of proof rules for the total correctness of parallel programs expressed in nondeterministic form.

The proof of soundness and completeness is novel in that we show that the weakest pre-conditions for the correctness criteria are actually fixed-points (least or greatest) of continuous functions over the complete lattice of total predicates. We have obtained proof rule schemata which can universally be applied to least or greatest fixed-points of continuous functions. Therefore, a system of proof rules is a priori sound and complete once it is shown that certain weakest pre-conditions are extremum fixed-points. The relationship between true parallelism and nondeterminism is also discussed.

Key words. parallel programs, nondeterministic programs, verification, correctness, semantics, fixed-points, completeness of axiomatic proof rules, deadlock, starvation

1. Introduction. Parallel programming is becoming more common in computer applications and systems implementation as a result of the rapidly growing development of asynchronous multiprocessor computers. These computers are finding their way into more and more applications wherein a very high degree of reliability is essential. As software is one target of this reliability, and in view of the increasingly common presence of parallel constructs in programming languages, it is important to study the formal semantics of parallel computation and develop methods for reasoning about parallel program correctness.

Several earlier attempts, both formal and informal, have been made in this area. Many rely on specific synchronization primitives, for example, semaphores [12], conditional critical regions [3], monitors [16], [17] and path expressions [10]. All deal primarily with weak correctness properties. Recently more formal results have appeared, notably [18], [19], [22]. The last approach is the one most widely known, and it includes a complete axiomatization of weak correctness.

Dijkstra [7] introduced predicate transformers to deal with both weak and strong correctness of nondeterministic sequential programs. Later it was realized that there are other notions of strong correctness for parallel programs distinct from termination, and that predicate transformers are useful for describing them formally. Van Lamswerde and Sintzoff [20] and Flon and Suzuki [9], [11] used predicate transformers to describe various aspects of the total correctness of parallel programs, such as blocking and deadlock. No one has previously produced an axiomatization of total correctness.

In the following we present a sound and complete axiomatization of total correctness. The axioms are based upon the notions defined in [11]. For the first time we can deal formally with the important strong correctness issues of blocking, deadlock, and starvation.

Section 2 indicates that parallel programs with conditional critical regions as the synchronization mechanism can be simply transformed into nondeterministic

* Received by the editors November 14, 1978, and in final form March 17, 1980. A version of this paper was presented at the Nineteenth Annual IEEE Conference on the Foundations of Computer Science, Ann Arbor, Michigan, October 1978.

[†] Department of Computer Science, University of Southern California, Los Angeles, California 90007.

[‡] Xerox Palo Alto Research Center, Palo Alto, California 94304.

programs. In § 3 we introduce a formal model of computation and define correctness criteria, such as deadlock and starvation, in terms of this model.

In § 4 an axiomatization for proving these correctness criteria is introduced, and in § 5 we review the analysis of the predicate transformers $wp(\mathbf{while} B \mathbf{do} S, R)$ and $wlp(\mathbf{while} B \mathbf{do} S, R)$ as the least and greatest fixed-points of the same continuous function over the complete lattice of total predicates. We suspect here that the weakest pre-conditions associated with the correctness criteria for our parallel programs may also be fixed-points. Therefore, we invent sound and complete proof rule schemas for the least and greatest fixed-points of certain recursive functions. The close relationship between weak correctness proof rules and fixed-points has also been observed by Park [23], de Bakker and Scott [1], Hitchcock and Park [15] and Clarke [5]. Some properties of monotonic functions and continuous functions are also reviewed. A result of § 5 is the discovery of a new least fixed-point proof rule which is valid for noncontinuous properties (Harel and Pratt [14] gave a rule which requires continuity).

In § 6 we show that the weakest pre-conditions of our correctness properties actually are least or greatest fixed-points of particular recursive functions. Thus, soundness and completeness of the axiomatization of § 4 follows as a direct consequence of the results of § 5. Section 7 contains a formal treatment of fairness in a system of parallel programs. As it turns out, inevitability under our definition of fairness is monotonic but noncontinuous, since that property is the least fixed-point, or the greatest fixed-point of a continuous function. The new least fixed-point proof rule discovered in § 5 is thus useful. The result is consistent with an observation made by Emerson and Clarke [8], based upon the results by Chandra [4] that inevitability under fair scheduling is not expressible in first-order arithmetic. This is because alternation of the greatest fixed-point and the least fixed-point requires the negation of universal quantification over functions, which introduces quantifiers over functions. Section 8 discusses an implication of our completeness results with respect to the work in [22] and § 9 describes the practical application of our proof method. Section 10 contains a brief example.

Notation. Logical connectives and arithmetic operators have their usual meanings unless otherwise specified. Following are some special notational conventions used in this paper.

\Rightarrow	implication
$Q _e^x$	substitution of all the free occurrences of x by e in Q
$(\forall i \in 1 \cdots n) B_i$	finite conjunction, $B_1 \wedge B_2 \wedge \cdots B_n$
$(\exists i \in 1 \cdots n) B_i$	finite disjunction, $B_1 \vee B_2 \vee \cdots B_n$
\sqsubseteq	partial ordering
$\sqcup X$	the least upper bound of the set X
$\sqcup_i \{a_i\}$	the join of the sequence $\{a_0, a_1, \cdots\}$ where $a_0 \sqsubseteq a_1 \sqsubseteq \cdots$
$\sqcap x$	the greatest lower bound of the set X
$\sqcap_i \{a_i\}$	the meet of the sequence $\{a_0, a_1, \cdots\}$ where $\cdots \sqsubseteq a_1 \sqsubseteq a_0$
$\mu x. \tau(x)$	the least fixed-point of $x = \tau(x)$, where $\tau(x)$ is monotonic
$\nu x. \tau(x)$	the greatest fixed-point of $x = \tau(x)$, where $\tau(x)$ is monotonic

2. Nondeterministic programs. The key to our approach is the recognition that parallel programs (executing on conventional digital computers where mutual exclusion of access to a single memory cell is the rule) have equivalent nondeterministic sequential forms. For example, the two-process program

$$(2.1) \quad \mathbf{cobegin} \ x := x + 1 // x := x + 1 \ \mathbf{coend}$$

has precisely the same effect on x (on a machine lacking an increment instruction) as the following nondeterministic program:

(2.2) $p_1 := p_2 := 0;$
do
 $p_1 = 0 \rightarrow t_1 := x; p_1 := 1 \parallel$
 $p_1 = 1 \rightarrow x := t_1 + 1; p_1 := 2 \parallel$
 $p_2 = 0 \rightarrow t_2 := x; p_2 := 1 \parallel$
 $p_2 = 1 \rightarrow x := t_2 + 1; p_2 := 2$
od,

where **do-od** is Dijkstra's construct for expressing nondeterminism. The t_j represent unique temporaries, and the p_j program counters.

However, the use of **do-od** to represent the nondeterministic equivalent of a parallel program is not sufficient, since a blocked parallel program (one in which no process is executable) will result in termination of its **do-od** representation since $(\forall i \in 1 \cdots n) \neg B_i$ will hold. In order to make the semantics of parallel and nondeterministic programs equivalent, we define the construct REP,

(2.3) $\mathbf{rep} B_1 \rightarrow S_1 \parallel B_2 \rightarrow S_2 \parallel \cdots \parallel B_n \rightarrow S_n \mathbf{per}.$

REP behaves similarly to **do-od** except that

- a) If $(\forall i \in 1 \cdots n) \neg B_i$, REP is said to be *blocked*. Computation ceases, but in a manner distinct from termination.
- b) The command **exit** is introduced to provide for termination if so desired. **Exit** is defined by $wp(\mathbf{exit}, R) = R$ and $wlp(\mathbf{exit}, R) = R$.
- c) All of the commands S_i must be deterministic.

Let \mathcal{L} be the assertion language, which is a superset of the programming language expressions. We denote the language of nondeterministic programs by $\text{REP}[\mathcal{L}]$. The syntax of $\text{REP}[\mathcal{L}]$ can be formally defined by the following derivative of BNF, where $\{k\}$ represents the repetition of k , possibly zero times:

$\langle \text{program} \rangle := \langle \text{initialization} \rangle \langle \text{nondeterministic part} \rangle$
 $\langle \text{initialization} \rangle := \langle \text{deterministic part} \rangle$
 $\langle \text{nondeterministic part} \rangle := \mathbf{rep} \langle \text{guarded command} \rangle \{ \langle \text{guarded command} \rangle \} \mathbf{per}$
 $\langle \text{guarded command} \rangle := \langle \text{Boolean expression} \rangle \rightarrow \langle \text{deterministic part} \rangle$
 $\langle \text{deterministic part} \rangle := \mathbf{exit} \mid \langle \text{simple statement} \rangle.$

We give numbers to the guarded commands of a **rep-per** statement, starting with 1 for the first command. The semantics of $\text{REP}[\mathcal{L}]$ is defined rigorously in § 3.

The transformation from a parallel program to its nondeterministic form is clearly effective, although it will in general be governed by the precise semantics of the source language and that of the object code for the target machine. A sketch of the algorithm is

given in [11]. The translation of (2.1) becomes:

```

 $p_1 := p_2 := 0;$ 
rep
 $p_1 = 0 \rightarrow t_1 := x; p_1 := 1 \parallel$ 
 $p_1 = 1 \rightarrow x := t_1 + 1; p_1 := 2 \parallel$ 
 $p_2 = 0 \rightarrow t_2 := x; p_2 := 1 \parallel$ 
 $p_2 = 1 \rightarrow x := t_2 + 1; p_2 := 2 \parallel$ 
 $p_1 = 2 \wedge p_2 = 2 \rightarrow \mathbf{exit}$ 

```

per.

Under the transformation, programs written with conditional critical regions have very close nondeterministic counterparts. Programs written with other synchronization primitives can still be transformed, but in general that requires somewhat more changes to the original.

3. Defining the correctness properties of parallel programs. In this section we introduce a model of computation and define various criteria for the correctness of nondeterministic programs that do not necessarily terminate. The definitions are expressed in terms of the set of sequences of states, where each sequence represents a possible execution path of the nondeterministic program. We shall also describe the obvious relationship between the correctness of a parallel program and that of its nondeterministic counterpart.

Model of computation. We define the semantics of **rep-per** by introducing an interpretive model in a way similar to Cook [6]. The difference is that we are interested in nondeterministic and often nonterminating computations.

As before, we denote the language of assertions and expressions of the program by \mathcal{L} . $\text{REP}[\mathcal{L}]$ is the programming language whose semantics we define.

An interpretive model $\mathcal{M}[\mathcal{I}]$ for the language $\text{REP}[\mathcal{L}]$ consists of an interpretation \mathcal{I} , a set of states, a computation history, and computation functions Hist and Comp . An interpretation \mathcal{I} for the language \mathcal{L} is $\langle D, P, F \rangle$, where D is a nonempty domain, P is the set of relations on D interpreting the predicate symbols of \mathcal{L} , and F is the set of operations on D interpreting the function symbols of \mathcal{L} .

A state of $\mathcal{M}[\mathcal{I}]$ is a total map $s: (\{\text{variables of } \mathcal{L}\} \rightarrow D) \cup \{\text{blocked}, \Omega\}$. We use the notation $P(s)$ to denote the substitution

$$P|_{s(y_1)}^{y_1} |_{s(y_2)}^{y_2} \cdots |_{s(y_n)}^{y_n},$$

where y_1, y_2, \dots, y_n are all the free variables of P , that is, P with its free variables simultaneously replaced by their values in state s . Then we say a formula P of \mathcal{L} is true in \mathcal{I} under state s , if $P(s)$ is true in \mathcal{I} .

The computation history associated with program A is a set of sequences of pairs $\langle i, s \rangle$ where i is the number of a guarded command, and s is the state reached after executing the i th command in the immediately preceding state. The computation history represents all possible executions of A when started in a given initial state. Therefore, all sequences which are members of the computation history must satisfy a causality relation: if a sequence is of the form $\langle \cdots \langle i, s_1 \rangle \langle j, s_2 \rangle \cdots \rangle$ then when $B_j(s_1)$ is true (the guard of the j th command in state s_1) the execution of S_j results in s_2 when started in s_1 .

LastState and LastCommand are functions defined on execution sequences. If t is a finite execution sequence of the form $\langle \dots \langle i, s \rangle \rangle$, then $\text{LastState}(t) = s$ and $\text{LastCommand}(t) = i$. Both are undefined for infinite sequences.

The function $\text{Comp}(S, s)$ assigns to a deterministic statement S and a state s the resulting state when the computation terminates. If the computation does not terminate, $\text{Comp}(S, s) = \Omega$.

The function $\text{Hist}(A, t)$, for a **rep-per** statement A and a computation history t , yields the set of possible execution sequences determined by A when started in any state in $\{\text{LastState}(u) \mid u \in t\}$.

Hist is defined as:

$$\begin{aligned} & \text{Hist}(\mathbf{rep} B_1 \rightarrow S_1 \parallel B_2 \rightarrow S_2 \parallel \dots \parallel B_n \rightarrow S_n \text{ per}, t) \\ &= \{\text{seq} \mid \text{seq} \in t \wedge (\text{LastState}(\text{seq}) = \Omega \vee \text{LastState}(\text{seq}) = \text{blocked} \\ & \quad \vee S_{\text{LastCommand}(\text{seq})} = \mathbf{exit})\} \\ & \cup \{\text{seq} \sim \langle 0, \text{blocked} \rangle \mid \text{seq} \in t \wedge (\forall i \in 1 \dots n) B_i(\text{LastState}(\text{seq})) = \text{False}\} \\ & \cup \text{Hist}(\mathbf{rep} B_1 \rightarrow S_1 \parallel \dots \parallel B_n \rightarrow S_n \text{ per}, \\ & \quad \{\text{seq} \sim \langle i, \text{Comp}(S_i, \text{LastState}(\text{seq})) \rangle \mid \text{seq} \in t \\ & \quad \quad \wedge B_i(\text{LastState}(\text{seq})) = \text{True}\}). \end{aligned}$$

We now give rigorous definitions of the predicate transformers wp and wlp .

DEFINITIONS.

$Wp(A, Q)$ is true in state s under interpretation \mathcal{I} , iff $\text{Comp}(A, s)$ is defined and Q is true in state $\text{Comp}(A, s)$ under \mathcal{I} .

$Wlp(A, Q)$ is true in state s under \mathcal{I} , iff either $\text{Comp}(A, s)$ is undefined or $\text{Comp}(A, s)$ is defined and Q is true in state $\text{Comp}(A, s)$ under \mathcal{I} .

For example, suppose that the following program is executed:

$x := 0;$

rep

(1) $x = 0 \rightarrow x := 1 \parallel$

(2) $x = 0 \rightarrow \mathbf{while} \text{ True } \mathbf{do} \text{ skip } \parallel$

(3) $x = 1 \rightarrow x := 0 \parallel$

(4) $x = 1 \rightarrow x := 2$

per.

Then part of the derivation of its computation history is given as follows:

$$\begin{aligned} & \{\langle \langle 0, x = 0 \rangle \rangle\} \\ & \{\langle \langle 0, x = 0 \rangle \langle 1, x = 1 \rangle \rangle, \langle \langle 0, x = 0 \rangle \langle 2, \Omega \rangle \rangle\} \\ & \{\langle \langle 0, x = 0 \rangle \langle 1, x = 1 \rangle \langle 3, x = 0 \rangle \rangle, \\ & \quad \langle \langle 0, x = 0 \rangle \langle 1, x = 1 \rangle \langle 4, x = 2 \rangle \rangle, \langle \langle 0, x = 0 \rangle \langle 2, \Omega \rangle \rangle\} \\ & \{\langle \langle 0, x = 0 \rangle \langle 1, x = 1 \rangle \langle 3, x = 0 \rangle \langle 1, x = 1 \rangle \rangle, \langle \langle 0, x = 0 \rangle \langle 1, x = 1 \rangle \langle 3, x = 0 \rangle \langle 2, \Omega \rangle \rangle, \\ & \quad \langle \langle 0, x = 0 \rangle \langle 1, x = 1 \rangle \langle 4, x = 2 \rangle \langle 0, \text{blocked} \rangle \rangle, \langle \langle 0, x = 0 \rangle \langle 2, \Omega \rangle \rangle\} \\ & \vdots \end{aligned}$$

In terms of our model of computation we define the following correctness properties for the REP construct:

1) *Invariance*. We say that a predicate R is *invariant* in the computation started in state s if and only if R is true in every state of every sequence in $\text{Hist}(\text{REP}, \{\langle 0, s \rangle\})$, unless the state is *blocked* or Ω .

2) *Potentiality*. We say that the program has the *potential* to establish a predicate R in the computation started in state s if and only if there exists a finite sequence r such that r is an initial section of some sequence in $\text{Hist}(\text{REP}, \{\langle 0, s \rangle\})$ and R holds in state $\text{LastState}(r)$.

3) *Inevitability*. We say that the program will *inevitably* establish R in the computation started in state s if and only if for every sequence t in $\text{Hist}(\text{REP}, \{\langle 0, s \rangle\})$, there is an initial section r of t such that R holds in state $\text{LastState}(r)$.

4) *Blocking*. We say that the program is *blocking free* in the computation started in state s if and only if for all finite sequences t in $\text{Hist}(\text{REP}, \{\langle 0, s \rangle\})$, $\text{LastState}(t) \neq \text{blocked}$.

5) *Deadlock*. We say that the program is *deadlock free* for command j in the computation started in state s if and only if for all initial sections r of all sequences in $\text{Hist}(\text{REP}, \{\langle 0, s \rangle\})$, there is a sequence q such that $r \sim q$ is an initial section of some sequence in $\text{Hist}(\text{REP}, \{\langle 0, s \rangle\})$, and $S_{\text{LastCommand}(q)} = \text{exit}$ or B_j is true in state $\text{LastState}(q)$. Informally, we require that at all times there be some continuation of the computation that leads either to completion or to a state where B_j is true.

6) *Starvation*. We say that the program is *starvation free* for command j in the computation started in state s if and only if for all initial sections r of all sequences t in $\text{Hist}(\text{REP}, \{\langle 0, s \rangle\})$, there exists a sequence q such that $r \sim q$ is an initial section of t and $S_{\text{LastCommand}(q)} = \text{exit}$ or B_j is true in state $\text{LastState}(q)$. Informally, we require that at all times whatever continuation is chosen will eventually lead either to completion or to a state where B_j is true. Note that if the program is starvation free for command j then it is deadlock free for command j , and if it is deadlock free for some j then it is blocking free.

It should be clear how these six criteria can express the parallel program notions of weak correctness, deadlock and starvation. In particular, a parallel program is weakly correct in the sense of Floyd (i.e., a post-condition R is true at termination) if and only if its nondeterministic equivalent has the predicate $P \Rightarrow R$ invariant where R is the post-condition and P is the disjunction of the guards of all exit statements. The parallel program is termination correct if and only if $P \Rightarrow R$ is invariant and $P \wedge \neg Q$ is inevitable, where Q is the disjunction of the other guards. The parallel program is blocking free if and only if the nondeterministic program is blocking free. Process P_j is deadlock free if and only if the guard of some command derived from the main loop of the process is deadlock free, and similarly for starvation free if fair scheduling is not assumed. (Fairness is discussed in § 7.)

4. Axiomatization. The following are rules of inference for reasoning about the correctness properties described in the previous section. Wlp and wlp are the predicate transformers for sequential programs defined in [7], and we assume the existence of complete proof rules for them (see, for example, [14]; in the notation of dynamic logic [13], for a deterministic program S , $wlp(S, R)$ is the same as $\langle S \rangle R$ and $wp(S, R)$ is the same as $[S]R$). Even though wlp and wp appear in the assumptions of the proof rules, they never appear in the assertions placed in programs, but rather only in proofs. Wip , wpp , wep , wbp , wdp , and wsp are weakest pre-condition predicate transformers for the corresponding correctness properties. For example, $wpp(\text{REP}, R)$ is true in state s (under interpretation \mathcal{I}) if and only if the potential to establish R exists when the computation is started in s .

1. *Invariance*

$$\frac{(\forall k \in 1 \cdots n)[I \wedge B_k \Rightarrow wlp(S_k, I)], I \Rightarrow R}{I \Rightarrow wip(\text{REP}, R)};$$

2. *Potentiality*

$$\frac{\neg Q(0), Q(m+1) \wedge \neg R \Rightarrow (\exists k \in 1 \cdots n)[B_k \wedge wp(S_k, Q(m))]}{Q(m) \Rightarrow wpp(\text{REP}, R)};$$

3. *Inevitability*

$$\frac{\neg Q(0), Q(m+1) \wedge \neg R \Rightarrow (\exists i \in 1 \cdots n)B_i \wedge (\forall k \in 1 \cdots n)[B_k \Rightarrow wp(S_k, Q(m))]}{Q(m) \Rightarrow wep(\text{REP}, R)};$$

4. *Blocking free*

$$\frac{(\forall k \in 1 \cdots n)[I \wedge B_k \Rightarrow wp(S_k, I)], I \Rightarrow (\exists k \in 1 \cdots n)B_k}{I \Rightarrow wbp(\text{REP})};$$

5. *Deadlock free*

$$\frac{(\forall k \in 1 \cdots n)[I \wedge B_k \Rightarrow wp(S_k, I)], I \Rightarrow wpp(\text{REP}, B_j) \vee Q}{I \Rightarrow wdp_j(\text{REP})};$$

6. *Starvation free*

$$\frac{(\forall k \in 1 \cdots n)[I \wedge B_k \Rightarrow wp(S_k, I)], I \Rightarrow wep(\text{REP}, B_j) \vee Q}{I \Rightarrow wsp_j(\text{REP})};$$

where Q is the disjunction of all the guards of **exit** commands.

5. The fixed-point approach to obtaining sound and complete proof rules. We must of necessity justify the axiomatization just presented and prove it to be sound and complete. Soundness and completeness proofs usually require a model of computation and an interpretation for predicate transformers in that model. An axiom system is said to be sound if all theorems are true in the model. The system is relatively complete if all true formulas are provable.

It has been observed by Basu and Yeh [2], Wadsworth (in [24]) and Clarke [5] that weakest pre-conditions and weakest liberal pre-conditions for loops are actually the least and greatest fixed-points of continuous functions over predicates. Once this fact is shown using a model of computation, soundness and completeness proofs can be accomplished by simple formal reasoning [5].

We shall extend these results by proving several metatheorems relating axiomatic proof rules to predicate transformers. In particular, if the predicate transformer is shown, in the computational model, to be either the least or greatest fixed-point of some recursive function (subject to certain constraints), then there is an applicable sound and complete proof rule schema for that predicate transformer. The theorems enable us to argue the soundness and completeness of our axiomatization independently of a model. (The model is needed only to show that predicate transformers are extremum fixed-points.) We first examine briefly the fixed-point properties of the weak and strong correctness of regular sequential programs without procedure calls.

5.1. Lattice of total predicates. The domain of all total predicates forms a complete lattice defined by the partial ordering

$$p \sqsubseteq q \quad \text{iff} \quad (\forall x)(p \Rightarrow q),$$

where x is the list of all free variables occurring in p and q . For this lattice False is the least element and True is the greatest element. It is also known that monotonic functions over a complete lattice have both a least and a greatest fixed-point [25].

Let us consider the function

$$\tau(q, R) = (\neg B \wedge R) \vee (B \wedge pt(S, q)),$$

where S is a program composed of assignment, composition and conditionals. Pt is a predicate transformer defined by

$$pt(x := e, q) = q|_e^x,$$

$$pt(S1; S2, q) = pt(S1, pt(S2, q)),$$

$$pt(\text{if } C \text{ then } S_1 \text{ else } S_2, q) = (C \wedge pt(S_1, q)) \vee (\neg C \wedge pt(S_2, q))$$

It is easy to see that $pt(S, q)$ is continuous and so is $\tau(q, R)$; therefore, τ has both a least and a greatest fixed-point. The following theorem describes the relations between fixed-points and predicate transformers for loops.

THEOREM (Basu and Yeh, Wadsworth, Clarke). *The greatest fixed-point of $\tau(q, R)$ with respect to q is the weakest liberal pre-condition and the least fixed-point of $\tau(q, R)$ with respect to q is the weakest pre-condition of the program “while B do S ” for the post-condition R .*

DEFINITIONS.

A function f is *monotonic* iff $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$.

A function f is *continuous from below* iff for any directed set D , $f(\sqcup D) = \sqcup\{f(x) \mid x \in D\}$.

A function f is *continuous from above* iff for any directed set D , $f(\sqcap D) = \sqcap\{f(x) \mid x \in D\}$.

A function f is *continuous* iff it is either continuous from below or continuous from above.

Examples.

$wp(S, p)$, where S is a deterministic program, is continuous from below with respect to p .

$wlp(S, p)$, where S is a deterministic program, is continuous from above with respect to p .

The set theoretical characterization of the least fixed-point of a function $\tau(x)$ continuous from below is

$$\sqcup_i \{\tau^i(\text{False})\},$$

i.e., the limit reachable from the least element of the lattice by repeated applications of τ . Symmetrically the greatest fixed-point of a function $\tau(x)$ continuous from above is

$$\sqcap_i \{\tau^i(\text{True})\}.$$

Another characterization is due to Tarski (in [25]) and applies to monotonic functions. The least fixed-point is

$$\sqcap\{x \mid \tau(x) \sqsubseteq x\},$$

and the greatest fixed-point is

$$\sqcup\{x \mid x \sqsubseteq \tau(x)\}.$$

We review some of the properties of monotonic and continuous functions and their least and greatest fixed-points.

a) If $h(x)$ is continuous, then $h(x)$ is monotonic.

b) If $g(x, y)$ is continuous from below [above] with respect to x and y , then $\mu x.g(x, y)$ [$\nu x.g(x, y)$] is continuous from below [above] with respect to y .

c) If $g(x, y)$ is continuous from below [above] with respect to x and y , then $\nu x.g(x, y)$ [$\mu x.g(x, y)$] is monotonic (not necessarily continuous) with respect to y .

d) If $g(x, y)$ is monotonic with respect to x and y , then $\mu x.g(x, y)$ and $\nu x.g(x, y)$ are both monotonic with respect to y .

Proof of d. Because of Tarski's theorem $\mu x.g(x, y)$ exists and is $\sqcap\{k \mid g(k, y) \sqsubseteq k\}$. For r, s such that $r \sqsubseteq s$, let $r' = \mu x.g(x, r)$ and $s' = \mu x.g(x, s)$. Then $g(s', r) \sqsubseteq g(s', s) = s'$. Therefore, $s' \in \{k \mid g(k, r) \sqsubseteq k\}$. Thus, $r' = \sqcap\{k \mid g(k, r) \sqsubseteq k\} \sqsubseteq s'$, or $r' \sqsubseteq s'$. The monotonicity of $\nu x.g(x, y)$ can be shown similarly. \square

5.2. Proof rule schemas. We now state three metatheorems about proof rules of the form

$$P \Rightarrow W,$$

where W is a predicate transformer found to be either the least or the greatest fixed-point of a function $\tau(x)$.

We will show that proof rules described by the metatheorems are sound and relatively complete for proving formulas of the above form. We must first extend Cook's notion of relative completeness [6], since in some cases these fixed-points cannot be expressed in a first-order language. In Theorem 2 we will need arithmetic, and in Theorem 3 we will need a second-order language.

We say that a deductive system for formulas of the form

$$P \Rightarrow \text{pre}(S, Q)$$

(where P and Q are logical formulas, S is a program, and $\text{pre}(S, Q)$ is a predicate transformer) is (relatively) complete if, when $P \Rightarrow \text{pre}(S, Q)$ is true in the model and all true logical formulas are given as axioms, $P \Rightarrow \text{pre}(S, Q)$ is provable in the system.

THEOREM 1. (Greatest fixed-point proof rule for monotonic functions.) *If $\text{pre}(S, R)$ is the predicate transformer which is equivalent to the greatest fixed-point of $\tau(x)$, a monotonic function, and*

(1) *the assertion language can express the greatest fixed-point of τ ,*

(2) *there is a sound and relatively complete proof system (with modus ponens) to prove formulas of the form $P \Rightarrow \tau(Q)$,*
then

$$\frac{P \Rightarrow \tau(P)}{P \Rightarrow \text{pre}(S, R)}$$

is a sound and relatively complete proof rule. (This is a generalization of a result of Clarke [5].)

Proof. (Soundness.) Since $P \Rightarrow \tau(P)$, P must be a member of the set $\{x \mid x \sqsubseteq \tau(x)\}$ and therefore $P \sqsubseteq \sqcup\{x \mid x \sqsubseteq \tau(x)\}$. From the definition $\models \text{pre}(S, R) \equiv \nu x.\tau(x)$; hence

$$\models P \Rightarrow \sqcup\{x \mid x \sqsubseteq \tau(x)\}.$$

Therefore, the rule is sound.

(*Relative completeness.*) Suppose for some J

$$\vDash J \Rightarrow \text{pre}(S, R).$$

From assumption (1) we can choose P to be the formula equivalent to $\nu x.\tau(x)$, that is,

$$\vDash P \equiv \sqcup\{x \mid x \sqsubseteq \tau(x)\}.$$

Then, since $\vDash J \Rightarrow P$ and both J and P are logical formulas, $\vdash J \Rightarrow P$ from assumption (2). Since P is a fixed-point of τ , $\vDash P \Rightarrow \tau(P)$, and from assumption (2) $\vdash P \Rightarrow \tau(P)$. We have $\vdash J \Rightarrow P$ and $\vdash P \Rightarrow \tau(P)$; therefore, from the premise of our rule and modus ponens $\vdash J \Rightarrow \text{pre}(S, R)$. \square

THEOREM 2. (Least fixed-point proof rule for continuous functions.) *If $\text{pre}(S, R)$ is the predicate transformer which is equivalent to the least fixed-point of $\tau(x)$, where τ is continuous from below, and*

(1) *the assertion language includes arithmetic and there is a formula in the assertion language of the form $P(i)$ such that $\vDash P(i) \equiv \tau^i(\text{False})$,*

(2) *there is a sound and relatively complete proof system (with modus ponens) to prove formulas of the form $Q_1 \Rightarrow \tau(Q_2)$,*
then

$$\frac{J(n+1) \Rightarrow \tau(J(n)), \neg J(0)}{J(n) \Rightarrow \text{pre}(S, R)}$$

is a sound and complete proof rule. (This is a generalization of a result from [14].)

Proof. (Soundness.) From the premise of the rule,

$$\vDash J(0) \Rightarrow \text{False} \quad \text{and}$$

$$\vDash J(i) \Rightarrow \tau^i(\text{False}) \quad \text{for all } i > 0.$$

The latter is easily shown by induction and using the monotonicity of τ :

$$\vDash J(i+1) \Rightarrow \tau(J(i)) \quad (\text{from the premise}),$$

$$\vDash \tau(J(i)) \Rightarrow \tau(\tau^i(\text{False})) \quad (\text{monotonicity of } \tau).$$

Therefore, $\vDash J(i+1) \Rightarrow \tau^{i+1}(\text{False})$.

Thus,

$$\vDash J(n) \Rightarrow \sqcup_i \{\tau^i(\text{False})\},$$

and the rule is sound.

(*Relative completeness.*) Suppose $\vDash P \Rightarrow \text{pre } S, R$). From the expressibility assumption $\tau^i(\text{False})$ can be expressed by a formula $J(i)$. Then

$$\vdash J(0) \equiv \text{False} \quad \text{and}$$

$$\vDash J(n+1) \Rightarrow \tau(J(n)).$$

From assumption (2),

$$\vdash J(n+1) \Rightarrow \tau(J(n)).$$

By the least fixed-point proof rule and generalization, $\vdash (\exists n \geq 0) J(n) \Rightarrow \text{pre}(S, R)$. But $J(i)$ forms a chain, $\vDash J(0) \Rightarrow J(1), \dots, \vDash J(i) \Rightarrow J(i+1), \dots$, and τ is continuous, so $(\exists n \geq 0) J(n)$ is greater than or equal to a fixed-point of τ . Therefore, $\vDash \text{pre}(S, R) \Rightarrow (\exists n \geq 0) J(n)$, and $\vDash P \Rightarrow (\exists n \geq 0) J(n)$. From assumption (2), $\vdash P \Rightarrow (\exists n \geq 0) J(n)$. Thus, $\vdash P \Rightarrow \text{pre}(S, R)$. \square

We note here that the sound and complete proof rules for regular programs of [14] are instances of Theorems 1 and 2.

Theorem 2 is only applicable to functions which are continuous from below. Actually, all the properties we have dealt with in § 3 can be expressed by continuous functions and no further proof schemas are necessary. However, we will later have to deal with the least fixed-point of monotonic but noncontinuous functions for which we do not have a proof rule schema. Hence the following theorem:

THEOREM 3. (Least fixed-point proof rule for monotonic functions.) *If pre (S, R) is the predicate transformer which is equivalent to the least fixed-point of a monotonic function $\tau(f)$, and*

- (1) *the assertion language is second order and can express the least fixed-point of τ ,*
 - (2) *there is a sound and relatively complete proof system (with modus ponens) to prove formulas of the form $P \Rightarrow \tau(Q)$,*
- then*

$$\frac{(\tau(J(x)) \Rightarrow J(x)) \Rightarrow (P \Rightarrow J(x))}{P \Rightarrow \text{pre}(S, R)}$$

is a sound and complete proof rule, where J is a fresh predicate variable and x is the list of all free variables modifiable by τ .

Proof. (Soundness.) Assume that the premise of the rule is true in the model, that is, $\models (\tau(J(x)) \Rightarrow J(x)) \Rightarrow (P \Rightarrow J(x))$. Then for any member q of the set $\{q \mid \tau(q) \sqsubseteq q\}$, $P \Rightarrow q$. Therefore,

$$P \Rightarrow \sqcap \{q \mid \tau(q) \sqsubseteq q\} \quad \text{or} \quad P \Rightarrow \mu q. \tau(q)$$

is true, hence $\models P \Rightarrow \text{pre}(S, R)$. \square

(Completeness.) Suppose $\models P \Rightarrow \text{pre}(S, R)$. Since $\models \text{pre}(S, R) \equiv \mu q. \tau(q)$, $\models P \Rightarrow \sqcap \{q \mid \tau(q) \sqsubseteq q\}$. Therefore $\models (\forall J)(\tau(J(x)) \Rightarrow J(x)) \Rightarrow (P \Rightarrow J(x))$. Since we assume that the above formula is provable, $\vdash (\tau(J(x)) \Rightarrow J(x)) \Rightarrow (P \Rightarrow J(x))$. Therefore, $\vdash P \Rightarrow \text{pre}(S, R)$. \square

6. Fixed-point characterization of the correctness of parallel programs. As was described in the previous section, in order to obtain sound and complete proof rules, our task will be to show that the weakest pre-conditions for our correctness properties are actually extremum fixed-points of continuous functions. We will present only the proofs for invariance (a greatest fixed-point) and potentiality (a least fixed-point). These proofs are highly representative of the others.

1) *Invariance.* We claim that $wip(\text{REP}, R)$, the weakest pre-condition that assures that R holds at every state in the computation history which is neither blocked nor Ω , is the greatest fixed-point of the following formula $\iota(p)$:

$$(6.1) \quad \iota(p) = R \wedge (\forall i \in 1 \cdots n)[B_i \Rightarrow wlp(S_i, p)].$$

Proof. (Weakest.) Suppose I is a pre-condition guaranteeing the invariance of R . Then we claim that

$$I \Rightarrow \nu p. \iota(p).$$

As a lemma we claim that if s is a state such that $\iota^m(\text{True})(s)$ is true (under \mathcal{S}), then for any sequence r of $\text{Hist}(\text{REP}, \{\langle 0, s \rangle\})$, R is true in the first m states of r . The claim is true when $m = 1$, since $\iota^1(\text{True}) = R$. Suppose the lemma is true for $\iota^k(\text{True})$. Since $\iota^{k+1}(\text{True}) = R \wedge (\forall i \in 1 \cdots n)[B_i \Rightarrow wlp(S_i, \iota^k(\text{True}))]$, if execution is started in a state satisfying $\iota^{k+1}(\text{True})$, R will be true initially and $\iota^k(\text{True})$ will be true in the next state. Thus R will hold in the first $k + 1$ states, which completes the proof of the lemma.

Now for any i , $I \Rightarrow \iota^i(\text{True})$, and because of the completeness of the lattice $I \Rightarrow \sqcap_i \{\iota^i(\text{True})\}$. Since wlp is continuous from above, $\iota(p)$ is continuous from above and $\sqcap_i \{\iota^i(\text{True})\}$ is the greatest fixed-point.

(*Pre-condition.*) Next we show that every fixed-point of (6.1) is a valid pre-condition for invariance. Since a fixed-point g satisfies

$$g = \iota(g) = R \wedge (\forall i \in 1 \cdots n)[B_i \Rightarrow wlp(S_i, g)],$$

R is initially true. After the execution of any one command g is again true; hence R is true in every state. Thus, g is a pre-condition for invariance. \square

2) *Potentiality.* We claim that $wpp(\text{REP}, R)$, the weakest pre-condition that assures that REP has the potential to establish R , is the least fixed-point of the following formula $\pi(p)$:

$$(6.2) \quad \pi(p) = R \vee (\exists i \in 1 \cdots n)[B_i \wedge wp(S_i, p)].$$

Proof. (Weakest.) Let P be any pre-condition that guarantees the potentiality of R . Then there is an execution sequence such that R is true at some intermediate state of the sequence. That is, for any initial values of the program variables, there is an integer n such that

$$P \equiv p_n,$$

where

$$p_0 \Rightarrow R$$

and for all k ($0 < k \leq n$)

$$p_k \Rightarrow R \vee (\exists i \in 1 \cdots n)[B_i \vee wp(S_i, p_{k-1})].$$

That is, p_k is a pre-condition assuring the possibility of establishing R within k steps of execution. Then

$$p_0 \Rightarrow \pi(\text{False}), \quad \text{since } \pi(\text{False}) \equiv R, \quad \text{and}$$

$$p_k \Rightarrow \pi(p_{k-1}) \quad \text{for } 0 < k \leq n.$$

Define the chain π_k by

$$\pi_0 = \pi(\text{False}) \quad \text{and}$$

$$\pi_{k+1} = \pi(\pi_k).$$

Then it is easily shown by induction that

$$p_k \Rightarrow \pi_k \quad \text{for all } 0 \leq k \leq n;$$

that is,

$$P \Rightarrow \pi_n.$$

Since $\pi_0 \sqsubseteq \pi_1 \sqsubseteq \cdots \sqsubseteq \pi_n \sqsubseteq \cdots$, the least upper bound exists and is the least fixed-point of π . Hence, $P \Rightarrow \sqcup_i \{\pi_i\}$, which is the same as $P \Rightarrow (\exists i \geq 0) \pi_i$.

(*Pre-condition.*) From the fixed-point induction rule [25],

$$\frac{\text{False} \Rightarrow WP, q \Rightarrow WP \vdash \pi(q) \Rightarrow WP}{\mu q. \pi(p) \Rightarrow WP},$$

where WP is the weakest pre-condition. Since $\text{False} \Rightarrow WP$ is a tautology, we have only to prove $\pi(q) \Rightarrow WP$ assuming $q \Rightarrow WP$. If $\pi(q)$ is true, then either R is true now or there

is a way to establish q in one step. From the assumption that q guarantees the potentiality of R , $\pi(q)$ guarantees the potentiality of R . Therefore, $\pi(q) \Rightarrow WP$. \square

3) *Inevitability*. We say that R is inevitable if and only if no execution sequence can avoid establishing it. We claim that the weakest pre-condition that guarantees the inevitability of R , $wep(R)$, is the least fixed-point of the formula

$$(6.3) \quad \eta(p) = R \vee (\exists i \in 1 \cdots n) B_i \wedge (\forall i \in 1 \cdots n) [B_i \Rightarrow wp(S_i, p)].$$

The proof is similar to that of potentiality.

4) *Blocking*. We claim that the greatest fixed-point of the formula

$$(6.4) \quad \beta(p) = (\exists i \in 1 \cdots n) B_i \wedge (\forall i \in 1 \cdots n) [B_i \Rightarrow wp(S_i, p)]$$

is the weakest pre-condition $wbp(\text{REP})$ that assures safety from blocking. The proof is similar to that of invariance.

5) *Deadlock*. We say that command j is safe from deadlock if and only if B_j is always potentially establishable. The weakest pre-condition for deadlock freeness is the greatest fixed-point of

$$(6.5) \quad \delta(p) = (wpp(B_j) \vee Q) \wedge (\forall i \in 1 \cdots n) [B_i \Rightarrow wp(S_i, p)],$$

where Q is the disjunction of all the guards of **exit** commands. The proof is similar to that of invariance.

6) *Starvation*. We say that command j is starvation free if and only if B_j is always inevitable. The weakest pre-condition for starvation freeness is the greatest fixed-point of

$$(6.6) \quad \sigma(p) = (wep(R) \vee Q) \wedge (\forall i \in 1 \cdots n) [B_i \Rightarrow wp(S_i, p)].$$

where Q is the disjunction of all the guards of **exit** commands. The proof is similar to that of invariance.

From the results of this section and Theorems 1 and 2, we are able to conclude that the proof rules given in § 4 are sound and relatively complete.

7. Fairness. The model of parallel computation used up to this point, i.e., pure nondeterminism, does not take into account any notion of scheduling. The properties of weak correctness, blocking, and deadlock are not affected by scheduling, but inevitability and hence starvation are. What we have actually defined so far is inevitability under true nondeterminism: a worst case scheduler. Therefore, if we can prove absence of starvation using the proof system of § 4, we can guarantee that the program is starvation free under any scheduler, as long as that scheduler has the minimal property of guaranteeing progress to some executable process. However, that assumption is often too strong for realistic systems.

Consider the parallel program

```

cobegin
  repeat  $x := x + 1$  end //
  repeat  $y := y + 1$  end
coend.

```

If separate processors serve each process, it is a reasonable assumption that x and y will both become arbitrarily large. Under true nondeterminism the possibility exists that one of the processes is serviced continually while the other is ignored.

In this section we will consider the problem of fair scheduling in a two-process system, and obtain new weakest pre-conditions for inevitability and absence of starvation. The results are easily extendable to the multi-process case.

We take the definition of fair scheduling to be that every process continuously in an “enabled” state for an unboundedly long time must eventually execute. This is a weak definition of fairness, but one which we feel to be the appropriate choice. This definition ensures the unbounded growth of both x and y in the above program. However in the following program, it does not:

```

cobegin
    repeat  $B := \text{not } B; x := x + 1$  end  $\parallel$ 
    repeat when  $B$  do  $y := y + 1$  end
coend.

```

This is due to the fact that the second process is not *continuously* enabled for an unbounded time. Our definition corresponds, however, to an execution model in which each process executes on an independent processor, and in such a model it is indeed possible for the second program to starve its second process.

The binding of process to processor is likely to become increasingly valid with the increasing importance of microprocessors. Some would argue, however, that “fairness” is a stronger notion than this, and that the second example should not allow starvation of either process. We feel that to commit the execution model to this would inevitably result in overspecification, in the sense that the scheduler would frequently make decisions that are unnecessary. Thus, we feel that such scheduling criteria should be built into the program itself, or perhaps into some superimposed execution model, but not into the underlying model we are describing here.

To model our notion of fairness with nondeterminism we need to introduce, in addition to program counters, variables that represent process numbers. After decomposing each process into indivisible actions, we associate the corresponding process number with each command derived from the process. The nondeterministic version of the parallel program

```

cobegin
     $S_{11}; S_{12}; \dots; S_{1n}$   $\parallel$ 
     $S_{21}; S_{22}; \dots; S_{2m}$ 
coend,

```

where the S_{ij} are taken to be the indivisible actions of each process, becomes

```

rep
     $B_{11} \rightarrow_1 S'_{11} \parallel$ 
     $\vdots$ 
     $B_{1n} \rightarrow_1 S'_{1n} \parallel$ 
     $B_{21} \rightarrow_2 S'_{21} \parallel$ 
     $\vdots$ 
     $B_{2m} \rightarrow_2 S'_{2m}$ 
per,

```

where the B_{ij} include tests on the program counters, and the S'_{ij} are the S_{ij} concatenated with the necessary program counter updates. In the following we only consider the two-process case, but extension to the n -process case is straightforward.

In defining “fair inevitability” (i.e., inevitability under the processor-per-process execution model), we rely on the following basic assumption: whereas under pure nondeterminism a predicate R will be inevitable only if a state is reached in which the only executable commands must establish R , in the fair model all that is required is to reach a state such that some command remains continually enabled and needs only a single execution to guarantee establishing R . Of course reaching this state may itself require the fair execution model.

Consider the function

$$\begin{aligned} \varphi(q, R) = & \text{wep}(R \vee (\exists k \in 1 \cdots n)[\text{wip}_2(B_{1k} \wedge \text{wp}(S_{1k}, q))]) \\ & \vee (\exists k \in 1 \cdots m)[\text{wip}_1(B_{2k} \wedge \text{wp}(S_{2k}, q))], \end{aligned}$$

where $\text{wip}_1(p)$ is the weakest pre-condition for the invariance of p across process 1 only, and $\text{wip}_2(p)$ for process 2 only. In other words

$$\text{wip}_m(R) = \nu q.((\forall k)[R \wedge B_{mk} \Rightarrow \text{wlp}(S_{mk}, q)]).$$

$\varphi(\text{wep}(R), R)$ expresses the “one-shot” fair-inevitability for the process m described above. It says that R is fair-inevitable exactly when either it is inevitable under pure nondeterminism, or a state will inevitably be reached such that the truth of

$$B_{2k} \wedge \text{wp}(S_{2k}, \text{wep}(R))$$

is kept invariant by process 1, or the truth of

$$B_{1k} \wedge \text{wp}(S_{1k}, \text{wep}(R))$$

is kept invariant by process 2, for some k . The “one-shot” fairness predicate must be iterated for cases where the fairness principle must be involved more than once before the desired outcome. Thus, R is fair-inevitable if and only if either

- 1) R is established under one-shot fair-inevitability, or
 - 2) (1) is true now or a state will be reached in which process 1 is continuously enabled and is guaranteed to establish (1) upon execution, or process 2 is continuously enabled and is guaranteed to establish (1) upon execution, or
 - 3) same as (2) reading (2) for (1), or
- ⋮

We therefore conjecture that the weakest pre-condition for fair-inevitability, $\text{wfep}(R)$, is given by

$$\text{wfep}(R) = \mu q. \varphi(q, R).$$

(A proof appears in the Appendix.)

Note that $\text{wfep}(R)$ is not necessarily continuous, since it is the least fixed-point of the greatest fixed-point of a continuous function (refer to property (c) of 5.1). Thus, we cannot use Theorem 2 to generate a proof rule for $\text{wfep}(R)$. However, φ is monotonic and hence we can use Theorem 3 to generate the rule.

We can thus obtain a proof rule for the absence of starvation under fair-scheduling by simply replacing $\text{wep}(R)$ by $\text{wfep}(R)$ in formula (6.6).

8. Soundness and completeness. The soundness and completeness of the axiom system of § 4 is a direct consequence of Theorems 1 and 2, and § 6.

As a corollary to the soundness and completeness results we can make a claim about the requirements for ghost variables in weak correctness proofs. Owicki [21] has

shown that history sequences and location counters as ghost variables, with general arithmetic operations on them, are sufficient to prove weak correctness.

COROLLARY. *A proof of the weak correctness of a parallel program with conditional critical regions requires neither history sequences nor an arithmetic domain for ghost variables if we ease its requirements to allow assertions to refer to location counters of other processes.*

Proof. Let P be the disjunction of the guards of the exit commands. Then in order to prove the weak correctness of a given post-condition R , we need only prove the invariance of $P \Rightarrow R$. Thus we conclude that finite-valued location counters are all that is necessary. \square

9. Practical proof methods. In order to carry out the verification of practical programs, it is usually not convenient to transform a well-structured program into explicit nondeterministic form. The results we have obtained do not require the explicit transformation. It is only necessary to identify indivisible program segments in order to apply the proof rules.

Let us consider the program schema

```
{pre}
  cobegin
    A; with  $r$  when  $B$  do  $S$ ;  $C$  //
    D; with  $r$  when  $E$  do  $F$ ;  $G$ 
  coend
{post}.
```

Completeness guarantees that all we have to do is to introduce a location counter for each process and increment it after each indivisible action of the process, for example:

```
 $p_1 := p_2 := 0$ ;
cobegin
  A;  $p_1 := 1$ ; with  $r$  when  $B$  do  $S$ ;  $p_1 := 2$ ;  $C$ ;  $p_1 := 3$  //
  D;  $p_2 := 1$ ; with  $r$  when  $E$  do  $F$ ;  $p_2 := 2$ ;  $G$ ;  $p_2 := 3$ 
coend.
```

Then, to prove weak correctness, we can always invent an invariant J such that

$$\begin{aligned} \text{pre} \wedge p_1 = 0 &\Rightarrow \text{wlp}(A; p_1 := 1, J), \\ J \wedge B \wedge p_1 = 1 &\Rightarrow \text{wlp}(S; p_1 := 2, J), \\ J \wedge p_1 = 2 &\Rightarrow \text{wlp}(C; p_1 := 3, \text{post}), \\ \text{pre} \wedge p_2 = 0 &\Rightarrow \text{wlp}(D; p_2 := 1, J), \\ J \wedge E \wedge p_2 = 1 &\Rightarrow \text{wlp}(F; p_2 := 2, J), \\ J \wedge p_2 = 2 &\Rightarrow \text{wlp}(G; p_2 := 3, \text{post}). \end{aligned}$$

Further comparison to Owicki and Gries' method [22] is in order. Their method requires the introduction of sufficient ghost variables and ghost assignments so as to be

able to prove the independent sequential correctness of each process and the noninterference of one process on the intermediate assertions of another. We have proven that simple program counters are sufficient (although not necessarily easier to work with). Proving noninterference in the Owicki and Gries method is analogous to proving invariance in our method.

Consider the previous example again. Let Q_1, R_1 and Q_2, R_2 be interference-free pre- and post-conditions for the first and second conditional critical regions respectively. Then

$$(p_1 = 1 \Rightarrow Q_1) \wedge (p_2 = 1 \Rightarrow Q_2) \wedge (p_1 = 2 \Rightarrow R_1) \wedge (p_2 = 2 \Rightarrow R_2)$$

can be our invariant J . Note that the complexity of verification is the same for both methods once the necessary assertions are stated.

10. Example. Consider the following simple program which operates on three buffers:

cobegin

with x, y **when** $x > 0 \wedge y < n$ **do** $x := x - 1; y := y + 1$ **end** //

with y, z **when** $y > 0 \wedge z < n$ **do** $y := y - 1; z := z + 1$ **end** //

with z, x **when** $z > 0 \wedge x < n$ **do** $z := z - 1; x := x + 1$ **end**

coend.

We will show that the above program is starvation free on the first process with the pre-condition $0 \leq x, y, z \leq n \wedge 0 < x + y + z < 3n$. We postulate that $Q(m) \equiv 0 \leq x, y, z \leq n \wedge 0 < x + y + z < 3n \wedge m = x + 2z + 3y$. Then

$$Q(0) \equiv \text{False},$$

$$Q(m+1) \wedge (x \leq 0 \vee y \geq n) \Rightarrow (y > 0 \wedge z < n \vee z > 0 \wedge x < n)$$

$$\wedge [y > 0 \wedge z < n \Rightarrow wp(y := y - 1; z := z + 1, Q(m))]$$

$$\wedge [z > 0 \wedge x < n \Rightarrow wp(z := z - 1; x := x + 1, Q(m))].$$

Therefore, from the proof rule 3,

$$Q(m) \Rightarrow wep(\text{REP}, x > 0 \wedge y < n).$$

Since $(\exists m \geq 0)Q(m)$ is invariant over all the commands, the program is starvation free.

11. Conclusion. We have presented an axiom system that defines the total correctness of parallel programs expressed in nondeterministic-sequential form. By arguing about the relationship between weakest pre-conditions and fixed-points, we formulated three theorems about the soundness and completeness of proof rule schemas that allowed us to immediately conclude that the axiomatization itself is sound and complete. We also recognized the fairness problem introduced by the transformation to nondeterminism, and have presented a somewhat modified formalism in which true parallelism may be correctly modelled. The result is a rigorous formal definition of the total correctness of parallel programs. A corollary of our work is that the use of auxiliary variables and computations other than simple location counters is not necessary in proving the correctness of parallel programs.

Appendix. Proof of the weakest pre-condition for fair-inevitability. We prove that the least fixed-point

$$\mu q. \varphi(q, R), \quad \text{where} \quad \varphi(q, R) = \text{wep}(R \vee \text{wip}_1(B_2 \wedge \text{wp}(S_2, q)) \vee \text{wip}_2(B_1 \wedge \text{wp}(S_1, q))),$$

is the weakest pre-condition for fair-inevitability. Here we are simplifying by assuming exactly one command per process (hence S_1 instead of S_{1k}). However, the proof is straightforwardly, though tediously, extendable to the general case. The definition of the fair-inevitability of R is that R will be established under a scheduler which serves processes at random, except when a process is continuously enabled, in which case that process may be assumed to eventually execute.

(*Pre-condition.*) First we will prove that $\mu q. \varphi(q, R)$ is a valid pre-condition for fair-inevitability. Because the fixed-point induction rule [25]

$$\frac{\text{False} \Rightarrow P, q \Rightarrow P \vdash \varphi(q, R) \Rightarrow P}{\mu q. \varphi(q, R) \Rightarrow P}$$

is sound even for monotonic but noncontinuous functions φ , and $\text{False} \Rightarrow P$ is a tautology, it is sufficient to show that $q \Rightarrow P \vdash \varphi(q, R) \Rightarrow P$ in order to conclude

$$\mu q. \varphi(q, R) \Rightarrow P.$$

Therefore, we will show that $q \Rightarrow WP \vdash \varphi(q, R) \Rightarrow WP$, where WP is the weakest pre-condition for fair-inevitability of R .

Suppose $\varphi(q, R)$ is true. Then it is inevitable that a state is reached in which either R , $\text{wip}_1(B_2 \wedge \text{wp}(S_2, q))$, or $\text{wip}_2(B_1 \wedge \text{wp}(S_1, q))$ is true.

Case 1) R is true. Since R implies WP , WP is fair-inevitable.

Case 2) $\text{wip}_1(B_2 \wedge \text{wp}(S_2, q))$ is true. This means that $B_2 \wedge \text{wp}(S_2, q)$ must remain true as long as process 1 executes. Because the scheduler must service process 2 eventually, it is inevitable that q will be established. Since $q \Rightarrow WP$ from the assumption, WP is fair-inevitable.

Case 3) $\text{wip}_2(B_1 \wedge \text{wp}(S_1, q))$ is true. Argument mirrors Case 2.

Thus, $\varphi(q, R)$ implies WP is fair-inevitable, which in turn implies R is fair-inevitable. Therefore, $\varphi(q, R) \Rightarrow WP$ from the assumption that WP is the weakest pre-condition for the fair-inevitability of R .

(*Weakest.*) We now prove that $\mu q. \varphi(q, R)$ is the weakest of all valid pre-conditions.

Whenever one process is repeatedly executing while the other is continuously enabled, the fair scheduler guarantees that the continuously enabled process will eventually execute. We call this eventuality a *yielding*.

Our goal is to show that for any pre-condition r , $r \Rightarrow q$ is true for all q satisfying $\varphi(q, R) \Rightarrow q$. Since r is a pre-condition, if a computation is started in a state satisfying r , eventually a state will be reached in which R is true. There may be a number of yieldings during this computation. Thus we will achieve our goal by induction on the number of yieldings.

Define the sequence φ_i by

$$\begin{aligned} \varphi_0 &= \varphi(\text{False}, R) = \text{wep}(R), \\ \varphi_{i+1} &= \varphi(\varphi_i, R). \end{aligned}$$

Let r be any pre-condition that guarantees the inevitability of R under fair-scheduling. Then R must be established within some number of yieldings, say k . There

is a sequence of $k + 1$ logical formulas such that

$$p_k = r,$$

$$p_0 \Rightarrow \text{wep}(R),$$

and for all i , $1 \leq i \leq k$,

$$p_i \Rightarrow \varphi(p_{i-1}, R);$$

that is, p_i is a pre-condition assuring the inevitability of R within i yieldings. Then,

$$p_i \Rightarrow \varphi_i \quad \text{for all } i, 0 \leq i \leq k;$$

that is

$$r \Rightarrow \varphi_k.$$

Since $\varphi_0 \sqsubseteq \varphi_1 \sqsubseteq \dots \sqsubseteq \varphi_k \sqsubseteq \dots$, the least upper bound exists and is the least fixed-point of φ . Hence $r \Rightarrow \mu q. \varphi(q, R)$. \square

Here we are pleased to acknowledge Allen Emerson, who proved a similar result for a different formulation of the fair-inevitability transformer.

Acknowledgments. We are indebted to Dana Scott for suggesting the notion of continuity from below and above as the solution to a problem contained in an earlier draft. Comments from Krzysztof Apt, Edmund Clarke, Allen Emerson, David Harel, Jim Morris, and Susan Owicki have been very valuable.

REFERENCES

- [1] J. W. DE BAKKER AND D. SCOTT, *A Theory of Programs*, unpublished manuscript, 1969.
- [2] S. K. BASU AND R. T. YEH, *Strong verification of programs*, IEEE Trans. on Soft. Eng., SE-1 (1975), pp. 339-345.
- [3] P. BRINCH HANSEN, *A comparison of two synchronizing concepts*, Acta Informatica, 1 (1972), pp. 190-199.
- [4] A. CHANDRA, *Computable nondeterministic functions*, Proc. of 19th Annual IEEE Symposium on Foundations of Computer Science, 1978, pp. 127-131.
- [5] E. M. CLARKE, *Program invariants as fixed points*, CS-1977-5, Department of Computer Science, Duke University, 1977.
- [6] S. A. COOK, *Soundness and completeness of an axiom system for program verification*, this Journal, 7 (1978), pp. 70-90.
- [7] E. W. DIJKSTRA, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1976.
- [8] A. EMERSON, private communication, 1979.
- [9] L. FLON, *On the Design and Verification of Operating Systems*, Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1977.
- [10] L. FLON AND A. N. HABERMANN, *Towards the construction of verifiable software systems*, Proc. of the ACM Conf. on Data: Abstraction, Definition and Structure, SIGPLAN Notices, 8 (1976), pp. 141-148.
- [11] L. FLON AND N. SUZUKI, *Nondeterminism and the correctness of parallel programs*, Proc. of IFIP Working Conf. on the Formal Description of Programming Concepts, E. Neuhold., ed., North-Holland, Amsterdam, 1978.
- [12] A. N. HABERMANN, *Synchronization of communicating processes*, Comm. ACM, 15 (1972), pp. 171-176.
- [13] D. HAREL, *Logic of Programs: Axiomatics and Descriptive Power*, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1978.
- [14] D. HAREL AND V. PRATT, *Nondeterminism in logics of programs*, Conf. Record of the Fifth Annual ACM Symp. on POPL, January, 1978.
- [15] P. HITCHCOCK AND D. PARK, *Induction rules and termination proofs*, Automata, Languages and Programming, M. Nivat, ed., IRIA, North-Holland, Amsterdam, 1973.

- [16] C. A. R. HOARE, *Monitors: an operating system structuring concept*, Comm. ACM, 17 (1974), pp. 549–557.
- [17] J. HOWARD, *Proving monitors*, Comm. ACM, 19 (1976), pp. 273–278.
- [18] R. KELLER, *Formal verification of parallel programs*, Comm. ACM, 19 (1976), pp. 371–384.
- [19] L. LAMPORT, *Proving the correctness of multiprocess programs*, IEEE Trans. on Soft. Eng., SE-3 (1977), pp. 125–143.
- [20] A. VAN LAMSWERDE AND M. SINTZOFF, *Formal derivation of strongly correct parallel programs*, Acta Informatica, 12 (1979), pp. 1–31.
- [21] S. OWICKI, *A consistent and complete deductive system for the verification of parallel programs*, Proc. of the Eighth Annual ACM Symposium on Theory of Computing, 1976, pp. 73–86.
- [22] S. OWICKI AND D. GRIES, *Verifying properties of parallel programs: an axiomatic approach*, Comm. ACM, 19 (1976), pp. 279–285.
- [23] D. PARK, *Fixpoint induction and proofs of program properties*, Machine Intelligence 5, Edinburgh University Press, Edinburgh, 1969, pp. 59–78.
- [24] W. DE ROEVER, *Dijkstra's predicate transformer, nondeterminism, recursion, and termination*, Proc. of Conf. on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Springer-Verlag, New York, 1976.
- [25] D. SCOTT, *Lattice of flow diagrams*, Symposium on Semantics of Algorithmic Languages, Lecture Notes in Mathematics 188, Springer-Verlag, New York, 1971.

AN ALGORITHM FOR FINDING K MINIMUM SPANNING TREES*

N. KATO[†], T. IBARAKI[‡] AND H. MINE[‡]

Abstract. This paper presents an algorithm for finding K minimum spanning trees in an undirected graph. The required time is $O(Km + \min(n^2, m \log \log n))$ and the space is $O(K + m)$, where n is the number of vertices and m is the number of edges. The algorithm is based on three subroutines. The first two subroutines are used to obtain the second minimum spanning tree in $O(\min(n^2, m\alpha(m, n)))$ steps, where $\alpha(m, n)$ is Tarjan's inverse of Ackermann's function [12] which is very slowly growing. The third one obtains the k th minimum spanning tree in $O(m)$ steps when the j th minimum spanning trees for $j = 1, 2, \dots, k-1$ are given.

Key words. k th minimum spanning tree, graph algorithm, computational complexity

1. Introduction. Let $G = (V, E)$ be an undirected connected graph with no parallel edges, where V is a set of n vertices and E is a set of m edges. The weight $w(e)$ is associated with each edge $e \in E$. The minimum spanning tree problem is to find the spanning tree T_1 with the minimum weight, where the weight of a spanning tree T (viewed as a set of edges) is defined by $w(T) = \sum_{e \in T} w(e)$. This problem has been intensively studied by many authors including Kruskal [9], Prim [11], Dijkstra [6], Yao [14], Cheriton and Tarjan [5]. Especially [5] and [14] require $O(m \log \log n)$ steps.

The j th minimum spanning tree T_j is defined recursively as follows.

- (1) T_1 is the minimum spanning tree.
- (2) T_j ($j \geq 2$) is a spanning tree with the minimum weight among those different from T_1, T_2, \dots, T_{j-1} .

Algorithms for finding K minimum spanning trees T_1, T_2, \dots, T_K have been studied by Burns and Haff [3], Camerini, Frata and Maffioli [4] and Gabow [7]. Gabow's algorithm requires $O(Km\alpha(m, n) + m \log n)$ steps and $O(K + m)$ space, where α is Tarjan's inverse of Ackermann's function [12] and is very slowly growing. We propose an algorithm with $O(Km + \min(m \log \log n, n^2))$ time and $O(K + m)$ space. This is slightly faster than Gabow's algorithm, and the required space is of the same order. The basic idea for enumeration is similar to but slightly different from Gabow's approach. This slight difference and the use of some additional lists make it possible to reduce the run time.

Section 2 gives the definition and a property of edge exchanges. Sections 3 and 4 give the outline and a detailed description of the entire algorithm. Section 5 analyzes time and space requirements. Sections 6 and 7 describe subroutines computing edge exchanges to generate the second and the j th minimum spanning trees, respectively. Although the algorithm explained in §§ 3 and 4 requires $O(Km)$ space, it is reduced to $O(K + m)$ in § 8.

2. T -exchanges. Let T be a spanning tree in G . A T -exchange is a pair of edges $[e, f]$ such that $e \in T, f \notin T$, and $T - e \cup f$ is a spanning tree. The weight of T -exchange $[e, f]$ is $w[e, f] = w(f) - w(e)$; note that the weight of tree $T - e \cup f$ is $w(T) + w[e, f]$.

LEMMA 2.1. [7] *A spanning tree T has minimum weight if and only if no T -exchanges have negative weight.*

* Received by the editors June 22, 1978, and in revised form April 11, 1980.

[†] Department of Medical Engineering, The Center for Adult Diseases, Osaka, Nakamichi Higashinariku, Osaka 537, Japan.

[‡] Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, Kyoto 606, Japan.

3. The outline of the algorithm. Our algorithm consists of routines GEN and GENK like Gabow's algorithm [7]. GEN computes T_j when T_1, T_2, \dots, T_{j-1} are given, using the branch-and-bound type technique described by Lawler [10]. Our GEN is different from Gabow's, however, in that a slightly different scheme is used to partition the solution space, and in that more information is stored in conjunction with solution space partition. GENK generates all the K minimum spanning trees using GEN as a subroutine.

The following lemma is a basis for our algorithm.

LEMMA 3.1. [7] *Let T be a minimum spanning tree satisfying the constraints $IN \subset T$ and $OUT \subset E - T$, where IN and OUT are given subsets of E . Then a minimum spanning tree which is different from T and satisfies the same constraint is given by $T - e \cup f$, where $[e, f]$ is a minimum T -exchange satisfying $e \in T - IN$ and $f \in E - T - OUT$.*

Now assume that the first $j - 1$ ($j > 1$) minimum spanning trees have been generated. The set of remaining spanning trees is partitioned into $j - 1$ disjoint sets

$$P_i^{j-1} = \{T_k | k > j - 1, IN_i \subset T_k, OUT_i \subset E - T_k\}, \quad i = 1, 2, \dots, j - 1,$$

where IN_i and OUT_i ($1 \leq i < j$) are set of edges which will be specified later (in a way slightly different from Gabow's definition). For $i = 1, 2, \dots, j - 1$, let

$$Q_i^{j-1} = \{([e, f], r) | \text{for each } f \in E - T_i - OUT_i, e \in T_i - IN_i$$

gives the minimum T_i -exchange $[e, f]$ with weight $r = w[e, f]\}$.

Note that each Q_i^{j-1} contains $|E - T_i - OUT_i| = O(m)$ labels therein.

Sets IN_i and OUT_i defining P_i^{j-1} ($1 \leq i \leq j - 1$) are given as follows. Initially, when $j = 2$ (i.e., only T_1 is obtained), IN_1 and OUT_1 defining P_1^{j-1} and Q_1^{j-1} are given by

$$IN_1 = \phi \quad \text{and} \quad OUT_1 = \phi.$$

In general, assume that T_j is obtained from T_{i^*} by applying T_{i^*} -exchange $[e^*, f^*]$. Then IN_i and OUT_i are updated as follows:

$$IN_{i^*} \leftarrow IN_{i^*}, \quad OUT_{i^*} \leftarrow OUT_{i^*} \cup \{f^*\},$$

$$IN_j \leftarrow IN_{i^*} \cup \{f^*\}, \quad OUT_j \leftarrow OUT_{i^*}.$$

Other IN_i and OUT_i do not change. These new sets define P_i^j for $i = 1, 2, \dots, j$, and GEN computes the corresponding Q_i^j . Recall here that Gabow [7] uses the scheme $IN_{i^*} \leftarrow IN_{i^*} \cup \{e^*\}$, $OUT_{i^*} \leftarrow OUT_{i^*}$; $IN_j \leftarrow IN_{i^*}$, $OUT_j \leftarrow OUT_{i^*} \cup \{e^*\}$. Our definition is essential to make the subsequent computation possible.

By this definition, the next lemma is obvious.

LEMMA 3.2. *Let j be $2 \leq j \leq K$.*

(1) *For any $i = 1, 2, \dots, j - 1$, T_i is a minimum spanning tree satisfying $IN_i \subset T_i$ and $OUT_i \subset E - T_i$, and no other T_k ($k = 1, 2, \dots, i - 1, i + 1, \dots, j - 1$) satisfies this constraint.*

(2) *Any spanning tree T satisfies $IN_i \subset T$ and $OUT_i \subset E - T$ for exactly one i with $1 \leq i \leq j - 1$.*

When T_1, T_2, \dots, T_{j-1} are known, Lemma 3.2(2) implies that T_j is given as a minimum spanning tree in $\cup_{i=1}^{j-1} P_i^{j-1}$ (note that P_i^{j-1} excludes T_1, T_2, \dots, T_{j-1}). By Lemma 3.1 and Lemma 3.2(2), a minimum spanning tree in P_i^{j-1} is given by $T_i - e' \cup f'$, where $([e', f'], r')$ is a label in Q_i^{j-1} with the smallest r . Thus, if we let $([e^*, f^*], r^*)$ be a label in $\cup_{i=1}^{j-1} Q_i^{j-1}$ with the smallest $w(T_i) + r$, T_j is given by

$$T_j = T_{i^*} - e^* \cup f^*.$$

Using these t^* , e^* and f^* , new sets P_i^j and Q_i^j are defined and the computation proceeds as explained above.

Now we describe how to compute Q_i^j ($1 \leq i \leq j$) in GEN. While computing T_1 , Q_1^1 is obtained in $O(\min(n^2, m\alpha(m, n)))$ steps by a special subroutine COMPQ1 or COMPQ2 depending upon whether $n^2 < m\alpha(m, n)$ or not. These subroutines are explained in § 6. When $T_j = T_{j^*} - e^* \cup f^*$ is obtained in GEN, $Q_{i^*}^j$ is computed by

$$Q_{i^*}^j = Q_{i^*}^{j-1} - \{([e^*, f^*], r^*)\}.$$

Q_i^j ($i \neq i^*, j$) are simply obtained by $Q_i^j = Q_i^{j-1}$. Finally Q_j^j is obtained by calling subroutine COMPQ3 explained in § 7. Obviously $|Q_i^j| = O(m)$ for all i . The key point is that $Q_{i^*}^j$ and Q_j^j are both obtained in $O(m)$ steps from $Q_{i^*}^{j-1}$. Based on these $Q_{i^*}^j$, minimum trees in P_i^j can also be obtained in $O(m)$ steps (note that only $P_{i^*}^j$ and P_j^j are considered since minimum spanning trees in $P_{i^*}^{j-1}$ and P_i^j do not change for $i \neq i^*, j$). GEN is repeated for $j = 2, 3, \dots, K$, and the entire procedure is organized as GENK.

Finally, we briefly explain the actual data structures of some of the above mentioned lists. Each set P_i^{j-1} is represented in our algorithm by a tuple

$$P_i^{j-1} = (t', [e', f'], A_i, IN_i, OUT_i, i),$$

where $([e', f'], r')$ is a label in Q_i^{j-1} with the smallest r , and $t' = w(T_i) + r' (= w(T_i - e' \cup f'))$. A_i is the adjacency list of T_i : $A_i = \{A_i(u) | u \in V\}$ and $A_i(u) = \{v | \text{edge}(u, v) \in T_i\}$. The length of P_i^{j-1} is $O(m)$ since $|A_i| = O(n)$ and $|IN_i| + |OUT_i| = O(m)$. In our implementation, more information is associated with $A_i(u)$; actually it consists of the following tuples:

$$A_i(u) = \{(v, w(u, v), INFLAG) | (u, v) \in T_i\},$$

where $INFLAG = 0$ implies $(u, v) \notin IN_i$ and $INFLAG = 1$ implies $(u, v) \in IN_i$. We also prepare an adjacency list of G : $A_G = \{A_G(u) | u \in V\}$, $A_G(u) = \{(v, w(u, v)) | (u, v) \in E\}$.

4. Algorithm for finding K minimum spanning trees. This section describes algorithms GENK and GEN in an ALGOL-like language.

Procedure GENK ($G = (V, E), K$); **begin**
comment $K \geq 2$;

- 1 Find the adjacency list A_1 for a minimum weight spanning tree T_1 and its weight t_1 ; **output** (A_1);
- 2 **if** $n^2 < m\alpha(m, n)$ **then** call *COMPQ1* to obtain Q_1^1
else call *COMPQ2* to obtain Q_1^1 ;
- 3 Find a minimum weight exchange $([e', f'], r')$ in Q_1^1 ;
- 4 Let P_1^1 be $(t_1 + r', [e', f'], A_1, \phi, \phi, 1)$;
- 5 **For** $j = 2$ **until** K **do** call *GEN* ($P_i^{j-1}, Q_i^{j-1} | i = 1, 2, \dots, j-1$);
end GENK;

Procedure GEN ($P_i^{j-1}, Q_i^{j-1} | i = 1, 2, \dots, j-1$); **begin**

- 1 Find $P_{i^*}^{j-1} = (t^*, [e^*, f^*], A_{i^*}, IN_{i^*}, OUT_{i^*}, i^*)$ with the smallest weight t' among $P_i^{j-1} = (t', [e', f'], A_i, IN_i, OUT_i, i)$, $i = 1, 2, \dots, j-1$;
- 2 **if** $t^* = \infty$ **then stop** (all spanning trees have been output and G has only $j-1$ spanning trees);
else begin
- 3 $A_j \leftarrow A_{i^*}$ with edge e^* replaced by f^* (A_j is the adjacency list of T_j);
output (A_j);
- 4 $Q_{i^*}^j \leftarrow Q_{i^*}^{j-1} - \{([e^*, f^*], t^* - t_{i^*})\}$;

```

5      Call COMPQ3( $A_j, IN_{i^*} \cup f^*, OUT_{i^*}, f^*, Q_{i^*}^{j-1}$ ) to obtain  $Q_j^i$ ;
6       $Q_i^i \leftarrow Q_{i^*}^{j-1}$  for  $i \neq i^*, j$ ;
7      if  $Q_{i^*}^i \neq \phi$  then  $P_{i^*}^i \leftarrow (t_{i^*} + r', [e', f'], A_{i^*}, IN_{i^*}, OUT_{i^*} \cup f^*, i^*)$ , where
           ( $[e', f'], r'$ ) is a label in  $Q_{i^*}^i$  with the minimum  $r$  and  $t_{i^*}$  (the weight of
            $T_{i^*}$ ) can be computed by  $t_{i^*} = t^* - w[e^*, f^*]$ 
           else  $P_{i^*}^i \leftarrow (\infty, \phi, \phi, \phi, \phi, i^*)$ ;
8      if  $Q_j^i \neq \phi$  then  $P_j^i \leftarrow (t^* + r'', [e'', f''], A_j, IN_{i^*} \cup f^*, OUT_{i^*}, j)$ , where
           ( $[e'', f''], r''$ ) is a label in  $Q_j^i$  with the minimum  $r$ 
           else  $P_j^i \leftarrow (\infty, \phi, \phi, \phi, \phi, i^*)$ ;
9       $P_i^i \leftarrow P_{i^*}^{j-1}$  for  $i \neq i^*, j$ ;
           end
       return
     end GEN

```

5. The correctness and the time bound of the algorithm.

LEMMA 5.1. For each $j = 2, 3, \dots, K$, GEN correctly computes T_j , Q_i^j and P_i^j ($i = 1, 2, \dots, j$) in $O(m)$ steps.

Proof. Since the correctness follows from the result of [7] and the discussion given so far, we consider the time requirement only. Line 1 of GEN finds $P_{i^*}^{j-1}$ with the minimum t among P_i^{j-1} , $i = 1, 2, \dots, j-1$. This is done in $O(\log(j-1)) \leq O(\log K) = O(m)$ steps (since the number of spanning trees $\leq 2^m$) if an appropriate sorting technique is used (e.g., heap sort [8]) for the set $\{P_i^{j-1} | i = 1, 2, \dots, j-1\}$. Line 2 requires constant steps. Line 3 is done in $O(n)$ ($\leq O(m)$) steps by $|A_{i^*}| = O(n)$. Line 4 requires $O(m)$ steps since $|Q_{i^*}^{j-1}| = O(m)$. Line 5 calls COMPQ3 and it requires $O(m)$, steps as will be shown in § 7. Lines 6 and 9 require constant steps because these are accomplished simply by keeping the previous data. Lines 7 and 8 are done in $O(m)$ steps by $|Q_{i^*}^i| = O(m)$ and $|Q_j^i| = O(m)$. (Adjustment of data structure of $\{P_i^i | i = 1, 2, \dots, j\}$ (e.g., using heap) is also done in $O(\log j) \leq O(m)$ steps, as is well known.) Thus, all computation in GEN is done in $O(m)$ steps. \square

THEOREM 5.2. GENK correctly generates the K minimum spanning trees from T_1 to T_K in $O(Km + \min(n^2, m \log \log n))$ time.

Proof. The correctness of GENK follows from the previous discussion. The time requirement is analyzed here. Line 1 requires $O(\min(n^2, m \log \log n))$ steps (e.g., [5], [14]). Line 2 requires $O(\min(n^2, m\alpha(m, n)))$ steps as shown in § 6. Line 3 requires $O(m)$ steps by $|Q_1^1| = O(m)$. Line 4 requires constant time. Line 5 calls GEN $K-1$ times, and requires $O(Km)$ steps in total by Lemma 5.1. Thus, the total time is as shown above. \square

A straightforward implementation of GENK requires $O(Km)$ space mainly to store Q_i^{j-1} and P_i^{j-1} for $i = 1, 2, \dots, j-1$. This will be reduced to $O(K+m)$ in § 8.

6. Subroutines COMPQ1 and COMPQ2. This section briefly explains the two subroutines COMPQ1 and COMPQ2, computing Q_1^1 in $O(n^2)$ steps and in $O(m\alpha(m, n))$ steps respectively. These are based on the next lemma.

LEMMA 6.1. For a given edge $f \in E - T_1$, let e be the maximum weight edge ($\neq f$) on the unique cycle formed by adding f to T_1 . Then $[e, f]$ is the smallest T_1 -exchange with the given $f \in E - T_1$.

Proof. The proof immediately follows since the weight of edge exchange $w[e, f]$ is given by $w(f) - w(e)$. \square

Q_1^1 is therefore computed by finding $[e, f]$ and $r = w[e, f]$ of Lemma 6.1 for every edge $f = (u, v) \in E - T_1$. COMPQ1 is first outlined. It is a slight augmentation of Prim's

algorithm [11] which computes T_1 in $O(n^2)$ time; we assume the reader's familiarity with Prim's algorithm. Consider a computation stage when an edge (x, v) is added to the current fragment (subtree which is going to comprise T_1), where x is a vertex in the fragment and v is not in the fragment. For each vertex u in the fragment, let (h, k) be the maximum weight edge in $u \xrightarrow{*}_{T_1} x$ (which has already been computed and stored). Then a maximum weight edge in $u \xrightarrow{*}_{T_1} v$ for $f = (x, v)$ is obtained by taking the edge with the larger weight between (x, v) and (h, k) . (This property easily follows from Lemma 6.1 and is omitted.) Thus computation of such maximum weight edges for all (u, v) , such that u 's are vertices in the fragment is done in $O(n)$ time. Since this can be repeated $n - 1$ times until T_1 is constructed by Prim's algorithm, computing at the same time the maximum weight edge for every $f = (u, v) \in E$, the total time to compute Q_1^1 is $O(n^2)$.

THEOREM 6.1. *COMPQ1 computes Q_1^1 in $O(n^2)$ time. \square*

COMPQ2 is a straightforward adaptation of Tarjan's algorithm [13] for verifying in $O(m\alpha(m, n))$ time that a tree T in an undirected graph is a minimum spanning tree. His algorithm involves the computation of the maximum weight edge along the path $u \xrightarrow{*}_T v$, for each edge $f = (u, v) \notin T$. By Lemma 6.1, this portion of his algorithm can be directly used as COMPQ2 for computing Q_1^1 .

THEOREM 6.2. *COMPQ2 computes Q_1^1 in $O(m\alpha(m, n))$ time. \square*

7. Subroutine COMPQ3. This section describes subroutine COMPQ3($A_j, IN_j, OUT_j, f^*, Q_j^{i-1}$) for obtaining Q_j^i in $O(m)$ steps when $T_j = T_{i^*} - e^* \cup f^*$ is given, where $[e^*, f^*]$ is the minimum T_{i^*} -exchange in $\cup_{i=1}^{i-1} Q_j^{i-1}$ (see § 3). COMPQ3 is based on the following lemma.

LEMMA 7.1. *For T_j and the edge $f^* = (u^*, v^*) \in T_j$ defined above, let $T_j(u^*)$ and $T_j(v^*)$ be two trees obtained from T_j by deleting f^* , where $u^* \in V_j(u^*)$ and $v^* \in V_j(v^*)$. Here $V_j(x)$ is the set of vertices in the connected component $T_j(x)$. Let $f = (u, v)$ be an edge in $E - T_j - OUT_j$.*

- (1) *If $u, v \in V_j(u^*)$ or $u, v \in V_j(v^*)$, the label $([e, f], r)$ stored in Q_j^{i-1} is also in Q_j^i .*
- (2) *If $u \in V_j(u^*)$ and $v \in V_j(v^*)$, then $([e, f], r)$ stored in Q_j^i is determined by*

$$w(e) = \max \left[\max \left\{ w(g) \mid g \notin IN_j, g \text{ is on } u^* \xrightarrow{*}_{T_j(u^*)} u \right\}, \right. \\ \left. \max \left\{ w(h) \mid h \notin IN_j, h \text{ is on } v^* \xrightarrow{*}_{T_j(v^*)} v \right\} \right],$$

$$r = w(f) - w(e).$$

Proof.

(1) Assume $u, v \in T_j(u^*)$ without loss of generality (see Fig. 1). Since $T_j - f^* = T_{i^*} - e^*$ (i.e., $T_j(u^*) = T_{i^*}(u^*)$), then $u \xrightarrow{*}_{T_j} v = u \xrightarrow{*}_{T_{i^*}} v$, and f^* is not an edge on $u \xrightarrow{*}_{T_j} v$. Thus, by Lemma 6.1 the label $([e, f], r)$ stored in Q_j^{i-1} is also in Q_j^i .

(2) Since $u \xrightarrow{*}_{T_j} v$ is equal to $u \xrightarrow{*}_{T_j(u^*)} u^* \rightarrow v^* \xrightarrow{*}_{T_j(v^*)} v$ and $(u^*, v^*) (= f^*) \in IN_j$ (see Fig. 2), the edge e defining $([e, f], r) \in Q_j^i$ is the maximum weight edge $\notin IN_j$ on either $u \xrightarrow{*}_{T_j(u^*)} u^*$ or $v^* \xrightarrow{*}_{T_j(v^*)} v$ by Lemma 6.1. \square

To compute $([e, f], r) \in Q_j^i$ efficiently by Lemma 7.1, COMPQ3 preprocesses trees $T_j(u^*)$ and $T_j(v^*)$ by calling subroutines EDGEFIND(A'_j, u^*, IN_j) and EDGEFIND(A'_j, v^*, IN_j), where A'_j is the adjacency list of $T_j(u^*) \cup T_j(v^*)$.

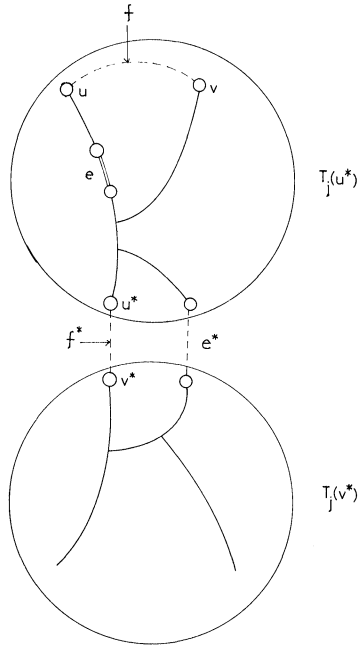


FIG. 1. Illustration of T_j -exchange in the proof of case (1) of Lemma 7.1.

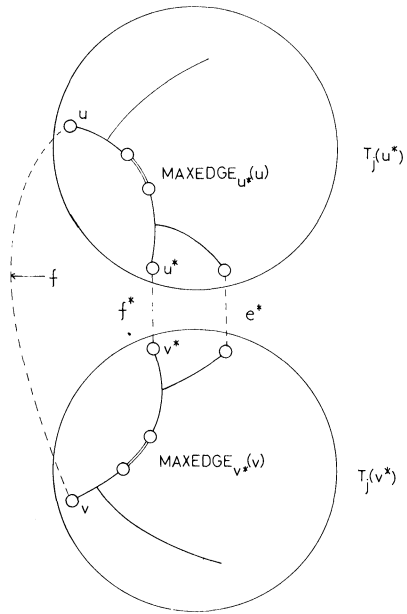


FIG. 2. Illustration of T_j -exchange in the proof of case (2) of Lemma 7.1.

EDGEFIND(A'_j, u^*, IN_j) finds the maximum weight edge $\text{MAXEDGE}_{u^*}(u) \in T_j(u^*) - IN_j$ on $u^* \xrightarrow{T_j(u^*)} u$ for each vertex $u \in T_j(u^*)$. Its weight is stored in $W_{u^*}(u)$.

(Subscript u^* is added to indicate MAXEDGE and W obtained by $\text{EDGEFIND}(A'_j, u^*, IN_j)$.) $\text{EDGEFIND}(A'_j, v^*, IN_j)$ is similar. After this, $([\text{MAXEDGE}_{w^*}(z), f], w(f) - W_{w^*}(z))$ is added to Q_j^i for each edge $f = (u, v) \in E - T_j - \text{OUT}_j$ with $u \in V_j(u^*)$ and $v \in V_j(v^*)$, where $W_{w^*}(z) = \max\{W_{u^*}(u), W_{v^*}(v)\}$.

For each $f = (u, v) \in E - T_j - \text{OUT}_j$ with $u, v \in V_j(u^*)$ or $u, v \in V_j(v^*)$, $([e, f], r)$ in Q_j^{i-1} is directly stored in Q_j^i .

Procedure COMPQ3($A_j, IN_j, \text{OUT}_j, f^* = (u^*, v^*), Q_j^{i-1}$); begin
 1 $Q_j^i \leftarrow \phi$; $A'_j(u^*) \leftarrow A_j(u^*) - \{v^*\}$; $A'_j(v^*) \leftarrow A_j(v^*) - \{u^*\}$; $A'_j(u) \leftarrow A_j(u)$ for $u \neq u^*, v^*$ ($A'_j = \{A'_j(u) | u \in V\}$ is the adjacency list of $T_j(u^*) \cup T_j(v^*)$);
 2 **for** $u \in V$ **do**
 3 **if** $u \in V_j(u^*)$ **then** $N(u) \leftarrow 1$ **else** $N(u) \leftarrow 0$ ($N(u)$ is a flag showing whether $u \in V_j(u^*)$ or $u \in V_j(v^*)$);
 4 **for** $u \in V$ **do** $W_{u^*}(u) \leftarrow W_{v^*}(u) \leftarrow -\infty$; $\text{MAXEDGE}_{u^*}(u) \leftarrow \text{MAXEDGE}_{v^*}(u) \leftarrow \phi$;
 5 Call $\text{EDGEFIND}(A'_j, u^*, IN_j)$ to obtain $\text{MAXEDGE}_{u^*}(u)$ and $W_{u^*}(u)$ for $u \in V_j(u^*)$;
 6 Call $\text{EDGEFIND}(A'_j, v^*, IN_j)$ to obtain $\text{MAXEDGE}_{v^*}(v)$ and $W_{v^*}(v)$ for $v \in V_j(v^*)$;
 7 **for** $f = (u, v) \in E - T_j - \text{OUT}_j$ **do**
 8 **if** $N(u) = N(v)$ **then** add label $([e, f], r)$ in Q_j^{i-1} to Q_j^i
 9 **else** add label $([\text{MAXEDGE}_{w^*}(z), f], w(f) - W_{w^*}(z))$ to Q_j^i , where $z \in \{u, v\}$ satisfies $W_{w^*}(z) = \max\{W_{u^*}(u), W_{v^*}(v)\}$ (if $W_{w^*}(z) = \infty$, do not add the label since $\text{MAXEDGE}_{w^*}(z)$ does not exist);
return
end COMPQ3;

Procedure EDGEFIND(A'_j, p^*, IN); begin
 Call $\text{DFS}(A'_j, \phi, p^*, IN)$;
return
end EDGEFIND;

Procedure DFS(A'_j, x, y, IN); begin
 1 **for** $z \in A'_j(y) - x (= A'_j(y)$ if $x = \phi$) and $(y, z) \in IN$ **do**
 2 **if** $W(y) < w(y, z)$ **then begin**
 $\text{MAXEDGE}(z) \leftarrow (y, z)$; $W(z) \leftarrow w(y, z)$;
 end
 3 **else begin**
 $\text{MAXEDGE}(z) \leftarrow \text{MAXEDGE}(y)$; $W(z) \leftarrow W(y)$;
 end
 4 Call $\text{DFS}(A'_j, y, z, IN)$;
return
end DFS

THEOREM 7.2. $\text{COMPQ3}(A_j, IN_j, \text{OUT}_j, f^*, Q_j^{i-1})$ correctly computes Q_j^i in $O(m)$ steps.

Proof. It is obvious that $A'_j = \{A'_j(u) | u \in V\}$ obtained from A_j at line 1, is the adjacency list of $T_j(u^*) \cup T_j(v^*)$. Thus, $\text{EDGEFIND}(A'_j, u^*, IN_j)$ and $\text{EDGEFIND}(A'_j, v^*, IN_j)$ correctly compute $\text{MAXEDGE}_{u^*}(u)$, $W_{u^*}(u)$ for all $u \in V_j(u^*)$ and $\text{MAXEDGE}_{v^*}(v)$, $W_{v^*}(v)$ for all $v \in V_j(v^*)$. Thus, lines 7–9 correctly compute Q_j^i by Lemma 7.1. Next, we analyze the time requirement. Line 1 requires

$O(n)$ steps since $|A_j| = O(n)$. Lines 2 and 3 are done in $O(n)$ steps by computing $V_j(u^*)$ and $V_j(v^*)$ using A_j , and associating flag $N(u)$ to $u \in V$ by using A'_j . Line 4 is also done in $O(n)$ steps. Lines 5 and 6 require $O(n)$ steps, by $|V_j(u^*)| = O(n)$, $|V_j(v^*)| = O(n)$ and $|A_j| = O(n)$. To execute lines 7–9 in constant time for each $f = (u, v) \in E - T_j - OUT_j$, note that

$$E - T_j - OUT_j = \{f | ([e, f], r) \in Q_i^{j-1}\} - f^* \cup e^*$$

holds. Furthermore, $N(u) \neq N(v)$ holds for $e^* = (u, v)$. Thus, adding label $([e, f], r)$ in Q_i^{j-1} to Q_j^i of line 8 is done in constant time if $f(\neq e^*)$ of line 7 is directly taken from Q_i^{j-1} (i.e., constant time is required to search $([e, f], r)$ in Q_i^{j-1} of line 8). Line 8, obviously, requires constant time. Thus, lines 7–9 require $O(m)$ steps in total by $|E| = m$. \square

8. Space reduction. The required space for GENK is reduced to $O(K + m)$ in this section, although GENK, explained in § 4, requires $O(Km)$ space to store P_i^{j-1} and Q_i^{j-1} ($i = 1, 2, \dots, j - 1$) (space required for other data is obviously $O(m)$).

First, in order to reduce the space requirement of P_i^{j-1} from $O(Km)$ to $O(K + m)$ we modify the data structure representing P_i^{j-1} in almost the same way as done by Gabow [7]. Namely, the data structure of P_1^{j-1} is the same as the one discussed in § 3 (P_1^{j-1} requires $O(m)$ space). P_i^{j-1} ($i > 1$) are modified to

$$P_i^{j-1} = (t', [e', f'], [e^*(i), f^*(i)], i^*(i), b(i), s(i), i), \quad i = 2, 3, \dots, j - 1$$

(thus, P_i^{j-1} ($i = 2, 3, \dots, j - 1$) require $O(j) \leq O(K)$ space), where $t', [e', f']$ are defined in § 3, and T_i is obtained from $T_{i^*(i)}$ by $T_{i^*(i)}$ -exchange $[e^*(i), f^*(i)]$. The derivation of T_2, T_3, \dots from T_1 is represented by a rooted tree; T_{i^*} is a father of T_i (or T_i is a son of T_{i^*}) if T_i is derived from T_{i^*} by a T_{i^*} -exchange, and T_i and T_k are brothers if their fathers coincide. T_i is placed to the left of its brother T_k if $i < k$. $b(i)$ and $s(i)$ in P_i^{j-1} denote the brother $T_{b(i)}$ immediately to the left of T_i ($b(i) = 0$ if T_i is the leftmost brother) and the rightmost son $T_{s(i)}$. Obviously, T_1 is the root of this tree. Based on the new lists, T_i, IN_i and OUT_i can be constructed in $O(m)$ time by following the path from T_i up to root T_1 . This technique is almost the same as Gabow's, and hence the details are omitted. The rest of the computation is then applied to the reconstructed P_i^{j-1} .

In order to reduce the space requirement of Q_i^{j-1} ($i = 1, 2, \dots, j - 1$) to $O(K)$ we execute the following cleanup step from time to time. Note that each T_i -exchange $([e, f], r) \in Q_i^{j-1}$ induces a spanning tree $T_i - e \cup f$ with weight $w(T_i) + r$. However, only $K - (j - 1)$ smallest spanning trees induced from $\cup_{i=1}^{j-1} Q_i^{j-1}$ are necessary to compute T_j, T_{j+1}, \dots, T_K , as justified below. Therefore, the cleanup step removes all $([e, f], r)$'s from $\cup_{i=1}^{j-1} Q_i^{j-1}$, except those with $K - (j - 1)$ smallest $w(T_i) + r$'s. The cleanup step is done in $O(K')$ steps, where $K' = |\cup_{i=1}^{j-1} Q_i^{j-1}|$, by finding the $K - (j - 1)$ th smallest element in $O(K')$ steps by the fast algorithm [2], and then removing all $([e, f], r)$'s with larger $(w(T_i) + r)$'s.

The cleanup step is justified as follows. Suppose that a T_i -exchange $[e, f]$ corresponding to $f \in E - T_i - OUT_i$ is removed from Q_i^{j-1} by a cleanup step. Later, a T_j may be obtained from T_i by $T_i - e^* \cup f^*$, and Q_j^j is computed from Q_i^{j-1} by COMQ3. Note that Q_i^{j-1} is obtained from Q_i^{j-1} and therefore Q_j^j does not contain the minimum T_j -exchange $[e', f]$ corresponding to the removed f . At this point, it is necessary to show that such $[e', f]$ in Q_j^j can be ignored for the rest of the computation. However, this is obvious because $w(T_i - e \cup f) \leq w(T_j - e' \cup f)$ can be easily proved and hence $T_j - e' \cup f$ is not a member of the K minimum spanning trees.

Now execute the cleanup step whenever $K' > 2K$ is satisfied. Then $K' = O(K)$ always holds. Since K' increases at most by m at every iteration, the cleanup step is necessary at every $\lceil K/m \rceil$ iterations, where $\lceil x \rceil$ denotes the smallest integer not smaller than x . Hence, the cleanup step is executed $O(K) \cdot O(m/K) = O(m)$ times before the entire computation is completed. Therefore, $O(Km)$ steps are required for the cleanup computation, but the time bound of the entire algorithm does not change.

Consequently, we have the next theorem.

THEOREM 8.1. GENK requires $O(Km + \min(n^2, m \log \log n))$ time and $O(K + m)$ space.

Acknowledgment. The authors would like to thank Professor H. N. Gabow of Colorado University for his helpful comments, especially for pointing out that Q_1^1 can be obtained in $O(m\alpha(m, n))$ time (in addition to mentioning Tarjan's result [13], he also proposed a new way of accomplishing this time bound, which is not included here).

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. BLUM ET AL., *Time bounds for selection*, J. Comput. System Sci., 7 (1973), pp. 448–461.
- [3] R. N. BURNS AND C. E. HAFF, *A ranking problem in graphs*, Proceedings of the 5th Southeast Conference on Combinatorics, Graph Theory and Computing, 19 (1974), pp. 461–470.
- [4] P. M. CAMERINI, L. FRATTA AND F. MAFFIOLI, *The K shortest spanning trees of a graph*, Int. Rep. 73–10, IEEE-LCE Politechnico di Milano, Italy, 1974.
- [5] D. CHERITON AND R. E. TARJAN, *Finding minimum spanning trees*, this Journal, 5 (1976), pp. 724–742.
- [6] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.
- [7] H. N. GABOW, *Two algorithms for generating weighted spanning trees in order*, this Journal, 6 (1977), pp. 139–150.
- [8] D. E. KNUTH, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA., 1973.
- [9] J. KRUSKAL, *On the shortest spanning subtree of a graph and the travelling salesman problem*, Proc. Amer. Math. Soc., 2 (1956), pp. 48–50.
- [10] E. L. LAWLER, *A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem*, Management Sci., 18 (1972), pp. 401–405.
- [11] R. C. PRIM, *Shortest connection networks and some generalizations*, Bell System Tech. J., 36 (1957), pp. 1389–1401.
- [12] R. E. TARJAN, *Efficiency of a good but not linear set union algorithm*, J. Assoc. Comp. Mach., 22 (1975), pp. 215–225.
- [13] ———, *Applications of path compression on balanced trees*, J. Assoc. Comp. Mach., 26 (1979), pp. 690–715.
- [14] A. C. YAO, *An $O(|E| \log \log |V|)$ algorithm for finding minimum spanning trees*, Inform. Process. Lett., 4 (1975), pp. 21–23.

SCHEDULING UNIT-TIME TASKS WITH ARBITRARY RELEASE TIMES AND DEADLINES*

M. R. GAREY†, D. S. JOHNSON†, B. B. SIMONS‡ AND R. E. TARJAN§

Abstract. The basic problem considered is that of scheduling n unit-time tasks, with arbitrary release times and deadlines, so as to minimize the maximum task completion time. Previous work has shown that this problem can be solved rather easily when all release times are integers. We are concerned with the general case in which noninteger release times are allowed, a generalization that considerably increases the difficulty of the problem even for only a single processor. Our results are for the one-processor case, where we provide an $O(n \log n)$ algorithm based on the concept of "forbidden regions".

Key words. scheduling, release time, deadline, computational complexity

1. Introduction. The scheduling problems we will be considering in this paper are all special cases of the following general scheduling problem. We are given n tasks, T_1, T_2, \dots, T_n , each requiring one unit of execution time. Each task T_i has associated with it an arbitrary release time $r_i \geq 0$ and a deadline $d_i \geq r_i + 1$. In addition, there may be a partial order $<$ imposed on the tasks. We wish to schedule the given tasks nonpreemptively on m identical processors so that

- (i) Each task T_i is started no earlier than its release time r_i and is completed no later than its deadline d_i .
- (ii) Whenever $T_i < T_j$, T_j does not start before T_i has been completed.
- (iii) The maximum completion time (or *makespan*) is minimized.

Previous results on related problems include the following. If the tasks are allowed to have unequal lengths, a simple transformation from the 3-PARTITION problem [4] shows that the problem is NP-complete in the strong sense [4], even for one processor and integer release times and deadlines. If the partial order is allowed to be arbitrary, then the problem with unit-time tasks and a variable number of processors is NP-complete [10], even if all release times are 0 and there is only a single overall task deadline. On the other hand, good algorithms are known for the following special cases: If no partial order is imposed and the release times are all integers, then the "earliest deadline scheduling rule" [5], [7] can be used to solve the problem in $O(n \log n)$ time for any number of processors. Indeed, this method can be used for one processor even with an arbitrary partial order, since (as we observe in § 2) the presence of a partial order is essentially irrelevant to the one-processor case. For two processors, integer release times and an arbitrary partial order, an $O(n^3 \log n)$ algorithm is given in [3].

As observed in [3], these problems seem to be considerably more difficult when the release times are not required to be integers (i.e., are not multiples of the common task length), even when there is only a single processor. In this paper we shall consider the version in which *arbitrary* release times are allowed, concentrating on the one-processor case. The first polynomial time algorithm for this problem was obtained recently by Simons [8], and it has time complexity $O(n^2 \log n)$. An alternative

* Received by the editors July 19, 1979, and in final form May 5, 1980.

† Bell Laboratories, Murray Hill, New Jersey 07974.

‡ Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California; now at IBM, San Jose, California. Much of the work of this author was done while she was a graduate student at University of California, Berkeley, and partially supported by the National Science Foundation under grant MCS 77-09906.

§ Computer Science Department, Stanford University, Stanford, California 94305. The work of this author was supported in part by the National Science Foundation under grant MCS-7826858 and by the U.S. Office of Naval Research under contract N00014-76-C-0330. Some of this research was done while the author was visiting Bell Laboratories.

algorithm with the same time complexity has since been obtained by Carlier [2]. Using the new concept of “forbidden regions”, we shall describe an algorithm which, when suitably modified to use appropriate data structures, runs in time $O(n \log n)$ and space $O(n)$.

The paper is divided into five sections. In § 2 we make some simple preliminary observations, including a “normalization” lemma and a lemma showing that partial orders are essentially irrelevant when there is only one processor. § 3 gives an $O(n^2)$ algorithm for the one processor problem based on “forbidden regions”, and § 4 improves the algorithm to $O(n \log n)$ through the careful use of appropriate data structures and several new ideas. Finally, § 5 concludes the paper by mentioning several problems that remain open, particularly with regard to the multi-processor case.

2. Preliminary observations. We shall represent a schedule (or a partial schedule) by giving a starting time s_i for each task T_i . Sometimes we will use $f_i = s_i + 1$ to denote the finishing time for T_i . A schedule is *feasible* if it satisfies the release times and deadlines (i.e., $r_i \leq s_i \leq d_i - 1$ for all i), obeys the partial order constraints (i.e., $T_i < T_j$ implies $f_i \leq s_j$), and executes at most m tasks at a time (i.e., for any time t , there are at most m tasks T_i for which t belongs to the execution interval $[s_i, f_i)$).

A task T_i is *ready* at time t if $r_i \leq t$. We shall say that a schedule is *normal* if, for any two tasks T_i and T_j , $s_i < s_j$ implies that either $d_i \leq d_j$ or $r_j > s_i$. In other words, a normal schedule has the property that, whenever one or more tasks begin execution at some time t , those tasks have the earliest deadlines among all remaining tasks that are ready at t . The following lemma can be proved by straightforward interchange arguments.

LEMMA 1. *For any $m \geq 1$, if there are no partial order constraints, then the existence of a feasible m -processor schedule implies the existence of a schedule that both minimizes maximum completion time and is normal.*

Lemma 1 tells us that in the absence of a partial order we can restrict our attention to normal schedules. The next lemma will show how, for $m = 1$, we can restrict ourselves to normal schedules even in the presence of a partial order.

Given a partial order $<$ on the tasks, we say the release times and deadlines are *consistent* with the partial order if $T_i < T_j$ implies $r_i + 1 \leq r_j$ and $d_i \leq d_j - 1$. We can make release times and deadlines consistent with the partial order by processing the tasks once in topological order [6] assigning $r_j \leftarrow \max(\{r_j\} \cup \{r_i + 1 : T_i < T_j\})$ and once in reverse topological order assigning $d_i \leftarrow \min(\{d_i\} \cup \{d_j - 1 : T_i < T_j\})$. This requires time linear in the size of the partial order and does not alter the feasibility of any schedule. Furthermore, in the one-processor case it allows us subsequently to ignore the partial order constraints. \square

LEMMA 2. *If the release times and deadlines are consistent with a partial order, then any normal one-processor schedule that satisfies the release times and deadlines must also obey the partial order.*

Proof. Consider any normal one-processor schedule, and suppose that $T_i < T_j$ but that $s_j < f_i$ (which, since there is only one processor, implies $s_j < s_i$). By the consistency assumption we have $r_i < r_j$ and $d_i < d_j$. However, these, together with $s_j < f_i$, cause a violation of the assumption that the schedule is normal, a contradiction from which the result follows. \square

Lemma 2 means that a partial order is essentially irrelevant when scheduling on one processor. Henceforth we shall assume that no partial order is imposed, and we will consider only normal schedules.

3. One-processor scheduling using forbidden regions. Our one-processor scheduling algorithms depend upon discovering “forbidden regions”. A *forbidden region* is an

interval of time (open both on the left and right) during which *no* task can start if the schedule is to be feasible.

The following algorithm forms a basic building block of our main algorithm. Suppose we are given k unit-time tasks, all of which must be scheduled to finish by some time d , and a finite collection of forbidden regions F_j . Ignoring the individual release times and deadlines of the tasks, we would like to find the latest time by which the first such task must start if all of them are to be completed by time d (without starting any of them in a forbidden region).

We do this using the following naive algorithm: Order the tasks arbitrarily as T_1, T_2, \dots, T_k and schedule them from the back of the schedule in order of *decreasing* index. When scheduling task T_i , start it at the latest time less than or equal to $s_{i+1} - 1$ (or $d - 1$, if $i = k$) which does not fall in a forbidden region. We call this the *Backscheduling Algorithm*.

LEMMA 3. *The starting times s_1 found for T_1 by the Backscheduling Algorithm is such that, if all the given tasks were to start at times strictly greater than s_1 , with none of them starting in one of the given forbidden regions, then at least one of them would not be completed by time d .*

Proof. Consider the schedule found by the Backscheduling Algorithm. Let $h_0 = s_1$, let h_1, h_2, \dots, h_j be the starting times of the idle periods (if any) in the schedule and let $h_{j+1} = d$. See Fig. 1.

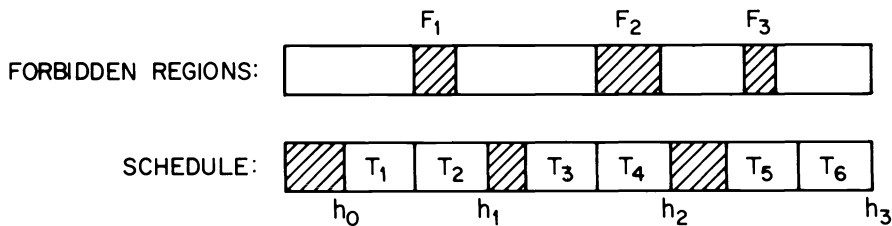


FIG. 1. *Scheduling by the Backscheduling Algorithm to avoid forbidden regions.*

Notice that whenever (t_1, t_2) is an idle period, it must be the case that $(t_1 - 1, t_2 - 1]$ is part of some forbidden region, for otherwise the Backscheduling Algorithm would have scheduled some task to overlap or finish during $(t_1, t_2]$. Now consider any interval $(h_i, h_{i+1}]$, $0 \leq i \leq j$. By the preceding observation, no task can possibly be scheduled to finish after h_i but before (or at) the starting time of the first task in the interval. Furthermore, by definition of the $\{h_i\}$, the tasks that are finished in the interval are scheduled with no idle periods separating them and with the rightmost one finishing at time h_{i+1} . It follows that the Backscheduling Algorithm finishes the *maximum* possible number of tasks in each interval $(h_i, h_{i+1}]$. Since there is no idle time in the schedule during $[h_0, h_1]$, any other schedule that started all the tasks later than time s_1 and finished them all by time d would have to exceed this maximum number of tasks in some interval $(h_i, h_{i+1}]$, $1 \leq i \leq j$, a contradiction. \square

We shall use the Backscheduling Algorithm as follows. Consider any task ready time r_i and any task deadline $d_j \geq d_i$. Suppose that we have already found a collection of forbidden regions in the interval $[r_i, d_j]$ and that we then apply the Backscheduling Algorithm, with $d = d_j$ and with the forbidden regions we have already found, to the set of all tasks T_k satisfying $r_i \leq r_k \leq d_k \leq d_j$. Let s be the latest possible start time found by the Backscheduling Algorithm in this case. There are two possibilities which are of

interest. First, if $s < r_i$, then we know that there can be *no* feasible schedule, since all these tasks must be completed by time d , none of them can be started before r_i , but at least one *must* be started by time $s < r_i$ if all are to be completed by d . Second, if $r_i \leq s < r_i + 1$, then we know that $(s - 1, r_i)$ can be declared to be a forbidden region, since any task started in that region would not belong to our set (its release time is less than r_i) and it would force the first task of our set to be started later than s , thus preventing these tasks from being completed by d .

Our first algorithm for the one-processor problem essentially applies the Back-scheduling Algorithm to *all* such pairs of release times and deadlines, in such a manner as to find forbidden regions from right to left. We do this by processing the release times in order from largest to smallest. To process a release time r_i , we determine for each deadline $d_j \geq d_i$ the number of tasks which cannot start before r_i and which must be completed by d_j . We then use the Backscheduling Algorithm with $d = d_j$ to determine the latest time at which the earliest such task can start. This time is called the *critical time* c_j for deadline d_j (with respect to r_i). Letting c denote the minimum of all these critical times with respect to r_i , we then declare failure if $c < r_i$ or declare $(c - 1, r_i)$ to be a forbidden region if $r_i \leq c < r_i + 1$. Notice that by processing release times from largest to smallest, all forbidden regions to the right of r_i will have been found by the time that r_i is processed. In order to make the entire process more efficient, we do not completely recompute the critical time for a deadline when a new release time is processed, but instead we update the old value.

Once we have found forbidden regions in this way, we schedule the full set of tasks forward from time 0 using the “earliest deadline scheduling rule”. This proceeds by initially setting t to the least nonnegative time not in a forbidden region and then assigning start time t to a task with lowest deadline among those ready at t . At each subsequent step, we first update t to the least time which is greater than or equal to the finishing time of the last scheduled task, greater than or equal to the earliest ready time of an unscheduled task, and which does not fall in a forbidden region, and we then assign start time t to a task with lowest deadline among those ready (but not previously scheduled) at t . The entire algorithm is specified below.

ALGORITHM A. Index the tasks (arbitrarily) so that $r_1 \leq r_2 \leq \dots \leq r_n$.

Part I. (Forbidden Region Declaration). Initially no forbidden regions have been declared. For each task T_i , in order of decreasing index, perform the following two steps:

- Step 1.* For each task T_j with $d_j \geq d_i$, update its critical time c_j as follows:
 - 1a. If c_j is undefined, set $c_j \leftarrow d_j - 1$; otherwise set $c_j \leftarrow c_j - 1$.
 - 1b. While $c_j \in F$ for some declared forbidden region F , set $c_j \leftarrow \inf(F)$.
- Step 2.* If $i = 1$ or $r_{i-1} < r_i$, set $c \leftarrow \min\{c_j : c_j \text{ is defined}\}$ and proceed as follows:
 - 2a. If $c < r_i$, declare failure and halt.
 - 2b. If $r_i \leq c < r_i + 1$, declare $(c - 1, r_i)$ to be a forbidden region.

Part II. (Schedule Generation). Initially no tasks are scheduled and $t = 0$. Repeat the following three steps until all tasks have been scheduled.

- Step 1.* If no unscheduled task is ready at time t , set $t \leftarrow \min\{r_i : T_i \text{ has not yet been scheduled}\}$.
- Step 2.* While $t \in F$ for some forbidden region F , set $t \leftarrow \sup(F)$.
- Step 3.* Select an unscheduled task T_j that has the least deadline among all such tasks that are ready at t . Set $s_j \leftarrow t$ and set $t \leftarrow t + 1$.

Fig. 2 illustrates the application of this algorithm to the tasks described by Table 1.

THEOREM 1. *In any feasible schedule, no task starts at a time that Algorithm A declares to be forbidden. If Algorithm A declares failure, then there is no feasible schedule.*

RELEASE TIMES

	9	$8\frac{2}{3}$	$8\frac{1}{3}$	5	$4\frac{2}{3}$	$4\frac{1}{3}$	$3\frac{1}{2}$	$1\frac{2}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	0
--	---	----------------	----------------	---	----------------	----------------	----------------	----------------	---------------	---------------	---

DEADLINES	$5\frac{2}{3}$								$4\frac{2}{3}$	$4\frac{2}{3}$	$4\frac{2}{3}$
6								5	$3\frac{2}{3}$	$3\frac{2}{3}$	$3\frac{2}{3}$
$6\frac{1}{3}$					$5\frac{1}{3}$	$5\frac{1}{3}$	$5\frac{1}{3}$	$4\frac{1}{3}$	$2\frac{2}{3}$	$2\frac{2}{3}$	$2\frac{2}{3}$
$6\frac{2}{3}$					$5\frac{2}{3}$	$4\frac{2}{3}$	$4\frac{2}{3}$	$3\frac{2}{3}$	$2\frac{2}{3}$	$2\frac{2}{3}$	$2\frac{2}{3}$
$7\frac{2}{3}$					$6\frac{2}{3}$	$5\frac{2}{3}$	$4\frac{2}{3}$	$3\frac{2}{3}$	$2\frac{2}{3}$	$2\frac{2}{3}$	$2\frac{2}{3}$
8				7	6	5	$3\frac{2}{3}$	$2\frac{2}{3}$	$1\frac{2}{3}$	$1\frac{2}{3}$	$1\frac{2}{3}$
10				9	$7\frac{1}{3}$	$6\frac{1}{3}$	$5\frac{1}{3}$	$4\frac{1}{3}$	$2\frac{2}{3}$	$1\frac{2}{3}$	$1\frac{2}{3}$
$10\frac{1}{3}$	$9\frac{1}{3}$	$9\frac{1}{3}$	$9\frac{1}{3}$	$8\frac{1}{3}$	$7\frac{1}{3}$	$6\frac{1}{3}$	$5\frac{1}{3}$	$4\frac{1}{3}$	$2\frac{2}{3}$	$1\frac{2}{3}$	$1\frac{2}{3}$
$11\frac{1}{3}$	$10\frac{1}{3}$	$9\frac{1}{3}$	$8\frac{1}{3}$	$7\frac{1}{3}$	$6\frac{1}{3}$	$5\frac{1}{3}$	$4\frac{1}{3}$	$2\frac{2}{3}$	$1\frac{2}{3}$	$\frac{2}{3}$	$\frac{2}{3}$
$12\frac{1}{3}$	$11\frac{1}{3}$	$10\frac{1}{3}$	$9\frac{1}{3}$	$8\frac{1}{3}$	$7\frac{1}{3}$	$6\frac{1}{3}$	$5\frac{1}{3}$	$4\frac{1}{3}$	$2\frac{2}{3}$	$1\frac{2}{3}$	$\frac{2}{3}$

FIG. 2a. Table of critical times.

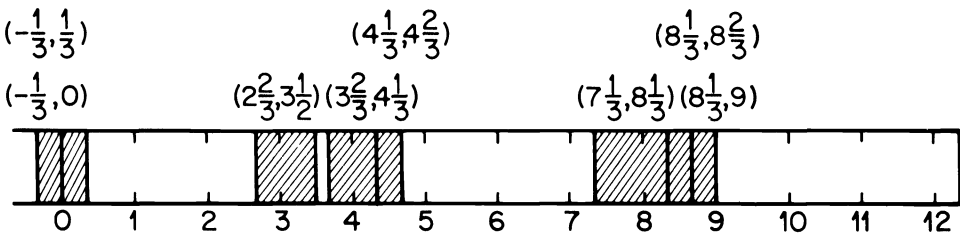


FIG. 2b. Forbidden regions.

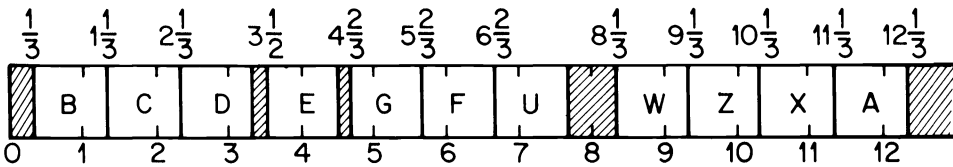


FIG. 2c. Schedule generated for Algorithm A for tasks in Table 1.

TABLE 1.

A set of tasks to schedule. Capital letters represent tasks, with release time first and deadline second.

A: $0, 12\frac{1}{3}$	B: $\frac{1}{3}, 10$	C: $\frac{2}{3}, 5\frac{2}{3}$	D: $1\frac{2}{3}, 6$
E: $3\frac{1}{2}, 7\frac{2}{3}$	F: $4\frac{1}{3}, 6\frac{2}{3}$	G: $4\frac{2}{3}, 6\frac{1}{3}$	U: $5, 8$
W: $8\frac{1}{3}, 11\frac{1}{3}$	X: $8\frac{2}{3}, 11\frac{1}{3}$	Z: $9, 10\frac{1}{3}$	

Proof. The proof is by a straightforward induction on the number of forbidden regions, using Lemma 3, and is omitted. \square

THEOREM 2. *If Algorithm A does not declare failure, then it finds a feasible schedule.*

Proof. Suppose we have a counterexample to the theorem, and let T_j be the first (earliest scheduled) task that fails to meet its deadline. Without loss of generality, we may assume that all idle times in $[0, s_j]$ belong to forbidden regions. (Otherwise, let $t' = \sup \{t : t \text{ is an idle time in } [0, s_j] \text{ that does not belong to a forbidden region}\}$, and let r denote the smallest release time among tasks started at time t' or later. Then, by the earliest deadline scheduling rule, $r \geq t'$ and there are no nonforbidden idle times in $[r, s_j]$. Furthermore, all tasks completed before t' must have release times less than $r - 1$, so they played no role in determining the forbidden regions after r . Thus we can obtain a new counterexample with the desired property by deleting all tasks completed before r and then subtracting r from the release times and deadlines of all remaining tasks.) We now consider two cases:

Case 1. Some task scheduled before T_j has a deadline later than d_j . Let T_i be such a task with maximum starting time, and let r denote the smallest release time among the tasks that start in $(s_i, s_j]$. By the earliest deadline scheduling rule, we know that r must exceed s_i . Index the tasks that start in $(s_i, s_j]$, in order of increasing starting times, as T'_1, T'_2, \dots, T'_k (note that $T'_k = T_j$), and consider the result of applying the Backscheduling Algorithm to these tasks with the given indexing and with $d = d_j$.

We claim that the Backscheduling Algorithm will assign each of these tasks a starting time that is strictly less than its starting time in the original schedule. This is clearly true for $T'_k = T_j$, since the Backscheduling Algorithm assigns it a starting time less than or equal to $d_j - 1$ (depending on whether or not $d_j - 1$ falls in a forbidden region). Inductively, suppose for some $l > 1$ that the claim holds for T'_l, \dots, T'_k and consider how the Backscheduling Algorithm schedules T'_{l-1} . If in the original schedule T'_l started immediately at the time T'_{l-1} finished, the fact that T'_l is started earlier by the Backscheduling Algorithm trivially implies that T'_{l-1} must also be started earlier. On the other hand, if in the original schedule T'_l was separated from T'_{l-1} by a block of idle time $[a, b)$, the fact that all idle times in the block belong to forbidden regions (by our choice of counterexample) implies that the Backscheduling Algorithm must have assigned T'_l a starting time strictly before a , and hence before the old finishing time of T'_{l-1} . Once again it follows that the Backscheduling Algorithm must start T'_{l-1} earlier than in the original schedule, and the claim follows by induction.

The import of the claim is that the critical time c_j (and hence the minimum critical time c) found by Algorithm A when processing release time r must have been strictly less than the starting time assigned to T'_1 by Algorithm A. Furthermore, since T'_1 started in that schedule at the first time not in a forbidden region in the interval $[f_i, s_j]$, it must be the case that $c \leq c_j < f_i$. If $c < r$, then Algorithm A would have declared failure, a contradiction. If $c \geq r$, then we have

$$s_i < r \leq c < f_i = s_i + 1 < r + 1,$$

which implies that Algorithm A would have declared a forbidden region $(c - 1, r)$ containing s_i , contradicting the fact that Algorithm A never starts a task in a forbidden region. It follows that Case 1 cannot occur.

Case 2. All tasks scheduled before T_j have deadlines less than or equal to d_j . Let r be the smallest release time among the tasks started in $[0, s_j]$, and let T'_1, T'_2, \dots, T'_k be the collection of all such tasks, indexed in order of increasing starting times (again note that $T'_k = T_j$). As in the previous case, if the Backscheduling Algorithm is applied to these tasks with the given indexing and with $d = d_j$, it will assign each such task a starting time strictly less than its starting time in the original schedule. Since T'_1 started in the

original schedule at the earliest time not in a forbidden region, it follows that the minimum critical time c found by Algorithm A when processing release time r must be less than 0. Therefore, since $r \geq 0$, Algorithm A would have declared failure, a contradiction to Case 2.

Since both Cases 1 and 2 result in contradictions, and since they include all possibilities, it follows that the assumed counterexample cannot exist, and Theorem 2 is proved. \square

THEOREM 3. *If Algorithm A finds a schedule, that schedule has minimum makespan among all feasible schedules.*

Proof. Suppose we have a counterexample, and let T_i be the first (earliest scheduled) task to be completed after the minimum makespan d^* . As in the previous proof, we may assume that all idle times in $[0, s_i]$ belong to forbidden regions. Let T'_1, T'_2, \dots, T'_k denote the tasks that start in the interval $[0, s_i]$, indexed in order of increasing starting times. The same reasoning as in the previous proof shows that, if the Backscheduling Algorithm were applied to these tasks with the given indexing and with $d = d^*$, it would assign each of the tasks a starting time that is strictly less than its original starting time. Indeed, since in the original schedule T'_1 started at the *earliest* time not in a forbidden region, the Backscheduling Algorithm will assign T'_1 a starting time that is strictly less than 0. But, by Lemma 3, this says that no feasible schedule can possibly complete this many tasks by time d^* , contradicting the fact that d^* is the (achievable) minimum makespan for the assumed counterexample. The theorem follows.

Thus Algorithm A will find a feasible schedule with minimum makespan whenever there is a feasible schedule, and otherwise will correctly declare that no feasible schedule exists.

In implementing Algorithm A, we note that, although the forbidden regions found in Part I may overlap, each region has left and right endpoints no greater than the corresponding endpoints for the region declared just previous to it. Thus we can maintain the forbidden regions in a stack, combining overlapping regions as they occur. For each deadline d_j we maintain a pointer into the stack which indicates the latest forbidden region that precedes the critical time c_j . It is then easy to see that there is an overall time bound of $O(n^2)$ on each of steps 1 and 2, and hence on Part I. Part II requires at most $O(n \log n)$ additional time, for a total of $O(n^2)$ time. In the next section we shall see how to reduce this bound of $O(n^2)$ to $O(n \log n)$ by modifying the algorithm to make more sophisticated use of data structures.

4. Improving the algorithm to $O(n \log n)$. Examination of Algorithm A reveals three places where $\Omega(n^2)$ time might be used, all of them in Part I. In the process of updating critical times c_j with respect to a new task T_i , each of steps 1a and 1b can contribute $\Omega(n)$ time, giving a total of $\Omega(n^2)$ time. Each computation of $c \leftarrow \min \{c_j : c_j \text{ is defined}\}$ in step 2 can also contribute $\Omega(n)$ time, again for a total of $\Omega(n^2)$ time.

The key to obtaining a speed-up from $\Omega(n^2)$ to $O(n \log n)$ involves a basic shift in the way we deal with critical times. Instead of keeping track of each c_j individually, so that the current value of any c_j can be found in constant time (the approach of Algorithm A), we shall keep track of a smaller amount of information, which will be sufficient for determining the current value of any c_j in time $O(\log n)$. This will permit us to use more efficient procedures for organizing and updating the data structures needed for computing the c_j values.

We first observe that each critical time can be decomposed into two components. After some task T_i is processed, each critical time c_j is smaller than the corresponding

deadline d_i by an amount that depends on both the number of tasks seen so far that have deadlines d_i or less and the locations of previously declared forbidden regions. We can keep track of these factors through the *task load* n_i and the *offset* o_i . The task load n_i is essentially the contribution to c_i from step 1a of Algorithm A; that is, n_i is the number of tasks T_k with $k \geq i$ for which $d_k \leq d_i$. (Note that a critical time c_i is “defined” if and only if $n_i > 0$.) The *offset* o_i is the contribution to c_i of step 1b; that is, the total distance c_i has been moved to keep it from being in a forbidden region. Thus we can compute c_i from these two components by the formula $c_i = d_i - (n_i + o_i)$.

The task loads can be maintained easily within an overall time bound of $O(n \log n)$, primarily because whenever we add 1 to n_i we must also add 1 to every n_k such that $d_k \geq d_i$. We store the task loads in a *task load tree*, which is a binary search tree [1, p. 115] having a vertex corresponding to each deadline d_i . In addition, we associate a numerical value with each vertex in such a way that the task load corresponding to any d_i is obtained by summing the values along the path from the root to the vertex for d_i (e.g., see [1, p. 141]). By the choice of an appropriate underlying data structure we can insure that no such path has length exceeding $O(\log n)$, and hence the time for determining the value of any n_i will be $O(\log n)$. Similarly, the cost of updating the tree when a new task is processed (which can involve changes to $\Omega(n)$ c_i values) will be only $O(\log n)$, for an overall updating cost of $O(n \log n)$, as claimed.

Maintaining the offsets is somewhat more complicated. In fact, we will not keep track of the offsets themselves, but rather certain related quantities which we will call *pseudo-offsets*. These are defined as follows:

Suppose that F is the set of forbidden regions declared before the start of processing for task T_i . For any deadline d_i and nonnegative integer n , let $b_i(n)$ denote the earliest starting time that would be assigned if the Backscheduling Algorithm of § 3 were applied to n tasks with deadline d_i and with the forbidden regions in F . (If $n = 0$, we let $b_i(n) = d_i$.) For each n , let

$$o'_i(n) = d_i - b_i(n) - n,$$

and define the *pseudo-offset* o'_i by $o'_i = \lim_{n \rightarrow \infty} o'_i(n)$. Observe that o'_i is well defined and finite, since as soon as $b_i(n) < r_i$ we must have $o'_i(n+1) = o'_i(n)$.

Notice that, unlike the offset o_i , the pseudo-offset o'_i does not depend on the current value of the task load n_i . This is the property that will allow us to maintain the pseudo-offsets efficiently. We postpone for the moment the discussion of how pseudo-offsets can be used in place of offsets for finding the same forbidden regions. Instead we first fill in the details of how the pseudo-offsets can be maintained (first-time readers may wish to skip over these details for now).

Pseudo-offsets are all initially 0 and change only when a new forbidden region (a, b) is declared. In this case the pseudo-offset for a given deadline $d_i > b$ changes if and only if there exists a nonnegative integer n such that $d_i - o'_i - n \in (a, b)$. If this occurs, the increase in the pseudo-offset is exactly $d_i - o'_i - n - a$. Thus the change depends only on the *fractional offset* $q_i = (d_i - o'_i) \pmod{1}$, i.e., the fractional part of $d_i - o'_i$.

To keep track of the pseudo-offsets, we use two data structures: a *fractional offset tree* and a *pseudo-offset forest*. The former is just a binary search tree with a vertex for each distinct fractional offset value. The latter is a standard union-find data structure (see [1]) with a vertex for each deadline d_i and with deadlines having the same fractional offset value belonging to the same tree in the forest. Each vertex in the fractional offset tree will contain a pointer to the root of the union-find tree for that fractional offset value. Each vertex in the pseudo-offset forest will contain a numerical value such that the sum of the values along the path from a vertex to the corresponding root is exactly

the pseudo-offset for the deadline represented by that vertex. Initially both data structures are empty.

When a new forbidden region (a, b) is created, we update these data structures as follows: First, if there is any deadline $d_i > a + 1$ not represented in the data structures, we add it with pseudo-offset $o'_i = 0$ and fractional offset value $q_i = d_i[\text{mod } 1]$, merging as necessary. Second, we determine the set $Q(a, b)$ of offsets affected by (a, b) which is given by

$$Q(a, b) = \begin{cases} \{q_i: q_i \in (a[\text{mod } 1], b[\text{mod } 1])\} & \text{if } a[\text{mod } 1] < b[\text{mod } 1], \\ \{q_i: q_i \in (0, b[\text{mod } 1]) \cup (a[\text{mod } 1], 1)\} & \text{otherwise.} \end{cases}$$

If $Q(a, b)$ is empty, no pseudo-offset is affected by (a, b) and the two data structures can remain unchanged. If $Q(a, b)$ is nonempty, then all the corresponding entries in the fractional offset tree are replaced by a single entry with value $a[\text{mod } 1]$, and all trees in the pseudo-offset forest that correspond to the fractional offset value $a[\text{mod } 1]$ are merged together. Finally, the auxiliary values in the pseudo-offset forest are changed to reflect the corresponding changes in the pseudo-offsets of the members of $Q(a, b)$. The appropriate amount of this change is $q_i - a[\text{mod } 1]$ if $a[\text{mod } 1] < q_i$, or $1 + q_i - a[\text{mod } 1]$ if $a[\text{mod } 1] > q_i$.

With appropriate data structures for the fractional offset tree and the pseudo-offset forest (again, see [1] and also [9]), the overall time bound for maintaining this information will be $O(n \log n)$. Each deadline is added to this structure once at a cost of $O(\log n)$. The construction of each set $Q(a, b)$ requires time $O((|Q(a, b)| + 1) \cdot \log n)$, and this will be $O(n \log n)$ overall since the sum over all $Q(a, b)$ of $|Q(a, b)|$ is bounded by $2n$. (Once two deadlines are merged because they have the same fractional offset value, they stay merged henceforth.) The time for merging two trees in the pseudo-offset forest is $O(1)$ per merge and hence $O(n)$ overall. Moreover, this can be done so that no tree ever has depth exceeding $O(\log n)$, so any particular pseudo-offset can be computed in time $O(\log n)$, as required. Finally, the changes in pseudo-offsets caused by a set $Q(a, b)$ can be incorporated in time $O(|Q(a, b)|)$ and hence $O(n)$ overall. Thus, as claimed, the overall time required for maintaining the pseudo-offset data structures is $O(n \log n)$. Specific details of the implementation are left to the reader.

We now wish to argue that we can still identify the same forbidden regions by using the pseudo-offsets. Recall that the critical time for deadline d_i is defined to be $d_i - o_i - n_i$. Define the *pseudo-critical time* c'_i to be $c'_i = d_i - o'_i - n_i$, and observe that the pseudo-critical time for a deadline never exceeds its critical time, though it may be smaller. In particular, if the *pseudo-task load* $n'_i = \min\{n: o'_i(n) = o'_i\}$ exceeds the task load n_i , then $c'_i < c_i$. The following lemma shows that we can use pseudo-critical times in place of critical times in step 2 of Algorithm A and still compute the same forbidden regions.

LEMMA 4. *If, after task T_i is processed, the minimum pseudo-critical time $c' = \min\{c'_j: n_j > 0\}$ satisfies $c' < r_i + 1$, then the minimum critical time $c = \min\{c_j: n_j > 0\}$ equals c' .*

Proof. The proof is by induction on the number of tasks processed. The lemma clearly holds if no tasks have been processed, since initially there are no forbidden regions and all critical times and pseudo-critical times equal their corresponding deadlines by definition. Suppose the lemma holds after processing task T_{i+1} but not after processing task T_i (recall that we process tasks in order of decreasing index). Then after processing task T_i there must be a deadline d_j such that $c'_j < r_i + 1$ is the minimum pseudo-critical time, but the minimum critical time c exceeds c'_j . In particular, this means that the minimum critical time c_0 after T_{i+1} is processed must obey $c_0 \geq c > c'_j$.

We also must have $c_j > c'_j$, and hence $n_j < n'_j$ (and hence $n'_j > 0$). This means that

$$\begin{aligned} c'_j &= d_j - (o'_j + n_j) \geq d_j - (o'_j(n'_j) + n'_j) + 1 \\ &\geq b_j(n'_j) + 1. \end{aligned}$$

Since $n'_j > 0$, we must have $o'_j(n'_j) > o'_j(n'_j - 1)$ by definition. Therefore $b_j(n'_j - 1) - 1$ must be in a forbidden region and $b_j(n'_j)$ must be the left endpoint of a forbidden region, by the operation of the Backscheduling Algorithm. Thus, if l is the left endpoint of the leftmost forbidden region while T_i is being processed, we must have

$$c'_j \geq b_j(n'_j) + 1 \geq l + 1.$$

But, by the way forbidden regions are defined in step 2 of Algorithm A, we must have $l \geq c_0 - 1$. Hence $c'_j \geq c_0$, a contradiction. Thus the lemma does indeed hold after processing T_i , and, by induction holds after processing any task.

As a consequence of Lemma 4, we know that if critical times are replaced by pseudo-critical times in step 2 of Algorithm A, the same forbidden regions will be declared. Hence, by replacing steps 1a and 1b by the computation of task loads and pseudo-offsets, we will not affect the critical regions and we will reduce two of the potential sources of $\Omega(n^2)$ computation steps to $O(n \log n)$. The remaining potential difficulty is in the calculation of $c' = \min \{c'_j : n_j > 0\}$ in step 2. If we had to look at all the c'_j each time we calculated c' , this could now conceivably take time $\Omega(n^2 \log n)$. Fortunately, we do not need to do this. The key observation is contained in the following lemma.

LEMMA 5. *If, after task T_i is processed, we have $c'_j \leq c'_k$ for some deadlines $d_k \leq d_j$, then at all times in the future we will have $c'_j \leq c'_k$.*

Proof. The pseudo-critical time for a given deadline changes only when either (a) the task load changes or (b) a new forbidden region is defined and changes the pseudo-offset. Since $d_k \leq d_j$, each change in the pseudo-critical time for d_k due to an increase in task load must be balanced by an identical change for d_j . Thus (b) is all that we need consider. The only way that a pseudo-critical time c'_k can be altered by a new forbidden region (a, b) is if there is some integer n such that $c'_k - n \in (a, b)$, in which case the pseudo-offset increases by $c'_k - n - a$ and the new pseudo-critical time becomes $c'_k - (c'_k - n - a) = n + a$. However, since $c'_j \leq c'_k$, we must have either $c'_j - n \leq a$, in which case $c'_j \leq n + a$, or else $c'_j - n \in (a, b)$ and hence c'_j also becomes $n + a$. In either case we have that the new values obey $c'_j \leq n + a = c'_k$, as claimed.

Thus, as our algorithm proceeds, certain pseudo-critical times become unnecessary for our computations of $c' = \min \{c'_j : n_j > 0\}$. To formalize this idea, let us say that a deadline d_i is *relevant* at a point in the updating process if for no d_j with $d_j \geq d_i$ is $c'_j < c'_i$. A deadline is *irrelevant* if it is not relevant. Initially all deadlines are relevant, though some may become irrelevant as the computation proceeds. Note that if we sort the relevant deadlines into nondecreasing order, their pseudo-critical times must also be in nondecreasing order. If we maintain a pointer into this list to the first deadline with a "defined" pseudo-critical time (i.e., with $n_j > 0$), then at any time we can determine c' in time $O(\log n)$ by merely computing the pseudo-critical time for the deadline to which the pointer points, using the data structures for task loads and pseudo-offsets discussed earlier. The total time for computing values of c' will thus be $O(n \log n)$. Moreover, since we keep the relevant deadlines sorted by deadline rather than pseudo-critical time, we need not do any reordering except to delete newly irrelevant deadlines. This will allow us to maintain data structures for the relevant deadlines in a way that also obeys an $O(n \log n)$ bound.

We store the relevant deadlines in a *relevant deadline tree*, which is a binary search tree structured to allow deletions in $O(\log n)$ time and can itself be initialized in time $O(n \log n)$ —again, see [1]. The pointer to the first relevant deadline with nonzero task load is initially undefined. In updating we make use of the following lemma, whose proof is much like that of Lemma 5 and is omitted.

LEMMA 6. *A relevant deadline can only become irrelevant as a result of the change of task loads during the processing of a task T_i , and not as the result of a change in pseudo-offsets during the processing of a forbidden region. If d_j is the minimum relevant deadline not less than d_i when T_i is processed, then the deadlines that become irrelevant are precisely those relevant deadlines $d_k < d_j$ whose old pseudo-critical times exceed the new value of the pseudo-critical time c'_j .*

Thus, after updating the task load tree, we need only identify d_j (in time $O(\log n)$), compute its pseudo-critical time (again in time $O(\log n)$), and then begin comparing this to the pseudo-critical times for those relevant deadlines d_k with $d_k < d_j$, in order, starting with the latest and ending as soon as one is found with $c'_k \leq c'_j$. All those with $c'_k > c'_j$ are deleted, at a cost of $O(\log n)$ per deletion. Since a deadline can only be deleted once, the overall cost for comparisons and deletions will then be $O(n \log n)$ as claimed. In addition, during this process the pointer to the first relevant deadline with nonzero task load can be updated if necessary at an overall cost of $O(n)$. Thus step 2 of Algorithm A can be replaced by a procedure which accomplishes the same task in a running time of $O(n \log n)$.

This guarantees that the overall algorithm can be made to run in time $O(n \log n)$. We shall call the revised algorithm Algorithm B. It differs from Algorithm A only in Part I, as Part II already runs in time $O(n \log n)$. The revised Part I proceeds as follows:

ALGORITHM B.

Part I. (Forbidden Region Declaration). Initially, there are no forbidden regions, the relevant deadline tree contains all deadlines, its pointer is undefined, the task load tree contains all deadlines with their task loads initialized to 0 and the two offset data structures are empty. For each task T_i , in order of decreasing index, we then perform the following steps:

Step 1.

- 1a. Modify the task load tree to add 1 to the task load for each deadline $d_j \geq d_i$.
- 1b. Set $d' \leftarrow \min \{d_j : d_j \text{ is relevant and } d_j \geq d_i\}$. While the relevant deadline d'' that immediately precedes d' in the sorted order of deadlines has a pseudo-critical time exceeding that for d' , delete d'' from the relevant deadline tree and, if the pointer was undefined or pointed to d'' , reset the pointer to point to d' .

Step 2.

If $i = 1$ or $r_{i-1} < r_i$, set $c' \leftarrow \min \{c'_j : d_j \text{ is relevant and } n_j > 0\}$ and proceed as follows:

2a. If $c' < r_i$, declare failure and halt.

2b. If $r_i \leq c' < r_i + 1$, declare $(c' - 1, r_i)$ to be a forbidden region and update the data structures as follows:

Add to the fractional offset tree and the pseudo-offset forest all deadlines $d_j \geq c'$ that are not currently represented, merging if necessary. Compute $Q(c' - 1, r_i)$ and replace all corresponding entries in the fractional offset tree by a single entry with value $c' - 1$. Merge all the corresponding sets in the pseudo-offset forest and update the pseudo-critical times appropriately.

The reader should be able to verify from the preceding discussion that the algorithm does indeed compute the same forbidden regions as Algorithm A, but now does it in an overall time bound of $O(n \log n)$. Certain additional efficiencies can be obtained, for instance by using path compression in the pseudo-offset forest (which is not required for the $O(n \log n)$ bound) and by combining the relevant deadline tree and the task load tree into a single data structure, but these will not yield any improvement in the basic $O(n \log n)$ bound and so we leave such details to those readers interested in actually implementing the procedure. We also leave to the reader the straightforward verification of the fact that none of the data structures used by this algorithm (or Algorithm A) require more than linear space. Fig. 3 illustrates the application of Algorithm B to the tasks of Table 1, and can be compared to the analogous Fig. 2 for Algorithm A.

5. Conclusion. In this paper we have studied the problem of scheduling unit-time tasks with arbitrary release times and deadlines, and we have showed how the idea of “forbidden regions” could be used to construct an algorithm which solved the problem of minimizing makespan on one processor in time $O(n \log n)$.

Several interesting open problems remain. Carlier [2] has recently shown that the general m -processor case can be solved in time $O(n^{m+1} \log n)$. Can the general problem be solved in time polynomial in *both* m and n , perhaps by a suitable generalization of the concept of forbidden regions (which we were unable to find)? What happens if we add precedence constraints to the problem? We have already seen that this does not affect the one-processor algorithm, but what of the case of two processors, where a polynomial time algorithm *is* known for the case when all release times are integers [3]?

		RELEASE TIMES										
		9	$8 \frac{2}{3}$	$8 \frac{1}{3}$	5	$4 \frac{2}{3}$	$4 \frac{1}{3}$	$3 \frac{1}{2}$	$1 \frac{2}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	0
DEADLINES	$5 \frac{2}{3}$	0	0	0	0	0	0	0	0	1	1	1
	6	0	0	0	0	0	0	0	1	2	2	2
	$6 \frac{1}{3}$	0	0	0	0	1	1	1	2	3	3	3
	$6 \frac{2}{3}$	0	0	0	0	1	2	2	3	4	4	4
	$7 \frac{2}{3}$	0	0	0	0	1	2	3	4	5	5	5
	8	0	0	0	1	2	3	4	5	6	6	6
	10	0	0	0	1	2	3	4	5	6	7	7
	$10 \frac{1}{3}$	1	1	1	2	3	4	5	6	7	8	8
	$11 \frac{1}{3}$	1	2	3	4	5	6	7	8	9	10	10
	$12 \frac{1}{3}$	1	2	3	4	5	6	7	8	9	10	11

FIG. 3a. Table of task loads.

FORBIDDEN REGIONS

$(8\frac{1}{3}, 9)$ $(8\frac{1}{3}, 8\frac{2}{3})$ $(7\frac{1}{3}, 8\frac{1}{3})$ $(4\frac{1}{3}, 4\frac{2}{3})$ $(3\frac{2}{3}, 4\frac{1}{3})$ $(2\frac{2}{3}, 3\frac{1}{2})$ $(-\frac{1}{3}, \frac{1}{3})$ $(-\frac{1}{3}, 0)$

5 $\frac{2}{3}$	-	-	-	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0
6	-	-	-	0	$\frac{2}{3}$ $\frac{1}{3}$	$\frac{2}{3}$ $\frac{1}{3}$	$\frac{2}{3}$ $\frac{1}{3}$	$\frac{2}{3}$ $\frac{1}{3}$
6 $\frac{1}{3}$	-	-	-	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{2}{3}$ $\frac{2}{3}$	$\frac{2}{3}$ $\frac{2}{3}$	$\frac{2}{3}$ $\frac{2}{3}$
6 $\frac{2}{3}$	-	-	-	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0
7 $\frac{2}{3}$	-	-	-	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0	$\frac{2}{3}$ 0
8	-	-	-	0	$\frac{2}{3}$ $\frac{1}{3}$	$\frac{2}{3}$ $\frac{1}{3}$	$\frac{2}{3}$ $\frac{1}{3}$	$\frac{2}{3}$ $\frac{1}{3}$
10	0	0	$\frac{1}{3}$	$\frac{1}{3}$ $\frac{2}{3}$	$\frac{1}{3}$ $\frac{2}{3}$	$\frac{2}{3}$ $\frac{4}{3}$	$\frac{2}{3}$ $\frac{4}{3}$	$\frac{2}{3}$ $\frac{4}{3}$
10 $\frac{1}{3}$	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{2}{3}$ $\frac{2}{3}$	$\frac{2}{3}$ $\frac{2}{3}$	$\frac{2}{3}$ $\frac{2}{3}$
11 $\frac{1}{3}$	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{2}{3}$ $\frac{2}{3}$	$\frac{2}{3}$ $\frac{2}{3}$	$\frac{2}{3}$ $\frac{2}{3}$
12 $\frac{1}{3}$	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{1}{3}$ 0	$\frac{2}{3}$ $\frac{2}{3}$	$\frac{2}{3}$ $\frac{2}{3}$	$\frac{2}{3}$ $\frac{2}{3}$

DEADLINES

FIG. 3b. Table of fractional and pseudo-offsets.

	9	$8\frac{2}{3}$	$8\frac{1}{3}$	5	$4\frac{2}{3}$	$4\frac{1}{3}$	$3\frac{1}{2}$	$1\frac{2}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	0
5 $\frac{2}{3}$	$5\frac{2}{3}$	$5\frac{2}{3}$	$5\frac{2}{3}$	$5\frac{2}{3}$							
6	6	6	6	6							
6 $\frac{1}{3}$	$6\frac{1}{3}$	$6\frac{1}{3}$	$6\frac{1}{3}$	$6\frac{1}{3}$	$5\frac{1}{3}$						
6 $\frac{2}{3}$	$6\frac{2}{3}$	$6\frac{2}{3}$	$6\frac{2}{3}$	$6\frac{2}{3}$	$5\frac{2}{3}$	$4\frac{2}{3}$					
7 $\frac{2}{3}$	$7\frac{2}{3}$	$7\frac{2}{3}$	$7\frac{2}{3}$								
8	8	8	8	7	6	5	$3\frac{2}{3}$	$2\frac{2}{3}$	$1\frac{2}{3}$		
10											
10 $\frac{1}{3}$	$9\frac{1}{3}$	$9\frac{1}{3}$									
11 $\frac{1}{3}$	$10\frac{1}{3}$	$9\frac{1}{3}$	$8\frac{1}{3}$	$7\frac{1}{3}$	$6\frac{1}{3}$	$5\frac{1}{3}$	$4\frac{1}{3}$	$2\frac{2}{3}$	$1\frac{2}{3}$	$\frac{2}{3}$	$\frac{2}{3}$
12 $\frac{1}{3}$	$11\frac{1}{3}$	$10\frac{1}{3}$	$9\frac{1}{3}$	$8\frac{1}{3}$	$7\frac{1}{3}$	$5\frac{1}{3}$	$5\frac{1}{3}$	$3\frac{2}{3}$	$2\frac{2}{3}$	$1\frac{2}{3}$	$\frac{2}{3}$

FIG. 3c. Table of pseudo-critical times for relevant deadlines when Algorithm B is applied to tasks in Table 1. Circled entries are the minimum pseudo-critical times for deadlines with nonzero task loads. Cross-hatched regions are for irrelevant deadlines.

Note added in proof. B. Simons has recently resolved one of our open problems by showing that the general m -processor case (with no procedure constraints) can be solved in time $O(n^3 \log n)$. [*A fast algorithm for multiprocessor scheduling*, IEEE 21st Annual Symposium on Foundations of Computer Science, Long Beach, California, 1980, pp. 50–53.]

REFERENCES

- [1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.
- [2] J. CARLIER, *Problème à une machine dans le cas où les tâches ont des durées égales*, Technical Report (1979) Institut de Programmation, Université Paris VI, Paris.
- [3] M. R. GAREY AND D. S. JOHNSON, *Two-processor scheduling with start-times and deadlines*, this Journal, 6 (1977), pp. 416–426.
- [4] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, California, 1979.
- [5] J. R. JACKSON, *Scheduling a production line to minimize maximum tardiness*, Research Report 43 (1955), Management Science Research Project, University of California at Los Angeles.
- [6] D. E. KNUTH, *The Art of Computer Programming*, Vol. 1: *Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1968.
- [7] T. LANG AND E. B. FERNÁNDEZ, *Scheduling of unit-length independent tasks with execution constraints*, Information Processing Letters, 4 (1976), pp. 95–98.
- [8] B. SIMONS, *A fast algorithm for single processor scheduling*, IEEE 19th Annual Symposium on Foundations of Computer Science, Long Beach, California, 1978, pp. 246–252.
- [9] R. E. TARJAN, *Applications of path compression on balanced trees*, J. Assoc. Comput. Mach., submitted.
- [10] J. D. ULLMAN, *NP-complete scheduling problems*, J. Comput. System Sci., 10 (1975), pp. 384–393.

APPROXIMATION ALGORITHMS FOR SEVERAL GRAPH AUGMENTATION PROBLEMS*

GREG N. FREDERICKSON[†] AND JOSEPH JA'JA'[‡]

Abstract. Graph augmentation problems on a weighted graph involve determining a minimum-cost set of edges to add to a graph to satisfy a specified property, such as biconnectivity, bridge-connectivity or strong connectivity. These augmentation problems are shown to be NP-complete in the restricted case of the graph being initially connected. Approximation algorithms with favorable time complexity are presented and shown to have constant worst-case performance ratios.

Key words. approximation algorithm, augmentation, biconnectivity, bridge-connectivity, connectivity, graph, NP-complete problem, strong connectivity

1. Introduction. A number of graph problems can be viewed as augmentation problems: Given a complete graph $G = (V, E)$, a subgraph $G_0 = (V, E')$ and a cost function on E , find a set of edges $E'' \subseteq E - E'$ of minimum cost such that $(V, E' \cup E'')$ satisfies a specified property. For instance, if $E' = \emptyset$ and the property is graph connectivity, then the corresponding problem is that of finding a minimum-cost spanning tree on V . This problem can of course be solved efficiently [P], [Y], [J], [CT]. An analogous problem for directed graphs, in which the property is having vertices in V be reachable from a given vertex, involves finding a minimum-cost spanning arborescence. This problem can also be solved in an efficient manner [CL], [E], [T1]. Further examples of augmentation problems are cited in [ET].

We consider three properties studied by Eswaran and Tarjan [ET], strong connectivity for directed graphs, and biconnectivity and bridge-connectivity for undirected graphs. The latter properties may be viewed as certain desirable features of networks. Given a network, we wish to augment the network with additional edges so that these features are achieved. Networks that are biconnected and bridge-connected can survive single element failures: A biconnected network can survive a node failure, and a bridge-connected network can survive a communication line failure.

If the edges in the graph all have equal weight, then the various augmentation problems also have efficient algorithms [ET], [RG]. However, if the edges have unequal weights, then the augmentation problems are NP-complete [ET]. The problems remain NP-complete if G_0 is restricted to be a connected subgraph with edge weights chosen from the set $\{1, 2\}$, as we shall see in § 3. In particular, augmenting a directed acyclic graph to be strongly connected, or augmenting even a tree to be bridge-connected or biconnected, is NP-complete.

Membership in the NP-complete [C], [K1], [K2], [GJ2] class of problems appears to indicate that there is no efficient algorithm for solving such a problem exactly. A reasonable alternative is to design efficient algorithms that yield near-optimal, or approximate, solutions [J], [GJ1]. In §§ 4, 5 and 6 we present approximation algorithms for our augmentation problems, and bound their performance in terms of a worst-case ratio of the cost of a generated augmentation to the cost of an optimal

* Received by the editors June 6, 1979, and in revised form May 30, 1980.

[†] Department of Computer Science, Pennsylvania State University, University Park, Pennsylvania 16802. The work of this author was supported in part by the National Science Foundation under grant MCS-7909259.

[‡] Department of Computer Science, Pennsylvania State University, University Park, Pennsylvania 16802. The work of this author was supported in part by the National Science Foundation under Grant MCS-7827600.

augmentation. Our algorithms run in $O(|V|^2)$ time, and have a similar strategy: Use an algorithm for finding a spanning arborescence to strongly connect a weakly connected graph. We consider graphs with no special cost restrictions, such as the triangle inequality, but note that improved bounds can be achieved for certain problems if the graphs are so restricted [FJ].

2. Definitions. An *undirected graph* $G = (V, E)$ consists of a set of *vertices* V and a set of (*undirected*) *edges* E , where each edge is an unordered pair (u, v) of vertices. A *directed graph* $G = (V, A)$ consists of a set of vertices V and a set of *directed edges* A . A directed edge $\langle u, v \rangle$ is an ordered pair of vertices such that the edge *leaves* u and *enters* v . A graph G is *complete* if, given its vertex set, its edge set contains all possible edges. A graph G is *weighted* if there is a cost function from its edge set into a set with a total order. A *subgraph* G' of graph G is a graph whose vertex and edge sets are subsets of the vertex and edge sets of G . G' is a *spanning subgraph* of G if their vertex sets are equal.

A *path* from v_1 to v_n in an undirected graph $G = (V, E)$ is a sequence of edges $(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$. A *cycle* is a path such that $v_1 = v_n$ and all vertices v_1, v_2, \dots, v_{n-1} are distinct. A (*strongly*) *directed path* from v_1 to v_n in a directed graph $G = (V, A)$ is a sequence of edges $\langle v_1, v_2 \rangle, \langle v_2, v_3 \rangle, \dots, \langle v_{n-1}, v_n \rangle$. A *directed cycle* is a directed path such that $v_1 = v_n$ and all vertices v_1, v_2, \dots, v_{n-1} are distinct. A *weakly directed path* from v_1 to v_n is a sequence of edges e_1, e_2, \dots, e_{n-1} , where e_i is $\langle v_i, v_{i+1} \rangle$ or $\langle v_{i+1}, v_i \rangle$ for $1 \leq i < n$. A graph is *acyclic* if it contains no (directed) cycles.

An undirected graph $G = (V, E)$ is *connected* if, for any two vertices u and v , there is a path from u to v . A *bridge* is an edge whose removal from E leaves G not connected. A graph is *bridge-connected* if its edge set contains no bridges. A *cutvertex* is a vertex whose removal from V , along with the removal of edges incident on it from E , leaves G not connected. A graph is *biconnected* if it contains no cutvertices. A directed graph $G = (V, A)$ is *strongly connected* if, for any two vertices u and v , there is a directed path from u to v . G is *weakly connected* if, for any two vertices u and v , there is a weakly directed path from u to v . The connected (*bridge-connected*, *biconnected*, *strongly connected*) *components* of a graph are its maximal connected (*bridge-connected*, *biconnected*, *strongly connected*) subgraphs. Biconnected components are also called *blocks*.

A *tree* is an undirected, connected acyclic graph. A *leaf* in a tree is a vertex with one edge incident on it. A *spanning tree* of a graph G is a spanning subgraph that is a tree. An *arborescence* is a directed acyclic graph with one vertex, the *root*, having no entering edges, and all other vertices having exactly one entering edge. A *spanning arborescence* of a directed graph G is a spanning subgraph that is an arborescence. A *branching* of a directed graph G is a graph whose weakly connected components are arborescences.

3. NP-completeness results. We state formal definitions of the bridge-connectivity, biconnectivity, and strong connectivity augmentation problems below, and then proceed to show that restricted version of these problems are NP-complete:

BRIDGE-CONNECTIVITY AUGMENTATION (BRA).

Instance. Complete undirected graph $G = (V, E)$, weight function $c(e) \in \mathbb{Z}^+$ for $e \in E$, subgraph $G_0 = (V, E')$, with $E' \subseteq E$ and positive integer B .

Question. Is there a set $E'' \subseteq E - E'$ such that $\sum_{e \in E''} c(e) \leq B$ and $(V, E' \cup E'')$ is bridge-connected?

BICONNECTIVITY AUGMENTATION (BIA).

Same as BRA except with biconnected replacing bridge-connected.

STRONG CONNECTIVITY AUGMENTATION (STA).

Instance. Complete directed graph $G = (V, A)$, weight function $c(a) \in \mathbb{Z}^+$ for $a \in A$, subgraph $G_0 = (V, A')$, with $A' \subseteq A$ and positive integer B .

Question. Is there a set $A'' \subseteq A - A'$ such that $\sum_{a \in A''} c(a) \leq B$ and $(V, A' \cup A'')$ is strongly connected?

Eswaran and Tarjan [ET] have shown all three augmentation problems to be NP-complete. Their reductions are from Hamiltonian circuits, and choose G_0 to be the empty subgraph (V, \emptyset) . It is possible that the problems could be less difficult if they are restricted to a connected G_0 , in particular, a tree. Let the connected bridge-connectivity augmentation problem (CBRA) be the same as BRA, except that G_0 must be a connected subgraph of G . Let the connected biconnectivity augmentation problem (CBIA) be similarly defined. Let the connected strong-connectivity augmentation problem (CSTA) be the same as STA, except that G_0 must be weakly connected. We show that all three restricted problems, CBRA, CBIA and CSTA, are NP-complete.

Eswaran and Tarjan [ET] claimed, but did not provide proofs, that CBIA and CBRA are NP-complete. Tarjan [T2] has provided a proof only for CBIA, using a transformation from directed Hamiltonian circuits. It appears that his transformation for CBIA does not carry over to CBRA.

Our transformation for CBRA is from 3-dimensional matching:
 3-DIMENSIONAL MATCHING (3DM).

Instance. A set $M \subseteq W \times X \times Y$, where W, X and Y are disjoint sets having the same number q of elements.

Question. Does M contain a matching, that is, a subset $M' \subseteq M$ such that $|M'| = q$ and no two elements of M' agree in any coordinate?

The transformations for CSTA and CBIA are also from 3DM, and are essentially the same. All three restricted augmentation problems remain NP-complete if edge weights are either 1 or 2. Handling G_0 where G_0 is a tree is no easier, since our reductions are to a graph that is a tree. Since it is easy to see that all three problems are in NP, we prove that 3DM is reducible to each of our problems, e.g., $3DM \propto CSTA$.

THEOREM 1. $3DM \propto CSTA$.

Proof. Let $M \subseteq W \times X \times Y$ be an instance of 3DM, with $|M| = p$ and $W = \{w_i | i = 1, 2, \dots, q\}$, $X = \{x_i | i = 1, \dots, q\}$ and $Y = \{y_i | i = 1, \dots, q\}$. We define an instance of CSTA as follows:

$$V = \{r\} \cup \{w_i, x_i, y_i | i = 1, \dots, q\} \cup \{a_{ijk}, \bar{a}_{ijk} | (w_i, x_j, y_k) \in M\},$$

$$A = \{(u, v) | u, v \in V \text{ and } u \neq v\},$$

$$A' = \{(r, w_i), (r, x_i), (y_i, r) | i = 1, \dots, q\} \cup \{(a_{ijk}, w_i), (w_i, \bar{a}_{ijk}) | (w_i, x_j, y_k) \in M\}.$$

Let $c(\bar{a}_{ijk}, a_{ijk}) = c(x_j, a_{ijk}) = c(\bar{a}_{ijk}, y_k) = 1$, for $(w_i, x_j, y_k) \in M$. Let all other edges in A have weight 2. Let $B = p + q$.

We claim that M contains a matching M' iff there is a set A'' of cost no more than $B = p + q$ such that $(V, A' \cup A'')$ is strongly connected. Suppose M contains a matching M' . For each triple (w_i, x_j, y_k) in M' , insert the directed edges (x_j, a_{ijk}) and (\bar{a}_{ijk}, y_k) into A'' . This will cause w_i, x_j and y_k to be on a (directed) cycle in $A' \cup A''$ containing r . Since $|M'| = q$, the total cost will be $2q$ and all w_i, x_j and y_k will be on cycles containing r . For each triple (w_i, x_j, y_k) in $M - M'$, insert (\bar{a}_{ijk}, a_{ijk}) into A'' . This will cause each a_{ijk} and \bar{a}_{ijk} not already on a cycle in $A' \cup A''$ to be on a cycle containing w_i . Total cost of the edges introduced in this step will be $|M| - |M'| = p - q$. Hence, the total cost of A'' will be $p + q$. It is easy to see that every pair of vertices lie on a cycle in $A' \cup A''$. Hence, $(V, A' \cup A'')$ is strongly connected.

We now prove the “if” part: Assume that there is a set of A'' of cost no greater than $p + q$ such that $(V, A' \cup A'')$ is strongly connected. There are $2(p + q)$ leaves in (V, A') . For $(V, A' \cup A'')$ to be strongly connected, A'' must contain at least $p + q$ edges, since an edge can connect at most two leaves. Since A'' is of cost $p + q$, A'' contains exactly $p + q$ edges, each of cost 1. Hence, each leaf in (V, A') has exactly one edge of A'' incident on it.

Each leaf x_j will have one edge (x_j, a_{ijk}) from A'' incident on it. Hence, (\bar{a}_{ijk}, a_{ijk}) is not in A'' , and thus (\bar{a}_{ijk}, y_k) is in A'' . No other edge in A'' will be incident to y_k . Hence, for each x_j , the corresponding (w_i, x_j, y_k) will be in M' . The remaining $p - q$ edges in A'' will be of the form (\bar{a}_{ijk}, a_{ijk}) , and thus contribute nothing to the matching. \square

THEOREM 2. $3DM \propto CBRA$.

Proof. Given an instance of 3DM, we define an instance of CBRA exactly as we defined an instance of CSTA in Theorem 1, except that we interpret the edges as being undirected, and use E and E' instead of A and A' .

We claim that M contains a matching M' iff there is a set E'' of cost no more than $B = p + q$ such that $(V, E' \cup E'')$ is bridge-connected. As in Theorem 1, for every triple (w_i, x_j, y_k) in M' , two edges will be inserted into E'' and will cause w_i, x_j , and y_k to be a cycle in $(V, E' \cup E'')$ containing r . For every triple (w_i, x_j, y_k) in $M - M'$, an edge will be inserted into E'' , causing a_{ijk} and \bar{a}_{ijk} to be a cycle containing w_i . The cost is again $p + q$, and joining cycles gives that each vertex is on a cycle containing r . Hence, $(V, E' \cup E'')$ is bridge-connected.

Assume there is an E'' of cost no more than $p + q$ such that $(V, E' \cup E'')$ is bridge-connected. As in Theorem 1, we may deduce that there are exactly $p + q$ edges in E'' , each of cost 1. We may also infer the elements of M' as in Theorem 1. \square

While Tarjan [T2] has already provided a proof that CBIA is NP-complete, we provide our own here, since it fits the same form as the proofs of Theorems 1 and 2. To ensure biconnectivity instead of bridge-connectivity, we simply introduce double copies, \bar{r} and \bar{w}_i , of the crucial vertices r and w_i , where cycles intersect.

THEOREM 3. $3DM \propto CBIA$.

Proof. Given an instance of 3DM, we define an instance of CBIA as follows:

$$\begin{aligned}
 V &= \{r, \bar{r}\} \cup \{w_i, \bar{w}_i, x_i, y_i \mid i = 1, \dots, q\} \cup \{a_{ijk}, \bar{a}_{ijk} \mid (w_i, x_j, y_k) \in M\}, \\
 E &= \{(u, v) \mid u, v \in V \text{ and } u \neq v\}, \\
 E' &= \{(r, \bar{r})\} \cup \{(w_i, \bar{w}_i), (\bar{w}_i, r), (\bar{r}, x_i), (y_i, r) \mid i = 1, \dots, q\} \\
 &\quad \cup \{(a_{ijk}, w_i), (\bar{w}_i, \bar{a}_{ijk}) \mid (w_i, x_j, y_k) \in M\}.
 \end{aligned}$$

The costs are defined as in Theorem 2.

We claim that M contains a matching M' iff there is a set E'' of cost no more than $B = p + q$ such that $(V, E' \cup E'')$ is biconnected. The argument is similar to that of Theorem 2. Suppose there is a matching M' . For each triple (w_i, x_j, y_k) in M' there is a cycle containing $\bar{r}, x_j, a_{ijk}, w_i, \bar{w}_i, \bar{a}_{ijk}, y_k$ and r . All w_i, \bar{w}_i, x_j and y_k will be on such cycles. For each triple (w_i, x_j, y_k) in $M - M'$, there is a cycle containing a_{ijk}, w_i, \bar{w}_i and \bar{a}_{ijk} . Hence, all a_{ijk} and \bar{a}_{ijk} are on cycles containing w_i and \bar{w}_i . Since any such cycle shares two vertices with any other cycle that it shares a vertex with, the removal of a single vertex cannot disconnect $(V, E' \cup E'')$. Hence, $(V, E' \cup E'')$ is biconnected. The cost of E'' , as in Theorem 2 will be $p + q$.

The proof of the if part of the claim is identical to that in Theorem 2. \square

4. Strong connectivity augmentation algorithm. We now present an algorithm that will find a set of directed edges that makes a subgraph strongly connected. Our

strategy is to add edges to the augmenting set A'' in two stages. In the first stage we find a *reverse spanning arborescence*, i.e., a directed acyclic spanning subgraph with one vertex, the root, having no leaving edge, and all other vertices having exactly one leaving edge. There is a path from every vertex to the root in such a directed spanning tree. We use an algorithm for finding optimum branchings [E], [CL], [T1] to find the reverse spanning arborescence.

In the second stage we find a spanning arborescence forcing the root to be the same as was returned from the first stage. In this directed spanning tree, all vertices have exactly one entering edge, except the root, which has none. We can force a particular vertex to be the root by setting costs on all entering edges to be very large. Hence, there will be a path from the root to every vertex. Combining the two directed spanning trees will yield a graph in which there is a path from any vertex u to the root and a path from the root to any vertex v . Thus, the graph will be strongly connected.

We note that our algorithm generates an optimal solution in the case that either a spanning arborescence or a reverse spanning arborescence is contained in G_0 . This is essentially the same observation that [ET] made with respect to one special case that they observed was efficiently solvable.

ALGORITHM STC.

Input. $G = (V, A)$ a complete directed graph with cost function $c: A \rightarrow \mathbb{Z}^+$, and $G_0 = (V, A')$ a subgraph of G .

Output. A set of edges $A'' \subset A - A'$ such that $(V, A' \cup A'')$ is strongly connected.

1. For each edge $\langle u, v \rangle$ in A , let $d\langle u, v \rangle = 0$ if $\langle v, u \rangle \in A'$ and let $d\langle u, v \rangle = c\langle v, u \rangle$, otherwise.
2. Find a minimum weight spanning arborescence $T = (V, A'_a)$ with root $r \in V$ on (V, A) using d . Set $A_a = \{\langle u, v \rangle | \langle v, u \rangle \in A'_a\}$.
3. For each edge $\langle u, v \rangle$ in A , let $d'\langle u, v \rangle = c\langle u, v \rangle$. If $\langle u, v \rangle \in A' \cup A_a$, then set $d'\langle u, v \rangle = 0$. For all u , set $d'\langle u, r \rangle = \infty$.
4. Find a minimum weight spanning arborescence $T' = (V, A_b)$ on (V, A) using d' .
5. Set $A'' = (A_a \cup A_b) - A'$.

LEMMA 1. *Let $G = (V, A)$ be a complete directed graph with cost function $c: A \rightarrow \mathbb{Z}^+$, and let $G_0 = (V, A')$ be a subgraph of G . Then algorithm STC finds a set of edges $A'' \subset A - A'$ such that $(V, A' \cup A'')$ is strongly connected.*

Proof. Let x and y be any two vertices of V . We will prove that x and y are reachable from each other in the graph $(V, A' \cup A'')$. Step 2 generates a reverse spanning arborescence with root r ; hence, there exist two directed paths from x and y to r . Since we assign an infinite cost to any edge of the form $\langle u, r \rangle$ at Step 3, the spanning arborescence found at Step 4 will have r as its root. It follows that there exist directed paths from r to x and y , and therefore x and y are reachable from each other in $(V, A' \cup A'')$. \square

THEOREM 4. *Let $G = (V, A)$ be a complete directed graph with cost function $c: A \rightarrow \mathbb{Z}^+$, and $G_0 = (V, A')$, a subgraph of G . Let C^* be the cost of an optimal augmentation A^* that makes G_0 strongly connected, and let \hat{C} be the cost of the edges A'' generated by algorithm STC. Then*

$$\frac{\hat{C}}{C^*} \leq 2.$$

Furthermore, there exists a family of examples for which the bound can be approached arbitrarily closely.

Proof. Step 2 finds an augmentation $A_a - A'$ of minimum cost such that there is a path from every vertex in $(V, A' \cup A_a)$ to r . Since $(V, A' \cup A^*)$ is strongly connected,

there is a path from every vertex in $(V, A' \cup A^*)$ to r . Since $A_a - A'$ is of minimum cost, the cost is no greater than the cost C^* of A^* . Step 4 finds an augmentation $A_b - (A' \cup A_a)$ that strongly connects $(V, A' \cup A_a)$. Since $A_b - (A' \cup A_a)$ is of minimum cost, and A^* also strongly connects $(V, A' \cup A_a)$, $A_b - (A' \cup A_a)$ is of cost no greater than A^* . Hence, $(A_a \cup A_b) - A'$ is of cost no greater than $2C^*$.

To see that the bound is approachable, consider the graph in Fig. 1. Let all edges $\langle u, v \rangle$ not shown have the cost of the shortest directed path from u to v . Let $A' = \emptyset$. The

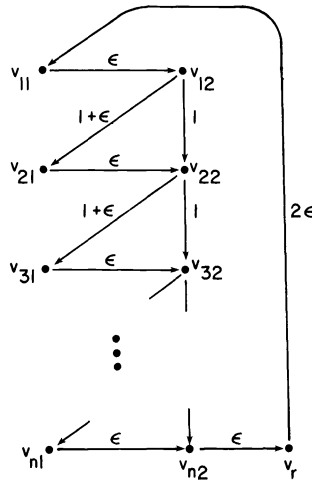


FIG. 1.

set A_a generated in Step 2 will contain all edges of cost ϵ or 1, for a total cost of $n - 1 + (n + 1)\epsilon$. The set $A_b - A_a$ generated in Step 4 will contain edges of costs $1 + \epsilon$ and 2ϵ , for a total cost of $(n - 1)(1 + \epsilon) + 2\epsilon$. Hence, $\hat{C} = 2n - 2 + (2n + 2)\epsilon$. The set A^* will contain the edges of costs $1 + \epsilon$, 2ϵ and ϵ , for a total cost of $n - 1 + (2n + 2)\epsilon$. Thus, \hat{C}/C^* can be arbitrarily close to 2. \square

We note that the worst-case example for completely disconnected graphs would also hold for partially connected graphs. If the edges of cost ϵ are in the edge set A' , then the same worst-case behavior is still obtained. Also, the edge costs in the example satisfy the triangle inequality. This suggests that the triangle inequality does not help in the worst case, when edge costs are not symmetric. If the edge costs are symmetric but the triangle inequality does not hold, then we have a bridge-connectivity problem, with one important modification. A directed graph allows for two edges between a pair of vertices, but an undirected graph (not a multigraph) allows for only one edge.

We also note that our algorithm runs in $O(|V|^2)$ time. Finding a minimum-cost arborescence can be done in time $O(|V|^2)$, using an algorithm by Tarjan [T1], and the remaining work of setting up the distance functions is $O(|A|)$. In fact, for $|A| < O(|V|^2)$, the average time complexity of our algorithm matches that of Tarjan's.

5. Bridge-connectivity augmentation algorithm. We first consider the problem of bridge-connecting a graph G_0 , where G_0 is connected. In § 3, we have shown that this problem is NP-hard. We make several observations about solution strategies, and then proceed to a description of our algorithm. First, we note that we may actually consider bridge-connecting a subgraph that is a tree. If we are given G_0 with nontrivial bridge-connected components, we shrink the vertex sets of the bridge-connected

components into corresponding single vertices, resulting in a tree whose edges are bridges in G_0 .

The second observation is that an algorithm should try to connect two leaves together at a time. If we use a greedy heuristic of joining a leaf to the nearest vertex, the edge set so identified may be arbitrarily more expensive than the optimal augmenting set. As an example, consider the graph in Fig. 2. The tree consists of the path from u to v . Edges that may be used in the augmentation are shown with their costs. All other

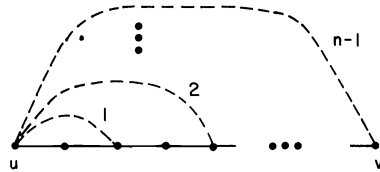


FIG. 2.

edges have cost n . The optimal augmentation is $\{(u, v)\}$, of cost $n - 1$. If the shortest edge from a leaf is always chosen (and the resulting cycle collapsed to a leaf), then all the augmentation edges will be used, at cost $n(n - 1)/2$.

Our final observation is that there is an element of directedness to this problem, which at first glance appears undirected. Consider the tree in Fig. 3, which consists of a path from u to v . Suppose the optimal augmenting set is $\{(u, t), (s, v)\}$. The path used to

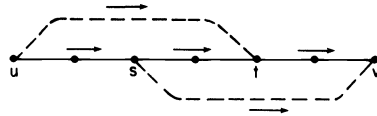


FIG. 3.

connect leaves u and v is $(u, t), (t, s), (s, v)$, where (t, s) is already in G_0 . In fact, if we think of this augmentation as going from u to v , then we note that by traversing from t to s , we are temporarily going backwards in the tree.

We thus see motivation for directing edges in our tree. We arbitrarily choose a leaf as a root r in our tree, and direct all edges in the tree toward r . We then use a minimum weight arborescence algorithm to find a directed tree out of r . The edges in the arborescence, when added to the directed tree, will form a strongly connected graph. The corresponding edges will form a bridge-connected graph.

Because we are transforming an undirected problem into a directed problem, we must be careful in handling an example such as the graph in Fig. 4a. If x is chosen as the root, then strongly connecting the directed tree in Fig. 4b appears to be expensive. However, the bridge augmentation would only use (u, v) as opposed to the edge from $\{(t, u), (x, u), (w, u)\}$. We thus develop the following distance function, which counteracts poor choices in directing edges in the tree. Let $d(u, v) = \min\{c(x, y) \mid u \text{ and } v \text{ are on a path from } x \text{ to } y \text{ in } T\}$. We present algorithm DIST for computing d .

ALGORITHM DIST.

Input. $G_s = (V_s, E_s)$, a complete undirected graph with weight function $c': E_s \rightarrow \mathbb{Z}^+ \cup \{\infty\}$ and edge-naming function b , and $T = (V_s, E'_s)$, a spanning tree of G .

Output. Weight function $d: E_s \rightarrow \mathbb{Z}^+ \cup \{\infty\}$ such that $d(u, v) = \min\{c(x, y) \mid u \text{ and } v \text{ are on a path from } x \text{ to } y \text{ in } T\}$ and $b': E_s \rightarrow E_s$ such that $c'(b'(u, v)) = d(u, v)$.

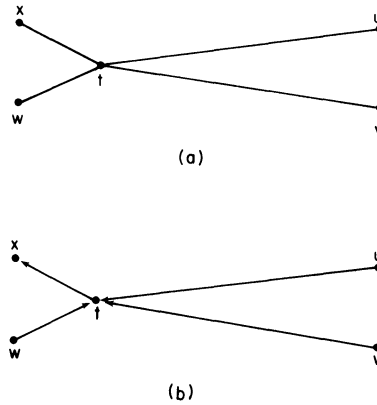


FIG. 4.

1. For each pair of vertices u and v , find $a(u, v)$, the number of edges on the path between u and v in T , and $s(u, v)$, the vertex adjacent to v on this path. Let $d(u, v) = c'(u, v)$ and $b'(u, v) = b(u, v)$.

2. Bucketsort the edges (u, v) of $E_s - E'_s$ into nonincreasing order of $a(u, v)$. For each edge (u, v) in $E_s - E'_s$ do the following in its sorted order: If $d(u, v) < d(u, s(u, v))$, then set $d(u, s(u, v)) = d(u, v)$ and $b'(u, s(u, v)) = b'(u, v)$. If $d(u, v) < d(s(v, u), v)$, then set $d(s(v, u), v) = d(u, v)$ and $b'(s(v, u), v) = b'(u, v)$. \square

LEMMA 2. Algorithm DIST correctly computes the distance function d and edge-naming function b' , and runs in $O(|V|^2)$ time.

Proof. The algorithm is a realization of the following dynamic programming optimality requirement:

$$d(u, v) = \min \{ \{c(u, v)\} \cup \{d(u, t) \mid v \text{ is adjacent to } t \text{ and on the path from } u \text{ to } t\} \cup \{d(t, v) \mid u \text{ is adjacent to } t \text{ and on the path from } v \text{ to } t\} \},$$

for every pair u and v in V .

For a given vertex u and all vertices v , $a(u, v)$ and $s(u, v)$ can be calculated for all v in $O(|V|)$. Hence, Step 1 requires $O(|V|^2)$.

The bucketsort will require $O(|E|)$ time. The updating generated from each edge (u, v) will require constant time; hence, all edges will require $O(|E|)$ time. Since the graph is complete $|E| = O(|V|^2)$. \square

If G_0 is not a tree (but is of course connected), we first shrink all vertices in a bridge-connected component to a single vertex representing that bridge-connected component. Thus, G is shrunk to a complete graph $G_s = (V_s, E'_s)$. The cost c' of an edge (u, v) is the minimum of $\{\infty\} \cup \{c(x, y) \mid (x, y) \in E - E', \text{ and } x \text{ is in the component represented by } u \text{ and } y \text{ is in the component represented by } v\}$. The edge (x, y) so used is referenced by a backpointer $b(u, v) = (x, y)$. If $c'(u, v) = \infty$, then $b(u, v) = (u, v)$.

We now proceed to the specification of algorithm BRC, which bridge-connects a connected subgraph. We note that our handling of distance d' in Step 4b allows us to slide back on edges in the tree, and also forces us to use node r as the root of the arborescence. Also note that we, of course, eliminate duplicates that might occur from insertions into E'' .

ALGORITHM BRC.

Input. $G = (V, E)$, a complete undirected graph with weight function $c: E \rightarrow \mathbb{Z}^+$, and $G_0 = (V, E')$, a spanning subgraph of G .

Output. A set of edges $E'' \subset E - E'$ such that $(V, E' \cup E'')$ is bridge-connected.

1. Shrink G to $G_s = (V_s, E_s)$ and G_0 to $T = (V_s, E'_s)$ such that each bridge-connected component in G_0 is mapped to a different vertex in V_s . Find weight function $c': E_s \rightarrow \mathbb{Z}^+ \cup \{\infty\}$ and $b: E_s \rightarrow E$.

2. Find $d: E_s \rightarrow \mathbb{Z}^+$ and $b': E_s \rightarrow E_s$ using Algorithm DIST.

3. Choose any leaf in V_s to be r . Let A' be the set of directed edges generated by directing each edge in E'_s toward r . Let $T' = (V_s, A')$.

4a. Let $A = \emptyset$. For each edge $(u, v) \in E_s$, insert $\langle u, v \rangle$ and $\langle v, u \rangle$ into A . Set $d'\langle u, v \rangle = d'\langle v, u \rangle = d(u, v)$.

b. If $\langle u, v \rangle \in A'$, then set $d'\langle u, v \rangle = 0$. For all u , set $d'\langle u, r \rangle = \infty$.

5. Find a minimum weight arborescence $T'' = (V_s, A'')$ on (V_s, A) , using d' .

6. For each edge $\langle u, v \rangle$ in A'' with $d'\langle u, v \rangle > 0$, insert the corresponding edge $b(b'\langle u, v \rangle)$ into E'' . \square

THEOREM 5. *Algorithm BRC generates a set E'' such that $(V, E' \cup E'')$ is bridge-connected.*

Proof. Step 5 finds a set A'' , such that $(V_s, A' \cup A'')$ is strongly connected. Thus $(V_s, E_s \cup \tilde{E})$ is bridge-connected (where \tilde{E} are the undirected versions of edges in A'' , and we allow \cup to be a multiset union). If not, then there would be a bridge (u, v) with $\langle u, v \rangle \in A' \cup A''$. There would be no path from v to u in $(V_s, A' \cup A'')$, hence a contradiction to $(V_s, A' \cup A'')$ being strongly connected.

Now replacing \tilde{E} with $E'' = b'(\tilde{E})$ will leave the graph bridge-connected, since if (u, v) creates a cycle in (V_s, E'_s) , certainly $b'(u, v)$ will create a cycle containing all vertices in the cycle created by (u, v) . Reconstituting $(V_s, E'_s \cup E''_s)$ into $(V, E' \cup E'')$, where $E'' = b(E''_s) - E'$, will leave the resulting graph bridge-connected. \square

LEMMA 3. *Let $E^* \subset E - E'$ be a minimum-cost set of edges such that $(V, E' \cup E^*)$ is bridge-connected. Then there is an augmentation $A''' \subset A - A'$, of cost no more than twice that of E^* , such that $(V_s, A' \cup A''')$ is strongly connected.*

Proof. Let (u, v) be an edge in E^* . Let (x, y) be the corresponding edge when V is shrunk to V_s . Consider the weakly directed path between x and y in T' . If the edge directions do not change along the path, direct (x, y) to complete a directed cycle and insert it into A''' . If the edge directions change, then they change at most once, say at vertex w . Insert $\langle x, y \rangle$ and $\langle w, x \rangle$ into A''' . By the definition of the distance functions d', d and c' , $d'\langle w, x \rangle \leq c'(x, y) \leq c(u, v)$. Hence, each edge (u, v) in E^* accounts for at most two directed edges in A''' , each of cost no more than (u, v) . Since every edge in T must be on a cycle containing exactly one edge in E^* , then every directed edge in T' must be on a directed cycle in $(V_s, A' \cup A''')$, and hence, this graph is strongly connected. \square

THEOREM 6. *Let $G = (V, E)$ be a complete undirected graph with cost function $c: E \rightarrow \mathbb{Z}^+$ and $G_0 = (V, E')$ a connected subgraph of G . Let C^* be the cost of an optimal augmentation E^* that makes G_0 bridge-connected, and let \hat{C} be the cost of the edges E'' generated by algorithm BRC. Then*

$$\frac{\hat{C}}{C^*} \leq 2.$$

Furthermore, there exists a family of examples with all edges of cost 1 such that the bound can be approached arbitrarily closely.

Proof. From Lemma 3, we know that there is an augmentation A''' of cost no greater than $2C^*$, such that $(V_s, A' \cup A''')$ is strongly connected. From A''' , we may construct an arborescence for (V, A') using d' , and with any particular vertex that we choose to be the root. The cost \hat{C} will be no greater than the cost of the minimum-cost

arborescence, which will be no greater than the cost of the induced arborescence, which will be no greater than $2C^*$.

To see that the bound is approachable, consider $G_0 = (V, E')$, which is a binary tree with n leaves. Let $c(u, v) = 1$ for all $u, v \in V, u \neq v$. The algorithm will choose a leaf at random to be the root r , and direct all tree edges toward r . A spanning arborescence must have a directed edge entering each of the remaining $n - 1$ leaves, so that a potential solution is the set of edges connecting r with each other leaf, which is of cost $\hat{C} = n - 1$. An optimal solution will have $n/2$ edges, each edge connecting two leaves, for a cost $C^* = n/2$. \square

We note that Algorithm BRC has time complexity $O(|V|^2)$. Step 2 requires $O(|V|^2)$ time, and Step 5 can be accomplished in $O(|V|^2)$ time [T1]. Edge costs may be contained in an adjacency matrix, and may be manipulated in $O(|V|^2)$ time.

If the subgraph G_0 is not connected, then we modify BRC so that the connected components are joined in minimum cost fashion. Prim's algorithm for finding a minimum spanning tree [P] may be used to accomplish this in $O(|V|^2)$ time. Since the cost of the minimum connection will be no larger than the cost of an optimum bridge-connection C^* , the worst-case bound on this algorithm is $\hat{C}/C^* \leq 3$.

The bound is approachable, as can be seen in the example of Fig. 5. Here, G_0 consists of k copies of a connected component containing vertices $v_{i1}, v_{i2}, w_{i1}, w_{i2}$. All

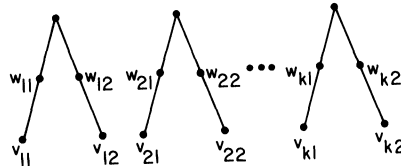


FIG. 5.

edge costs are 1. A minimum cost connection may consist of edges $\{(w_{i2}, w_{i+1,1}) | 1 \leq i < k\}$ of total cost $k - 1$. If v_{k2} is chosen as a root, then a minimum cost augmentation would consist of $\{(v_{i1}, v_{i2}) | 1 \leq i \leq k\} \cup \{(v_{i2}, v_{i+1,1}) | 1 \leq i < k\}$ of cost $2k - 1$. Thus, $\hat{C} = 3k - 2$. An optimal augmentation would be $\{(v_{k2}, v_{11})\} \cup \{(v_{i2}, v_{i+1,1}) | 1 \leq i < k\}$ of cost $C^* = k$.

For G_0 completely disconnected, use the previous example with edges that were previously given in the k components now having cost ϵ . A spanning tree of cost $(k - 1)(1 + 4\epsilon)$ will yield the same difficult topology, and the algorithm can have $\hat{C} = 3k - 2 + (k - 1)4\epsilon$ and $C^* = k + (k - 1)4\epsilon$. As ϵ goes to zero, the bound is 3. If the cost function satisfies the triangle inequality, then we can do much better with another algorithm [FJ].

6. Biconnectivity augmentation algorithm. Finding an augmentation that biconnects a graph seems to involve difficulties that are not present in the bridge-connectivity augmentation problem. In the case of finding an optimal augmentation for unweighted graphs, the $O(|V| + |E|)$ algorithm for biconnection in Rosenthal and Goldner [RG] is considerably more involved than the $O(|V| + |E|)$ algorithm for bridge connection in Eswaran and Tarjan [ET]. We have found a similar increase in complication in the weighted case in moving from bridge-connectivity to biconnectivity augmentation approximation algorithms.

The block-cutvertex tree described below is used to handle not only biconnected components (blocks), but cutvertices as well. In Step 3, when the directed tree is set up, we must add new vertices "upwind" of the cutvertices. The way in which we handle

edges that enter and leave the cutvertices and new vertices will ensure that strongly connecting the graph guarantees not just bridge-connectivity but also biconnectivity in the original graph.

The structure of this section is much the same as § 5: we first present an algorithm for biconnecting a connected graph. The approach will be quite similar to that presented in § 5, and the reader will notice a similar structure in the algorithm and the proofs. However, the increased level of complication in the algorithm will find its way into the proofs of correctness and of bound behavior. We now define the block-cutvertex tree.

Let $G_0 = (V, E')$ be a connected subgraph of graph $G = (V, E)$. Let $G^i = (V^i, E^i)$, $1 \leq i \leq t$, be the biconnected components (blocks) of G_0 , and let V_c be the cutvertices of G_0 . The *block-cutvertex tree* $T_s = (V_s, E'_s)$ is defined as follows. Let V_b be a set of t new vertices, called block-vertices, representing the blocks of G_0 . Let $V_s = V_c \cup V_b$. Let $E'_s = \{(v_i, v_j) | v_i \in V_b \wedge v_j \in V_c \wedge v_j \in V^i\}$; that is, each edge connects a block vertex v_i with a cutvertex v_j such that the cutvertex is in the corresponding block. Hence, cutvertices and block vertices alternate in the block-cutvertex tree T_s . The *block-cutvertex graph* $G_s = (V_s, E_s)$ induced by G_0 will have $E_s = \{(v_i, v_j) | v_i, v_j \in V_s \wedge i \neq j\}$.

We define cost function c' on E_s in terms of cost function c , but being careful about the way cutvertices are handled. If $v_i, v_j \in V_c$, then set $c'(v_i, v_j) = c(v_i, v_j)$. If $v_i \in V_b$ and $v_j \in V_c$, then set $c'(v_i, v_j) = \min \{\{\infty\} \cup \{c(v_k, v_j) | v_k \in (V^i - V_c)\}\}$. If $v_i, v_j \in V_b$, then set $c'(v_i, v_j) = \min \{\{\infty\} \cup \{c(v_k, v_m) | v_k \in (V^i - V_c) \wedge v_m \in (V^j - V_c)\}\}$. Our cost function c' has been defined as though we do not include cutvertices in blocks. We can do this since we have individual vertices in V_s representing the cutvertices, and hence need not include them in the block vertices. We allow the possibility of an edge with cost ∞ , since all vertices in some block may be cutvertices. Finally, for each edge (u, v) in E_s , let $b(u, v)$ be a pointer to an edge (x, y) in E with $c(x, y) = c'(u, v) < \infty$. If $c'(u, v) = \infty$, let $b(u, v) = (u, v)$.

Before presenting the algorithm, we give an example, in Fig. 6, of how the cutvertices must be handled carefully. In Fig. 6a, we have a block-cutvertex tree. In Fig.

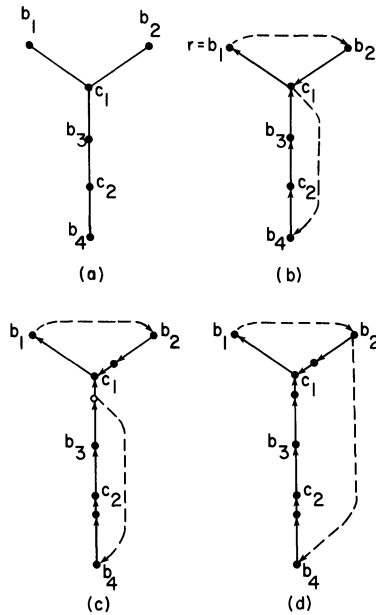


FIG. 6.

6b, we give a directed tree without introducing the new vertices of Step 3b. The dotted edges give a strong connection for the directed tree, but a cutvertex still remains in the corresponding undirected graph. In Fig. 6c, we show a directed tree with the new vertices included. If the same augmentation is used, as shown, then this augmentation does not strongly connect the tree. We cannot use an edge $\langle c_1, b_4 \rangle$ in the augmentation, since no such edge is introduced in Step 4a. Hence, an augmentation such as that in Fig. 6d must be used, and this both strongly connects the tree, and also induces a biconnection in the original graph.

ALGORITHM BIC.

Input. $G = (V, E)$, a complete undirected graph with weight function $c: E \rightarrow \mathbb{Z}^+$, and $G_0 = (V, E')$, a spanning subgraph of G .

Output. A set of edges $E'' \subset E - E'$ such that $(V, E' \cup E'')$ is biconnected.

1. Shrink G_0 to its block-cutvertex tree $T_s = (V_s, E'_s)$ and G to $G_s = (V_s, E_s)$, the block-cutvertex graph induced by G_0 . Find weight function $c': E_s \rightarrow \mathbb{Z}^+ \cup \{\infty\}$ and $b: E_s \rightarrow E$.

2. Find $d: E_s \rightarrow \mathbb{Z}^+$ and $b': E_s \rightarrow E_s$ using Algorithm DIST.

3a. Choose any leaf in V_s to be r . Let A' be the set of directed edges generated by directing each edge in E'_s toward r . Set V'_s to V_s .

b. For each edge $\langle u, v \rangle$ in A' such that $v \in V_c$, insert new vertex x_{uv} in V'_s , and replace $\langle u, v \rangle$ in A' with $\langle u, x_{uv} \rangle$ and $\langle x_{uv}, v \rangle$. Let $T'_s = (V'_s, A')$.

4a. Set A to A' . For each edge $(u, v) \in E_s - E'_s$, do the following. If $u \notin V_c$ or there is no path from v to u in T'_s , then insert $\langle u, v \rangle$ into A with $d'\langle u, v \rangle = d(u, v)$. Otherwise, let x be the (new) vertex preceding u in T'_s , and insert $\langle x, v \rangle$ into A with $d'\langle x, v \rangle = d(u, v)$ and set $b'(x, v) = b'(u, v)$. Perform the above with the roles of u and v reversed.

b. If $\langle u, v \rangle \in A'$, then set $d'\langle u, v \rangle = 0$. For all u , set $d'\langle u, r \rangle = \infty$.

5. Find a minimum cost arborescence $T''_s = (V'_s, A'')$ on (V'_s, A') using d' .

6. For each edge $\langle u, v \rangle$ in A'' with $d'\langle u, v \rangle > 0$, insert the corresponding edge $b(b'(u, v))$ into E'' . \square

THEOREM 7. *Algorithm BIC generates a set E'' such that $(V, E' \cup E'')$ is biconnected.*

Proof. Step 5 finds a set A'' such that $(V'_s, A' \cup A'')$ is strongly connected. Let v be a cutvertex in G_0 that separates vertices u and w . We shall show that there is a weakly directed path from w to u in $(V'_s, A' \cup A'')$ that does not contain v . Hence, we shall show that the edges in A'' , when undirected, will eliminate vertices in V_c as cutvertices.

Let P_v be the set of vertices in V'_s such that there is a strongly directed path in T'_s from each vertex in P_v to v . Since $(V'_s, A' \cup A'')$ is strongly connected, there is a strongly directed path from v to any other vertex. The first edge in the path cannot be $\langle v, t \rangle$, where $t \in P_v$, since no edge $\langle v, t \rangle$ is added to A in Step 3b such that $v \in V_c$ and $t \in P_v$. Hence, a directed path in $(V'_s, A' \cup A'')$ from v to any other node x must contain a directed subpath from s to x , where $s \notin P_v$ and v is not in the subpath. Thus, there is a directed subpath not containing v from s_1 to u for some $s_1 \notin P_v$. Similarly, there is a directed subpath from s_2 to w for some $s_2 \notin P_v$. Furthermore, since $s_1, s_2 \in P_v$ and v has only one successor in T'_s , there is a weakly directed path from s_1 to s_2 in T'_s that does not contain v . Hence, there is a weakly directed path from u to s_1 to s_2 to w in $(V'_s, A' \cup A'')$ that does not contain v .

As in Theorem 5, we note that replacing $\langle x, y \rangle$ in A'' with the corresponding edge $b(b'(x, y))$ does not adversely affect biconnectivity. Let $b(b'(x, y)) = (p, q)$. If $\langle x, y \rangle$ is on a weakly directed path that eliminates v as a cutvertex between u and w , then v cannot be on the path from x to p in T or on the path from y to q in T . \square

LEMMA 4. Let $E^* \subset E - E'$ be a minimum cost set of edges such that $(V, E' \cup E^*)$ is biconnected. Then there is an augmentation $A''' \subset A - A'$, of cost no more than twice that of E^* , such that $(V_s, A' \cup A''')$ is strongly connected.

Proof. We describe an algorithm for generating A''' from E^* . Start with $A''' = \emptyset$. Mark all vertices inaccessible except the root of T'_s . While not all vertices are accessible, do the following: Choose an accessible vertex $w \in V_b$ such that w is on the path from x to y in T_s for some edge (x, y) corresponding to an unexamined edge $(u, v) \in E^*$. If $w \neq y$, insert $\langle w, y \rangle$ into A''' , and if $w \neq x$, insert $\langle w, x \rangle$ into A''' . Mark all vertices on the weakly directed path from x to y in T'_s accessible.

Since $w \in V_b$, edges $\langle w, y \rangle$ and $\langle w, x \rangle$ exist in A and have d' cost no greater than $c(u, v)$. Also, if w is accessible from the root then introduction of the two directed edges will make all vertices on the weakly directed path from x to y accessible from the root.

It remains to show that an edge (u, v) from E^* and suitable accessible vertex w always exist. At any point in the execution of the above algorithm the vertices V_e marked as accessible represent a set of vertices that is biconnected. If E^* has not been completely examined then there must be an edge (x, y) corresponding to an unexamined edge (u, v) in E^* such that the path between x and y in T_s contains more vertices in V_e than just one cutvertex. Otherwise, E^* cannot neutralize all cutvertices from V_c . If there is more than one vertex from V_e on a path between x and y in T_s , then there must be a $w \in V_b \cap V_e$ on this path, since blockvertices and cutvertices alternate in T_s . \square

THEOREM 8. Let $G = (V, E)$ be a complete undirected graph with cost function $c: E \rightarrow \mathbb{Z}^+$ and $G_0 = (V, E')$ a connected subgraph of G . Let C^* be the cost of an optimal augmentation E^* that makes G_0 biconnected, and let \hat{C} be the cost of the edges E'' generated by Algorithm BIC. Then

$$\frac{\hat{C}}{C^*} \leq 2.$$

Furthermore, there exists a family of examples with all edges of cost 1 such that the bound can be approached arbitrarily closely.

Proof. Similar to the proof of Theorem 6. \square

Like Algorithm BRC, Algorithm BIC has the complexity $O(|V|^2)$. If the subgraph G_0 is not connected, then we first connect the graph, in minimum-cost fashion, and then run BIC. The worst-case bound is 3, as in the case of bridge-connection.

REFERENCES

- [CT] D. CHERITON AND R. TARJAN, *Finding minimum spanning trees*, this Journal, 5(1976), pp. 724-742.
- [CL] Y. J. CHU AND T. H. LIU, *On the shortest arborescence of a directed graph*, Sci. Sinica, 14(1965), pp. 1396-1400.
- [C] S. COOK, *The complexity of theorem-proving procedures*, Proc. 3rd Annual ACM Symposium on the Theory of Computing, 1971, pp. 151-158.
- [E] J. EDMONDS, *Optimum branchings*, J. Res. Nat. Bur. Standards Sect. B, 71(1967), pp. 233-240.
- [ET] K. P. ESWARAN AND R. E. TARJAN, *Augmentation problems*, this Journal, 5(1976), pp. 653-665.
- [FJ] G. N. FREDERICKSON AND J. JA'JA', *On the relationship between the biconnectivity augmentation and traveling salesman problems*, TR CS-79-41, Pennsylvania State University, 1979.
- [GJ1] M. R. GAREY AND D. S. JOHNSON, *Approximation algorithms for combinatorial problems: an annotated bibliography*, in Algorithms and Complexity: Recent Results and New Directions, J. F. Traub, ed., Academic Press, New York, 1976, pp. 41-52.

- [GJ2] ———, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.
- [J] D. S. JOHNSON, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9(1974), pp. 256–278.
- [J̄] D. B. JOHNSON, *Priority queues with update and finding minimum spanning trees*, Inform. Process. Lett., 4(1975), pp. 53–57.
- [K1] 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. 85–104.
- [K2] ———, *On the complexity of combinatorial problems*, Networks, 5(1975), pp. 45–68.
- [P] R. C. PRIM, *Shortest connection networks and some generalizations*, Bell System Tech. J., 36(1957), pp. 1389–1401.
- [RG] A. ROSENTHAL AND A. GOLDNER, *Smallest augmentations to biconnect a graph*, this Journal, 6(1977), pp. 55–66.
- [T1] R. TARJAN, *Finding optimum branchings*, Networks, 7(1977), pp. 25–35.
- [T2] ———, private communication, 1979.
- [Y] A. C. YAO, *An $O(|E|\log \log |V|)$ algorithm for finding minimum spanning trees*, Inform. Process. Lett., 4(1975), pp. 21–23.

THE RATIONAL INDEX: A COMPLEXITY MEASURE FOR LANGUAGES*

LUC BOASSON,† BRUNO COURCELLE‡ AND MAURICE NIVAT†

Abstract. With every language L we associate an increasing function called its rational index. We obtain this function by comparing L with rational languages of increasing complexity.

We show that the rational indices of two languages related by a rational transduction are polynomially related. From this, we can define new rational cones of languages in terms of rational indices.

We then focus our attention on the rational index of context-free languages and raise several questions closely related to the open problems concerning the subcones of the family of context-free languages.

Key words. complexity, context-free language, rational transduction, generator language

Introduction. Let \mathcal{L} be the family of languages defined by a family \mathcal{A} of “devices” (e.g., grammars, automata, Turing machines). Each “device” A has a finite size $|A|$ ($|A| \in \mathbb{N}$) and defines a language $L(A)$.

The relation between the size of a device A and the length of a shortest word in $L(A)$ measures in some sense the complexity of the family \mathcal{A} . It can be expressed by a function over the integers,

$$\rho_{\mathcal{A}}(n) = \text{Max} \{ \text{Min} \{ |u| \mid u \in L(A), L(A) \neq \emptyset \text{ and } |A| \leq n \} \}.$$

For \mathcal{A} the family of finite automata (and $|A|$ the number of states of A in \mathcal{A}) $\rho_{\mathcal{A}}$ is bounded by a linear function.

For \mathcal{A} the family of context-free grammars (and $|A|$ the length of the grammar A written as a word made of all production rules) $\rho_{\mathcal{A}}$ is bounded by $\lambda n \cdot n^n$ but not by any linear function.

In order to study the complexity of one language L , we choose to consider the family \mathcal{L} of languages of the form $L \cap L(A)$, where A is a finite nondeterministic automaton such that $L \cap L(A) \neq \emptyset$. We introduce the rational index of L as the function $\rho_L: \mathbb{N} \rightarrow \mathbb{N}$ defined by

$$\rho_L(n) = \text{Max} \{ \text{Min} \{ |u| \mid u \in L \cap L(A), L \cap L(A) \neq \emptyset, A \in \mathcal{A}_n \} \},$$

where $L \subseteq X^*$ and \mathcal{A}_n is the set of all finite nondeterministic automata with input alphabet X and at most n states.

The order of ρ_L (linear, polynomial, exponential) will be used as a measure of the complexity of L .

This complexity measure behaves properly through the AFL operations. In particular, if L is the image of L' by a rational transduction, then ρ_L and $\rho_{L'}$ are polynomially related. Thus, we can define rational cones in terms of this complexity.

We focus our attention on context-free languages and raise several questions closely related to the open problems concerning the cones of nongenerator languages. We show in particular, that every generator is of exponential order.

* Received by the editors July 20, 1979, and in revised form May 15, 1980. A first version of this paper was presented at the Conference on Theoretical Computer Science, Waterloo, Ontario, Canada, August, 1977.

† Université de Paris VII, L.I.T.P. tour 55; 2, Place Jussieu, 75221 Paris Cedex 05, France.

‡ Université de Bordeaux I, Mathématiques et Informatique; 351, Cours de la Libération, 33405 Talence Cedex, France.

Rational indices have been used by J. M. Steyaert for studying ETOL languages [10] and by W. Damm to show that a certain hierarchy of languages which extends the Chomsky hierarchy is strict [4].

1. The rational index of a language. Let us begin with some definitions and notation.

Let \mathbb{N}_+ and \mathbb{R}_+ denote the sets of positive integers and reals respectively.

By *function* we mean an increasing mapping of \mathbb{N}_+ into \mathbb{R}_+ . We shall use the notation borrowed from the lambda-calculus: $\lambda n.f(n)$ is the function which maps n to $f(n)$ for $n \in \mathbb{N}_+$. For instance, $\lambda n.n$ is the identity function and $\lambda n.n^2$ the square function. We shall also use small letters f, g, h, \dots to denote functions.

Our functions are ordered as follows:

$$f \leq g \text{ if and only if, for all } n, f(n) \leq g(n).$$

Hence, by $\text{Max}(f, g)$ we mean $\lambda n.\text{Max}\{f(n), g(n)\}$, which is clearly the least upper bound of f and g for the order \leq .

For a function f , we define the set $\Omega(f)$ of all functions g such that for some $M \in \mathbb{R}_+$, $g \leq M.f$.

The relation $g \in \Omega(f)$ is a preorder that we shall denote by $g \leq f$. By $g < f$, we shall mean that $g \leq f$ holds but $f \leq g$ does not.

Remark 1. For any two functions f and g , $f \leq g$ if and only if there exist M and N such that $f(n) \leq M.g(n)$ for all $n \geq N$.

By $f + g$ and $f.g$ we shall mean respectively $\lambda n.(f(n) + g(n))$ and $\lambda n(f(n).g(n))$. The identity function $\lambda n.n$ will be denoted by Id .

We now introduce some notation concerning languages.

Let $\mathcal{A}_n(X)$ be the family of nondeterministic finite automata with at most n states and some input alphabet X . Let $\text{Rat}_n(X)$ be the family of languages accepted by the automata in \mathcal{A}_n ; i.e., $\text{Rat}_n(X) = \{L(A) \mid A \in \mathcal{A}_n(X)\}$.

The general automaton will be of the form $\langle X, Q, \lambda, q_0, F \rangle$, where X is the input alphabet, Q the set of states, q_0 the initial state, F the set of final states and $\lambda : Q \times X \rightarrow P(Q)$ the nondeterministic transition function.

The function λ is canonically extended into $\lambda^* : Q \times X^* \rightarrow P(Q)$ by $\lambda^*(q, \varepsilon) = \{q\}$ and $\lambda^*(q, au) = \cup \{\lambda^*(q', u) \mid q' \in \lambda(q, a)\}$, where $u \in X^*$ and $a \in X$.

The length of a word u in X^* is denoted by $|u|$, and for any $L \subseteq X^*$ we denote by $\text{MIN}(L)$ the set of words of L of minimal length. The empty word is denoted by ε .

For any two languages K and L such that $K \cap L \neq \emptyset$, we define $\delta_{K,L}$ to be the common length of all words in $\text{MIN}(K \cap L)$.

We are now able to define the *rational index* ρ_L of a nonempty language $L \subseteq X^*$:

$$\rho_L = \lambda n.\max \{\delta_{K,L} \mid K \cap L \neq \emptyset \text{ and } K \in \text{Rat}_n(X)\}.$$

Since $\text{Rat}_n(X) \subseteq \text{Rat}_{n+1}(X)$ for all n , ρ_L is a function in our sense.

For any class of functions \mathcal{C} we define $\Omega(\mathcal{C})$ as the union of all the $\Omega(f)$ for all f in \mathcal{C} . We define a class of languages as follows:

$$\Omega(\mathcal{C}) = \{L \subseteq X^* \mid \rho_L \in \Omega(\mathcal{C})\}.$$

We also define $\pi(f)$ as the set of functions g such that $f \leq g$ (i.e., $f \in \Omega(g)$) and $\pi(\mathcal{C})$ as the union of the $\pi(f)$ for $f \in \mathcal{C}$. We also get

$$\pi(\mathcal{C}) = \{L \subseteq X^* \mid \rho_L \in \pi(\mathcal{C})\}.$$

The function ρ_L seems to depend on the alphabet X , but this is not the case, provided

$L \subseteq X^*$. Let Y be the least alphabet such that $L \subseteq Y^*$. Then, for all $A \in \mathcal{A}_n(X)$,

$$L \cap L(A) = L \cap L(A'),$$

where $A' \in \mathcal{A}_n(Y)$ is the automaton obtained from A by deleting every arc labeled by a symbol not in Y .

Hence, in the rest of the paper we shall always consider “sufficiently large alphabets” without needing to be very precise on this point.

We now begin to investigate the relation between a language and its rational index.

Remark 2. A language L is infinite if and only if $\text{Id} \leq \rho_L$.

Remark 3. For every function $f \in \pi(\text{Id})$ there exists a language L such that $f < \rho_L$.

This means that there exist arbitrarily complex languages. Let us show this. Let

$$X = \{x, y\}, \quad L = \{x^n y^{(n+2)f(n+2)} \mid n \in \mathbb{N}_+\}, \quad K = x^{n-2} y^*.$$

Note that $K \in \text{Rat}_n(X)$ and that $\delta_{K,L} = n - 2 + nf(n)$. Hence, $f \leq \rho_L$ and $\rho_L \not\leq f$.

We shall now relate the rational indices of two languages L and L' with the rational index of $L \cup L'$, LL' , L^* , $\tau(L)$, where τ is a rational transduction.

LEMMA 1. $\rho_{L \cup L'} = \text{Max}(\rho_L, \rho_{L'})$.

We leave the proof to the reader and state

LEMMA 2. $\rho_{LL'} \leq \rho_L + \rho_{L'}$.

Proof. Let $K \in \text{Rat}_n(X)$ be $L(A)$ for the automaton $A = \langle X, Q, \lambda, q_0, F \rangle$. Let $u \in \text{MIN}(K \cap LL')$. There exist $q \in Q$, $q' \in F$ and $u_1, u_2 \in X^*$ such that

$$u_1 \in L \quad \text{and} \quad u_2 \in L',$$

$$u = u_1 u_2,$$

$$q \in \lambda^*(q_0, u_1),$$

$$q' \in \lambda^*(q, u_2).$$

Let $A_1 = \langle X, Q, \lambda, q_0, \{q\} \rangle$ and $A_2 = \langle X, Q, \lambda, q, F \rangle$; since u is minimal (in length) in $K \cap LL'$, u_1 is minimal in $K_1 \cap L$ and u_2 is minimal in $K_2 \cap L'$. Hence,

$$\rho_{LL'}(n) \leq |u| = |u_1| + |u_2| \leq \rho_L(n) + \rho_{L'}(n). \quad \square$$

This bound cannot be improved, as shown by the example of $L = L' = a^*b$. By taking $K = (a^q b)^*$ we get $\delta_{K,L} = \delta_{K,L'} = q + 1$ and $\delta_{K,LL'} = 2q + 2$.

LEMMA 3. $\rho_{L^*} \leq \text{Id} \cdot \rho_L$.

This means that for all n , $\rho_{L^*}(n) \leq n \cdot \rho_L(n)$.

Proof. Let A be as in the proof of Lemma 2 and $u \in \text{MIN}(K \cap L^*)$ with $u \neq \varepsilon$. Then for some $u_0, u_1, \dots, u_k \in X^+$, some p_1, p_2, \dots, p_{k+1} in Q ,

$$u = u_0 u_1 u_2 \dots u_k,$$

$$p_1 \in \lambda^*(q_0, u_0),$$

$$p_2 \in \lambda^*(q_1, u_1),$$

$$\vdots$$

$$p_{k+1} \in \lambda^*(p_k, u_k),$$

$$p_{k+1} \in F.$$

Since u is minimal, $q_0, p_1, p_2 \cdots p_{k+1}$ must be distinct. Hence, $k + 2 \leq n$. Also, $|u_i| \leq \rho_L(n)$ for $i = 0, 1, \dots, k + 1$ for the same reason. Hence,

$$|u| \leq (k + 1)\rho_L(n) \leq n \cdot \rho_L(n). \quad \square$$

LEMMA 4. Let $R \in \text{Rat}_q(X)$. Then $\rho_{L \cap R} \leq \lambda n \cdot \rho_L(qn)$.

Proof. If $K \in \text{Rat}_n(X)$ and $R \in \text{Rat}_q(X)$ then $K \cap R \in \text{Rat}_{qn}(X)$ (see Eilenberg [5]). And $\delta_{K, L \cap R} = \delta_{K \cap R, L}$, whence the result. \square

A homomorphism of free monoids $\Phi: X^* \rightarrow Y^*$ is *alphabetic* if $\Phi(X) \subseteq Y \cup \{\varepsilon\}$.

LEMMA 5. Let Φ be an alphabetic homomorphism. Then $\rho_{\Phi(L)} \leq \rho_L$.

Proof. Let us recall from Eilenberg [5] that if $K \in \text{Rat}_n(X)$ then $\Phi^{-1}(K) \in \text{Rat}_n(X)$.

Then $\Phi(L) \cap K = \Phi(L \cap \Phi^{-1}(K))$. Let $u \in \text{MIN}(\Phi(L) \cap K)$; then $u = \Phi(v)$ for some $v \in \text{MIN}(L \cap \Phi^{-1}(K))$. Hence, $|u| \leq |v| \leq \delta_{L, \Phi^{-1}(K)} \leq \rho_L(n)$, if we assume that $K \in \text{Rat}_n(X)$. \square

LEMMA 6. Let Φ be an alphabetic homomorphism. Then, $\rho_{\Phi^{-1}(L)} \leq \lambda n \cdot (n + 1)\rho_L(n)$; hence, $\rho_{\Phi^{-1}(L)} \leq \text{Id} \cdot \rho_L$.

Proof. Let $K \in \text{Rat}_n(X)$ and $u \in \text{MIN}(\Phi^{-1}(L) \cap K)$. Then u can be written as

$$\begin{aligned} u &= w_0 a_1 w_1 a_2 \cdots w_{n-1} a_m w_m, \\ \Phi(w_i) &= \varepsilon \quad \text{for } i = 0, 1, \dots, m, \\ \Phi(a_i) &= a'_i \neq \varepsilon \quad \text{for } i = 1, \dots, m. \end{aligned}$$

Since u is minimal, $u' = a'_1 a'_2 \cdots a'_m \in \text{MIN}(L \cap \Phi(K))$ and $m \leq \rho_L(n)$ since $\Phi(K) \in \text{Rat}_n(X)$. By an application of the pumping lemma, one can see that $|w_i| \leq n - 1$ for $i = 0, 1, \dots, m$. Hence,

$$|u| \leq (n - 1)\rho_L(n) + n - 1 \leq (n + 1)\rho_L(n). \quad \square$$

Remark 4. Let us show that Lemma 6 cannot be improved.

Let $L = \{a^n b^m \mid m, n \in \mathbb{N}_+, m \leq n\}$. One shows with the pumping lemma that $\rho_L \leq \text{Id}$. Now let $\Phi: \{a, b, c\}^* \rightarrow \{a, b\}^*$ such that $\Phi(a) = a$, $\Phi(b) = b$, $\Phi(c) = \varepsilon$ and $L' = \Phi^{-1}(L)$. Then $\lambda n \cdot n^2 \leq \rho_{L'}$. To show this, one takes $K = (ac^q)^* b^q \in \text{Rat}_{2q+1}(\{a, b, c\})$. Then $\delta_{KL'} = q^2 + q$. Hence, $\rho_{L'}(n) \geq \lambda n \cdot n^2 / 4$. \square

Putting together Lemmas 4, 5 and 6, we get the fundamental theorem:

THEOREM 1. Let L be a language and τ a rational transduction. There exists an integer p such that $\rho_{\tau(L)} \leq \lambda n \cdot (pn + 1)\rho_L(pn)$.

Proof. Let us recall from Nivat [8] that a rational transduction $\tau: X^* \rightarrow Y^*$ can be written as $\tau(u) = \psi(\Phi^{-1}(u) \cap R)$, where $R \in \text{Rat}_p(Z)$ for some $p \in \mathbb{N}_+$ and some finite alphabet Z , and Φ, ψ are alphabetic homomorphisms of $Z^* \rightarrow X^*$ and of $Z^* \rightarrow Y^*$ respectively.

From all this and with Lemmas 4, 5 and 6, we get

$$\begin{aligned} \rho_{\tau(L)} &\leq \rho_{\Phi^{-1}(L) \cap R} \\ &\leq \rho_{\Phi^{-1}(L)}(pn) \\ &\leq (pn + 1)\rho_L(pn). \end{aligned} \quad \square$$

This theorem allows us to define new families of languages.

DEFINITIONS. A class \mathcal{C} of functions is *extensive* if:

- (1) $\text{Id} \cdot f \in \Omega(\mathcal{C})$ for all f in \mathcal{C} ,
- (2) $\lambda n \cdot f(pn) \in \Omega(\mathcal{C})$ for all f in \mathcal{C} and p in \mathbb{N}_+ .

We give a “dual” definition: a class \mathcal{C} of functions is *antiextensive* if

- (1) $\lambda n.(f(n)/n) \in \pi(\mathcal{C})$ for all f in \mathcal{C} ,
- (2) $\lambda n.f(\lceil n/p \rceil) \in \pi(\mathcal{C})$ for all f in \mathcal{C} , and p in \mathbb{N}_+ , where $\lceil n/p \rceil$ denotes the integral part of n/p .

Examples. The family $\text{Pol} = \{\lambda n.n^k \mid k \in \mathbb{N}_+\}$ is extensive and the family $\text{Exp} = \{\lambda n.2^{\alpha n} \mid \alpha \in \mathbb{R}_+\}$ is antiextensive.

Let us recall that a *cone* is a family \mathcal{L} of languages which is closed by rational transduction, i.e., such that $\tau(L) \in \mathcal{L}$ if $L \in \mathcal{L}$ and τ is a rational transduction.

Dually, we define an *anticone* as a family \mathcal{L} such that $\tau(L) \in \mathcal{L}$ implies $L \in \mathcal{L}$, where τ is a rational transduction.

COROLLARY 1. *If \mathcal{C} is extensive then $\Omega(\mathcal{C})$ is a cone (and even a full AFL); if \mathcal{C} is antiextensive then $\pi(\mathcal{C})$ is an anticone.*

We shall make a special use of $\mathbf{Pol} = \Omega(\text{Pol})$ the cone of *polynomial* languages and of $\mathbf{Exp} = \pi(\text{Exp})$ the anticone of *exponential* languages.

After having studied the basic AFL operations, we consider substitutions.

Let $L \subseteq X^*$ and $\sigma: X \rightarrow P(Y^*)$ be a mapping which associates with every a in X some language $\sigma(a) \subseteq Y^*$. Then, $\sigma(L)$ is the language

$$\sigma(L) = \{u_1 u_2 \cdots u_n \mid \text{there exists } a_1 a_2 \cdots a_n \text{ in } L, \text{ such that } u_i \in \sigma(a_i), \text{ for } i = 1, 2, \dots, n\}.$$

If $\sigma(a) \in \mathcal{L}$ for all $a \in X$ where \mathcal{L} is a family of languages, we call σ an \mathcal{L} -*substitution*. A family of language \mathcal{L} is *substitution closed* if for all $L \in \mathcal{L}$ and all \mathcal{L} -substitutions σ , $\sigma(L)$ belongs to \mathcal{L} .

We shall also use the *syntactic substitution* $L_1 \circ L_2$ of L_2 into L_1 defined as follows for $L_1 \subseteq X^*$ and $L_2 \subseteq Y^*$ with $X \cap Y = \emptyset$:

$$\begin{aligned} L_1 \circ L_2 &= \sigma(L_1), \\ \sigma(a) &= aL_2 \quad \text{for } a \in X. \end{aligned}$$

Equivalently,

$$\begin{aligned} L_1 \circ L_2 &= \{a_1 u_1 a_2 u_2 \cdots a_n u_n \mid a_1, a_2, \dots, a_n \in X, a_1 a_2 \cdots a_n \in L_1 \\ &\quad \text{and } u_i \in L_2 \quad \text{for } i = 1, 2, \dots, n\}. \end{aligned}$$

LEMMA 7. $\rho_{L \circ L'} \leq \lambda n. \rho_L(n)(1 + \rho_{L'}(n))$.

Proof. Let $K = L(A)$ for some finite automaton $A = \langle X \cup Y, Q, \lambda, q_0, F \rangle$ with at most n states. For $q, q' \in Q$, let us define $K_{q,q'}$ as $\{u \in Y^* \mid q' \in \lambda^*(q, u)\}$. It is clear that each $K_{q,q'}$ belongs to $\text{Rat}_n(Y)$.

Let $u \in \text{MIN}((L \circ L') \cap K)$. Then u can be written as $u = a_1 u_1 a_2 u_2 \cdots a_m u_m$, where $u_i \in L'$ for each i and $a_1 a_2 \cdots a_m \in L$. Since u belongs also to K , we get a sequence $p_1, p'_1, p_2, \dots, p_m, p'_m$ in Q such that the following conditions hold:

$$(C) \quad \left\{ \begin{array}{l} p_1 \in \lambda^*(q_0, a_1), \\ p'_1 \in \lambda^*(p_1, u_1), \quad \text{i.e., } u_1 \in K_{p_1, p'_1}, \\ p_2 \in \lambda^*(p'_1, a_2), \\ \vdots \\ p_m \in \lambda^*(p'_{m-1}, a_m), \\ p'_m \in \lambda^*(p_m, u_m), \quad \text{i.e., } u_m \in K_{p_m, p'_m}, \\ p'_m \in F. \end{array} \right.$$

Let K' be the set of all words $x \in X^*$ which can be written $x = a_1 a_2 \cdots a_m$ for some $a_i \in X$, some $m \in \mathbb{N}$, some $u_i \in Y^*$, some $p_1, p'_1, \dots, p_m, p'_m$ satisfying (C). It is not difficult to see that $K' \in \text{Rat}_n(X)$.

The minimality of u implies that each u_i is in $\text{MIN}(K_{p_i, p'_i} \cap L')$; hence $|u_i| \leq \rho_L(n)$. It implies also that $a_1 a_2 \cdots a_m \in \text{MIN}(K' \cap L)$; hence, $m \leq \rho_L(n)$. We therefore get

$$|u| \leq m(1 + \rho_L(n)) \leq \rho_L(n)(1 + \rho_L(n)). \quad \square$$

A class \mathcal{C} of functions is *multiplicative* if for all $f, g \in \mathcal{C}$ then $f.g \in \Omega(\mathcal{C})$.

THEOREM 2. *Let \mathcal{C} be an extensive and multiplicative class of functions. Then $\Omega(\mathcal{C})$ is a substitution-closed full AFL.*

Proof. The family $\mathcal{L} = \Omega(\mathcal{C})$ is closed by union, product and star operation by Lemmas 1, 2 and 3. It is closed by rational transduction by Theorem 1. Hence, \mathcal{L} is a full AFL.

Let σ be an \mathcal{L} substitution and L a language in \mathcal{L} . One can find in \mathcal{L} a language L_σ and a rational transduction such that $\sigma(L) = \tau(L \circ L_\sigma)$. But $L \circ L_\sigma$ belongs to \mathcal{L} by Lemma 7 and so does $\sigma(L)$. Hence, \mathcal{L} is substitution closed. \square

COROLLARY 2. *The family **Pol** of polynomial languages is a substitution-closed full AFL.*

2. Rational indices of context-free languages. Let us first give an upper bound to the rational index of a context-free language in terms of the size of a grammar which generates it.

For a context-free grammar G , let $|G|_1$ be the cardinality of its nonterminal alphabet and $|G|_2$ be the maximum length of the right-hand side of a production in G . By looking at derivation trees, one establishes easily that the length of a shortest word in $L(G)$ (if any) is at most $|G|_2^{|G|_1}$.

PROPOSITION 1. *Let L be the context-free language generated by some grammar G . Then $\rho_L \leq \lambda n.2^{pn^2}$ where $p = |G|_1 \log_2(|G|_2)$.*

Proof. Let $K \in \text{Rat}_n(X^*)$ be given by some finite automaton with set Q of (at most n) states. In order to define $L(G) \cap K$, one constructs a context-free grammar \bar{G} with nonterminals of the form (q, S, q') for S nonterminal in G and $q, q' \in Q$, such that $L(\bar{G}, (q, S, q')) = L(G) \cap K_{q, q'}$ (notation of Lemma 7) for all S, q , and q' . It is clear that $|\bar{G}|_1 \leq n^2 |G|_1$ and $|\bar{G}|_2 = |G|_2$. Hence the result. \square

The best we can do is to construct context-free languages such that $\rho_L \geq \lambda n.2^{pn}$ (see Lemma 8 below).

Open problem 1. Does there exist a context-free language L such that $\rho_L > \lambda n.2^{pn}$ for all p ?

We shall now consider various subfamilies of the family **CF** of all context-free languages.

Let us recall the definitions of some classical languages.

Let $Z_2 = \{a_1, a_2\} \cup \{\bar{a}_1, \bar{a}_2\}$. We define on Z_2^* the congruence \equiv as being the congruence generated by $a_1 \bar{a}_1 \equiv a_2 \bar{a}_2 \equiv \varepsilon$. We denote by $D_2'^*$ the Dyck set over Z_2 ; i.e., the class of ε with respect to this congruence.

We shall consider the language $D_1'^* = D_2'^* \cap \{a_1, \bar{a}_1\}^*$ and the symmetric language $S_2 = D_2'^* \cap \{a_1, a_2\}^* \{\bar{a}_1, \bar{a}_2\}^*$.

We shall also use the language $\Delta_2' \subseteq Z_2^*$ defined by the grammar

$$S \rightarrow a_1 S S \bar{a}_1 + a_2 \bar{a}_2.$$

The least rational cone containing Δ_2' or $D_2'^*$ is the family **CF** of all context-free languages. The least cone containing S_2 (resp. $D_1'^*$) is the family **Lin** of linear languages (resp. the family **Oct** of one-counter languages).

The substitution closure of **Lin** (i.e., the least substitution-closed family of languages containing **Lin**) is the full AFL **Qrt** of quasi-rational languages. The substitution closure of **Oct** (resp. of $\mathbf{Lin} \cup \mathbf{Oct}$) is the family **Ict** of iterated counter languages (resp. the family **Gre** of *Greibach languages*. It has been shown in [1], [2] that **Gre** is strictly included in the family **N Gen** of languages which do not generate **CF**, contradicting a conjecture of Greibach [7]. The name we give to this family comes from our investigations concerning this family. Let us recall that a context-free language L generates **CF** (is a generator) if $\mathbf{CF} = \{\tau(L) \mid \tau \text{ is a rational transduction}\}$. We denote by **Gen** the set of generators.

Let us recall that **N Gen** is the maximal proper subcone of **CF** (see [7]). Note that **Gen** is not an anticone. We only have

$$L \in \mathbf{CF} \text{ and } \tau(L) \in \mathbf{Gen} \text{ implies } L \in \mathbf{Gen}.$$

But we can have a language L , not in **CF**, such that $\tau(L) \in \mathbf{Gen}$ for some rational transduction, for instance, $L = \{a^p u / p \text{ is prime and } u \in D_2^*\}$.

LEMMA 8. *The language Δ_2' belongs to **Exp**.*

Proof. Let us define the following regular sets

$$\begin{aligned} K_1 &= (a_2 \bar{a}_2)^*, \\ K_2 &= (a_1 K_1 \bar{a}_1)^*, \\ &\vdots \\ K_{i+1} &= (a_1 K_i \bar{a}_1)^* \quad \text{for } i \geq 1. \end{aligned}$$

It is easy to see that K_i is in $\text{Rat}_{i+1}(Z_2)$. Note also that for any g in K_i , $|g|_{a_1} = |g|_{\bar{a}_1}$; i.e., g has the same number of a_1 and of \bar{a}_1 . We shall show that $\Delta_2' \cap K_i$ is reduced to a single word f_i and that $|f_i| = 2^{i+1} - 2$.

If $i = 1$ then $\Delta_2' \cap K_1 = \{a_2 \bar{a}_2\}$, hence, $f_1 = a_2 \bar{a}_2$ and $|f_1| = 2^2 - 2 = 2$. Let us assume the property for i and prove it for $i + 1$. Let g be a word in $\Delta_2' \cap K_{i+1}$. It can be written as

$$g = a_1 g_1 \bar{a}_1 a_1 g_2 \bar{a}_1 \cdots a_1 g_n \bar{a}_1, \quad \text{with } g_j \in K_i \text{ for } j = 1, \dots, n.$$

and

$$g = a_1 h_1 h_2 \bar{a}_1, \quad \text{with } h_1, h_2 \in \Delta_2'.$$

Hence,

$$h_1 h_2 = g_1 \bar{a}_1 a_1 g_2 \cdots a_1 g_n.$$

Since $|g_1 \bar{a}_1|_{\bar{a}_1} = |g_1 \bar{a}_1|_{a_1} + 1$, the word $g_1 \bar{a}_1$ cannot be a left factor of any word in Δ_2' , in particular, of h_1 . Hence, $g_1 = h_1 g_1'$ and $g_1' \bar{a}_1 a_1 g_2 \cdots a_1 g_n = h_2$. Similarly, $|g_1' \bar{a}_1|_{\bar{a}_1} = |g_1' \bar{a}_1|_{a_1} + 1$, and we get a contradiction unless $n = 1$. So we get $h_1 h_2 \in K_i$. With the same remarks as before, we get $h_1 \in K_i$ and $h_2 \in K_i$ and hence, using the induction hypothesis, $h_1 = h_2 = f_i$.

Therefore, there is exactly one possible g in $K_{i+1} \cap \Delta_2'$, namely

$$g = f_{i+1} = a_1 f_i f_i \bar{a}_1,$$

and

$$|f_{i+1}| = 2 + 2 \cdot (2^{i+1} - 2) = 2^{i+2} - 2. \quad \square$$

THEOREM 3. **Gen** is included in **Exp**. In words, every generator of the family of context-free languages is exponential.

Proof. Let L be any generator. Then $\Delta_2' = \tau(L)$ for some rational transduction. Lemma 8 and Theorem 1 show then that $\rho_L \geq \lambda n \cdot 2^n$. \square

Remark 5. The converse of Theorem 3 is not true. Boasson has constructed a language which is exponential but is not a generator [3].

We now consider some families of nongenerator languages, and in particular **Lin** and **Oct**.

LEMMA 9. *The languages S_2 and $D_1'^*$ are polynomial.*

Proof. The language S_2 is generated by the grammar $S \rightarrow a_1S\bar{a}_1 + a_2S\bar{a}_2 + \varepsilon$. Let $A \in \mathcal{A}_n(\mathbb{Z}_2)$.

The construction recalled in Proposition 1 yields a linear grammar with n^2 nonterminals, generating $S_2 \cap L(A)$. Hence $\delta_{L(A), S_2} \leq 2n^2$ and $\rho_{S_2} \leq \lambda n \cdot 2n^2$. Let us now consider $D_1'^* \subseteq \{a_1, \bar{a}_1\}^*$. To simplify the notation, we replace a_1 by a and \bar{a}_1 by \bar{a} .

For each word u in $\{a, \bar{a}\}^*$, let us define

$$\beta(u) = |u|_a - |u|_{\bar{a}}.$$

It is well known that a word $u \in \{a, \bar{a}\}^*$ belongs to $D_1'^*$ if and only if $\beta(u) = 0$ and $\beta(v) > 0$ for all left factors v of u .

For $u \in D_1'^*$ let $h(u)$ be the *height* of u defined as $\text{Max} \{\beta(v) \mid v \text{ is a left factor of } u\}$. Let $w(u)$, the *width* of u , be the maximum number of left factors v of u having the same associated integer $\beta(v)$.

It is not difficult to check that, for all $u \in D_1'^*$,

$$|u| \leq h(u) \cdot w(u).$$

Now let $K \in \text{Rat}_n(\{a, \bar{a}\})$ and $u \in \text{MIN}(D_1'^* \cap K)$.

Claim 1. The width of u is bounded by n . Otherwise, u can be written as

$$u = u_1u_2 \cdots u_{n+1}u',$$

with

$$\beta(u_1) = \beta(u_2) = \cdots = \beta(u_{n+1}),$$

and we can find $1 \leq i < j \leq n+1$ and $q \in Q$, $q' \in F$ (cf. Lemma 7) such that $q \in \lambda^*(q_0, u_1 \cdots u_i)$, $q \in \lambda^*(q, u_{i+1} \cdots u_j)$ and $q' \in \lambda^*(q, u_{j+1} \cdots u_{n+1}u')$. Hence, $u_1 \cdots u_iu_{j+1} \cdots u_{n+1}u' \in D_1'^* \cap K$ and u is not minimal.

Claim 2. The height of u is bounded by n^2 . Otherwise, u can be written as

$$u = u_0au_1au_2 \cdots au_h\bar{a}v_{h-1}\bar{a} \cdots v_2\bar{a}v_1\bar{a}v_0,$$

where $u_0, u_1, \cdots, u_h, v_{h-1}, \cdots, v_0 \in D_1'^*$, $h \geq n^2$.

Then we can find $0 \leq i < j \leq h$, q and \bar{q} in Q and $q'' \in F$ such that

$$q \in \lambda^*(q_0, u_0a \cdots u_ia),$$

$$q \in \lambda^*(q_0, u_0a \cdots u_ja),$$

$$q' \in \lambda^*(q_0, u_0a \cdots au_h\bar{a}v_{h-1} \cdots v_j\bar{a}),$$

$$q' \in \lambda^*(q_0, u_0a \cdots au_h\bar{a}v_{h-1} \cdots v_{i+1}\bar{a}),$$

$$q'' \in \lambda^*(q', v_i\bar{a} \cdots \bar{a}v_0).$$

Hence, u is not minimal, since

$$u' = u_0a \cdots u_iau_{j+1}a \cdots au_h\bar{a}v_{h-1} \cdots v_j\bar{a}v_i\bar{a} \cdots \bar{a}v_0$$

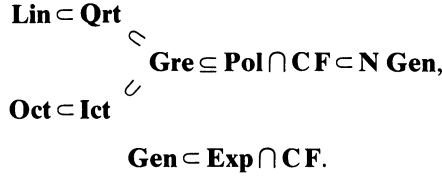
belongs to $D_1'^* \cap K$ and is shorter than u . Hence, $|u| \leq n^3$. So $\rho_{D_1'^*} \leq \lambda n \cdot n^3$. \square

Open problem 2. Does $\rho_{D_1'^*} > \lambda n \cdot n^2$?

THEOREM 4. *The rational cones **Lin** and **Oct** are included in **Pol**. The full AFL's **Qrt**, **Ict** and **Gre** are included in **Pol**.*

Proof. By Theorem 1 **Pol** is a rational cone containing S_2 (by Lemma 9) and **Lin** is the least one containing S_2 . So **Lin** \subseteq **Pol**. The proof is similar for **Oct** and the other families by using Theorem 2 and Lemma 9. \square

Let us summarize the relations between this various families of languages into a diagram.



On this diagram, all inclusions are known to be strict except **Gre** \subseteq **Pol**.

Conjecture 1. The inclusion **Gre** \subset **Pol** is strict.

A possible way to prove this conjecture would be to consider the language $\bar{E} = \{a, b, c, d\}^* - E$ where E is generated by the grammar

$$S \rightarrow aSbSc + d.$$

The language E is known to be a generator, and it satisfies the following property:

(*) For all $u, v, w \in \{a, b, c, d\}^*$ if uvw and uw belong to E , then $v = \varepsilon$.

We show that \bar{E} is polynomial. Let $u \in \text{MIN}(\bar{E} \cap K)$ for some $K \in \text{Rat}_n(\{a, b, c, d\})$ and assume that $|u| \geq 3n$. Then, by using the pumping lemma, we can write

$$u = u_1vu_2 \quad \text{with } |v| \geq n \text{ and } u_1u_2 \in K.$$

By a second application of the pumping lemma, we can write

$$v = v_1wv_2 \quad \text{with } |w| \geq 1, |v_1v_2| \geq 1 \text{ and } u_1v_1v_2u_2 \in K.$$

The minimality of u implies

$$\begin{aligned}
 u_1u_2 &\notin \bar{E} \cap K, \\
 u_1v_1v_2u_2 &\notin \bar{E} \cap K;
 \end{aligned}$$

hence, $u_1u_2 \in E$, $u_1v_1v_2u_2 \in E$, which contradicts property (*) of E .

We conjecture that $\bar{E} \notin \mathbf{Gre}$, which would prove that the inclusion **Gre** \subset **Pol** is strict.

There is yet another question.

Conjecture 2. Every context-free language is either in **Pol** or in **Exp**.

This is not at all obvious. There could exist a context-free language with rational index $\lambda n.2^{\sqrt{n}}$.

We conclude this section by examining more closely the quasi-rational languages. The following proposition shows in particular that Lemma 7 cannot be improved.

The family **Qrt** of quasi-rational languages is the union of all the **Qrt**(p) for $p \geq 1$, defined as follows:

$$\begin{aligned}
 \mathbf{Qrt}(1) &= \mathbf{Lin}, \\
 \mathbf{Qrt}(p+1) &= \{\sigma(L) / L \in \mathbf{Lin} \text{ and } \sigma \text{ is a } \mathbf{Qrt}(p)\text{-substitution}\}.
 \end{aligned}$$

PROPOSITION 2. For all $p \in \mathbb{N}_+$ there exists a language in $\mathbf{Qrt}(p)$ with a rational index in $\Omega(\lambda n.n^{2p}) \cap \pi(\lambda n.n^{2p})$.

Proof. Let

$$L = \{x^n y^m z^m t^n / n, m \geq 1\}$$

and

$$L_1 = \{a_1^n b_1^m c_1^m d_1^n | n, m \geq 1\}.$$

For $i \geq 1$, we define

$L_{i+1} = \sigma_i(1)$ where σ_i is the substitution such that

$$\begin{aligned} \sigma_i(x) &= a_{i+1}, \\ \sigma_i(y) &= b_{i+1}L_i, \\ \sigma_i(z) &= c_{i+1}, \\ \sigma_i(t) &= d_{i+1}, \end{aligned}$$

so that $L_i \in \mathbf{Qrt}(i)$ for all i .

An easy modification of Lemma 7 then shows that

$$\rho_{L_{i+1}} \leq \rho_L \cdot \rho_{L_i}.$$

Since $\rho_L \leq \lambda n.n^2$ by Lemma 9 part 1, we get $\rho_{L_i} \leq \lambda n.n^{2i}$ for all i . We now show that $\lambda n.n^{2i} \leq \rho_{L_i}$ for all i .

In order to show that $\lambda n.n^{2i} \leq \rho_{L_i}$, we shall define a family $K_n^{(i)}$ of rational languages (for $i, n \geq 1$) such that

$$K_n^{(i)} \text{ belongs to } \text{Rat}_{c.i.n} \text{ for some constant integer } c$$

and

$$\delta_{K_n^{(i)}, L_i} \geq n^{2i}.$$

We start with

$$K_n^{(1)} = a_1(b_1^n) * (c_1^{n+1}) * d_1,$$

which belongs to Rat_{2n+3} . Clearly, $\text{MIN}(L_1 \cap K_n^{(1)})$ consists of the word $a_1 b_1^{n(n+1)} c_1^{n(n+1)} d_1$ of length $2n^2 + 2n + 1$. Hence, $\delta_{K_n^{(1)}, L_1} \geq n^2$ and $\rho_{L_1} \geq \lambda n.n^2$.

Assuming now that $K_n^{(i-1)}$ is defined by an automaton $A_n^{(i-1)}$ with $a(i-1, n)$ states, we can build $K_n^{(i)}$ by defining $A_n^{(i)}$ as shown in Fig. 1, in such a way that $A_n^{(i)}$ has n arcs labeled by b_i .

Hence, $a(i, n) = a(i-1, n) + 2n + 4$. Since $a(1, n) = 2n + 3$, we get $a(i, n) = 2i(n+2) - 1$. We have constructed $A_n^{(i)}$ in such a way that $\text{MIN}(L_i \cap K_n^{(i)}) = \{f_n^{(i)}\}$, where

$$\begin{aligned} f_n^{(i)} &= a_i (g_n^{(i)})^{n+1} c_i^{n(n+1)} d_i, \\ g_n^{(i)} &= b_i a_{i-1} f_n^{(i-1)} d_{i-1} b_i a_{i-1}^2 f_n^{(i-1)} d_{i-1}^2 b_i \cdots b_i a_{i-1}^n f_n^{(i-1)} d_{i-1}^n. \end{aligned}$$

The proof is an induction on i for fixed n . For $i = 1$, we have $g_n^{(1)} = b_1$ and $f_n^{(1)} = a_1 b_1^{n(n+1)} c_1^{n(n+1)} d_1$. Let us sketch the proof of the inductive step.

Let f be any word in $\text{MIN}(L_i \cap K_n^{(i)})$. It must begin with a_i . Then, $b_i a_{i-1}$ must follow, and they are followed by a word u of $K_n^{(i-1)}$. It is easy to see that u must also be in L_{i-1} . Since f is minimal, u must be in $\text{MIN}(L_{i-1} \cap K_n^{(i-1)})$; hence, $u = f_n^{(i-1)}$.

Then $d_{i-1}b_i a_{i-1}^2$ must follow. Hence, our word f begins with

$$a_i b_i a_{i-1} f_n^{(i-1)} d_{i-1} b_i a_{i-1}^2 u,$$

for some u in $K_n^{(i-1)}$.

As above we get $u = f_n^{(i-1)}$, followed by $d_{i-1}^2 b_i a_{i-1}^3$; then, $f_n^{(i-1)}$ again and $d_{i-1}^3 b_i a_{i-1}^4$ and so on until we obtain for f a beginning of the form $a_i g_n^{(i)}$.

Note that $g_n^{(i)}$ is a loop in $A_n^{(i)}$ from state α to itself (see Fig. 1). Hence

$$f = a_i (g_n^{(i)})^p c_i^{(n+1)q} d_i,$$

for some integers $p, q \geq 1$. Note that these integers are the minimal ones such that $f \in L_i$. Since $g_n^{(i)}$ has n b_i 's we must have $p = n + 1$ and $q = n$, which proves the claim.

We get

$$|f_n^{(i)}| > n^2 |f_n^{(i-1)}|;$$

hence,

$$|f_n^{(i)}| > n^{2i} \text{ for } i \geq 1.$$

Therefore,

$$\rho_{L_i}(2i(n+2)-1) \geq n^{2i},$$

and finally

$$\lambda n \cdot n^{2i} \leq \rho_{L_i}.$$

□

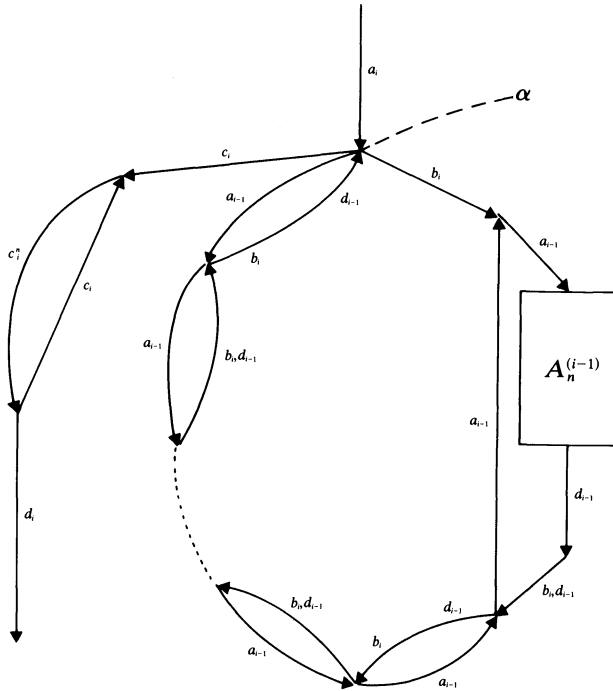


FIG. 1. Automaton $A_n^{(i)}$.

3. Comparison with other complexity measures. We shall compare the rational index with two similar complexity measures.

The first one, introduced by Paredaens and Vincke [9], is called a *T-measure*, and is defined as follows:

For any language $L \subseteq X^*$ let \sim_L be the following (classical) equivalence relation on X^* :

$$u \sim_L v \text{ if and only if for all } h \in X^*, uh \in L \Leftrightarrow vh \in L.$$

For every $n \in \mathbb{N}_+$, Let $f_L(n)$ be the index of the restriction of \sim_L to X^n , i.e., the number of equivalence classes of words of length n .

If $f_L(n)$ is bounded by a polynomial (a polynomial of degree r), then L is said to be in the class P (in the class P_r). Note that $f_L(n) \leq k^n$ where $k = |X|$.

This measure is quite different from the rational index. In particular, the language

$$K_r = \{a_1^{n_1} a_2^{n_2} \cdots a_r^{n_r} b a_r^{n_r} \cdots a_1^{n_1} \mid n_1, n_2, \dots \geq 1\}$$

is in $P_r - P_{r-1}$. But K_r belongs to $\Omega(\lambda n.n^2) \cap \pi(\lambda n.n^2)$ for all r .

On the other hand, for

$$S = \{w\tilde{w} \mid w \in \{a, b\}^*\},$$

then $f_s(n) = 2^n$. Hence, s is one of the most complex languages with respect to the *T-measure*; but it is also in $\Omega(\lambda n.n^2) \cap \pi(\lambda n.n^2)$.

So there are languages with a “small” rational index but a “large” *T-measure*. Conversely, for any function $f: \mathbb{N} \rightarrow \mathbb{N}$ such that $f(n) \geq n$ for all n , let

$$L_f = \{a^n b^{f(n)} \mid n \in \mathbb{N}\}.$$

This language is in P_0 and f is its rational index. So there are languages with a “small” *T-measure* but a complicated rational index. The connections between rational index and *T-measure* seem to be very loose. Nevertheless, let us give one more conjecture.

Conjecture 3. Any context-free language in P has a polynomial rational index (i.e., is in **Pol**).

The second complexity measure we mention was introduced by Goodrich, Ladner and Fischer [6]. For any subset L of X^n , there is a straight-line program which computes L by using the union, the concatenation, the empty word and the symbols of X . As an example, the language $L \subset \{a, b\}^n$ of all words having exactly two a 's can be computed by the following program.

```

begin
   $A_0 \leftarrow \{\varepsilon\}$ 
   $A_1 \leftarrow \emptyset$ 
   $A_2 \leftarrow \emptyset$ 
for  $i = 1$  to  $n$  do
     $A_2 \leftarrow A_1 a \cup A_2 b;$ 
     $A_1 \leftarrow A_0 a \cup A_1 b;$ 
     $A_0 \leftarrow A_0 b;$  od
end

```

At the end of the computation, the program variable A_i holds the set of words of length n having exactly i occurrences of a . The length of this program is said to be $3n + 3$.

For any language L , let $L_n = L \cap X^n$ and $C_{uc}(L_n)$ be the shortest length of a straight-line program computing L_n , and the mapping $\lambda n.C_{uc}(L_n)$ can be considered as a

complexity measure of L . It is shown that for context-free L , this function is in $\Omega(\lambda n.n^2)$. There is a “hardest” context-free language with respect to C_{uc} , namely,

$$T = \{ab^i c^j ab^i db^k ac^k b^i a \mid i, j, k \geq 1\} \in \mathbf{Qrt}(2),$$

for which

$$C_{uc}(T_n) \geq c.n^2 \quad \text{for all odd } n,$$

for some constant c .

It seems to us that this complexity will not help very much to differentiate context-free languages.

Note added at revision. Conjecture 1 has been proved by J. Gabbaro (but with an example other than \bar{E} ; we do not know whether \bar{E} belongs to \mathbf{Gre}).

Acknowledgments. We thank J. M. Steyaert for helpful comments on earlier versions of this paper.

REFERENCES

- [1] L. BOASSON, *Langages algébriques, paires itérantes et transductions rationnelles*, Theoret. Comput. Sci., 2 (1976), pp. 209–223.
- [2] ———, *The inclusion of the substitution closure of linear and one-counter languages in the largest sub-AFL of the family of context-free languages is proper*, Inform. Process. Lett., 2 (1973), pp. 135–140.
- [3] ———, *Un langage algébrique particulier*, RAIRO Inform. Théorique, 13(1979), pp. 203–218.
- [4] W. DAMM, *The IO and OI hierarchies*, Schriften zur Informatik and Angewandte Mathematik, RWTH Aachen, 1978.
- [5] S. EILENBERG, *Automata, Languages, Machines*, Vol. A, Academic Press, New York, 1974.
- [6] G. GOODRICH, R. LADNER AND M. FISHER, *Straight-line programs to compute finite languages*, Conference on Theoretical Computer Science, Aug. 1977, University of Waterloo, Ontario, Canada.
- [7] S. A. GREIBACH, *Chain of full AFL's*, Math. Systems Theory, 4 (1970), pp. 231–242.
- [8] M. NIVAT, *Transductions des langages de Chomsky*, Ann. Inst. Fourier, Grenoble, 18 (1968), pp. 339–456.
- [9] J. PAREDAENS AND R. VINCKE, *A class of measures on formal languages*, Acta Inform., 9 (1977), pp. 73–86.
- [10] J. M. STEYAERT, *Sur les index rationnels des feuillages de forêts linéaires*, Note C. R. Acad. Sci. Paris Sér A, 394, Paris, July, 1977.

EDGE-DELETION PROBLEMS*

MIHALIS YANNAKAKIS†

Abstract. If π is a property on graphs or digraphs, the edge-deletion problem can be stated as follows: find the minimum number of edges whose deletion results in a subgraph (or subdigraph) satisfying property π . Several well-studied graph problems can be formulated as edge-deletion problems.

In this paper we show that the edge-deletion problem is NP-complete for the following properties:

- (1) without cycles of specified length l , or of any length $\leq l$,
- (2) connected and degree-constrained,
- (3) outerplanar,
- (4) transitive digraph,
- (5) line-invertible,
- (6) bipartite,
- (7) transitively orientable.

For problems (5), (6), (7) we determine the best possible bounds on the node-degrees for which the problems remain NP-complete.

Key words. edge-deletion, maximum subgraph, graph property, NP-complete

1. Introduction. A graph (digraph) is a pair $G = (N, E)$, where N is a finite set of nodes and E , a set of unordered (ordered) pairs (u, v) of distinct nodes, is a set of edges.

Two nodes u and v are *adjacent* if $(u, v) \in E$. A set of nodes is *independent* if no two of them are adjacent. A graph is *complete* if every two nodes are adjacent. A path (cycle) that passes through all the nodes of a graph is called a *Hamiltonian path (cycle)*.

The *degree* of a node is the number of nodes adjacent to it. A *cubic* graph is a graph all of whose nodes have degree 3. A graph G is *connected* if there is a path between any two nodes of G ; it is *biconnected* if there are two node-disjoint paths between any two nodes of it (i.e., two paths that have no other common nodes besides the first and the last node). The maximal biconnected subgraphs of a connected graph G are called the *blocks* of G .

The *line-graph* $L(G)$ of a graph G has one node corresponding to each edge of G , and an edge connecting any two nodes corresponding to adjacent edges of G (i.e., two edges with a common node). A graph H is *line-invertible* if there exists a graph G such that $H = L(G)$.

A graph is *outerplanar* if it can be embedded in the plane so that all its nodes lie on the same face. A digraph is *transitive* if whenever $(x, y) \in E$ and $(y, z) \in E$ with $x \neq z$, then also $(x, z) \in E$. A graph is *transitively orientable* (or a comparability graph) if we can assign orientation to its edges so that the resulting digraph is transitive. A graph $G = (N, E)$ is called *bipartite* if N can be partitioned into two sets N_1, N_2 of independent nodes. A *star* on $n + 1$ nodes, denoted S_n , is the bipartite graph $(N_1 \cup N_2, N_1 \times N_2)$, with $|N_1| = 1, |N_2| = n$.

For more information regarding the properties defined above, as well as various characterizations of the graphs possessing them, the reader is referred to [H].

If $S \subseteq N$ is a subset of nodes of the graph $G = (N, E)$, the *subgraph of G induced by S* , denoted $\langle S \rangle$, is the graph (S, E_S) where $E_S = \{(u, v) \in E \mid u, v \in S\}$. If $F \subseteq E$ is a subset of edges of G , the graph $G - F$ formed by deleting F is $(N, E - F)$.

* Received by the editors May 22, 1979, and in revised form May 1, 1980.

† Bell Laboratories, Murray Hill, NJ 07974. This research was partially funded by the National Science Foundation under grant MCS76-15255 at Princeton University.

For π a property on graphs (or digraphs) we can define the corresponding *edge-deletion problem* as follows: given a graph (digraph) G , find a set of edges of minimum cardinality whose deletion results in a graph (digraph) satisfying π .

Several known NP-complete problems [C], [K] (such as the feedback arc set [K], the simple max-cut problem [GJS]) and polynomial problems (such as the arc-deletion [K], the maximum matching and the b -matching problems [E], [EJ]) can be formulated in an obvious way as edge-deletion problems by specifying the property π appropriately.

Much work has been done recently on the node-analogue, the node-deletion problems [KD], [L], [LDL], [Y1], which have been shown to be NP-complete for a fairly large class of properties, one that includes all the properties that we mentioned above. However, it is a common observation that problems on edges tend to be easier to solve (or harder to show NP-complete) than their node-analogues. In our case of deletion problems, this is exemplified by the properties $\pi_1 = \text{“acyclic graph”}$ (forest) and $\pi_2 = \text{“degree constrained”}$. The node-deletion version of these problems was shown to be NP-complete in [KD]. The edge-deletion version is the arc-deletion and the b -matching problem respectively.

In this paper we show the edge-deletion problem to be NP-complete for the following properties:

- (1) without cycles of specified length l , or of any length $\leq l$, with $l \geq 3$,
- (2) outerplanar,
- (3) transitive digraph,
- (4) line-invertible,
- (5) bipartite (simple max-cut problem),
- (6) transitively orientable.

Furthermore, we determine for problems (4), (5), (6) the best possible bounds on the node-degrees for which the problems remain NP-complete.

In [Y2] we studied the effect of adding a connectivity requirement to π (i.e., requiring the remaining subgraph to be connected) on the complexity of the node-deletion problem; we showed that for a large class of properties the problem remains NP-complete, and pointed out a case where the problem becomes polynomial. We show here that inclusion of a connectivity requirement does not affect the NP-complete status of the edge-deletion problems that we consider. Moreover, for the property π_2 (= “degree-constrained”), inclusion of a connectivity requirement makes the corresponding edge-deletion problem NP-complete. (For $\pi_1 = \text{“acyclic graph”}$ the problem obviously still remains polynomial.)

In order to prove that a problem L (in its language recognition version) is NP-complete, it suffices to show [K], [GJ] that

- (1) $L \in \text{NP}$, and
- (2) a known NP-complete problem can be reduced in polynomial time to it.

In our proofs we omit the first part, which is straightforward in all cases. For the second part we make use of the following NP-complete problems: SAT-3, node cover, Hamiltonian path [K], NOT-ALL-EQUAL 3SAT [S].

2. Edge-deletion problems. In all of the proofs (except the “line-invertible” case) the remaining subgraph after deleting a minimum set of edges turns out to be connected. This implies that inclusion of the connectivity requirement would not affect the optimal solution, and thus that the edge-deletion problems with the connectivity requirement are also NP-complete. For the “line-invertible” case we give a separate simple proof.

THEOREM 1. *The following edge-deletion problems are NP-complete:*

- (i) “without cycles of specified length l ”, for any fixed $l \geq 3$,
- (ii) for even l , the same problem restricted to bipartite graphs,
- (iii) “without any cycles of length $\leq l$ ”, restricted to bipartite graphs, for fixed $l \geq 4$.

Proof.

(i) The reduction is from the node cover problem. Let $G = (N, E)$ be a graph, input to the node cover problem. If $l = 3$, assume that G has no triangles by replacing each edge of G by a path of length 3. The graph G_1 thus formed has node cover number $\alpha_0(G_1) = e + \alpha_0(G)$, where e is the number of edges of G : if (u, v) is an edge of G and $u-w_1-w_2-v$ the path that replaced it, then a node cover V_1 for G_1 must contain at least two nodes from this path, and it contains exactly two, if these are either u and w_2 or v and w_1 . Let $V = [N \cap V_1] \cup \{v \in N \mid V_1 \text{ contains 3 nodes from a path that replaced an edge of } G \text{ incident to } v\}$. Then $|V| \leq |V_1| - e$, and V is clearly a node cover for G . Conversely, given a node cover V for G , we can add one node from each path that replaced an edge of G , to form a node cover for G_1 .

Now from the graph G (with the previous transformation first applied, if $l = 3$ and G has triangles) construct a graph G' as follows: add a new node c and edges from c to all nodes of G and replace every edge of G by a path of length $l - 2$. For example, if G is as in Fig. 1a and $l = 4$, then G' is shown in Fig. 1b.

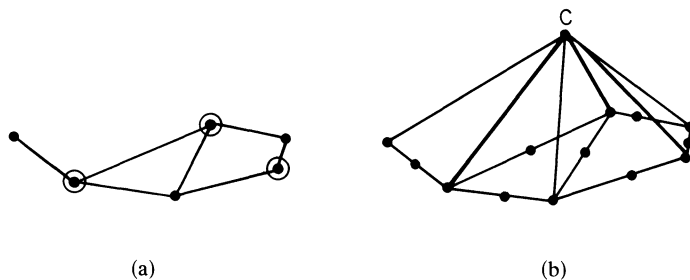


FIG. 1

We claim that there is a set of k edges of G' whose deletion results in a subgraph satisfying the desired property if and only if G has a node cover of k nodes.

(if). If V is a node cover for G , then the set of edges $A = \{(c, v) \mid v \in V\}$ is a solution to the edge-deletion problem. (In our example, if V consists of the circled nodes in Fig. 1a, then A is shown with heavy lines in Fig. 1b.)

(only if). Let A be a solution to the edge-deletion problem. Replace every edge in A that lies on a path $u-v$ by (c, u) or (c, v) , and let A' be the new set of edges. Clearly $|A'| \leq |A|$. Let $V = \{v \mid (c, v) \in A'\}$. Then V must form a node cover for G , since for every edge (u, v) of G , the edges (c, u) , (c, v) , together with the $u-v$ path that replaced (u, v) , form a cycle of length l in G' . Note that $G' - A$, with A defined as in the (if) part, is connected, assuming that G is.

(ii) If l is even, the graph G' constructed in part (i) is bipartite.

(iii) Construct the graph G' as in part (i) with l or $l - 1$, depending on whether l is even or odd. Note also that with A defined as in the (if) part, $G' - A$ has no cycles of length $\leq l$. \square

THEOREM 2. *The edge-deletion “connected, with maximum degree r ” problem is NP-complete, for every fixed $r \geq 2$.*

Proof. For $r = 2$, this problem contains the Hamiltonian cycle problem as a special case: A graph G with n nodes has a Hamiltonian cycle iff it contains a connected subgraph with maximum degree 2 and n edges.

For $r > 2$, we reduce the Hamiltonian cycle problem to it: given a graph G , for every node u of G add $r - 2$ new nodes u_1, \dots, u_{r-2} and edges from them to u , and let G' be the new graph.

Let n be the number of nodes of G . We claim that G' has a connected subgraph G_1 with $n(r - 1)$ edges and maximum degree r if and only if G has a Hamiltonian cycle.

(if). If G has a Hamiltonian cycle C , then G' has such a subgraph G_1 : G_1 consists of the cycle C and all the edges (u_i, u) .

(only if). Conversely, suppose that the optimal subgraph G_1 of G' contains $s (\leq n \cdot (r - 2))$ new nodes and $l (\leq n)$ original nodes. Then the number of edges of G_1 is $\leq (l \cdot r + s) / 2 \leq (n \cdot r + n(r - 2)) / 2 = n \cdot (r - 1)$, with equality iff $l = n$ and $s = n \cdot (r - 2)$, that is iff G_1 has the form described in the (if) part, in which case G has a Hamiltonian cycle. \square

THEOREM 3. *The edge-deletion “outerplanar” graph problem is NP-complete.*

Proof. By reducing the Hamiltonian path problem to it: given a graph G take two copies of it, G_1 and G_2 ; add two new nodes s, t and edges from them to all nodes of G_1, G_2 , and to each other. Let G' be the resulting graph. We claim that G' has an outerplanar subgraph H with $4n + 1$ edges if and only if G has a Hamiltonian path.

(if). If G has a Hamiltonian path, then G' contains an outer-planar subgraph H with $2 \cdot (2n + 2) - 3 = 4n + 1$ edges (see Fig. 2), that is a maximal outerplanar graph on $2n + 2$ nodes [H, p. 107].

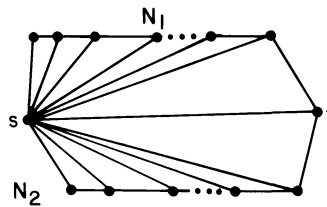


FIG. 2

(only if). If the optimal outerplanar subgraph H of G' has $4n + 1$ edges, then it is maximal outerplanar on $2n + 2$ (i.e., all) nodes and therefore biconnected. But then the boundary of the exterior face is a Hamiltonian cycle of G' , and since the node-sets N_1, N_2 of G_1 and G_2 are connected to each other only through s and t , H must have the form of Fig. 2 and consequently G must contain a Hamiltonian path. \square

A similar transformation to that of the previous theorem was used in [LG] to show the NP-completeness of the edge-deletion planar graph problem.

To prove our next theorem we will use a lemma on the NP-completeness of a restricted form of the SAT-3 problem (satisfiability with at most 3 literals per clause).

LEMMA 1. *The SAT-3 problem is NP-complete even if each variable x occurs (as x or \bar{x}) in 3 clauses and each literal in at most 2 clauses.*

A similar result is stated in [GJ, p. 259]. Lemma 1 can be proved by a straightforward reduction from SAT-3, and thus we omit its proof.

Let us note here that the requirements of the lemma are in a sense the best possible (unless $P = NP$), since if each variable appears (positively or negatively) in at most 2

clauses, then satisfiability can be decided easily in polynomial time. Consider the following algorithm, where initially $S = \{C_1, \dots, C_p\}$.

1. **while** S is not empty **do**
2. **if** S has some clause with only 1 literal **then**
3. let C be such a clause and a the single literal of C ;
4. **else** let C be any clause of S and a any literal of C ;
5. set $a = 1$ and delete C from S ;
6. **if** a occurs again in S , say in clause C' , **then** delete C' from S ;
7. **if** \bar{a} occurs in S , say in clause C'' , **then**
8. **if** C'' has at least 2 literals **then** delete \bar{a} from C'' ;
9. **else** report " S is not satisfiable" and **stop**;
10. **report** " S is satisfiable".

The algorithm tries at first to satisfy all 1-literal clauses; these can be satisfied in a unique way (if possible at all) by assigning 1 to their literals. If a contradiction arises (both a and \bar{a} having to receive truth value 1) then S cannot be satisfied (line 9). If however at some iteration line 4 is executed (all clauses have at least 2 literals) then the algorithm will terminate with a satisfying truth assignment: in all subsequent iterations S will have at most one clause with a single literal. In case S has such a clause, assigning 1 to the literal of this clause will produce at most one clause with a single literal, since each variable appears at most twice.

Lemma 1 appears to be useful in proving the NP-completeness of restricted problems. For example, from it and Karp's reduction to the node cover problem from the SAT-3 problem [K] follows a result of Garey, Johnson and Stockmeyer [GJS], that the node cover problem is NP-complete on graphs with maximum degree 3. We will use it in the proof of the next theorem to determine the best possible bound on the node-degrees for which the problem remains NP-complete.

THEOREM 4. *The edge-deletion "line invertible" graph problem on graphs with maximum degree 4 is NP-complete.*

Proof. By reducing it to the SAT-3 problem with the clauses satisfying the requirements of Lemma 1. We construct a Graph $G = (V, E)$ with nodes

$$\begin{aligned} V = & \{A_i, B_i | 1 \leq i \leq n\} \cup \{D_{ij} | \text{variable } x_i \text{ occurs in } C_j\} \\ & \cup \{E_{ij} | \bar{x}_i \text{ occurs in } C_j\} \cup \{D'_i, E'_i | 1 \leq i \leq n\} \\ & \cup \{C_j | 1 \leq j \leq p\} \cup \{C'_j | C_j \text{ has only 2 literals}\} \end{aligned}$$

and edges

$$\begin{aligned} E = & \{(A_i, B_i) | 1 \leq i \leq n\} \cup \left\{ \begin{array}{l} (A_i, D_{ij}), (A_i, E_{ik}) \\ (D'_i, D_{ij}), (E'_i, E_{ik}) \end{array} \middle| D_{ij}, E_{ik} \in V \right\} \\ & \cup \{(D_{ij}, C_j), (E_{ij}, C_j), (D_{ij}, D_{il}), (E_{ij}, E_{il}) | j \neq l; D_{ij}, D_{il}, E_{ij}, E_{il} \in V\} \\ & \cup \{(C_j, C'_j) | C'_j \in V\}. \end{aligned}$$

Example. If $C_1 = x_1 \vee x_2 \vee \bar{x}_3$, $C_2 = \bar{x}_2 \vee x_3$, $C_3 = \bar{x}_1 \vee x_2 \vee x_3$, the graph G is as in Fig. 3.

Let r be the total number of literal occurrences in the clauses. We claim that there is a set F of r edges whose deletion from G results in a line-invertible subgraph if and only if the clauses are satisfiable.

(if). Suppose that the clauses are satisfiable and let V_1 be the true variables in a

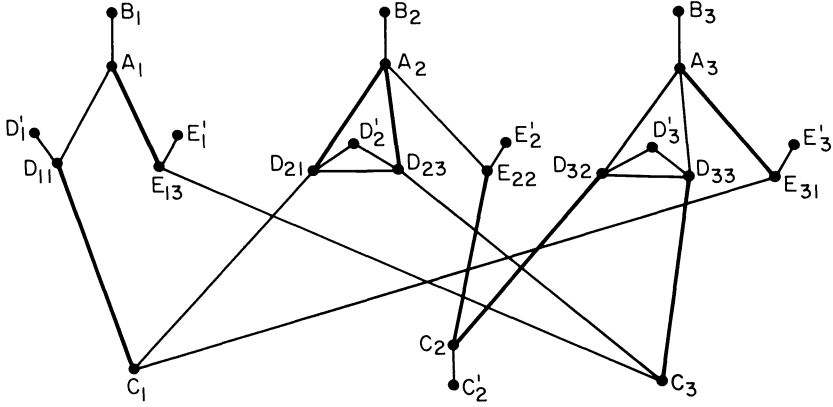


FIG. 3

satisfying truth assignment. Let

$$F = \{(A_i, D_{ij}), (E_{ij}, C_j) | x_i \in V_1; 1 \leq i \leq n, 1 \leq j \leq p\} \\ \cup \{(A_i, E_{ij}), (D_{ij}, C_j) | x_i \in V_1; 1 \leq i \leq n, 1 \leq j \leq p\}.$$

In our example the set F corresponding to the truth assignment $x_1 = 1, x_2 = 0, x_3 = 1$ is shown with heavy lines. Clearly $|F| = r$. We will show that $G - F$ is line-invertible using the following characterization (see, e.g., [H, p. 74]): a graph is line-invertible iff its edges can be partitioned into complete subgraphs in such a way that no node belongs to more than 2 subgraphs. We will call such a partition of the edges a *legal partition*.

In $G - F$ the only nodes with degree greater than 2 are $\{D_{ij} | x_i$ occurs twice $\}, \{E_{ij} | \bar{x}_i$ occurs twice $\}, \{A_i |$ the literal $(x_i$ or $\bar{x}_i)$ that occurs twice is true $\}$. Partition their incident edges as follows:

If $x_i \in V_1$, and x_i appears twice: $\langle A_i, D_{ij_1}, D_{ij_2} \rangle, \langle D'_i, D_{ij_1} \rangle, \langle D'_i, D_{ij_2} \rangle, \langle A_i, B_i \rangle$.

If $x_i \notin V_1$ and x_i appears twice: $\langle D'_i, D_{ij_1}, D_{ij_2} \rangle, \langle C_{j_1}, D_{ij_1} \rangle, \langle C_{j_2}, D_{ij_2} \rangle$,

where $x_i \in C_{j_1}, C_{j_2}$, and similarly for the symmetric cases.

In addition we have one complete subgraph for every edge both of whose nodes have degree at most 2. Clearly every node of $G - F$ belongs to at most 2 subgraphs of the partition we defined, and therefore the partition is legal.

(only if). Suppose that $G' = G - F$ is line-invertible, with $|F'| \leq r$. Replace an edge in F of the form (D_{ij}, D'_i) by (D_{ij}, C_j) and an edge (E_{ij}, E'_i) by (E_{ij}, C_j) , and let F' be the resulting set.

The nodes $\{D_{ij}, D'_i, A_i, C_j\}$ (and similarly for E_{ij}) form a star S_3 , a forbidden induced subgraph of line graphs [H, p. 75]. Therefore at least one edge incident to each D_{ij} , and each E_{ij} belongs to $F' \Rightarrow |F'| = r$ and no other edges are in F' . Define a truth assignment T as follows:

$$x_i = \begin{cases} 0 & \text{if there is no edge } (A_i, D_{ij}) \text{ in } G - F', \\ 1 & \text{otherwise.} \end{cases}$$

Because of the edges (A_i, B_i) we have: $x_i = 1 \Rightarrow G - F'$ does not contain any (A_i, E_{ij}) edge $\Rightarrow G - F'$ contains all (E_{ij}, C_j) edges. Because of the star formed by C_j and the 3 nodes corresponding to its literals (or C_j, C'_j and the 2 nodes corresponding to its literals if C_j has 2 literals), at least one edge incident to each C_j belongs to F' and the

corresponding literal is therefore true in T . That is, T is a satisfying truth assignment for the clauses. \square

Remark 1. The restriction of the maximum degree to 4 in Theorem 4 is best possible; i.e., for graphs with maximum degree 3 the edge-deletion “line-invertible” problem can be solved in polynomial time. Let G be a graph with maximum degree 3. We will show first how to transform G to a graph G' without triangles and then how to solve the problem on G' .

Let $\langle v_1, v_2, v_3 \rangle$ be a triangle of G and let w_1, w_2, w_3 be their other neighbors (some may be missing). We will construct from G a new graph G'' that does not contain this triangle (nor any triangles that do not exist in G) and show how to obtain an optimal solution for G from an optimal solution for G'' .

Note at first that if $w_1 = w_2 = w_3 = w$ then G is the complete graph on the 4 nodes v_1, v_2, v_3, w and therefore is line-invertible. (The reason is that the maximum degree of G is 3.) Thus, let us assume that the 3 nodes w_1, w_2, w_3 are not identical.

Case 1. $w_1 \neq w_2 \neq w_3 \neq w_1$. Since v_1, v_2, v_3 have no other neighbors in $G - \{v_1, v_2, v_3\}$, an edge (v_i, w_i) , if it appears in the largest line-invertible subgraph, must form a complete subgraph by itself in a legal partition of the edges. So we can remove the edges of the triangle as in Fig. 4a. An optimal solution F for the new graph G'' is also an optimal solution for G , and vice-versa: if P is a legal partition of the edges of $G'' - F$ into complete subgraphs, then P together with the triangle $\langle v_1, v_2, v_3 \rangle$ forms a legal partition for $G - F$. Conversely, if F is an optimal solution for G and P a legal partition for $G - F$, then the restriction of P on the edges of G'' is a legal partition of $G'' - F$.

Case 1 applies also if 2 of the three nodes w_1, w_2, w_3 are missing or if one is missing and the other two are not identical.

Case 2. Two v_i 's have a neighbor in common, say $w_1 = w_2$. Let x be the other neighbor of w_1 . If x or w_3 is missing then we let G'' be the graph obtained by deleting from G the nodes v_1, v_2 , and w_1 if x is missing or v_3 if w_3 is missing. It is easy to see that as in Case 1 an optimal solution for G'' is also an optimal solution for G . If both x and w_3 exist we distinguish three cases: (i) $x \neq w_3$ and x and w_3 are not adjacent, (ii) $x \neq w_3$ and they are adjacent, (iii) $x = w_3$. Note that in all three cases the graph formed by the v_i 's, the w_i 's and x is a forbidden induced subgraph for line-graphs (these are the graphs G_4, G_7, G_2 respectively in [H, Fig. 8.3]) and that deletion of either (w_1, x) or (v_3, w_3) makes it a line-graph (see Fig. 4b,c,d).

(i) $x \neq w_3$ and they are not adjacent. Apply the transformation of Fig. 4b, where u and u' are new nodes, and let G'' be the resulting graph. We claim that the number of edges that have to be deleted from G in order to form a line-graph is the same as that of G'' . Let F be an optimal solution for G . We can assume without loss of generality that F contains either (w_1, x) or (v_3, w_3) . Deleting from G'' the rest of F and edge (u, x) or (u, w_3) results in a line-invertible subgraph. Conversely, let F be an optimal solution for G'' . F must contain at least one edge incident to u (since the star S_3 is a forbidden subgraph for line-graphs). We can assume without loss of generality that (u, u') is not in F (otherwise replace it by one of the other 2 edges incident to u). Deleting from G the edges of F that are not incident to u , and (w_1, x) (resp. (v_3, w_3)) if (u, x) (resp. (u, w_3)) is in F , results in a line-invertible subgraph of G .

(ii) $x \neq w_3$ and they are adjacent. Let y_1, z_1 be their other neighbors, respectively. If y_1, z_1 are distinct and adjacent, let y_2, z_2 be their other neighbors and continue in this way until we arrive to (a) 2 nonadjacent nodes y_k, z_k (see Fig. 4c), or (b) 1 node, say y_k (i.e., z_{k-1} has degree 2), or (c) 2 identical nodes $y_k = z_k$, or (d) we exhaust the graph. In case (d) let y_{k-1}, z_{k-1} be the last nodes in the sequence (i.e., they have degree 2). Let $A_1 = \{(v_3, w_3), (x, y_1)\} \cup \{(y_{i-1}, y_i) \mid i \text{ odd} \leq k-1\} \cup \{(z_{i-1}, z_i) \mid i \text{ even} \leq k-1\}$, and

$A_2 = \{(w_1, x), (w_3, z_1)\} \cup \{(z_{i-1}, z_i) \mid i \text{ odd} \leq k-1\} \cup \{(y_{i-1}, y_i) \mid i \text{ even} \leq k-1\}$. If we delete A_1 or A_2 from G , the remaining graph (a path attached to the 2 triangles) is obviously line-invertible. Conversely, let F be an optimal solution for G . Since S_3 is a forbidden subgraph for line-graphs, F must contain at least one edge incident to x, w_3, y_i, z_i (for all $i \neq k-1$). In addition F must contain at least one edge incident to w_1 or v_3 . Consequently, F must contain at least k edges, and therefore A_1 (or A_2) is an optimal solution.

For the rest of the cases let $V = \{w_1, w_3, v_1, v_2, v_3, x, y_1, \dots, y_{k-1}, z_1, \dots, z_{k-1}\}$, and let G_1 be the graph obtained by deleting from G the set of nodes V . In case (c) ($y_k = z_k$) an optimal solution for G consists of an optimal solution for G_1 and an optimal solution for $\langle V \rangle$, the graph induced by V : if F is an optimal solution for G_1 , and we take (for example) A_1 to be the optimal solution for $\langle V \rangle$, then a legal partition of $G - (F \cup A_1)$ can be obtained by combining a legal partition of $G_1 - F$ with a legal partition of $\langle V \rangle - A_1$ with the triangle $\langle y_{k-1}, z_{k-1}, y_k \rangle$ in place of the edge (y_{k-1}, z_{k-1}) . (Note that y_k has degree 1 in G_1 .) Thus, in this case we let the new graph G'' be G_1 .

In case (b) (z_k is missing) we let G'' be the subgraph of G obtained by deleting all nodes of V but y_{k-1} . Clearly a solution for G must contain a solution for G'' and a solution for $\langle V \rangle$, since "line-invertible" is a property that is hereditary on induced subgraphs (i.e., if it holds for a graph it holds also for its induced subgraphs). It is easy to see that A_1 if k is even (A_2 if k is odd) can be combined with any solution for G'' to form a solution for G . Note that A_1 if k is even (A_2 if k is odd) contains (y_{k-2}, y_{k-1}) . Therefore, from an optimal solution for G'' we can obtain an optimal solution for G .

In case (a) ($y_k \neq z_k$) let G'' be the graph of Fig. 4c, where u and u' are new nodes. Let F and F'' be optimal solutions for G and G'' respectively. We claim that $|F| = |F''| + k$. F must contain as in case (d) at least k edges from $\langle V \rangle$; furthermore, if F contains exactly k such edges then it will have to contain also (y_{k-1}, y_k) or (z_{k-1}, z_k) because otherwise a star S_3 will be left around y_{k-1} or z_{k-1} . We let F'' consist of the edges of G_1 that belong to F and (u, y_k) if $(y_{k-1}, y_k) \in F$, (u, z_k) if $(z_{k-1}, z_k) \in F$ or either of the two if neither (y_{k-1}, y_k) nor (z_{k-1}, z_k) belongs to F . Clearly F'' is a solution for G'' , and $|F''| \leq |F| - k$.

Conversely, let F'' be an optimal solution for G'' . F'' must contain at least one edge incident to u , and we can assume as in case (i) that it does not contain (u, u') . We let F contain the edges of G_1 that are in F'' . In addition, if F'' contains (u, y_k) (resp. (u, z_k)) we include in F the edge (y_{k-1}, y_k) (resp. (z_{k-1}, z_k)) and the set A_1 if k is odd or A_2 if k is even (resp. A_2 or A_1). If F'' contains both (u, y_k) and (u, z_k) , then we include in F both (y_{k-1}, y_k) and (z_{k-1}, z_k) and either A_1 or A_2 (it does not matter which of the two). Clearly $|F| = |F''| + k$. The graph obtained by deleting F and G is either $G_1 - F$ connected to the two triangles via a path or is the disjoint union of $G_1 - F$ and a path attached to the two triangles. Consequently, $G - F$ is line-invertible, and F is an optimal solution for G .

(iii) $x = w_3$. Apply the transformation of Fig. 4d, and let G'' be the new graph. Let $V = \{v_1, v_2, v_3, w_1, x\}$. A solution F for G must contain a solution for $\langle V \rangle$ and a solution for G'' , since "line-invertible" is a hereditary property. The edge (w_1, x) together with an optimal solution for G'' forms a solution for G , which is therefore an optimal solution.

We have shown thus far how to obtain from a graph G a new graph G'' with at least one triangle fewer in such a way that an optimal solution for G can be easily derived from an optimal solution for G'' . Let G' be the result of the repeated application of the transformations, until all triangles have been eliminated. It suffices to show now how to find an optimal solution F for G' . Since G' contains no triangles, in a legal partition of the edges of $G' - F$ into complete subgraphs each edge must form a complete subgraph

by itself. This means that $G'-F$ must have maximum degree at most 2. Since any graph with maximum degree 2 (or less) is obviously line-invertible, F is a minimum set of edges whose deletion results in a subgraph with no node having degree more than 2. Finding such a set F is exactly the b -matching problem, which can be solved in polynomial time using matching techniques [E], [EJ].

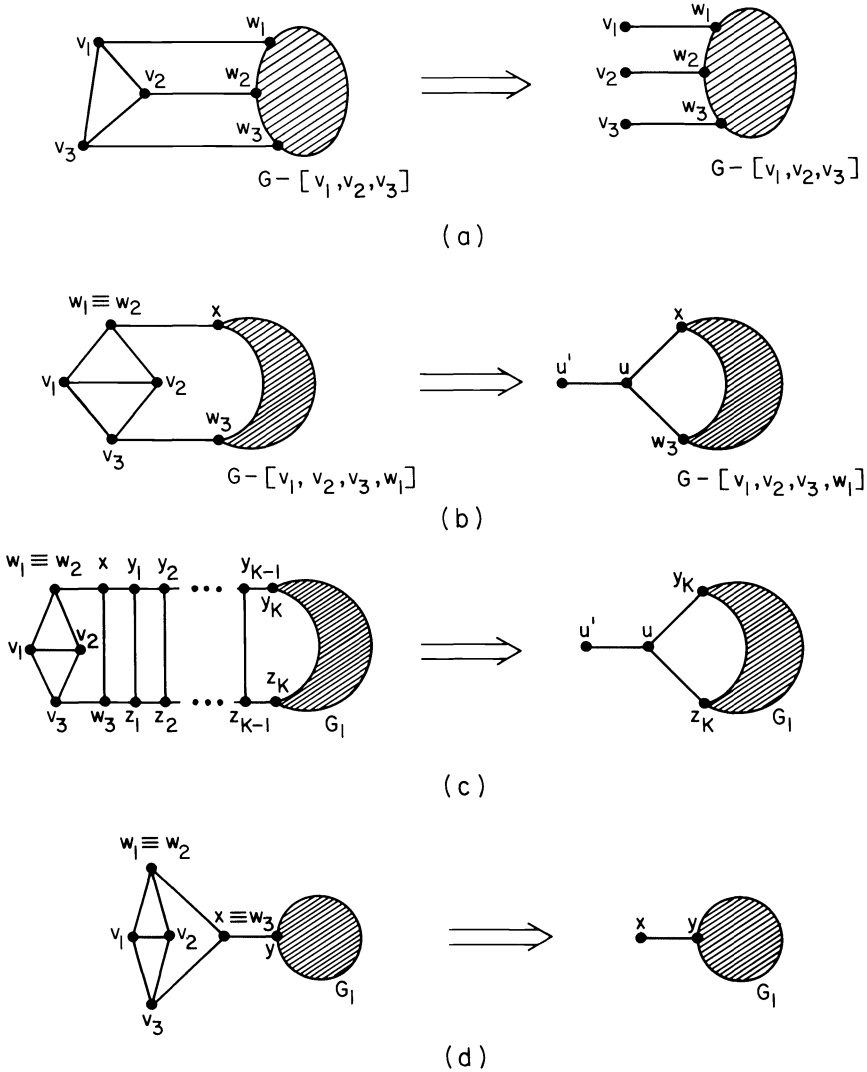


FIG. 4

Remark 2. The remaining graph in the proof of Theorem 4 is disconnected. However the same result can be easily shown if we include also the connectivity requirement, even for graphs with maximum degree 3. In [GJS] it is shown that the Hamiltonian cycle problem is NP-complete even for graphs that contain no triangles and have maximum degree 3. Since a connected line-invertible subgraph of a graph with no triangles is either a path or a cycle, it follows immediately that the same is true for the edge-deletion connected line-invertible subgraph problem.

THEOREM 5. *The simple max-cut problem restricted to cubic graphs without triangles is NP-complete.*

*Proof*¹. The reduction is from the NOT-ALL-EQUAL 3SAT problem [S]. An instance of the NOT-ALL-EQUAL 3SAT problem is a set $S = \{C_1, \dots, C_p\}$ of clauses, each with exactly 3 literals. The instance is satisfied by any truth assignment that does not leave any clause with all true or all false literals.

Given a set $S = \{C_1, \dots, C_p\}$ of clauses with $|C_i| = 3$ for each i , construct a graph $G = (N, E)$ as follows. Let G have $2p$ nodes $A_1, \dots, A_p, B_1, \dots, B_p$ for each variable x , connected in a path $A_1, B_1, A_2, B_2, \dots, A_p, B_p$. In addition, let G have one node for each literal occurrence, with literal occurrences in the same clause being connected by disjoint paths of length 3. If x occurs in C_i , connect by an edge x 's node A_i to the node that corresponds to the occurrence of x in C_i ; if \bar{x} occurs in C_i , connect B_i to the corresponding node.

Example. If $C_1 = x_1 \vee \bar{x}_2 \vee x_3$, $C_2 = \bar{x}_1 \vee x_2 \vee \bar{x}_3$, $C_3 = x_1 \vee x_2 \vee \bar{x}_3$, the graph G is as in Fig. 5.

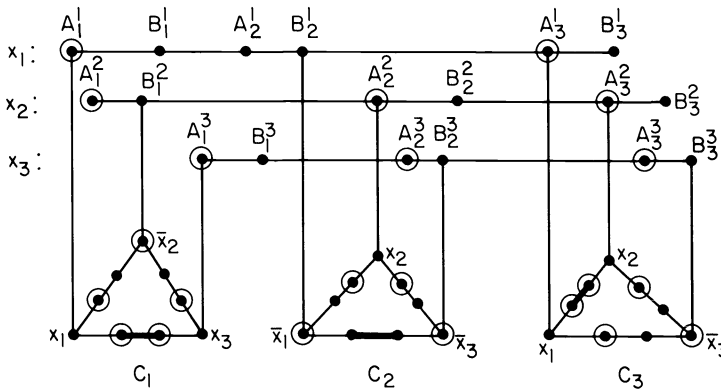


FIG. 5

We claim that there is a set F of p edges whose deletion results in a bipartite subgraph of G if and only if the instance of NOT-ALL-EQUAL 3SAT is satisfiable.

(*if*). Suppose that S is satisfiable and let T be a satisfying truth assignment. Let F contain one edge from each path (of length 3) that connects literal occurrences with the same truth value in T . Since T is a satisfying truth assignment, F contains one edge from each clause and therefore $|F| = p$. In our example the set F corresponding to the truth assignment $x_1 = x_2 = x_3 = 1$ is shown with heavy lines. The graph $G - F$ is bipartite: one side of the bipartition of the nodes of $G - F$ contains all A_i nodes of true variables, B_i nodes of false variables, all nodes that correspond to false literal occurrences and the nodes in the paths of the clause-constructions that are adjacent to true literal occurrences. (In Fig. 5 we have circled the nodes in this side of the bipartition.) The other side of the bipartition contains the rest of the nodes.

(*only if*). Let F be an optimal solution to the edge-deletion problem, and suppose that $|F| \leq p$. Since G contains an odd cycle (of length 9) for each clause, F must contain at least one edge from each such cycle. Thus, $|F| = p$, and F contains exactly one edge from each clause cycle. Let N_1, N_2 be a bipartition of the nodes of $G - F$. For each variable x , all A_i nodes must belong to the one set and all B_i nodes to the other, because

¹ This is a simplification by T. J. Schaefer of our original proof.

of the path $A_1, B_1, \dots, A_p, B_p$. Define a truth assignment by setting $x = 1$ if its A_i nodes belong to N_1 , and 0 otherwise. All clause nodes that correspond to false literal occurrences must belong to N_1 , and all those that correspond to true literal occurrences must be in N_2 . Suppose that there is a clause all of whose literals receive the same truth value, say true. Since F contains only one edge from this clause, there are two literals of it such that F does not contain an edge from the path of length 3 that connects the corresponding nodes. Thus, there are two nodes of N_2 that are connected by an odd path, a contradiction.

The graph G that we constructed has maximum degree 3 but is not cubic. We can easily make G cubic by attaching proper graphs to the nodes of degree 1 and 2. An optimal solution for G can be easily recovered from an optimal solution for the new graph, since the max-cut problem can be solved independently on the blocks of the graph (i.e., an optimal solution for a graph is the union of optimal solutions for its biconnected components). \square

The NP-completeness of the simple max-cut problem (without the restriction on the degrees) was shown in [GJS] (there it was called "simple max-cut" to stress the fact that the edges have no weights; the NP-completeness of the weighted version was shown in Karp's paper [K]). Another restriction of the max-cut problem, on planar graphs, has been proved to be polynomial even in the case of weights. ([OD] and see the comments in [GJS] on it; also [Ha].)

THEOREM 6. *The edge-deletion "transitively orientable" graph problem is NP-complete, even on cubic graphs without triangles.*

Proof. Let G be a graph without any triangles, and suppose that $G-F$ is transitively orientable with D the transitive digraph that results after a proper orientation of the edges. In D all edges incident to a node must be directed in the same way: either all going in or all coming out. That is, D induces a bipartition of the nodes: on the one side those nodes that have all edges going in and on the other those with all edges coming out.

Conversely, every bipartite graph is transitively orientable, since we can direct all edges from the one side of the bipartition to the other. Therefore, if G is a graph without triangles, any solution to the edge-deletion "transitively orientable" graph problem is also a solution to the edge-deletion "bipartite" problem. The result then follows from Theorem 5. \square

THEOREM 7. *The edge-deletion "transitive digraph" problem is NP-complete.*

Proof. The reduction is from the simple max-cut problem on graphs without triangles. Let G be such a graph. Construct a digraph D as follows. Replace each edge of G with two oppositely directed edges; for each node u of G add two new nodes u_1, u_2 and connect each of them with two oppositely directed edges to u . Let n be the number of nodes of G and e the number of edges. We claim that there is a set F of $e + 2n + k$ edges of D whose deletion results in a transitive digraph if and only if there is a set F' of k edges of G whose deletion from G results in a bipartite subgraph.

(if). Let F' be a solution to the simple max-cut problem for G with $|F'| = k$. Let N_1, N_2 be a bipartition of the edges of $G-F'$. Let F consist of the edges of D that replaced the edges in F' and of all the edges directed into nodes of N_1 and out of nodes of N_2 . Clearly $|F| = e + 2n + k$, and $D-F$ is transitive, since it does not contain any two consecutive edges (x, y) and (y, z) . Note also that $D-F$ is connected (i.e., its underlying undirected graph is connected) if $G-F'$ is.

(only if). Let $D_1 = D - F$ be transitive with F a minimum set and suppose that $|F| = e + 2n + k$. Since G does not contain any triangles, D_1 cannot have any pair of consecutive edges (x, y) and (y, z) with $x \neq z$.

Suppose that D_1 contains two opposite edges (x, y) and (y, x) . If x and y are both nodes of the original graph G , we can replace them by $(x, x_1), (x, x_2), (y, y_1), (y, y_2)$ to get a subdigraph larger than D_1 . If one of them is not a node of G , say $y = x_1$, let z be a node of G adjacent to x . If there are two opposite edges $(z, z_1), (z_1, z)$ incident to z , we can replace these four edges by $(x, x_1), (x, x_2), (x, z), (z_1, z), (z_2, z)$. If there are not two opposite edges incident to z , all edges incident to it must be either ingoing or outgoing, since D contains no transitive triangles. Assume without any loss of generality that they are all ingoing. Then we can replace $(x, x_1), (x_1, x)$ by $(x, x_1), (x, x_2), (x, z)$ to get a larger transitive subdigraph.

So D_1 cannot contain any two opposite edges. But then all edges incident to the same node must have the same direction and this induces a bipartition of the nodes of D and consequently also of G .

Therefore, if we let F' be the set of edges of G that are not in the underlying graph of D_1 , then $G - F'$ is bipartite, and $|F'| = k$, since F contains all edges of D that replaced edges of F' and one edge from each other pair of oppositely directed edges of D . \square

3. Conclusions. In this paper we showed several edge-deletion problems to be NP-complete. Unlike their node-analogues [KD], [L], [LDL], [Y1], [Y2], edge-deletion problems do not seem to be amenable in general to a unified approach. It would be interesting to find classes of properties for which this is possible, that is, classes of properties for which the edge-deletion problem can be shown NP-complete using a small number of reductions, or classes of properties for which there is a uniform polynomial algorithm that solves the edge-deletion problem (for example a uniform way of reducing it to the maximum matching problem). Note also that the special case of the edge-deletion problem with $\pi =$ "complement of a chordal graph" (which is equivalent to computing a minimum fill-in of a graph, a problem that arises in the solution of linear equations) remains still an open problem [GJ].

Acknowledgment. I wish to thank Professor J. D. Ullman for the careful reading of the manuscript and many helpful remarks.

REFERENCES

- [C] S. A. COOK, *The complexity of theorem-proving procedures*, Proc. of Third Annual ACM Symposium on Theory of Computing, 1970, pp. 151–158.
- [E] J. EDMONDS, *Paths, trees and flowers*, Canad. J. Math., 17 (1965), pp. 449–467.
- [EJ] J. EDMONDS AND E. L. JOHNSON, *Matching: a well-solved class of integer linear programs*, Combinatorial Structures and their Applications, Gordon and Breach, New York, 1970, pp. 89–92.
- [GJ] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to Theory of NP-completeness*, W. H. Freeman, San Francisco, 1978.
- [GJS] M. R. GAREY, D. S. JOHNSON AND L. STOCKMEYER, *Some simplified NP-complete graph problems*, Theoretical Comp. Sci., 1 (1976), pp. 237–267.
- [HA] F. HADLOCK, *Finding a maximum cut of a planar graph in polynomial time*, this Journal, 4 (1975), pp. 221–225.
- [H] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1970.
- [K] 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.
- [KD] M. S. KRISHNAMOORTHY AND N. DEO, *Node-deletion NP-complete problems*, this Journal, 8 (1979), pp. 619–625.
- [L] J. M. LEWIS, *On the complexity of the maximum subgraph problem*, Proc. of Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 265–274.
- [LDL] J. M. LEWIS, D. P. DOBKIN, AND R. J. LIPTON, *Graph properties defined by a forbidden subgraph*, Proc. of the 1977 Conference on Info. Sci. and System, 1977, pp. 108–112.

- [LG] P. C. LIU AND R. C. GELDMACHER, *On the deletion of nonplanar edges of a graph*, Technical report, Stevens Institute of Technology, 1976.
- [OD] G. I. ORLOVA, AND Y. G. DORFMAN, *Finding the maximum cut in a graph*, Eng. Cybernetics, 10 (1972), pp. 502–506.
- [S] T. J. SCHAEFER, *The complexity of satisfiability problems*, Proc. of Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 216–226.
- [Y1] M. YANNAKAKIS, *Node- and edge-deletion NP-complete problems*, Proc. of Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 253–264.
- [Y2] ———, *The effect of a connectivity requirement on the complexity of maximum subgraph problems*, J. Assoc. Comput. Mach., 26 (1979), pp. 618–630.

NODE-DELETION PROBLEMS ON BIPARTITE GRAPHS*

M. YANNAKAKIS†

Abstract. A set of problems which has attracted considerable interest recently is the set of node-deletion problems. The general node-deletion problem can be stated as follows: Given a graph, find the minimum number of nodes whose deletion results in a subgraph satisfying property π . In [LY] this problem was shown to be NP-complete for a large class of properties (the class of properties that are hereditary on induced subgraphs) using a small number of reduction schemes from the node cover problem. Since the node cover problem becomes polynomial on bipartite graphs, it might be hoped that this is the case with other node-deletion problems too.

In this paper we characterize those properties for which the bipartite restriction of the node-deletion problem is polynomial and those for which it remains NP-complete. Similar results follow for analogous problems on other structures such as families of sets, hypergraphs and 0,1 matrices. For example, in the case of matrices, our result states that if M is a class of 0,1 matrices which is closed under permutation and deletion of rows and columns, then finding the largest submatrix in M of a matrix is polynomial if the matrices of M have bounded rank and NP-complete otherwise.

Key words. node-deletion, maximum subgraph, bipartite graph, hereditary property, NP-complete, polynomial algorithm

1. Introduction. A common approach when faced with an NP-complete problem [C], [K], [GJ] is to restrict its input domain with the hope that the problem may become solvable in polynomial time under the restriction. In fact it turns out that sometimes this is the case: for example, the node-cover problem, NP-complete on general graphs [K], can be solved efficiently when restricted to bipartite graphs. (This follows from König's theorem that the node covering number of a bipartite graph equals the number of edges in a maximum matching, and moreover a minimum node cover can be found efficiently from a maximum matching when the graph is bipartite—see, for example, [La].) In other cases, however, the node cover problem continues to be NP-complete under restriction; for example the node cover problem on planar graphs [GJS].

Much work has been done recently [KD], [LY], [LDL] on a class of problems, called the node-deletion (or maximum subgraph) problems. The general *node-deletion problem* can be stated as follows: Given a graph (or digraph) G , find a set of nodes of minimum cardinality, whose deletion results in a graph or digraph satisfying a property π .

Several of the known NP-complete problems, such as the node cover, the max clique, the feedback-node set [K], as well as some polynomial problems, such as the connectivity of a graph [E], [T], can be formulated in an obvious way as node-deletion problems, by specifying appropriately the property π .

Consider the following two properties: (1) “transitively orientable” (or “comparability graph” in the terminology of [B]), and (2) “complete”. The corresponding node-deletion problems when restricted to bipartite graphs are polynomial for different reasons: the first one because all bipartite graphs are transitively orientable and therefore we never have to delete any nodes (the problem vanishes); the second one because the largest bipartite graph that is complete consists of a single edge. We say that a property is *nontrivial* on some input domain D if it is true for a single node and is not satisfied by all the graphs in D . For example, “transitively orientable” is a trivial property on bipartite graphs. Clearly, the nontriviality of a property is a necessary

* Received by the editors June 23, 1978, and in revised form May 1, 1980. Part of this work is based on the author's Ph.D. thesis at Princeton University.

† Bell Laboratories, Murray Hill, New Jersey 07974.

condition for the very existence of the corresponding node-deletion problem. We say that a property π is *interesting* on D if there are arbitrarily large graphs in D satisfying π . For example “complete” is not an interesting property on bipartite graphs. Suppose that π is not interesting on D . Then the corresponding node-deletion problem is polynomial in a trivial way: Let k be an upper bound on the orders of the graphs in D satisfying π . Given a graph G from D , examine all the (induced) subgraphs of G of order up to k and find the one with the largest order that satisfies π .

If a property π cannot be recognized in nondeterministic polynomial time, then obviously the corresponding node-deletion problem is not in NP. Thus we will assume henceforth that π is in NP (at least for bipartite graphs), although our results are valid even if this is not the case, with “NP-hard” replacing “NP-complete”.

We say that a property π is *hereditary on induced subgraphs* if whenever G satisfies π , then deletion of any node does not produce a graph violating π . In [LY] it was shown that for any (graph or digraph) property that is hereditary on induced subgraphs, nontrivial and interesting, the node-deletion problem is NP-complete, by using a small number of reduction schemes from the node-cover problem. The same was shown for the restriction to planar graphs. The fact, however, that the node-deletion problem used as a prototype there (the node-cover) can be solved efficiently on bipartite graphs suggests that more of these problems may become easier when restricted to bipartite graphs. Our first aim is to disprove this for a broad class of properties. We say that a property π is *determined by the components* if, whenever the components of a graph satisfy π then the whole graph does so too. (Note that if π is a nontrivial property determined by the components then π is immediately an interesting property: any independent set of nodes satisfies it.) Our first theorem states that for all (nontrivial on bipartite graphs) properties that are determined by the components and hereditary on induced subgraphs the node-deletion problem restricted to bipartite graphs is NP-complete with a single exception: the node cover problem.

If, however, we allow properties that are not determined by the components, this is not the case any more: there are properties for which the node-deletion problem can be solved efficiently on bipartite graphs by taking advantage of the polynomiality of the node cover problem. Our main theorem states that these are exactly those properties that hold for graphs with a bounded number of different neighborhoods. (For example, all nodes of an independent set of nodes—the property that corresponds to the node cover problem—have the same neighborhood, the empty set.) Similar results follow for other structures that can be represented as bipartite graphs, such as families of sets, hypergraphs and 0,1 matrices.

The rest of this paper is structured as follows. In § 2 we review the basic graph-theory terminology and the notation we use. In § 3 we examine properties that are determined by the components. Section 4 is devoted to the proof of the main theorem. In order to completely characterize the properties with an NP-complete node-deletion problem we need a Ramsey-type theorem for bipartite graphs; its proof is given at the end of the paper in an Appendix. Finally in § 5 we consider the implications for other structures.

2. Graph-theory terminology and notation. A *graph* (*digraph*) is a pair $G = (N, E)$, where N is a finite set of nodes and E , a set of unordered (ordered) pairs (u, v) of distinct nodes, is a set of edges. The *order* of G is the cardinality of N , denoted $|N|$ or $|G|$. Two nodes u and v are *adjacent* if $(u, v) \in E$. The *neighborhood* $\Gamma(v)$ of a node v is the set of nodes that are adjacent to v . A set of nodes is *independent* if no two of them are adjacent. A graph is *complete* if every two nodes are adjacent. An independent set of edges is a graph each component of which consists of a single edge.

A graph $G = (N, E)$ is called *bipartite* if N can be partitioned into two sets N_1, N_2 of independent nodes. (Note that one of the two sets may be empty.) A *complete bipartite* graph has $E = \{(u, v) | u \in N_1, v \in N_2\}$ and is denoted by $K_{n_1 n_2}$, where $n_i = |N_i|$. The complete bipartite graph $K_{1, n}$ is called a *star* of n nodes rooted at the single node of N_1 and is denoted by S_n .

If $S \subseteq N$ is a subset of nodes, the *subgraph* of G induced by S , denoted as $\langle S \rangle$, is the graph (S, E_S) , where $E_S = \{(u, v) \in E | u, v \in S\}$. The graph $G - S$ formed by deleting a subset $S \subseteq N$ of nodes from a graph G , is $\langle N - S \rangle$. A *cutpoint* of a connected graph is a node whose deletion disconnects the graph. If c is a cutpoint of G , and $\langle N_1 \rangle, \dots, \langle N_t \rangle$ the components of $G - c$, the subgraphs $\langle N_1 \cup \{c \} \rangle, \dots, \langle N_t \cup \{c \} \rangle$ are called the *components* of G relative to c .

If π is a property, we use $\gamma_\pi(G)$ to denote the minimum number of nodes whose deletion results in a subgraph of G satisfying π . Usually, when no ambiguity can arise, we drop the subscript π . By $\alpha_0(G)$ we denote the node-covering number of G ; i.e., $\alpha_0(G) = \gamma_{\pi_0}(G)$, with $\pi_0 =$ "independent set of nodes".

Several common properties (such as planar, outerplanar, chordal, line-invertible, et al.) will be used as examples in the paper. For their definition, as well as any other undefined graph-theory terminology, the reader is referred to [B] or [H].

3. Properties that are determined by the components. The node cover problem corresponds to the property $\pi_0 =$ "independent set of nodes". Let π be any property other than π_0 which is hereditary on induced subgraphs, nontrivial on bipartite graphs and determined by the components. Since π is nontrivial it is satisfied by the trivial graph, and since it is determined by the components, it is satisfied by any independent set of nodes. Because we assumed $\pi \neq \pi_0$, there is a graph with at least one edge satisfying π , and since π is hereditary on induced subgraphs, a single edge also satisfies π . Consequently, any graph with maximum degree 1 satisfies π , since π is determined by the components. The minimum number of nodes whose deletion from a graph G results in a subgraph with maximum degree 1 is called the *dissociation number* $\text{dis}(G)$ of G ; i.e., $\text{dis}(G) = \gamma_{\pi_1}(G)$, with $\pi_1 =$ "degree constrained with maximum degree 1" [PY]. First we will show that the dissociation number problem for bipartite graphs is NP-complete. Then we use this result to include the case that some star (other than S_1) does not satisfy π . Finally we take care of the case that all stars satisfy π , using the techniques of [LY].

LEMMA 1. *The dissociation number problem for bipartite graphs is NP-complete.*

Proof. The reduction is from the NOT-ALL-EQUAL SAT problem [S]. An instance of NOT-ALL-EQUAL SAT is a set of clauses, each containing 3 literals, satisfied by any truth assignment that leaves no clause with all true or all false literals. This problem was shown to be NP-complete in [S]. Let the set of clauses $S = \{C_1, \dots, C_p\}$ with variables x_1, \dots, x_n be an instance of the NOT-ALL-EQUAL SAT problem. We will construct a graph G such that $\text{dis}(G) \leq 2n + 5p$ if and only if S can be satisfied. The graph G contains a hexagon H_i for each variable x_i , as in Fig. 1a.

The nodes T_i and T'_i of H_i are associated with the literal x_i , and F_i and F'_i with \bar{x}_i . For each clause C_j , G contains a linked double hexagon DH_j as in Fig. 1b. These parts are connected to each other as follows. Node A_{jk} (resp. A'_{jk}) is connected by an edge to the primed (resp. unprimed) node associated with the k th literal of C_j , and node B_{jk} (resp. B'_{jk}) is connected to the primed (resp. unprimed) node associated with the negation of the k th literal of C_j . In Fig. 1b we show the connections for the clause $C_j = (x_i, \bar{x}_m, x_l)$. Clearly the constructed graph G is bipartite.

Let V be a *dissociation set*; i.e., $G - V$ has maximum degree 1. Each variable hexagon needs at least two nodes deleted, and these must be opposite nodes. Note that

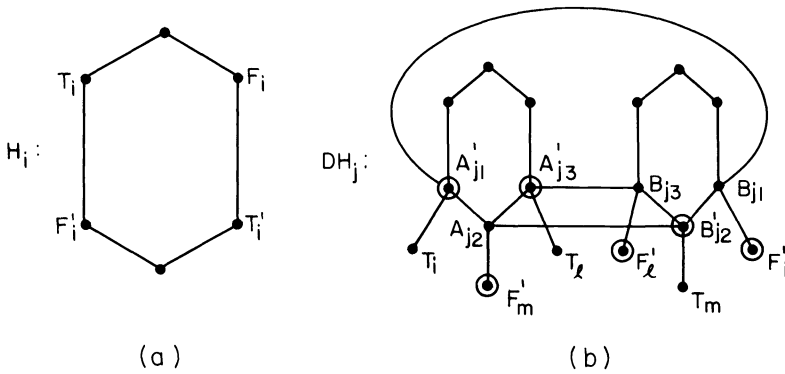


FIG. 1

after two opposite nodes are deleted from a hexagon the remaining 4 nodes all have some edge left incident to them. In a clause construction at least two nodes have to be deleted from each of the two hexagons. However, 4 nodes do not suffice: if we delete two opposite nodes from each of the two hexagons there will be two A and two B nodes left, and therefore for some k both A_{jk} and B'_{jk} (or A'_{jk} and B_{jk}) are left; these nodes will have degree at least 2. Thus, each clause construction needs at least 5 nodes deleted to satisfy π_1 . If exactly 5 nodes are deleted then at least two of them (one from each hexagon) must not have a label in Fig. 1b, and since each hexagon needs at least 2 nodes, some A node and some B node will be left.

Now suppose that the dissociation set V has at most $2n + 5p$ nodes. From the previous arguments it follows that $|V| = 2n + 5p$, and V contains exactly two opposite nodes from each H_i and five nodes from each DH_j . We can assume without loss of generality that the two opposite nodes of H_i in V are either both T_i and T'_i or both F_i and F'_i . Consider the truth assignment in which a literal is true if its associated nodes are in V . For each DH_j some A node is in $G - V$ and therefore the corresponding literal is true; also some B node is in $G - V$ and therefore the corresponding literal is false. Thus, if $\text{dis}(G) \leq 2n + 5p$ then S can be satisfied.

Conversely, suppose that S can be satisfied, and fix a satisfying truth assignment. Delete from the variable hexagons nodes that are associated with true literals. From each clause construction delete A nodes whose corresponding literal is false and B nodes whose corresponding literal is true. At this point, every H_i and each of the two hexagons of every DH_j is isolated from the rest of the graph. From each hexagon of a clause construction delete exactly one unlabeled node that is opposite to an already deleted A or B node; this is possible since we have a satisfying truth assignment. (In Fig. 1b we have circled the nodes deleted for the assignment $x_i = x_m = x_l = 0$.) The remaining graph has maximum degree 1 and consequently $\text{dis}(G) = 2n + 5p$. \square

COROLLARY 1. *Let π be any property which is hereditary on induced subgraphs, determined by the components and satisfied by a single edge, and suppose that π is not satisfied by all stars. Then the node-deletion π problem restricted to bipartite graphs is NP-complete.*

Proof. Let S_r be the star of the least order that does not satisfy π . Since a single edge (the star S_1) satisfies π , we have $r \geq 2$. The graph of Fig. 2 is called the (a, b) double star with roots v_1 and v_2 , where $a, b \geq 0$. Thus, for example, the star S_r is a $(r - 1, 0)$ and a $(0, r - 1)$ double star. Let q be the largest integer such that the $(r - 2, q)$ double star satisfies π . Since π is hereditary and is violated by S_r , we have $0 \leq q \leq r - 2$.

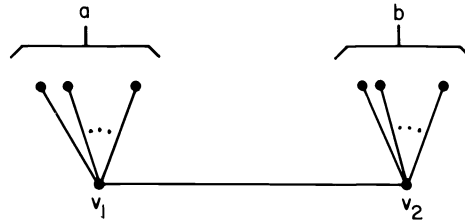


FIG. 2

Let G be a graph whose dissociation number we want to find, and let (P, Q) be a bipartition of its nodes. Attach $r - 2$ new nodes to every node in P and q new nodes to every node in Q , and let G' be the resulting graph. We claim that $\text{dis}(G) = \gamma_\pi(G')$.

Let V be a dissociation set of G . The components of $G' - V$ are $(r - 2, q)$ double stars, and stars S_{r-2} and S_q . Since π is determined by the components, $G' - V$ satisfies π and consequently $\gamma_\pi(G') \leq \text{dis}(G)$.

Let V be a set of nodes whose deletion results in a subgraph of G' satisfying π . We can assume without loss of generality that V contains no new nodes (otherwise a new node can be replaced by the node of G to which it is attached). Suppose that $G - V$ contains a node u of degree at least two; i.e., there are edges $(u, v), (u, w)$ in $G - V$. If $u \in P$, then $G' - V$ contains a star S_r rooted at u , consisting of u, v, w and the new nodes attached to u . If $u \in Q$ then $G' - V$ contains a $(r - 2, q + 1)$ double star with roots v, u : this double star consists of u, v, w and the new nodes attached to v and u . Thus, V is a dissociation set of G and consequently $\text{dis}(G) \leq \gamma_\pi(G')$. \square

We are ready now for the proof of the main theorem of this section.

THEOREM 1. *With the exception of the node cover, the node-deletion problem for graph-properties that are hereditary on induced subgraphs, determined by the components and nontrivial on bipartite graphs, restricted to bipartite graphs is NP-complete.*

Proof. As we pointed out at the beginning of this section, if π is any property (other than $\pi_0 =$ "independent set of nodes") satisfying the assumptions of the theorem, then π is satisfied by a single edge.

Case 1. π is not satisfied by all stars. The result follows from Corollary 1.

Case 2. π is satisfied by all stars. In [LY] the node cover problem is reduced to the node-deletion π problem on general graphs as follows. For every connected graph H we form a sequence α_H as follows. If c is a cutpoint of H , let $\alpha_{c,H} = \langle n_1, n_2, \dots, n_{j(c)} \rangle$, where $n_1 \geq n_2 \geq \dots \geq n_{j(c)}$ are the orders of the components of H relative to c . If H is not biconnected define $\alpha_H = \text{lexicographically}^1 \min \{ \alpha_{c,H} \mid c \text{ a cutpoint of } H \}$ and $c(H)$ any cutpoint of H that gives the minimum sequence α_H . If H is biconnected then $\alpha_H = |H|$ and $c(H)$ is any node of H . Let J be a graph with the lexicographically smallest α -sequence that violates π . Let J_0 be its largest component relative to $c(J)$, d any node of J_0 other than $c(J)$, and J' the graph obtained by deleting all nodes of J_0 except $c(J)$. It is shown in [LY] that if graph G' is obtained from graph G by attaching to each node of G a copy of J' through node $c(J)$, and replacing every edge (u, v) of G by a copy of J_0 attached through its $c(J)$ and d nodes (identified with u and v in an arbitrary way—see Fig. 3) then (1) deleting a node cover of G from G' leaves connected components with an α -sequence lexicographically smaller than that of J and therefore satisfying property π , and (2) at least $\alpha_0(G)$ nodes have to be deleted from G' if J is not to be contained in the remaining graph. Thus, $\alpha_0(G) = \gamma_\pi(G')$.

¹ If $\bar{a} = \langle a_1, \dots, a_p \rangle$ and $\bar{b} = \langle b_1, \dots, b_q \rangle$ are two sequences of integers (or of elements of any totally ordered set), we say that \bar{a} is lexicographically larger than \bar{b} (denoted as $\bar{a} >_L \bar{b}$) if either (i) there is an i with $1 \leq i \leq \min(p, q)$ such that $a_j = b_j$ for $1 \leq j \leq i - 1$ and $a_i > b_i$, or (ii) $q < p$ and $a_j = b_j$ for $1 \leq j \leq q$.

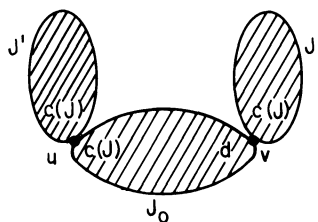


FIG. 3

Now, let J be the lexicographically smallest *bipartite* graph that violates property π . Since all stars satisfy π , J_0 has in its bipartition at least one more node in the same set with $c(J)$. Choose d to be any such node and apply the previous transformation. The resulting graph G' is obviously bipartite. Since “bipartite” is a hereditary property, deleting a node cover of G from G' will leave *bipartite* connected components with an α -sequence lexicographically smaller than that of J , and therefore $\gamma_\pi(G') \leq \alpha_0(G)$. Combining with (2) above, we have $\gamma_\pi(G) = \alpha_0(G)$. \square

COROLLARY 2. *The restriction to bipartite graphs of the node-deletion π problem for the following properties π is NP-complete: 1) planar; 2) outerplanar; 3) line-invertible; 4) chordal; 5) interval; 6) without cycles of length l for any fixed even $l \geq 4$; 7) without cycles of length $\leq l$, for any $l \geq 4$; 8) degree constrained with maximum degree $r \geq 1$; 9) acyclic graph (forest).*

These problems were first shown to be NP-complete on *general* graphs by Krishnamoorthy and Deo [KD]. However, their techniques sufficed to show the NP-completeness of the bipartite restriction only for property 9).

4. General hereditary properties. If we consider properties that are not determined by the components then there are some more node-deletion problems besides the node cover which become polynomial when restricted to bipartite graphs.

Example. Consider the property $\pi =$ “complete bipartite”. If $G = (N, E)$ is a bipartite graph with $N = P \cup Q$ a bipartition of the node set, let $G' = (N, E')$ be the graph with $E' = \{(u, v) | u \in P, v \in Q, (u, v) \notin E\}$. Then $\gamma_\pi(G) = \min \{\alpha_0(G), \alpha_0(G')\}$. For, consider the maximum induced subgraph $H = (N_1, E_1)$ with property π . Either H does not contain any edge, in which case N_1 is an independent set of G and $\gamma_\pi(G) = \alpha_0(G)$, or it contains some edge, in which case N_1 is an independent set of G' and $\gamma_\pi(G) = \alpha_0(G')$. \square

In this section we give an exact characterization of those properties which have a polynomial node-deletion problem (assuming of course $P \neq NP$).

At first we consider properties which are satisfied by some bipartite graphs with an arbitrarily large number of nontrivial components. (A component is nontrivial if it has at least two nodes.) Let $k(G)$ be the number of nontrivial components of graph G and let $k(\pi) = \sup \{k(G) | G \text{ is a bipartite graph satisfying } \pi\}$. We show that if $k(\pi) = \infty$ then the techniques of the previous section together with those of [LY] can be used to prove the NP-completeness of the corresponding node-deletion problem. As a corollary of this, we show that if the property π is satisfied by a certain set of graphs $\{B_t | t \geq 1\}$ (which we define later), then the corresponding node-deletion problem is also NP-complete.

Then we focus on properties π with $k(\pi) = 1$ and the graphs which can satisfy such properties, i.e., graphs which do not have an induced subgraph with more than one nontrivial component. Let us call \mathbf{G}_1 this class of graphs. We show that the node-deletion problem for the property π_1 which is satisfied exactly by the graphs of class \mathbf{G}_1

is NP-complete. We then extend this result to all properties that are satisfied by all graphs of \mathbf{G}_1 .

Finally, we conclude that if there are graphs with arbitrarily many different neighborhoods which satisfy a hereditary property π , then either $k(\pi) = \infty$ or π is satisfied by the set of graphs $\{B_i\}$ or π is satisfied by all graphs of \mathbf{G}_1 , and consequently the corresponding node-deletion problem is NP-complete. Then we prove that all remaining properties have a polynomial node-deletion problem.

4.1. Properties π with $k(\pi) = \infty$. An independent set of edges is a graph every connected component of which is a single edge. We will denote an independent set of t edges by I_t . Clearly, if π is a hereditary property with $k(\pi) = \infty$ then π is satisfied by any independent set of edges and vice-versa.

THEOREM 2. *The restriction to bipartite graphs of the node-deletion problem for properties π that are hereditary on induced subgraphs, nontrivial on bipartite graphs and are satisfied by any independent set of edges is NP-complete.*

Proof. Let G be a graph with components G_1, \dots, G_t and assume that $\alpha_{G_1} \geq_L \alpha_{G_2} \geq_L \dots \alpha_{G_t}$, where the α -sequence of a connected graph is defined as in the proof of Theorem 1. The β -sequence of G is defined as $\beta_G = \langle \alpha_{G_1}, \dots, \alpha_{G_t} \rangle$. Let J be a bipartite graph with the lexicographically smallest β -sequence that cannot be repeated arbitrarily many times without violating π ; i.e., there exists an $l \geq 1$ such that l independent copies of J (that is, with no interconnecting edges) violate π , but $l - 1$ independent copies of J satisfy π . The existence of such a J follows from the nontriviality of π . Let J_1, \dots, J_t be the connected components of J sorted according to their α -sequences. Let J_0 be the largest component of J_1 relative to $c(J_1)$, the node of J_1 that gave its α -sequence.

Case 1. J_0 is a single edge. Then J_1 is a star, say $J_1 = S_r$, and $J = J_1$, since all connected graphs with a smaller (or equal) α -sequence are stars of smaller (or equal) order, and therefore an appropriate number of repetitions of S_r contains as an induced subgraph repetitions of any graph J with $J_1 = S_r$. Since any independent set of edges satisfies π , $r \geq 2$. Let l be the minimum number of independent copies of S_r that violate π ; $l \geq 1$. Let q be the largest integer such that the $(r - 2, q)$ double star can be repeated arbitrarily many times without violating π ($r - 2 \geq q \geq 0$); i.e., there exists an $m \geq 1$ such that m independent copies of the $(r - 2, q + 1)$ double star violate π .

We can reduce now the dissociation number problem to the node-deletion π problem as follows. Given graph G , let G'' consist of $(l + m)(n + 1)$ independent copies of the graph G' constructed in the proof of Corollary 1, where n is the order of G . We claim that $\text{dis}(G) \leq h \Leftrightarrow \gamma_\pi(G'') \leq h(l + m)(n + 1)$. Suppose that $\text{dis}(G) \leq h$. Deleting a minimum dissociation set of G from each copy of G'' leaves a graph whose connected components are induced subgraphs of the $(r - 2, q)$ double star. Since this graph can be repeated arbitrarily many times without violating π , we have $\gamma_\pi(G'') \leq h(l + m)(n + 1)$.

Conversely, let V be an optimal solution to the node-deletion problem. Then $G'' - V$ can contain S_r in at most $l - 1$ copies of G' and the $(r - 2, q + 1)$ double star in at most $m - 1$ copies of G' . Thus, at least $n(l + m) + 2$ copies of G' do not contain either of the two after the deletion of V , and V must contain at least $\text{dis}(G)$ nodes from each of these copies. Suppose that $\text{dis}(G) \geq h + 1$. Then,

$$\gamma_\pi(G') \geq (h + 1)[n(l + m) + 2] = hn(l + m) + n(l + m) + 2(h + 1) > h(l + m)(n + 1),$$

since $n > \text{dis}(G) \geq h + 1$.

Case 2. J_0 is not a single edge. Then J_0 has in its bipartition at least one more node d in the same set with $c(J_1)$. We can now reduce the node cover problem to the

node-deletion π problem using the same method as above of taking sufficiently many copies of the graph G' constructed in the proof of Theorem 1 (see also [LY]). \square

Theorem 2 implies the NP-completeness of the node-deletion problem also for some properties π with $k(\pi) < \infty$. For example, let B_t , $t \geq 1$ be the graph with nodes $v_1, \dots, v_t, u_1, \dots, u_t$ and edges (v_i, u_j) for $i \neq j$ (in Fig. 4 we show B_4), and consider the property π which is satisfied by all B_t and their induced subgraphs. It is not hard to see that we can reduce the dissociation number problem to the node-deletion problem for π in the same way that the maximum complete bipartite problem was reduced to the node cover problem in the example at the beginning of this section. Let us formalize this reduction.

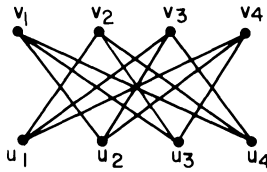


FIG. 4. The graph B_4 .

A bipartitioned graph $BG = (P, Q, E)$ is a bipartite graph $G = (N, E)$ together with a bipartition $N = P \cup Q$ of its nodes. The bipartite complement of (P, Q, E) is the bipartitioned graph b.c. $(BG) = (P, Q, \bar{E})$, where $\bar{E} = \{(u, v) | u \in P, v \in Q, (u, v) \notin E\}$. Thus, if $P \cup Q$ is any bipartition of I_t , the independent set of t edges, the bipartite complement of B_t has B_t as its underlying graph. Note, however, that if a graph G has some nonisomorphic components, then the underlying graph of a bipartite complement of it might depend on the bipartition chosen. Two bipartitioned graphs $BG = (P, Q, E)$ and $BG' = (P', Q', E')$ are isomorphic if their underlying graphs are isomorphic via an isomorphism that sends P to P' and Q to Q' . (Thus (P, Q, E) is not necessarily isomorphic to (Q, P, E) .) Note that BG and BG' are isomorphic if and only if their bipartite complements are. Let us call bipartite property π a property which is allowed to take into account also the bipartition of a graph; i.e., if BG, BG' are two isomorphic bipartitioned graphs, then π is satisfied by BG if and only if it is satisfied by BG' . Clearly, any graph-property π can be regarded also as a bipartite property: BG satisfies π if and only if its underlying graph does. If $BG = (P, Q, E)$ is a bipartitioned graph, the subgraph of it induced by a set of nodes S is $\langle S \rangle = (P \cap S, Q \cap S, E_S)$, where $E_S = \{(u, v) | u, v \in S, (u, v) \in E\}$. The subgraph of b.c. (BG) induced by S is obviously the bipartite complement of the subgraph of BG induced by S .

Now, let π be a bipartite property which is nontrivial, interesting and hereditary on induced subgraphs. Define the bipartite property $\bar{\pi}$ as follows: BG satisfies $\bar{\pi}$ if and only if b.c. (BG) satisfies π . Clearly, $\bar{\pi}$ is also nontrivial, interesting and hereditary on induced subgraphs. Moreover, the two node-deletion problems are equivalent: $\gamma_\pi(BG) = \gamma_{\bar{\pi}}(\text{b.c.}(BG))$. The proofs of NP-completeness for graph properties that we have given till now (as well as those that follow) can be used verbatim for bipartite properties as well.² Thus, from Theorem 2 we have:

COROLLARY 3. *The restriction to bipartite graphs of the node-deletion problem for properties π that are hereditary on induced subgraphs, nontrivial on bipartite graphs and are satisfied by all graphs B_t , is NP-complete.*

² We have preferred (and will continue in the sequel) to give the proofs in terms of graph-properties rather than bipartite properties in order not to unnecessarily complicate the notation.

4.2. The class \mathbf{G}_1 . For a graph G , let $i(G)$ be the maximum number of independent edges that it contains (as an induced subgraph). Clearly $i(G)$ is also the maximum number of nontrivial components of all induced subgraphs of G . Thus, \mathbf{G}_1 is the class of bipartite graphs G with $i(G) \leq 1$. For example, if G is an independent set of nodes $i(G) = 0$ and consequently $G \in \mathbf{G}_1$. At first we will give a characterization of the graphs in \mathbf{G}_1 .

LEMMA 2. *Let G be a bipartite graph. Then $i(G) = 1$ if and only if G consists of isolated nodes and a nontrivial component in which nodes in the same side of the bipartition are totally ordered by their neighborhoods.*

Proof.

(\Rightarrow) Since $i(G) = 1$, G has exactly one nontrivial component. Let (P, Q) be the bipartition of it. Let $\Gamma(v)$ be the neighborhood of a node v (the set of nodes adjacent to v). Suppose that there are nodes v_i, v_j of P such that $\Gamma(v_i) \not\subseteq \Gamma(v_j)$ and $\Gamma(v_j) \not\subseteq \Gamma(v_i)$. Let $u_i \in \Gamma(v_i) - \Gamma(v_j)$ and $u_j \in \Gamma(v_j) - \Gamma(v_i)$. The edges $(v_i, u_i), (v_j, u_j)$ are independent, contradicting $i(G) = 1$.

(\Leftarrow) If $(v_i, u_i), (v_j, u_j)$ are two independent edges with $v_i, v_j \in P, u_i, u_j \in Q$, then $\Gamma(v_i)$ and $\Gamma(v_j)$ are incomparable, and similarly for $\Gamma(u_i)$ and $\Gamma(u_j)$. \square

Lemma 2 has an interesting interpretation in terms of families of sets. Let X be a finite set and F a finite family of (not necessarily distinct) subsets of X . F is a *chain* if F is the multiset $\{S_1, S_2, \dots, S_t\}$ with $S_1 \subseteq S_2 \subseteq \dots \subseteq S_t$. A family of sets can be represented by a bipartite graph which has one node for every set and one node for every element, and an edge between a set-node and an element-node if the element is in the set. It follows then from Lemma 2 that the graphs of \mathbf{G}_1 represent exactly the chains.

Let G be a connected graph in \mathbf{G}_1 with bipartition (P, Q) and set of edges E . Let σ be the number of different neighborhoods of nodes in P ; i.e., $P = \{v_1, \dots, v_p\}$ with $\Gamma(v_1) = \Gamma(v_2) = \Gamma(v_{i_1}) \supseteq \Gamma(v_{i_1+1}) = \dots = \Gamma(v_{i_2}) \supseteq \dots \supseteq \Gamma(v_{i_{\sigma-1}+1}) = \dots = \Gamma(v_p)$. Let us denote by $\Gamma_1, \Gamma_2, \dots, \Gamma_\sigma$ these different neighborhoods. Let Q be $\{u_1, \dots, u_q\}$ with $\Gamma(u_1) \subseteq \dots \subseteq \Gamma(u_q)$. Let j_k be the largest index of an element of Q which is in Γ_k but not in Γ_{k+1} , for $k = 1, \dots, \sigma - 1$. We have $\Gamma_1 = Q$ (since G is connected). If $j \leq j_1$ then $\Gamma(u_j) \subseteq \Gamma(u_{j_1})$ and therefore u_j is not adjacent to a v_i with $i > i_1$. Consequently, $\Gamma_2 = \{u_j | j > j_1\}$, and $\Gamma(u_1) = \dots = \Gamma(u_{j_1}) = \{v_i | i \leq i_1\}$. Proceeding similarly we can show that $\Gamma_k = \{u_j | j > j_{k-1}\}$, and $\Gamma(u_{j_{k+1}}) = \dots = \Gamma(u_{j_k+1}) = \{v_i | i \leq i_{k+1}\}$, where for $k + 1 = \sigma$, $i_\sigma = p$ and $j_\sigma = q$. Thus, there are also exactly σ different neighborhoods of nodes in Q .

For $t \geq 0$, let D_t be the following graph. D_t has nodes $v_1, v_2, \dots, v_t, u_1, u_2, \dots, u_t$ and an edge (v_i, u_j) whenever $i \leq j$. Clearly $\Gamma(v_1) \supseteq \Gamma(v_2) \supseteq \dots \supseteq \Gamma(v_t)$ and $\Gamma(u_1) \subseteq \Gamma(u_2) \subseteq \dots \subseteq \Gamma(u_t)$. Thus $D_t \in \mathbf{G}_1$ and the parameter σ of D_t is equal to t .

LEMMA 3. *Let G be a connected graph in \mathbf{G}_1 with n nodes, let (P, Q) be a bipartition of it, and let σ be the number of different neighborhoods of nodes in P (or Q). Then:*

(1) G contains D_σ as an induced subgraph.

(2) G is an induced subgraph of D_t , where $t = n - \sigma$. (Note that $\sigma \leq |P|, |Q|$ and therefore $\sigma \leq n/2$.)

Proof. Let $P = \{v_1, \dots, v_p\}$ with $\Gamma(v_1) \supseteq \dots \supseteq \Gamma(v_p)$, $Q = \{u_1, \dots, u_q\}$ with $\Gamma(u_1) \subseteq \dots \subseteq \Gamma(u_q)$, and let $i_1, \dots, i_\sigma, j_1, \dots, j_\sigma$ be defined as above.

(1) Let G' be the graph induced by $v_{i_1}, v_{i_2}, \dots, v_{i_\sigma}, u_{j_1}, \dots, u_{j_\sigma}$. By our previous discussion v_{i_k} is adjacent to u_{j_l} with $j_l > j_{k-1}$, or equivalently $l \geq k$. Thus, G' is isomorphic to D_σ .

(2) Let $p_k = i_k - i_{k-1}, q_k = j_k - j_{k-1}$ for $k = 1, \dots, \sigma$ where $i_0 = j_0 = 0$. Note that G is uniquely determined by the p 's and the q 's (see Fig. 5—here we show schematically the different neighborhoods, σ from each side; nodes bracketed together have the same neighborhood). Let $D_t = \{\bar{P}, \bar{Q}, \bar{E}\}$ with $\bar{P} = \{\bar{v}_1, \dots, \bar{v}_t\}, \bar{Q} = \{\bar{u}_1, \dots, \bar{u}_t\}$, where $t =$

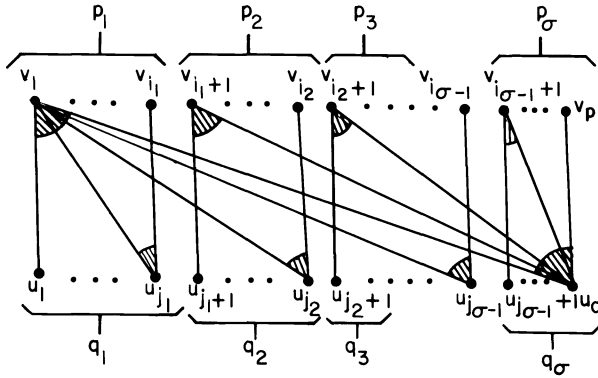


FIG. 5

$n - \sigma$ and $(\bar{v}_i, \bar{u}_i) \in \bar{E}$ if $i \leq j$. Let G' be the subgraph of D_i obtained as follows. From the set \bar{P} we keep the first p_1 nodes, delete the next $q_1 - 1$ nodes, keep the next p_2 nodes, delete the next $q_2 - 1$ nodes, \dots , keep the next p_σ nodes, delete the final $q_\sigma - 1$ nodes. We have

$$\sum_{k=1}^{\sigma} p_k + \sum_{k=1}^{\sigma} (q_k - 1) = \sum_{k=1}^{\sigma} p_k + \sum_{k=1}^{\sigma} q_k - \sigma = p + q - \sigma = n - \sigma.$$

From the set \bar{Q} we delete the first $p_1 - 1$ nodes, keep the next q_1 nodes, \dots , delete the next $p_\sigma - 1$ nodes, keep the final q_σ nodes. It is easy to see that the resulting subgraph G' is isomorphic to G . \square

4.3. Properties that are satisfied by all graphs of G_1 .

LEMMA 4. *The node-deletion π'_1 problem restricted to bipartite graphs is NP-complete, where π'_1 is the property satisfied exactly by the graphs of G_1 .*

Proof. The reduction is from the satisfiability problem with 3 literals per clause (3-SAT). Let $S = \{C_1, \dots, C_p\}$ be a set of clauses with variables x_1, \dots, x_n and 3 literals per clause. We will construct a graph G that $\gamma_{\pi'_1}(G) \leq n + 2p$ if and only if S is satisfiable. For each variable x_i , G has 4 nodes T_i, T'_i, F_i, F'_i which form two independent edges $(T_i, T'_i), (F_i, F'_i)$. Node T_i is associated with the literal x_i and F_i with \bar{x}_i . For each clause C_j , G contains a hexagon as in Fig. 6. Node A'_{jk} is connected by an edge to the node associated with the k th literal of C_j ($k = 1, 2, 3$). G has two more

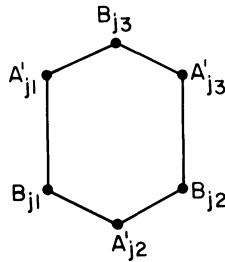


FIG. 6

adjacent nodes B, A' and the following additional edges:

$$\begin{aligned} & \{(T_i, T'_j), (T_i, F'_j), (F_i, T'_j), (F_i, F'_j) | i < j\} \\ & \cup \{(B, T'_i), (B, F'_i) | 1 \leq i \leq n\} \\ & \cup \{(B_{jk}, t'_i), (B_{jk}, F'_i), (B_{jk}, A') | 1 \leq j \leq p, 1 \leq k \leq 3, 1 \leq i \leq n\} \\ & \cup \{(B_{ik}, A'_{jl}) | i < j, 1 \leq k, l \leq 3\}. \end{aligned}$$

The intention of introducing these edges is that in a subgraph of G with property π'_1 , if primed nodes are ordered according to increasing neighborhoods, the A'_{jk} nodes should come first in increasing order of the first index j , then node A' and then the nodes associated with the literals in increasing order of their index, and similarly for the unprimed nodes. The constructed graph G is clearly bipartite.

Since we have a pair of independent edges per variable, at least one node must be deleted for each variable. Also two nodes must be deleted from each hexagon for the remaining graph not to contain two independent edges. Thus, $\gamma_{\pi'_1}(G) \geq n + 2p$. Suppose now that $\gamma_{\pi'_1}(G) = n + 2p$, and let V be such that $G - V$ satisfies π'_1 and $|V| = n + 2p$. Define a truth assignment by setting x_i true if and only if $T_i \in V$. Now look at the hexagon of clause C_j ; at least one of the A'_j nodes is not in V , say A'_{jk} . Since the edge (B, A') (which remains in $G - V$) is independent from the edge that connects A'_{jk} to the node associated with the k th literal of C_j , this node must be in V and consequently the corresponding literal is true. Thus, S is satisfied by this truth assignment.

Conversely, suppose that S is satisfiable, and fix a satisfying truth assignment. Delete the nodes associated with true literals. For each clause C_j keep a node A'_{jk_i} whose corresponding literal is true, and delete the other two A'_j nodes. We claim that the remaining subgraph G' satisfies π'_1 . From Lemma 2 it suffices to show that nodes in the same side of G' (either of the two sides) are totally ordered by their neighborhoods. We will show this for the primed nodes. At first, note that no A'_j node is connected in G' to a T_i or F_i node. Thus, $\Gamma(A'_{1k_1}) \subseteq \Gamma(A'_{2k_2}) \subseteq \dots \subseteq \Gamma(A'_{pk_p}) \subseteq \Gamma(A') \subseteq \Gamma(T'_i), \Gamma(F'_i)$ for any i . Also $\Gamma(T'_i), \Gamma(F'_i) \subseteq \Gamma(T'_j), \Gamma(F'_j)$ for $i < j$. In G , T'_i and F'_i were incomparable because of the two independent edges (T_i, T'_i) and (F_i, F'_i) . Since either T_i or F_i is deleted, this is not the case in G' . $\Gamma(T'_i) \subseteq \Gamma(F'_i)$ if x_i is true (and thus T_i is deleted), and $\Gamma(F'_i) \subseteq \Gamma(T'_i)$ if x_i is false. \square

Lemma 4 has an interesting corollary for families of sets. If F is a family of subsets of X , the *restriction of F on $X' \subseteq X$* is the family which consists of the intersections of the sets in F with X' . The family obtained from F by deleting the subfamily F_1 and the subset of elements X_1 is the restriction of $F - F_1$ on $X - X_1$. Lemma 4 then implies that it is NP-complete to find the minimum number of sets and elements whose deletion results in a chain. Note that if only elements or only sets are to be deleted, then the problem can be solved in polynomial time: both cases can be formulated as a longest-path problem in a directed acyclic graph, for which there exists an efficient dynamic programming algorithm (see, for example, [La]).

THEOREM 3. *Let π be a property with $k(\pi) < \infty$ which is hereditary on induced subgraphs and is satisfied by all graphs of \mathbf{G}_1 . The node-deletion π problem restricted to bipartite graphs is NP-complete.*

Proof. Recall the definition of D_t from § 4.2. Let l be the largest integer such that l independent copies of D_t , for any t , satisfy π ; i.e., there exists a t_0 such that $l + 1$ copies of D_{t_0} violate π . There exists such an l since $k(\pi) < \infty$ and $l \leq k(\pi)$. Since all graphs in \mathbf{G}_1 satisfy π , $l \geq 1$. From Lemma 3 it follows that any graph G with at most l nontrivial components K_j with $i(K_j) = 1$ satisfies π .

We will use a reduction from 3-SAT which is a slight modification of the one used in the proof of Lemma 4. If $S = \{C_1, \dots, C_p\}$ is a set of clauses with variables x_1, \dots, x_n and 3 literals per clause, let G be the graph constructed in the proof of Lemma 4, and $m = 4n + 6p + 2$ the order of G . We construct a graph G' as follows. G' has l connected components; $l - 1$ of them are D_t with $t = t_0 + rm$, $r = t_0 m^5$. The l th component K_l is obtained from G as follows. Every node is replaced by r nodes. Every edge $(T_i, T'_i), (F_i, F'_i), (B, A'), (B_{ji}, A'_{jk}), (A'_{jk}, T_i), (A'_{jk}, F_i)$ is replaced by a copy of D_r on the nodes that replaced the endpoints of the edge. (We assume a fixed ordering among

nodes that replaced the same node.) The rest of the edges are replaced by complete bipartite graphs.

We claim that S is satisfiable if and only if $\gamma_\pi(G') \leq r(n+2p)$.

Suppose that S is satisfiable and fix a satisfying truth assignment. We delete from G' all nodes that replaced the nodes that form the solution to the node-deletion π'_1 problem described in Lemma 4. It is easy to see, using arguments similar to those for Lemma 4, that the remaining part of the l th component of G' is in \mathbf{G}_1 . Since every graph in \mathbf{G}_1 is an induced subgraph of some D_b , and from our choice of l , it follows that $\gamma_\pi(G') \leq r(n+2p)$.

Conversely, suppose that $\gamma_\pi(G') \leq r(n+2p)$ and let V be a solution to the node-deletion problem with $|V| = \gamma_\pi(G')$. Let V_i be the subset of nodes in V that belong to the i th component K_i of G' . Let $K_1 = (P_1, Q_1, E_1)$ be the first component of G' (a copy of D_t) with $P_1 = \{v_1, \dots, v_t\}, Q_1 = \{u_1, \dots, u_t\}, E_1 = \{(v_i, u_j) | i \leq j\}$. Suppose that $v_j, u_j \notin V$ and let $i > j$ be such that $v_i \notin V$. Then, in $G' - V$ we have $u_i \in \Gamma(v_j) - \Gamma(v_i) \Rightarrow \Gamma(v_i) \subsetneq \Gamma(v_j)$. In $K_1 - V_1$ there are at least $t - 2|V_1|$ different j 's with $v_j, u_j \notin V$. Thus $K_1 - V_1$ has at least $t - 2|V_1| = \sigma$ different neighborhoods in each side, and from Lemma 3 it contains D_σ as an induced subgraph. Now

$$\begin{aligned} |V_1| &\leq |V| \leq r(n+2p) \\ \Rightarrow \sigma &\geq t - 2r(n+p) = t_0 + rm - 2r(n+p) > t_0 + r(2n+4p) > t_0. \end{aligned}$$

Therefore $K_1 - V_1$ (and similarly for $K_i - V_i, i \leq l-1$) contains D_{t_0} . Consequently, the remaining part of the last component $K_l - V_l$ cannot contain two independent copies of D_{t_0} .

For a node v of G , let the ordering of nodes that replaced v be v_1, v_2, \dots, v_r , where $\Gamma(v_1) \subseteq \Gamma(v_2) \subseteq \dots \subseteq \Gamma(v_r)$ if v is an unprimed node, and $\Gamma(v_1) \supseteq \Gamma(v_2) \supseteq \dots \supseteq \Gamma(v_r)$ if v is a primed node. From the constructions of G' we have, for any $i, (v, u)$ is an edge of G if and only if (v_i, u_i) is an edge of G' . Thus, for every i , the subgraph of G' induced by the v_i 's is a copy of G ; we call it the i th copy G_i of G in G' . Let $e = (v, u')$ be an edge of G that was replaced by a copy of D_r , and let $H(e) = \{i | v_i, u'_i \notin V\}$. Then, the graph induced by $\{v_i, u'_i | i \in H(e)\}$ is D_h , where $h = |H(e)|$. Let $e_1 = (v, u'), e_2 = (x, y')$ be two independent edges of G both of which were replaced by D_r , and let $H(e_1, e_2) = H(e_1) \cap H(e_2)$. The graph induced by $\{v_i, u'_i, x_i, y'_i | i \in H(e_1, e_2)\}$ consists of two independent copies of D_h , where $h = |H(e_1, e_2)|$. Thus, $|H(e_1, e_2)| < t_0$, since $K_l - V_l$ cannot contain two independent copies of D_{t_0} . Let H be the union of all $H(e_1, e_2)$ where e_1 and e_2 are as above, and $\bar{H} = \{1, \dots, r\} - H$. We have

$$|H| \leq \sum |H(e_1, e_2)| \leq \binom{m}{4} t_0 < m^4 t_0 \Rightarrow |\bar{H}| > r - m^4 t_0.$$

For every $i \in \bar{H}$, the remaining graph of the i th copy of G does not contain any pair of independent edges both of which were replaced by a D_r . Since the arguments in the proof of Lemma 4 were based solely on such pairs of independent edges, at least $\gamma_{\pi'_1}(G)$ nodes have to be deleted from every such copy of G . Now, if S is not satisfiable, $\gamma_{\pi'_1}(G) \geq n + 2p + 1$, and consequently

$$\begin{aligned} \gamma_\pi(G') &\geq |\bar{H}| \gamma_{\pi'_1}(G) > (r - m^4 t_0)(n + 2p + 1) \\ &= r(n + 2p) + r - m^4 t_0(n + 2p + 1) \\ &= r(n + 2p) + m^4 t_0(4n + 6p - n - 2p + 1) > r(n + 2p) \quad \square \end{aligned}$$

4.4 Properties with a polynomial node-deletion problem. For a bipartite graph G , let $\nu(G)$ be the number of different neighborhoods of its nodes. Thus, for example, $\nu(I_t) = \nu(B_t) = \nu(D_t) = 2t$. If π is a property, let $\nu(\pi) = \sup\{\nu(G) \mid \text{the bipartite graph } G \text{ satisfies } \pi\}$. All properties π we have considered till now have $\nu(\pi) = \infty$. In the Appendix the following Ramsey-type theorem is shown: For every t , there exists a number $M(t)$ such that every bipartite graph G with $\nu(G) \geq M(t)$ contains as an induced subgraph either I_t or B_t or D_t . Let π be a hereditary property with $\nu(\pi) = \infty$, and suppose that π is not satisfied by I_{t_1}, B_{t_2} and D_{t_3} . It follows from our Ramsey-type theorem that π cannot be satisfied by any graph G with $\nu(G) \geq M(t)$ where $t = \max\{t_1, t_2, t_3\}$, contradicting $\nu(\pi) = \infty$. Consequently, if π is a hereditary property with $\nu(\pi) = \infty$, then π is satisfied either by all I_n , or by all B_n , or by all D_n . That is, we have examined thus far all hereditary properties π with $\nu(\pi) = \infty$. We are going to prove in the remainder of this section that if $\nu(\pi) < \infty$, then the corresponding node-deletion problem is polynomial.

Let G be a graph with $\nu(G)$ different neighborhoods, $V = \{v_1, \dots, v_{\nu(G)}\}$ a set of nodes of G that have different neighborhoods, and $\Gamma_i = \Gamma(v_i)$. The nodes of G are partitioned into $\nu(G)$ subsets $V_1, \dots, V_{\nu(G)}$ according to their neighborhoods; i.e. if $u \in V_i$, then $\Gamma(u) = \Gamma(v_i)$. Let Γ'_i be the neighborhood of v_i in the graph induced by V . Clearly we have $\Gamma_i = \bigcup_{v_j \in \Gamma'_i} V_j$. Therefore, if we know $\langle V \rangle$, the subgraph of G induced by V and the cardinalities of the V_i 's, then we can reconstruct the graph G . A *characteristic graph* of G is a labeled graph G' with $\nu(G)$ nodes labeled $1, 2, \dots, \nu(G)$ which is isomorphic to $\langle V \rangle$ via the mapping $i \rightarrow v_i$. The *characteristic tuple* of G associated with G' is the tuple $\tau = \langle |V_1|, |V_2|, \dots, |V_{\nu(G)}| \rangle$ (see Fig. 7 for an example).

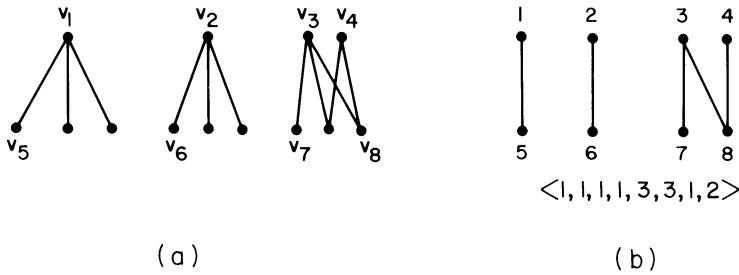


FIG. 7. (a) A graph G . (b) A characteristic graph G' of G and the associated tuple τ .

A graph G can have as many as $\nu(G)!$ pairs (G', τ) of characteristic graph and tuple; however, a pair (G', τ) specifies a unique G . If τ_1, τ_2 are two ν -tuples with $\tau_1 \neq \tau_2$, we say that τ_1 *dominates* τ_2 ($\tau_1 > \tau_2$) if every entry of τ_1 is at least as large as the corresponding entry of τ_2 . (Thus $>$ is a partial order among tuples of the same length.) Note that if τ_1, τ_2 are two tuples of length $\nu(G)$ with $\tau_1 > \tau_2$, then the graph (specified by) (G', τ_2) is an induced subgraph of (G', τ_1) ; and conversely, if G_2 is an induced subgraph of G_1 , $\nu(G_2) = \nu(G_1)$ and G' is a characteristic graph of both, then the associated characteristic tuples τ_2, τ_1 satisfy $\tau_1 > \tau_2$. We will use characteristic graphs and tuples to show that every property π with $\nu(\pi) < \infty$ has a finite description of a certain form. Let $\text{Chg}(\pi)$ be the set of characteristic graphs of all graphs that satisfy π . Clearly, if $\nu(\pi) < \infty$ then $\text{Chg}(\pi)$ is finite. Let $G \in \text{Chg}(\pi)$. With G we associate a set $S(\pi, G)$ of tuples of length $\nu(G) = |G|$. The tuples of $S(\pi, G)$ have entries from $N_\infty = \{1, 2, 3, \dots\} \cup \{\infty\}$. If τ is a tuple with entries from N_∞ , we say that (G, τ) satisfies π if for any tuple τ' obtained from τ by replacing ∞ entries with positive integers, the graph specified by (G, τ') satisfies π . $S(\pi, G)$ is defined to be the set of tuples τ for which (1)

(G, τ) satisfies π , and (2) there is no $\tau' > \tau$ such that (G, τ') satisfies π . (∞ is regarded to be larger than all positive integers.) Clearly, if G is a graph satisfying π , G' a characteristic graph of it and τ the characteristic tuple of G associated with G' , then either $\tau \in S(\pi, G')$ or τ is dominated by some tuple τ' of $S(\pi, G')$. Consequently, a hereditary property π is completely specified by $\text{Chg}(\pi)$ and the sets $S(\pi, G)$: π is satisfied by a graph G if and only if there is a $G' \in \text{Chg}(\pi)$ and a $\tau' \in S(\pi, G')$ such that G' is a characteristic graph of G , and the associated tuple τ of G satisfies $\tau \preceq \tau'$. We will show now that every $S(\pi, G)$ is finite.

LEMMA 5. *Let r be a positive integer and S a set of mutually incomparable (with respect to $<$) r -tuples with entries from N_∞ . Then S is finite.*

Proof. The proof is by induction on r . The basis ($r = 1$) is trivial: S can have at most one tuple, since any two 1-tuples are comparable. So let us assume the lemma for all $m < r$. Let F_1, \dots, F_t be all the distinct subsets of $\{1, \dots, r\}$ such that for each F_i there is a tuple τ_i in S with an ∞ entry exactly at the places of F_i . We have $t \leq 2^r$. Let $\bar{F}_i = \{1, \dots, r\} - F_i$. Let b_{ij} be the j th entry of τ_i with $j \in \bar{F}_i$, $b_{ij} < \infty$. For each $i = 1, \dots, t$, $j \in \bar{F}_i$, $k = 1, 2, \dots, b_{ij}$, let T_{ijk} be the set of tuples in S which have ∞ at the places of F_i (possibly also in other places) and k at the j th place. Let T'_{ijk} be the subtuples of T_{ijk} formed by deleting the j th entry and the entries in F_i . The tuples of T'_{ijk} have length $r - |F_i| - 1$. (Note here that if $|F_i| \geq r - 1$, then τ_i has all entries but the j th ∞ , and therefore $|T_{ijk}| = 1$.) For a tuple $\tau \in T_{ijk}$ let τ' be the subtuple of it in T'_{ijk} . Since any two tuples $\tau, \sigma \in T_{ijk}$ agree in the places that we deleted, we have for the corresponding subtuples $\tau \neq \sigma \Leftrightarrow \tau' \neq \sigma'$ and $\tau > \sigma \Leftrightarrow \tau' > \sigma'$. Thus, $|T_{ijk}| = |T'_{ijk}|$ and T'_{ijk} is a set of mutually incomparable tuples of smaller length. Therefore, we have from the inductive hypothesis that T'_{ijk} is finite, and therefore T_{ijk} too.

Now, let $\tau \in S$, and let F_i be the places in which τ has an ∞ entry, and suppose $\tau \neq \tau_i$. Since τ is incomparable to τ_i , there is at least one place $j \in \bar{F}_i$ in which τ has a smaller entry, say k , than that of τ_i . Thus, $\tau \in T_{ijk}$. Therefore S is the union of the sets T_{ijk} ; that is, S is a finite union of finite sets, and therefore is finite. \square

Note however, that even if $r = 2$ there is no absolute bound on the size of S . For example, for any t , the set $S_t = \{(t - i, t + i) \mid i = 0, 1, \dots, t - 1\}$ has size t . That is, S can be arbitrarily large, though finite.

THEOREM 4. *If π is a hereditary property with $\nu(\pi) < \infty$, then the corresponding node-deletion problem on bipartite graphs can be solved in polynomial time.*

Proof. Let $\text{Chg}(\pi)$ and $S(\pi, G')$ for $G' \in \text{Chg}(\pi)$ be defined as above. Let $c_1(\pi)$ be $\sum_{G' \in \text{Chg}(\pi)} |S(\pi, G')|$. For tuple τ , let $c_2(\tau)$ be the sum of the entries of τ that are not ∞ , and $c_2(\pi) = \max \{c_2(\tau) \mid \tau \in S(\pi, G'), G' \in \text{Chg}(\pi)\}$. Given a bipartite graph $G = (N, E)$ of order n , we solve the node-deletion π problem as follows. At first we choose a pair (G', τ) with $G' \in \text{Chg}(\pi)$, $\tau \in S(\pi, G')$. There are $c_1(\pi)$ choices of (G', τ) . Let $|G'| = r$, $r \leq \nu(\pi) = \nu$. Now we choose a mapping of the nodes of G' into r nodes of G (n^r choices). Let $V = \{v_1, \dots, v_r\}$ be these nodes of G . We verify that G' is isomorphic to $\langle V \rangle$ under this mapping. For every node u we compute $\Gamma(u) \cap V$. If $\Gamma(u) \cap V$ is not equal to any $\Gamma(v_i) \cap V$, then node u is deleted. The remaining nodes are thus partitioned into r sets V_1, V_2, \dots, V_r . Let $F \subseteq \{1, \dots, r\}$ be the places where τ has an ∞ entry, $\bar{F} = \{1, \dots, r\} - F$, and let b_j be the j th entry of τ for $j \in \bar{F}$. For every $j \in \bar{F}$ we pick at most $b_j - 1$ nodes from V_j . Let V' be the set of these nodes. There are at most $[\prod_{j \in \bar{F}} (b_j - 1)] n^{c_2(\tau)} \leq c_2(\pi) n^{c_2(\pi)}$ ways of choosing V' . We verify that a node $u \in V' \cap V_i$ has the same neighborhood in $\langle V \cup V' \rangle$ as v_i . For every node $u \in V_j$ with $j \in F$ we check if $\Gamma(u) \cap (V \cup V') = \Gamma(v_j) \cap (V \cup V')$; if not we delete it. Let V'_j be the remaining nodes and $N' = \bigcup_{j \in F} V'_j$. We construct a (bipartite) graph $H = (N', E')$ on N' with edges $E' = \{(u, v) \mid u \in V'_j \text{ and } v \in (\Gamma(v_j) - \Gamma(u)) \cup (\Gamma(u) - \Gamma(v_j))\}$, and solve the node

cover problem on H . Let V'' be the maximum independent set of H and $L = V \cup V' \cup V''$. We claim that $\langle L \rangle$ satisfies π . It suffices to show that if $u \in V_i \cap L$ then u has the same neighborhood as v_i in $\langle L \rangle$. Suppose without loss of generality that $(u, v) \in E$ and $(v_i, v) \notin E$, for some $v \in L$. If $i \in \bar{F}$, then v cannot be in $V \cup V'$ because we checked that u agrees with v_i in $\langle V \cup V' \rangle$. Thus, $v \in V'_j$ for some $j \in F$. Since $(v_i, v) \notin E$ we have $(v_i, v_j) \notin E$ and consequently $(u, v_j) \notin E$. But then $u \in \Gamma(v) \cap (V \cup V') - \Gamma(v_j) \cap (V \cup V')$, and therefore v cannot be in V'_j . If $i \in F$, then v cannot be in N' because of the edge (u, v) in H . Thus $v \in V \cup V'$ and u shouldn't be in V'_i .

Let L_{\max} be the set L of largest cardinality for all choices of (G', τ) and the mappings we defined, and let $l_{\max} = |L_{\max}|$. We claim that $\gamma_\pi(G) = n - l_{\max}$. Since $\langle L_{\max} \rangle$ satisfies π we have $\gamma_\pi(G) \leq n - l_{\max}$. For the other direction, let $\langle L' \rangle$ be the largest induced subgraph of G with property π , G' a characteristic graph of $\langle L' \rangle$, τ' the associated tuple, $V = \{v_1, \dots, v_r\}$ a set of nodes of $\langle L' \rangle$ with $\langle V \rangle$ isomorphic to G' via the map $v_i \rightarrow i$. Let τ be a tuple in $S(\pi, G')$ with $\tau' \leq \tau$, and F and \bar{F} as before. Let V' (resp. V'') be the set of nodes of $L' - V$ with the same neighborhood in $\langle L' \rangle$ as some v_i , $i \in \bar{F}$, (resp. $i \in F$). At some point, the algorithm will try G', τ, V and V' . Since every $u \in V''$ agrees with some v_i , $i \in F$ in $\langle L' \rangle$ it agrees also in $V \cup V'$. Thus, $V'' \subseteq N'$. Moreover V'' is an independent set of the graph H that will be constructed. Thus, $|L'| \leq l_{\max}$ and $\gamma_\pi(G) \geq n - l_{\max}$. The running time of the algorithm is at most $c_1(\pi)n^v c_2^v(\pi)^{c_2(\pi)} n^3 = c'(\pi)n^{c(\pi)}$. \square

Summarizing our results we have

THEOREM 5. *Let π be a nontrivial property on bipartite graphs which is hereditary on induced subgraphs. The restriction of the node-deletion problem for π on bipartite graphs is polynomial or NP-complete according to whether $\nu(\pi) < \infty$ or $\nu(\pi) = \infty$.*

5. Extension to other structures. As we mentioned in § 4.1, the same proofs go through also for bipartite properties (i.e., properties on bipartitioned graphs). Note now that there is a 1-1 correspondence between bipartitioned graphs and families of sets, hypergraphs, and 0,1 matrices. A bipartitioned graph $BG = (P, Q, E)$ corresponds to the family $F = \{\Gamma(v) | v \in P\}$ of subsets of Q , to the hypergraph with node set Q and edge set F and to the 0,1 matrix with set of rows P and set of columns Q . Thus, Theorem 5 has an analogue in each of these structures. We will state here the result only for 0,1 matrices. Let \mathbf{M} be a class of 0,1 matrices which is closed under permutation and deletion of rows and columns; i.e., if $A \in \mathbf{M}$ then deleting and permuting rows and columns of A results in a matrix in \mathbf{M} . If A is a 0,1 matrix, let $\nu_1(A)$ (resp. $\nu_2(A)$) be the number of distinct rows (resp. columns) of A , $r(A)$ its rank in Z_2 , the integers mod 2. If BG is the bipartitioned graph that corresponds to A we have $\nu(BG) = \nu_1(A) + \nu_2(A) = \nu(A)$. Let $\nu_1(\mathbf{M}) = \sup \{\nu_1(A) | A \in \mathbf{M}\}$, and similarly for $\nu_2(\mathbf{M})$, $\nu(\mathbf{M})$, $r(\mathbf{M})$. The class \mathbf{M} of matrices corresponds to a bipartite property π which is hereditary on induced subgraphs, and $\nu(\mathbf{M}) = \nu(\pi)$. For any 0,1 matrix A we have $\log \nu_1(A)$, $\log \nu_2(A) \leq r(A) \leq \nu_1(A)$, $\nu_2(A)$, since r linearly independent vector span 2^r distinct vectors (in Z_2). Therefore $\nu(\mathbf{M}) = \infty$ if and only if $r(\mathbf{M}) = \infty$.³ Therefore, if we take the size of a matrix to be "number of rows+number of columns", we have

COROLLARY 4. *If \mathbf{M} is a class of 0,1 matrices which is closed under permutation and deletion of rows and columns, then finding the largest submatrix in \mathbf{M} of a matrix is polynomial if the matrices of \mathbf{M} have bounded rank and NP-complete otherwise.*

Examples of classes \mathbf{M} that satisfy the assumptions of Corollary 4 (and have unbounded rank) are: totally unimodular, balanced, with the consecutive ones property, with the circular ones property (see [B], [GJ] for definitions).

³ The same result holds if we take ranks r' in Q , the rationals, since $r(A) \leq r'(A) \leq \nu(A)$.

Appendix. We will show that every 0,1 matrix A of “sufficiently large” rank $r(A)$ (in Z_2 or Q) contains either a “large” identity matrix I_t , or the complement B_t of a “large” identity, i.e., the matrix formed by changing in I_t 0’s to 1’s and 1’s to 0’s, or a “large” lower triangular $t \times t$ matrix D_t with all entries in the lower triangle and the diagonal being 1, and all entries in the upper triangle being 0.

PROPOSITION. For every $t \geq 1$, there is a number $R(t)$ such that every 0,1 matrix A of rank $r(A) \geq R(t)$ contains (after permuting rows and columns) as a submatrix either I_t , or B_t , or D_t .

First we will prove two claims.

CLAIM 1. Let k, t be any positive integers and $m_1(k, t) = 2k(t - 1) + 1$. If A is any 0,1 matrix with $r(A) \geq m_1(k, t)$ and every row and column of A has no more than k 1’s, then A contains I_t .

Proof. We use induction on t with k fixed. The basis ($t = 1$) is trivial. For the induction step, suppose that A is an $r \times r$ 0,1 matrix of rank $r = r(A) \geq m_1(k, t)$ satisfying the conditions of the claim. Assume without loss of generality that the first row has 1 in the first g columns and 0 in the rest, and that the first column has 1 in the first f rows and 0 in the rest. We have $1 \leq f, g \leq k$. Let B be the submatrix of A formed by the last $r - f$ rows and C the submatrix of B formed by the last $r - g$ columns (see Fig. A1). We have $r(B) = r - f \Rightarrow r(C) \geq r - f - g \geq r - 2k \geq 2k(t - 1) + 1 - 2k = 2k(t - 2) + 1 = m_1(k, t - 1)$.

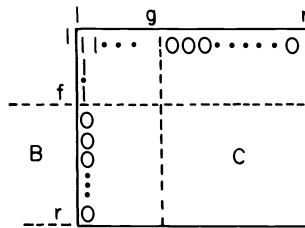


FIG. A1

Since every row and column of C does not contain more than k 1’s, and $r(C) \geq m_1(k, t - 1)$, it follows from the induction hypothesis that C contains I_{t-1} as a submatrix. This submatrix combined with the first row and column gives I_t . \square

CLAIM 2. Let k, t be any positive integers, and $m_2(k, t) = (k + 1)m_1(k, t)$. If A is any 0,1 matrix of rank $r(A) \geq m_2(k, t)$ and every row of A has no more than k 1’s, then A contains I_t .

Proof. Let S_1 be the set of columns with at most k 1’s and S_2 the rest of the columns. The number of 1’s in the $r \times r$ matrix of full rank $r = r(A) \geq m_2(k, t)$ is at most rk since every row has at most k 1’s. Thus, $|S_2|(k + 1) \leq rk \Rightarrow (r - |S_1|)(k + 1) \leq rk \Rightarrow |S_1|(k + 1) \geq r \geq m_2(k, t) \Rightarrow |S_1| \geq m_1(k, t)$. If A' is the submatrix of A formed by the columns in S_1 , then A' satisfies the conditions of Claim 1 and therefore contains I_t . \square

A similar result holds if the columns of A (instead of the rows) satisfy the condition of Claim 2. Also, similar arguments can be used to show that if A is a 0,1 matrix of rank $r(A) \geq m_2(k, t)$ every row (or every column) of which has at most k 0’s, then A contains B_t . We are ready now for the proof of the proposition.

Proof of proposition. We will show that for any (fixed) positive integer t there is a function $f_t(s)$ where $s \geq 1$, such that if the rank of a 0,1 matrix A satisfies $r(A) \geq f_t(s)$ then A contains either I_t or B_t or D_s . The proposition then follows by taking $R(t) = f_t(t)$.

We use induction on s . For $s = 1$ we have $f_t(1) = 1$. For the induction step let $f_t(s) = m_1(m_2(f_t(s - 1), t), t)$, where m_1 and m_2 are defined as in Claims 1 and 2. Let A

be an $r \times r$ matrix of rank $r \geq f_t(s)$. If every row and every column of A has no more than $m_2(f_t(s-1), t)$ 1's, then from Claim 1 A has to contain I_t . So assume without loss of generality that the last row of A has at least $m_2(f_t(s-1), t) + 1$ 1's. Let A' be the submatrix of A formed by the columns in which the last row has an 1 and all the rows but the last (see Fig. A2). We have $r(A') \geq m_2(f_t(s-1), t)$, since all columns of A are linearly independent. Let C be a submatrix of A' formed by $r(A')$ linearly independent

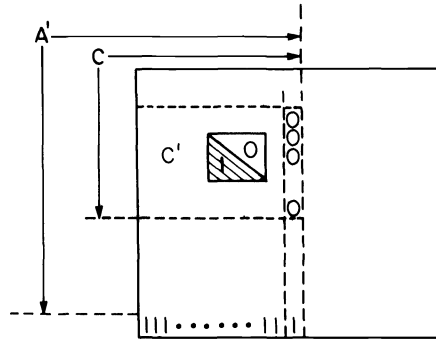


FIG. A2

rows of A' (and all columns). If every column of C has at most $f_t(s-1)$ 0's, then by Claim 2 C contains B_t . Thus, let us assume that the last column of C has a 0 in more than $f_t(s-1)$ rows of C and let C' be the submatrix of C formed by these rows and all columns of C but the last (see Fig. A2). All rows of C' are linearly independent and thus $r(C') \geq f_t(s-1)$. By the induction hypothesis C' contains either I_t or B_t or D_{s-1} . In the last case D_{s-1} together with the last row and column of A' forms D_s . \square

Note. An upper bound for $R(t)$ can be computed from the recursive equation we gave for $f_t(s)$. Since all we need for our purposes is the existence of $R(t)$ we have taken no care in giving a tight proof, and thus we don't expect this upper bound to be tight. It would be an interesting combinatorial problem to find better bounds for $R(t)$.

Following our discussion in § 5, the proposition we just proved implies similar results for families of sets and bipartite graphs. For families of sets the result reads: For every $t \geq 1$ there is a number $N(t)$ such that if F is any family of at least $N(t)$ distinct subsets of a set X , then there is a subset X' of X such that the restriction of F on X' contains either t disjoint nonempty sets, or t sets whose complements with respect to X' are disjoint and nonempty, or a chain of t different nonempty sets (i.e., a *proper chain* of t nonempty sets). An upper bound for $N(t)$ is $2^{R(t)}$.

In terms of bipartite graphs we have: For every t there is a number $M(t)$ such that every bipartite graph G with $\nu(G) \geq M(t)$ contains as an induced subgraph either a set of t independent edges (I_t) or the graph B_t or D_t . It suffices, for example, to take $M(t) = 2N(t)$.

Acknowledgments. I would like to thank J. D. Ullman for the careful reading of an earlier version of the manuscript, and a referee for greatly simplifying the proof of Lemma 1.

REFERENCES

[B] C. BERGE, *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.
 [C] S. A. COOK, *The complexity of theorem-proving procedures*, Proc. Third Annual ACM Symposium on Theory of Computing, 1971, pp. 151-158.

- [E] S. EVEN, *Maximal flow in a network and the connectivity of a graph*, Proc. Eighth Annual Princeton Conf. on Info. Sci. and Systems, 1974.
- [GJ] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1978.
- [GJS] M. R. GAREY, D. S. JOHNSON AND L. STOCKMEYER, *Some simplified NP-complete graph problems*, Theoret. Comp. Sci., 1 (1976), pp. 237–267.
- [H] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1970.
- [K] 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.
- [KD] M. S. KRISHNAMOORTHY AND N. DEO, *Node-deletion NP-complete problems*, this Journal, 8 (1979), pp. 619–625.
- [La] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
- [LDL] J. M. LEWIS, D. P. DOBKIN AND R. J. LIPTON, *Graph properties defined by a forbidden subgraph*, Proc. of the 1977 Conference on Info. Sci. and Systems, John Hopkins Univ., Baltimore, MD., pp. 108–112.
- [LY] J. M. LEWIS AND M. YANNAKAKIS, *The node-deletion problem for hereditary properties is NP-complete*, J. Comput. Systems Sci., 20 (1980), pp. 219–230.
- [S] T. J. SCHAEFER, *The complexity of satisfiability problems*, Proc. Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 216–226.
- [T] R. E. TARJAN, *Testing graph connectivity*, Proc. Sixth Annual ACM Symposium on Theory of Computing, 1974, pp. 185–193.

AN $O(n \log^2 n)$ ALGORITHM FOR THE k th LONGEST PATH IN A TREE WITH APPLICATIONS TO LOCATION PROBLEMS*

N. MEGIDDO, A. TAMIR, E. ZEMEL AND R. CHANDRASEKARAN†

Abstract. Many known algorithms are based on selection in a set whose cardinality is superlinear in terms of the input length. It is desirable in these cases to have selection algorithms that run in sublinear time in terms of the cardinality of the set. This paper presents a successful development in this direction. The methods developed here are applied to improve the previously known upper bounds for the time complexity of various location problems.

Key words. polynomial algorithm, selection, location theory, tree, p -center

1. Introduction. It is now well known that the k th largest element of an ordered set S can be found in linear time in the cardinality of S [1]. Since the discovery of that fact, it has been observed by several authors that in some structured sets, the k th largest element may be found even faster. For example, if $S = X + Y$ (where both X and Y consist of n numbers) then the k th largest element of S can be found in $O(n \log n)$ time, even though $|S| = n^2$. This was first achieved by Jefferson, Shamos and Tarjan [15] and by Johnson and Mizoguchi [11], and later generalized and improved by Frederickson and Johnson [6]. A more general case is the following. Suppose that the set S is partitioned into m sorted subsets such that the k th largest element in each subset can be found in constant time. Fox [5] finds the k th largest element of S in $O(m + k \log m)$ time. Galil and Megiddo [7] solve the problem in $O(m \log^2(|S|/m))$ time. The basic idea of [15] can be used to solve this problem in $O(m \log(|S|/m))$ steps. This was improved by Frederickson and Johnson [6], who solve the same problem in $O(\max\{m, c \log(k/c)\})$ time, where $c = \min(k, m)$. This is also proved to be an asymptotically optimal bound [6], [10].

The structure of S in this latter example is quite abstract. It remains an open question how other structured sets should be handled. For example, suppose that S is the set of all pairs of nodes of a graph, ordered according to the distance (along a shortest path) between the members of the pair. How can we exploit this structure on S for finding the k th largest element? Another interesting example is when S is the set of maximum flows between pairs of source-sinks in a capacitated network.

In this paper we develop an algorithm for the k th largest element in the set of all simple paths in a tree with edge-lengths. The cardinality of this set is $O(n^2)$ (n is the number of nodes in the tree and each simple path is characterized by its two endpoints). However, our algorithm runs in $O(n \log^2 n)$ time. This fast method of selecting an internodal distance is shown to be very useful in the solution of different combinatorial location problems.

The organization of the paper is as follows. In § 2 we review the two basic approaches to selection in an ordered set with sorted subsets. In § 3 we discuss a decomposition scheme for trees, on which the partition of the set of paths is based. The partitioning itself is developed in § 4 and the solution of the selection problem in the set of paths is summarized in § 5. A brief survey of four different location problems is given in § 6. In § 7 we apply the methods developed in this paper to obtain improved algorithms for the location problems defined in § 6. In § 8 we briefly discuss the more general case of weighted location problems.

* Received by the editors May 5, 1979, and in revised form April 21, 1980.

† Northwestern University, Graduate School of Management, Evanston, Illinois 60201.

2. An overview of selection algorithms. Suppose that an ordered set S is partitioned into m subsets S_1, S_2, \dots, S_m such that the k th largest element in each subset can be found in constant time. We distinguish between two methods of selection in S . The first one, which we like to call "trimming," is attributed to Jefferson, Shamos and Tarjan [15] and is also used in Frederickson and Johnson [6]. The second, which we call "splitting," is a generalization of linear-time median finding [1] and is used in Johnson and Mizoguchi [11] and in Galil and Megiddo [7]. For simplicity of exposition, we assume in this section that all members of S are distinct. Handling the general case of S being a multiset is similar (see the above references). At a given iteration, let $S'_i \subseteq S_i$ be the set of elements still under consideration with $S' = \bigcup_{i=1}^m S'_i$.

A. Trimming. Suppose that we are looking for the k th largest element x of S' , and assume without loss of generality that $k \leq \frac{1}{2}|S'|$. We first find the lower quartile, y_i , in each S'_i . Next, we consider the set $Y = \{y_1, \dots, y_m\}$ where each y_i is weighted by $|S'_i|$, and we find the (weighted) lower quartile y of Y . Obviously, at least one half of the elements of S' are greater than y , and hence $y \leq x$. We can now reduce the set S' by discarding the lower quarter of each subset S'_i for which $y_i \leq y$. This amounts to discarding $\frac{1}{16}$ of the set S' , and the problem now reduces to finding the k th largest element of the remaining set.

B. Splitting. In this method we first find the median z_i in each S'_i . Next, we find the weighted median z of $Z = \{z_1, \dots, z_m\}$ (relative to the weights $|S'_i|$). Obviously, z is between the lower and upper quartiles of S' . By computing the rank of z in S' , we can tell whether $z \geq x$ or $z < x$. In the latter case, the lower half of every S'_i such that $z_i \leq z$ can be discarded, and we look for the k th largest element in the remaining set. Otherwise, the upper half of every S'_i such that $z_i \geq z$ is discarded, and we look for the $(k - \frac{1}{4}|S'|)$ th largest element of the remaining set.

It is interesting to compare the logic and overall efficiency of splitting and trimming. One difference between the two methods lies in the position of elements they eliminate. At any given iteration, splitting may eliminate elements from the upper or lower quartiles of S' , depending on the outcome of a logical test. In contrast, the elimination process of trimming is not based on any test, and the elements eliminated always come from lower parts of S' if $k \leq \frac{1}{2}|S'|$ and from its upper part if the reverse condition holds. As will be pointed out in § 7, a procedure similar to splitting (i.e., based on a logical test) turns out to be preferable for solving various location problems on a tree. As for the efficiency of identifying the k th element of S , we note that, in the worst case, splitting eliminates at each iteration more variables than trimming (four times as many in the formulation given above, although the difference can be reduced by a slight modification of the trimming procedure). However, the corresponding reduction in the number of iterations enjoyed by the splitting method is more than offset by the effort involved in identifying the rank of z in S' which is necessary to support the logical test. Thus, while the overall complexity achieved by the splitting procedure is $O(m \log^2(|S|/m))$ the corresponding complexity for trimming is only $O(m \log(|S|/m))$.

Our algorithm for the k th longest simple path in a tree exploits the structure of the set S of paths in the following way. We partition S into $m = O(n \log n)$ subsets S_1, \dots, S_m with the following properties.

- (i) The k th largest element in any S_i , as well as its length, can be computed in constant time.
- (ii) All the elements of each S_i are paths leading from the same node v_i to other nodes of the tree.

(iii) The k largest and the k smallest elements of any S_i can be discarded in constant time.

(iv) The partitioning process is carried out in $O(n \log^2 n)$ time and $O(n \log n)$ space.

Once this partition is obtained, one can employ the trimming algorithm for finding the k th longest path. This amounts to a time bound of $O(n \log^2 n)$. Details are worked out in the following sections. The partitioning is carried out by a divide-and-conquer algorithm on the tree T . The first step in this direction is an efficient decomposition of the tree which we describe in the next section.

3. Decomposition of trees. In this section we show how to decompose a tree T into three (or fewer) subtrees such that precisely one node of T belongs to more than one subtree and such that each subtree has no more than $n/2 + 1$ nodes (where the set of nodes of T is $N = \{1, \dots, n\}$).

Suppose that the tree T is given in the form of lists $N(i)$ of all the neighbors of a node i ($i = 1, \dots, n$). If i and j are neighbors, then by removing the edge (i, j) two subtrees of T are induced. We denote by $K(i, j)$ the number of nodes in that subtree which contain node i . (Note that K is defined on *ordered* pairs of neighboring nodes.) It is easy to verify the following:

- (1) If i is a leaf where $N(i) = \{j\}$ then $K(i, j) = 1$.
- (2) $K(i, j) + K(j, i) = n$ for all pairs of neighbors.
- (3) For all j , $\sum_{i \in N(j)} K(i, j) = n - 1$.
- (4) If $j, k \in N(i)$ ($j \neq k$) then $K(j, i) < K(i, k)$.

In order to decompose T in the manner described above, we need to find a node x such that for all $i \in N(x)$, $K(i, x) \leq n/2$. The existence of such a node, referred to as the *centroid* of T , was observed by Jordan in 1869 [12]. Linear time algorithms for finding the centroid appear in Goldman [8] and Kariv and Hakimi [13]. For the sake of completeness we provide such an algorithm below.

We first note that the computation of all the $K(i, j)$ s can be carried out in $O(n)$ time. This is done as follows. Fix one of the nodes r as the "root," so that every other node i has a "father" $f(i)$ relative to r (i.e., $f(i)$ is the node following i on the path from i to r).

The quantities $K(i, f(i))$ ($i \neq r$) can be computed recursively by $K(i, f(i)) = 1 + \sum_{j: f(j)=i} K(j, f(j))$, and the computation of all $K(i, j)$ s can be completed by (2). The whole process takes $O(n)$ time.

Once all the $K(i, j)$ s are known, the following process can be used to find a node x such that $K(i, x) \leq n/2$ for all $i \in N(x)$.

- (1) $x \leftarrow 1$
if $K(i, x) \leq n/2$ for all $i \in N(x)$ **then stop**
else (there is precisely one $i \in N(x)$ such that $K(i, x) > n/2$) $x \leftarrow i$
go to 1

This procedure generates a path $1 = x_1, \dots, x_k = x$ such that $K(x_{j+1}, x_j) > n/2$ ($j = 1, \dots, k - 1$). By (4) and (2), the function $m(x_j) \equiv \max_{i \in N(x_j)} K(i, x_j)$ is monotone decreasing along that path, and hence an $x_k = x$ is reached for which $m(x) \leq n/2$. Obviously, this procedure takes $O(n)$ time.

We now claim that the set $N(x)$ can be partitioned into three or fewer subsets N_1, N_2, N_3 such that $\sum_{i \in N_j} K(i, x) \leq n/2$. This is easily proved as follows. Assume $N(x) =$

$\{v_1, \dots, v_p\}$. By (3) there is s ($1 \leq s \leq p$) such that

$$\sum_{i=1}^{s-1} K(v_i, x) \leq \frac{n-1}{2} \quad \text{and} \quad \sum_{i=s+1}^p K(v_i, x) \leq \frac{n-1}{2}.$$

The desired subsets are $N_1 = \{v_1, \dots, v_{s-1}\}$, $N_2 = \{v_s\}$, $N_3 = \{v_{s+1}, \dots, v_p\}$.

Finally, the partition of $N(x)$ induces a decomposition of T into three or fewer subtrees T_1, T_2, T_3 ; namely, T_j is the subtree consisting of x and all the nodes accessible from x via a member of N_j ($j = 1, 2, 3$). Obviously, x is the only node of T that belongs to more than one such subtree, and also in each T_j there are no more than $(n/2) + 1$ nodes. The decomposition is carried out in $O(n)$ time.

4. Partition of the set of paths in a tree. In the preceding section we described a decomposition of a tree into three subtrees with a single node x common to the three of them. We refer in this section to that node x as the “decomposer.” In this section, S is the set of all simple paths in a tree T . Since there is a one-to-one correspondence between pairs of nodes and simple paths in a tree, we also consider S as the set of pairs of nodes, ordered according to the internodal distances. We partition S into subsets such that the k th largest element in any subset can be found in constant time.

The essence of the partitioning algorithm is as follows. First, we find a decomposer x (see § 3) and we look at the three subtrees T_1, T_2, T_3 in the corresponding decomposition. For each T_i ($i = 1, 2, 3$), we compute all the distances from the node x to all other nodes of T_i , and we sort the set S_i of all simple paths leading from x into T_i according to these distances. Thus, the node x contributes three sorted subsets to our partition of S . Next, for each node $j \neq x$ in T_1 we can easily compute the sorted set of distances from j to all nodes of T_2 , since this is obtained by adding a constant (namely, the distance between j and x) to all elements of S_2 . Analogously, for each $j \neq x$ in either T_1 or T_2 , we compute the sorted set of distances from j to all nodes of T_3 , by adding the distance between j and x to all elements of S_3 . Thus, each node $j \neq x$ of T_1 contributes at this stage two sorted subsets and each $j \neq x$ in T_2 contributes one sorted subset to our partition of S . We proceed by decomposing the subtrees T_1, T_2, T_3 , each along the same lines described above, until all the paths (or equivalently, pairs) are enumerated. Throughout this process, we skip paths leading to or from nodes that have previously served as decomposers, to make sure that each pair of nodes is taken into account precisely once.

The number of subsets created during the partitioning process is estimated as follows. Let $M(n)$ denote the maximum number of subsets in such a partition of S for a tree with n vertices. The tree is decomposed into three subtrees. If n_1, n_2, n_3 are the numbers of nodes in these subtrees, then $n_1 + n_2 + n_3 = n + 2$ and $n_i \leq n/2 + 1$. Each node contributes no more than three subsets to the partition of S , and we proceed, recursively, with the subtrees. Hence

$$M(n) \leq 3n + M(n_1) + M(n_2) + M(n_3),$$

and it follows that $M(n) = O(n \log n)$.

We now estimate the running time $T(n)$ of the partitioning process. It is very essential to note here what is meant by “creating” subsets. The creation of the subsets contributed by the first decomposer requires $O(n \log n)$ time, since we need to compute all distances from the decomposer to all other nodes and then sort them. However, the creation of other subsets (i.e., subsets contributed by nondecomposers) requires only a few pointers, as discussed later in this section. Thus, the general step in the partitioning process consists of: (i) tree decomposition, $O(n)$; (ii) computing all distances from a

single node, $O(n)$; (iii) sorting the set of all these distances and discarding those associated with previous decomposers, $O(n \log n)$; (iv) creating the subsets, pointers and constants, $O(n)$. Thus, the recursive relation is

$$T(n) \leq Cn \log n + T(n_1) + T(n_2) + T(n_3),$$

and therefore $T(n) = O(n \log^2 n)$.

Next, we discuss the storage aspects of the partitioning algorithm. Whenever a node x serves as a decomposer for a subtree T_1 , three sorted sets R_1, R_2, R_3 of distances from x into T_1 are generated. We distinguish between the sets R_i and the subsets S_i that actually constitute our partition. Each set is stored as an array, and the total space for storing these arrays is $O(n \log n)$. (This can be proved by induction.) The partition of S into subsets S_1, \dots, S_m , as well as the reduced forms of S that are processed by the trimming or splitting procedures (see § 2), are handled as follows. Each S_i is characterized by four items. First is a pointer to the corresponding R_j from which S_i is created. Second is a constant number that should be added to an element of R_j in order to get an element of S_i . Third and fourth are two pointers needed to specify the boundary of that portion of R_j from which S_i is generated. (These two pointers are at the start the same for all the S_i s that rely on the same R_j , but during the trimming or splitting process they may become different.) Thus, the total amount of storage that we need is $O(n \log n)$. In addition, at most $O(n \log n)$ storage is required in order to properly maintain the set of trees T_i which are generated throughout the algorithm.

We conclude this section with a pidgin Algol description of the partitioning process. It receives as input a tree T with a set of nodes $N = \{1, \dots, n\}$ and produces as output a partition S_1, \dots, S_m of the set of internodal distances of T , where $m = O(n \log n)$. The sets S_1, \dots, S_m satisfy the properties (i)–(iii) of § 2. The overall complexity bounds for the algorithm are $O(n \log^2 n)$ time and $O(n \log n)$ space. The procedure uses the following terminology:

Q	current set of subtrees not yet subdivided.
B	current set of nodes which have not as yet served as decomposers.
R_j	j th sorted set of distances between a decomposer and the nodes of a subtree.
k	index for set S_k used in the partition.
$\gamma(k)$	a label identifying the index of subset R_j used to create S_k .
$\beta(k)$	the constant increment which must be added to each element of R_j to get the corresponding element of S_k .

In addition, the procedure uses the following subroutines in the course of its execution:

CENTROID (T)	Given a tree T returns its centroid.
SUBTREE (T, x, i)	Given a tree T , its centroid x and an index $i = 1, \dots, 3$, returns the subtree T_i (see last paragraph of § 3).
DISTANCE (T, A, B)	Given a tree T and two sets of nodes A and B returns a vector of all the distances $d(i, j)$, $i \in A, j \in B, i \neq j$.
SORT (D)	Given a vector D , returns the entries in a sorted way.

Procedure DECOMPOSE (T)

```

begin
   $Q \leftarrow T$ 
   $B \leftarrow N$ 
   $j \leftarrow 0$ 
   $k \leftarrow 0$ 

```

```

while  $Q \neq \emptyset$  do
  begin
    choose  $T'$  from  $Q$ 
     $x \leftarrow \text{CENTROID}(T')$ 
    for  $i = 1, \dots, 3$  do
      begin
         $T_i \leftarrow \text{SUBTREE}(T, x, i)$ 
         $N_i \leftarrow \text{Nodes of } T_i$ 
         $N'_i \leftarrow N_i \cap B \setminus \{x\}$ 
         $D_i \leftarrow \text{DISTANCE}(T_i, x, N'_i)$ 
         $j \leftarrow j + 1$ 
         $v_i \leftarrow j$ 
         $R_j \leftarrow \text{SORT}(D_i)$ 
      end
    for each  $j \in N'_1$  do
      begin
         $k \leftarrow k + 1$ 
         $\gamma(k) \leftarrow v_2$ 
         $\beta(k) \leftarrow d(j, x)$ 
         $k \leftarrow k + 1$ 
         $\gamma(k) \leftarrow v_3$ 
         $\beta(k) \leftarrow d(j, x)$ 
      end
    for each  $j \in N'_2$  do
      begin
         $k \leftarrow k + 1$ 
         $\gamma(k) \leftarrow v_3$ 
         $\beta(k) \leftarrow d(j, x)$ 
      end
    if  $x \in B$  do
      begin
        for  $i = 1, \dots, 3$  do
          begin
             $k \leftarrow k + 1$ 
             $\gamma(k) \leftarrow v_i$ 
             $\beta(k) \leftarrow 0$ 
          end
         $B \leftarrow B \setminus \{x\}$ 
      end
    for  $i = 1, \dots, 3$  do
      begin
        if  $|N_i| \geq 3$  then  $Q \leftarrow Q \cup T_i$ 
        else
           $D_i \leftarrow \text{DISTANCE}(T, N'_i, N'_i)$ 
           $j \leftarrow j + 1$ 
           $R_j \leftarrow \text{SORT}(D_i)$ 
           $k \leftarrow k + 1$ 
           $\gamma(k) \leftarrow j$ 
           $\beta(k) \leftarrow 0$ 
        end
      end
  end
end

```


5. The k th longest path in a tree. Once the partition of the set S of all paths into $m = O(n \log n)$ subsets is established, one can use the techniques introduced in § 2 to find the k th longest path. This amounts to an effort of $O(m \log n/m) = O(n \log^2 n)$ if one uses trimming and an inferior bound, of $O(n \log^3 n)$, if splitting is used. As the effort involved in generating the partition of S is also $O(n \log^2 n)$, we can conclude that the overall effort for finding the k th longest path in T is $O(n \log^2 n)$.

Can this bound be further beaten down? Possibly, but the margin for improvement is slim. An $O(n \log n)$ lower bound on the complexity of the problem can be obtained in a number of ways. The following simple reduction was offered to us by one of the referees. Consider the tree of Fig. 1 where the heavy line in the center is chosen long enough to ensure that the longest paths in T include one element from X and one from Y . Thus, the well-known $O(n \log n)$ bound on selection in $X + Y$ is valid for our problem as well.

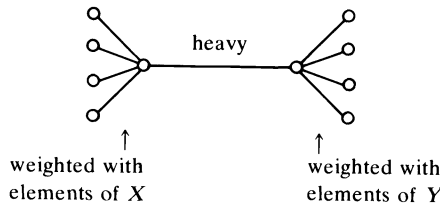


FIG. 1

6. Location problems. We consider here the following different problems of location. First, we assume that a tree T is embedded in the Euclidean plane, so that the edges are line segments whose endpoints are the nodes and whose edges intersect one another only at nodes. Moreover, each edge has a positive length. (Any tree with positive edge-weights can be so embedded in R^2). This embedding enables us to talk about points, not necessarily nodes, on the edges. We then denote by $d(x, y)$ the distance, measured along the edges of the tree, between any two points x, y of the tree.

In a typical location problem, one has to select p points of the tree under different assumptions depending on the particular model considered. In each model, we distinguish between the “supply” set Σ (this is the set from which we select the p points) and the “demand” set Δ , with reference to which the objective function is defined. The p -center problem seeks to choose p points x_1, \dots, x_p from Σ so as to minimize $\sup_{y \in \Delta} \min_{1 \leq i \leq p} d(x_i, y)$. The four special cases, where the sets Σ and Δ are either the set of all nodes or the set of all points of the tree, have been discussed and given different algorithms in [2], [3], [4], [9], [13].

Following Handler [9], we use the categorization scheme $\{N\}_A/\{A\}_N/p$, interpreted as follows. The first cell describes the supply set Σ , which could be either the set N of all nodes or the set A of all points. The second cell describes the demand set Δ , which could also be either N or A . The third cell indicates the number of points that we have to select from Σ . For example, $N/A/2$ refers to selecting two nodes so as to minimize the maximum (over all points of the tree) of a distance between a point of the tree and the selected node that is nearest to that point. Kariv and Hakimi [13] provide $O(n^2 \log n)$ algorithms for $A/N/p$ and $N/N/p$. Chandrasekaran and Tamir [3], using a unified approach, solve $A/N/p$, $N/N/p$ and $N/A/p$ in $O(n^2 \log n)$ time. The $A/A/p$ problem is solved in [4] in $O(n^2 \log^2 p)$ time.

All the algorithms mentioned above are based on the same principle. First, a finite set R of real numbers, which is known to contain the optimal objective function value, is identified. Next, we search R for the minimum value which is feasible in the following

sense. A value $r > 0$ is feasible if there exists a set of p points x_1, \dots, x_p of Σ such that the distance between any demand point y and its nearest x_i , is not greater than r . Efficient algorithms are known for deciding whether a given r is feasible, and hence the location problem can be solved by a binary search of R using such a feasibility test. For all four problems this test runs in $O(n)$ time. (See [13] for $N/N/p$ and $A/N/p$ and [4] for $N/A/p$ and $A/A/p$.) The set R of relevant values in the four different problems is given in Table 1 (see [3], [4], [13]).

TABLE 1

Model	The set R
$N/N/p$	$\{d(i, j)\}_{i, j \in N}$
$A/N/p$	$\{\frac{1}{2}d(i, j)\}_{i, j \in N}$
$N/A/p$	$\{d(i, j), \frac{1}{2}d(i, j)\}_{i, j \in N}$
$A/A/p$	$\{(1/2k)d(i, j)\}_{i, j \in N, k=1, \dots, p}$

Along the lines discussed above, each one of these problems can be solved by computing the set R and then searching R by repeatedly using linear-time median finding [1]. This amounts to $O(|R| + n \log |R|)$ time where $|R|$ is the dominant term. Thus, in order to improve this upper bound, one has to bypass the computation of the set R and still be able to search in that set. This is essentially where we apply the techniques developed in the previous sections.

7. Improved algorithms for location problems. The sets R of relevant values for the various versions of the p -center problem bear a close resemblance to the set S of internodal distances on T . Thus, we can use any algorithm for finding the k th longest element in S to support a binary search over R . Such search involves at each iteration identifying the median element of R , performing the feasibility test and finally discarding one half of the elements. However, we note that identifying the median element at each iteration may be more than one needs. In fact, one can do better by applying a search strategy similar to that of splitting.

Assume that the set R is partitioned into m subsets R_1, \dots, R_m such that the k th largest element in each subset can be found in constant time. We can employ the following procedure. First, we find the median element z_i in each subset R_i . Next, we find the median element z in the set $z = (z_1, \dots, z_m)$ relative to the weights $|R_i|$. Thus, z is between the lower and upper quartiles of R . This value z can now be tested for feasibility. The test takes $O(n)$ time and determines whether the optimal value v is greater than z (this is when z is not feasible) or not. If $v > z$, we discard the lower half (including z_i) from each R_i such that $z_i \leq z$. If $v \leq z$, we discard the upper half (including z_i) from each R_i such that $z_i \geq z$, with the exception that z itself is not discarded. The search then proceeds with the reduced set R until the optimal value is singled out.

Since each reduction eliminates one quarter of the remaining set, the number of such stages is $O(\log |R|)$. Following is a more detailed analysis for the particular cases.

A. $N/N/p$. Here R is the set of the internodal distances. It follows from § 4 that R can be appropriately partitioned into $O(n \log n)$ subsets where the partitioning process takes $O(n \log^2 n)$ time. During the searching process, in each iteration we need $O(n \log n)$ time for identifying the element z and $O(n)$ time for the feasibility test. Thus, the searching stage takes $O((n + n \log n) \log |R|)$ time, and hence the location problem is solved in $O(n \log^2 n)$ time.

B. $A/N/p$. Since $R = \{\frac{1}{2}d(i, j) : i, j \in N\}$ in this case, the location problem is solved by the same partition which is used in $N/N/P$. Hence, the time bound for this case is $O(n \log^2 n)$.

C. $N/A/p$. The relevant set here is $R = \{d(i, j), \frac{1}{2}d(i, j) : i, j \in N\}$. Thus, we use essentially the same partition as the one for $N/N/p$ and $A/N/p$, but in terms of pairs of nodes, each pair is counted twice: once for the distance $d(i, j)$ and once for the number $\frac{1}{2}d(i, j)$. This implies the same time bound of $O(n \log^2 n)$ for this case too.

D. $A/A/p$. This last case is slightly more complicated than the previous ones. Since $R = \{(1/2k)d(i, j) : i, j \in N, k = 1, \dots, p\}$ in this case, one way of partitioning R is by using the partition of § 4 for the set of pairs of nodes and replicating each subset p times, so that each $d(i, j)$ is multiplied by all the numbers $\frac{1}{2}, \frac{1}{4}, \frac{1}{6}, \dots, 1/2p$. Thus, R is partitioned into $m = O(pn \log n)$ subsets while $|R| = O(pn^2)$. By applying the searching method, R is successively reduced by a factor of one quarter. Let the set of remaining variables at a given iteration be R' , and denote by $T(|R'|)$ the time required by the algorithm to handle this set. We now have

$$T(R') \leq C_1 m + c_2 n + T(3/4|R'|).$$

When the cardinality of R' reaches the level $|R'| = O(m)$ we can search over R' directly using the method of linear-time median finding. This involves, at each iteration, finding the median element and performing the test. The total effort involved in the identification of *all* the median elements is clearly $O(m)$. Also, since the number of iterations is $O(\log m)$, and since each test requires an effort of $O(n)$, the total effort associated with handling a set of cardinality $O(m)$ is $O(n \log m) = O(m)$. Solving the recursion relation with the initial condition $T(m) = O(m)$ we then get that $T(|R'|) = O(m \log (|R'|/m))$ and hence the location problem is solved in this approach in time

$$O\left(pn \log n \log \left(\frac{n}{\log n}\right)\right) = O(pn \log^2 n).$$

There is an alternative partition that in some cases leads to a better upper bound. We first compute the $\frac{1}{2n}(n-1)$ internodal distances (in $O(n^2)$ time). Then we partition R into $m = \frac{1}{2}n(n-1)$ subsets of the form $R_{ij} = \{(1/2k)d(i, j) : k = 1, \dots, p\}$, where computing the k th largest element in each set is trivial. Applying the searching procedure we obtain the following bound:

$$O(m \log (|R|/m)) = O(n^2 \log p).$$

To summarize, the different cases are solved with the upper bounds in Table 2.

TABLE 2

Model	Upper bound
$N/N/p$	$O(n \log^2 n)$
$A/N/p$	$O(n \log^2 n)$
$N/A/p$	$O(n \log^2 n)$
$A/A/p$	$O(n \min \{p \log^2 n, n \log p\})$

8. Location problem with weighted demands. A more general type of location problem is where the demand is weighted. Specifically, when $\Delta = N$ we may have weights $w_i > 0$ ($i \in N$) and seek to select $x_1, \dots, x_p \in \Sigma$ so as to minimize

$$\max_{i \in N} \{w_i \cdot \min_{1 \leq j \leq p} d(x_j, i)\}.$$

It is shown in [3], [13] that the relevant set R generalizes to $\{w_i d(i, j)\}_{i, j \in N}$ in the $N/N/p$ case and to $\{(w_i w_j / (w_i + w_j)) d(i, j)\}_{i, j \in N}$ in the $A/N/p$ case. Both cases are solvable in $O(n^2)$ time [4], [13], but based on our method an $O(n \log^2 n)$ algorithm for the weighted $N/N/p$ case can be constructed as follows.

Essentially, we consider the set S' of all *ordered* pairs of nodes together with the linear order induced by the weighted distances $w_i d(i, j)$. The set S' is partitioned, along lines similar to those of § 4, into $O(n \log n)$ subsets. All the pairs (i, j) in any subset are with the same i , hence the restriction of the order to each subset is independent of the weight w_i . All we have to do during the algorithm, is to multiply the k th largest distance in a set corresponding to i by the weight w_i . Thus the partition satisfies all the properties that are required to obtain a bound of $O(n \log^2 n)$.

REFERENCES

- [1] M. BLUM, R. W. FLOYD, V. R. PRATT, R. L. RIVEST AND R. E. TARJAN, *Time bounds for selection*, J. Comp. Sys. Sci., 7 (1972), pp. 448–461.
- [2] R. CHANDRASEKARAN AND A. DAUGHETY, *Location on tree networks: p -center and n -dispersion problems*, Math. Oper. Res., to appear.
- [3] R. CHANDRASEKARAN AND A. TAMIR, *Polynomially bounded algorithms for locating p -centers on a tree*, Discussion Paper No. 358, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL, 1978.
- [4] ———, *An $O((n \log P)^2)$ algorithm for the continuous p -center problem on a tree*, Discussion Paper No. 367, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL, 1978.
- [5] B. L. FOX, *Discrete optimization via marginal analysis*, Management Sci., 13 (1966), pp. 210–216.
- [6] G. N. FREDERICKSON AND D. B. JOHNSON, *Optimal algorithms for generating quantile information in $X + Y$ and matrices with sorted columns*, Proceedings of the 1979 Conference on Information Sciences and Systems, The Johns Hopkins University, to appear.
- [7] Z. GALIL AND N. MEGIDDO, *A fast selection algorithm and the problem of optimum distribution of effort*, J. Assoc. Comput. Mach., 26 (1979), pp. 58–64.
- [8] A. J. GOLDMAN, *Optimal center location in simple networks*, Transportation Sci., 5 (1971), pp. 212–221.
- [9] G. Y. HANDLER, *Finding two-centers of a tree: The continuous case*, Transportation Sci., 12 (1978), pp. 93–106.
- [10] D. B. JOHNSON AND D. S. KASHDAN, *Lower bounds for selection in $X + Y$ and other multisets*, J. Assoc. Comput. Mach., 25 (1978), pp. 556–570.
- [11] D. B. JOHNSON AND T. MIZOGUCHI, *Selecting the K th element in $X + Y$ and $X_1 + X_2 + \dots + X_m$* , this Journal, 7 (1978), pp. 147–153.
- [12] C. JORDAN, *Sur les Assemblages des lignes*, J. Reine Angew. Math., (1869), pp. 185–190.
- [13] O. KARIV AND S. L. HAKIMI, *An algorithmic approach to network location problems. I: The p -centers*, SIAM J. Appl. Math., 37 (1979), 513–538.
- [14] ———, *An algorithmic approach to network location problems. II: The p -medians*, SIAM J. Appl. Math., 37 (1979), pp. 539–560.
- [15] M. I. SHAMOS, *Geometry and Statistics: Problems at the Interface*, in Algorithms and Complexity: New Directions and Recent Results, J. F. Traub, ed., Academic Press, New York, 1976, pp. 251–280.

OPTIMIZATION PROBLEMS ON GRAPHS WITH INDEPENDENT RANDOM EDGE WEIGHTS*

GEORGE S. LUEKER†

Abstract. Optimization problems on complete graphs with edge weights drawn independently, from a fixed distribution, are considered. Several methods for analyzing these problems are discussed, including greedy methods, applications of Boole's inequality, and exploitation of relationships with results about random unweighted graphs. These techniques are illustrated in the case in which the edge weights are drawn from a normal distribution; in particular, we investigate the expected behavior of the minimum weight clique on k vertices. We describe the asymptotic behavior (in probability and/or almost surely) of the random variable which describes the optimum; we also discuss the asymptotic behavior of its mean. Finally techniques are demonstrated by which we may determine an asymptotic description of the behavior of a greedy algorithm for this problem.

Key words. random graphs, optimization problems, normal distribution, weighted graphs, probabilistic analysis, traveling salesman problem, cliques, Boole's inequality, greedy algorithms, convergence in probability, almost sure convergence

1. Introduction. Many results have been proven about the properties of random graphs. Some of these [1], [3], [9], [10], [12], [19], [26], [27], [29] deal with graphs constructed by letting each possible edge be present with a specified probability; one then tries to estimate the probability that a subgraph of a given type will be present. ([11] may be considered a paper about random directed graphs.) We will call such a problem a *subgraph existence* problem. Another area of interest is algorithms on graphs in which all edges are present but weights are assigned to the edges according to some distribution; one then tries to find the minimum weight subgraph of a given type. We will call such problems *subgraph optimization* problems; they are the subject of this paper. For example, if a traveling salesman problem is constructed using the Euclidean distance between n points chosen from a uniform distribution in the unit square, then asymptotically the optimum solution tends to be proportional to $n^{1/2}$ [2]; very efficient algorithms have been designed whose asymptotic behavior tends to be optimal [23], [21]. In this paper we assume that the edge weights are independent; see [31], where a similar, though slightly more general, model is discussed. The assignment problem for the case in which edge weights are chosen independently from various distributions has been analyzed by Borovkov [4]. For this problem, it appears that tight bounds are particularly difficult to obtain in the case in which edge weights are chosen from a uniform distribution; this case has been further pursued by Walkup [30]. Similarly, for the traveling salesman problem it appears to be particularly difficult to obtain tight bounds on the behavior of the true optimum in the case where the edges are drawn from a uniform distribution; see [24] for an algorithm for this case (with directed graphs) which tends to give nearly optimum solutions for large n .

In § 2, we present some basic definitions and facts. In § 3, we will discuss a very general technique which has been used by a number of researchers for obtaining lower

* Received by the editors March 21, 1978, and in final revised form May 30, 1980. This work was supported by the National Science Foundation under grants MCS77-04410 and MCS79-04997. This paper presents and extends the results in *Maximization problems on graphs with edge weights chosen from a normal distribution*, presented at the 10th ACM Symposium on Theory of Computing, held in San Diego in May, 1978.

† Department of Information and Computer Science, University of California, Irvine, California 92717.

bounds on the values of solutions to optimization problems, based on Boole's inequality. Section 4 discusses a very general technique which has been used by a number of researchers for obtaining upper bounds on these values, using theorems about subgraph existence problems. Surprisingly, combining the bounds discussed in §§ 3 and 4 sometimes enables us to make rather precise statements about the asymptotic behavior of the minimum, as will be demonstrated in § 5. Since many optimization problems are NP-complete [17], [22], it is useful to investigate the behavior of heuristics. In § 6 we will investigate the behavior of some greedy algorithms.

2. Definitions. We will frequently discuss probabilities and expected values. If X is a random variable and A and B are events, let $P\{A\}$ be the probability of A , $P\{A|B\}$ be the probability of A given B , $E[X]$ be the expected value of X , and $E[X|A]$ be the expected value of X given A .

Throughout this paper, \mathcal{G}_n will be a random structure which is a complete, weighted, labeled graph on n vertices; we will assume the vertices are labeled $1, 2, \dots, n$. Weights are chosen, independently, from a distribution whose probability density function (pdf) is f and whose (cumulative) probability distribution function (PDF) is F ; we assume that F is continuous. X will denote the random variable whose PDF is F . G will denote some particular weighted complete graph. The weight of the edge joining vertex v and w will be denoted $d(v, w)$. Depending on the application, \mathcal{G}_n may be undirected or directed; in the former case, $d(v, w)$ is of course symmetric. When we make asymptotic statements about the behavior of some random variable which is a function of \mathcal{G}_n , we will assume that an infinite sequence $\mathcal{G}_n, n = 1, 2, \dots$, is considered, with each graph drawn independently.

Let S_n be a set of labeled graphs on n vertices; again, the vertices are labeled $1, 2, \dots, n$, so there is a natural one-to-one correspondence between the vertices of an element of S_n and the vertices of \mathcal{G}_n . All elements of S_n are assumed to have the same number of edges. For any H in S_n , and any weighted graph G , let $W(G, H)$ be the number found by summing, over all edges in H , the weight of the corresponding edge in G . For a given G , we wish to choose H in S_n so as to minimize $W(G, H)$; this minimum will be called $W_{\min}(G)$. Note that, for example, if S_n is the set of the $(n-1)!/2$ cycles on n vertices in an undirected graph, $W_{\min}(G)$ gives the solution to the traveling salesman problem. We wish to investigate the expected behavior of $W_{\min}(\mathcal{G}_n)$. (Often in an optimization problem we wish to maximize some quantity; for uniformity, however, we will always assume that we are minimizing quantities. The methods used here could also be applied to maximization problems.)

In this paper we will often wish to discuss inequalities which hold approximately, most of the time, for large enough n . In order to make such statements precisely, we need to introduce some notation. Let Y_n and Z_n be sequences of reals. For any n and $\varepsilon > 0$, consider the following two propositions:

- (1) $Y_n \leq Z_n + \varepsilon |Z_n|,$
- (2) $Y_n \geq Z_n - \varepsilon |Z_n|.$

If for every $\varepsilon > 0$, (1) (respectively (2)) holds except for finitely many n , we will write $Y_n \leq Z_n$ (respectively $Y_n \geq Z_n$). If for every $\varepsilon > 0$, both (1) and (2) hold except for finitely many n , we write $Y_n \sim Z_n$.

Now let Y_n and Z_n be sequences of random variables; we will not assume that Y_n and Z_n are independent, but we will assume that variables with different indices are independent. (In our applications, each Z_n will often be a constant.) Note that now (1) and (2) are events rather than simple predicates. Let $P_1(n, \varepsilon)$ be the probability that (1)

fails and $P_2(n, \epsilon)$ be the probability that (2) fails. If for each $\epsilon > 0$, $P_1(n, \epsilon)$ (respectively $P_2(n, \epsilon)$) goes to 0 as n approaches infinity, we will say $Y_n \leq Z_n$ (respectively $Y_n \geq Z_n$) *in probability*. If both P_1 and P_2 approach zero for all $\epsilon > 0$, we write $Y_n \sim Z_n$ *in probability*. (The phrase “in probability” will be abbreviated “(pr)”.)

A much stronger notion is that of almost sure behavior. An asymptotic statement holds *almost surely* if the set of sequences Y_n and Z_n which do not obey the statement has probability measure 0. Suppose that, for each $\epsilon > 0$, with probability 1, the sequences Y_n and Z_n satisfy (1) except for only finitely many n . Then by an argument like that of [5, Theorems 4.1.1, 4.2.2] we may write

$$(3) \quad Y_n \leq Z_n \quad \text{almost surely.}$$

(“Almost surely” will be abbreviated “(a.s.)”.) By the Borel-Cantelli lemmas (see, for example, [5]), an equivalent definition of (3) is

$$\forall \epsilon > 0, \quad \sum_{n=0}^{\infty} P_1(n, \epsilon) < \infty.$$

We may similarly define statements that $Y_n \geq Z_n$ or $Y_n \sim Z_n$ almost surely. Sometimes we will show that an asymptotic statement which is true in probability is not true almost surely; it then follows that the statement is almost surely false (see [5, Corollary, p. 78]), even though it is true in probability!

Note that statements about probabilistic convergence and convergence of expected values are somewhat independent. In particular, either, both or neither of the following two statements may be true:

$$\begin{aligned} E[Y_n] &\sim E[Z_n], \\ Y_n &\sim Z_n \quad (\text{a.s.}). \end{aligned}$$

For more information and examples, see [5] and [31].

We will illustrate the methods discussed in this paper in the case in which f is the unit normal distribution. If X is some random variable, let $X_{i:n}$ denote the random variable obtained by selecting the i th smallest of n independent observations of X ; this is called an *order statistic* of X . For more information about order statistics, see [7], [28]; in particular, it is known that in the unit normal case, as n approaches infinity, the PDF of $(2 \log n)^{1/2}[X_{n:n} - (2 \log n)^{1/2}]$ approaches $\exp(-e^{-x})$. The rate of approach is quite slow, however (an observation which [7] attributes to [16]). The following related observations, which are well known or easily established, are useful.

FACT 1. *Let X be a unit normal variable and let A be an event with probability p . Then, as $p \rightarrow 0$,*

- (a) $|E[X|A]| \leq (2 \log p^{-1})^{1/2}$.
- (b) $E[|X||A] \leq (2 \log p^{-1})^{1/2}$.

FACT 2. *Let X be a unit normal variable. Then as $n \rightarrow \infty$,*

- (a) $E[X_{1:n}] \sim -(2 \log n)^{1/2}$.

Moreover, for any ϵ with $0 < \epsilon < 1$, as $n \rightarrow \infty$,

- (b) $P\{X_{1:n} \geq -(1 - \epsilon)(2 \log n)^{1/2}\} \leq \exp(-n^\epsilon)$.
- (c) $P\{X_{1:n} \leq -(1 + \epsilon)(2 \log n)^{1/2}\} \sim nF(-(1 + \epsilon)(2 \log n)^{1/2})$
 $= \Theta(n^{-2\epsilon - \epsilon^2}(\log n)^{-1/2})$.

Note that for any k , and any $\epsilon > 0$,

$$\lim_{n \rightarrow \infty} n^k \exp(-n^\epsilon) = 0;$$

we will describe this by saying that $\exp(-n^\epsilon)$ swallows polynomials. Note also that we may conclude from part (b) that

$$X_{1:n} \leq -(2 \log n)^{1/2} \quad (\text{a.s.}),$$

while on the other hand, from part (c),

$$X_{1:n} \geq -(2 \log n)^{1/2} \quad (\text{pr. but not a.s.}).$$

Intuitively, this is because the minimum cannot be greater (algebraically) than some bound unless *all* n observations are greater than the bound; for it to be less than the bound, only a single observation needs to be low. This sort of behavior will arise again when we consider the problem of finding minimum weight cliques.

FACT 3. Let X be a unit normal variable and let F be its PDF. Then, as $p \rightarrow 0$,

$$P\{X \leq (1 + \epsilon)F^{-1}(p) | X \leq F^{-1}(p)\} = \Theta(p^{2\epsilon + \epsilon^2}(-\log p)^{\epsilon + \epsilon^2/2}).$$

Now let F again be an arbitrary PDF, and f the corresponding pdf. Let X^{*m} be the random variable corresponding to the sum of m random variables chosen independently according to F , and let F^{*m} be the corresponding PDF. Note that if F is unit normal, then

$$(4) \quad F^{*m}(x) = F(m^{-1/2}x).$$

In order to discuss minimization problems, we will need to discuss the expected value of a sum given that a certain event is true; the following notation will be helpful. If m is a positive integer and p is a real in $(0, 1]$, let

$$B(m, p, F) = E[X^{*m} | F^{*m}(X^{*m}) \leq p].$$

See Fig. 1. Note that if A is any event with probability p , then

$$E[X^{*m} | A] \geq B(m, p, F);$$

note also that if F is unit normal,

$$(5) \quad B(m, p, F) \sim -(2m \log p^{-1})^{1/2}.$$

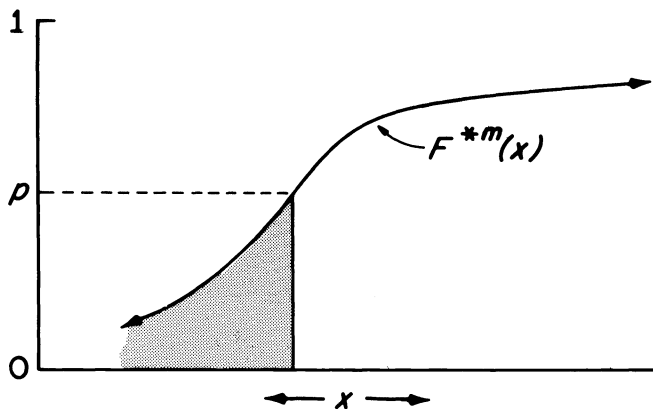


FIG. 1. Illustration of $B(m, p, F)$. The shaded area in this figure illustrates the event $F^{*m}(X^{*m}) \leq p$. The expected value of X^{*m} in this event is $B(m, p, F)$.

3. A lower bound. In this section we derive a simple bound on the expected behavior of $W_{\min}(\mathcal{G}_n)$ and on the PDF of $W_{\min}(\mathcal{G}_n)$. The method is a straightforward application of Boole's inequality, which can be a useful tool for examining distributions

of extrema; see [6], [7], [14]. Donath [8] used a combinatorial argument which, upon examination, is quite similar to the method to be used here. See [15] for another application of this inequality to an optimization problem; there a problem involving points distributed uniformly over the Euclidean plane was investigated. Let M_n be the cardinality of S_n ; recalling that each element of S_n has the same number of edges, let m_n be this common number.

LEMMA 1.

$$P\{W_{\min}(\mathcal{G}_n) \leq x\} \leq M_n F^{*m_n}(x).$$

Proof. We have

$$\begin{aligned} P\{W_{\min}(\mathcal{G}_n) \leq x\} &= P\{\exists H \in S_n \text{ such that } W(\mathcal{G}_n, H) \leq x\} && \text{(by the definition of } W_{\min}) \\ &\leq \sum_{H \in S_n} P\{W(\mathcal{G}_n, H) \leq x\} && \text{(by Boole's inequality [14, p. 23])} \\ &= \sum_{H \in S_n} F^{*m_n}(x) && \text{(since each } H \text{ has } m_n \text{ edges)} \\ &= M_n F^{*m_n}(x). \end{aligned}$$

□

COROLLARY 1.

$$E[W_{\min}(\mathcal{G}_n)] \geq B(m_n, M_n^{-1}, F).$$

Proof. Note that if a random variable had a PDF of $\min(1, M_n F^{*m_n})$, its expectation would be precisely $B(m_n, M_n^{-1}, F)$. See Fig. 2. □

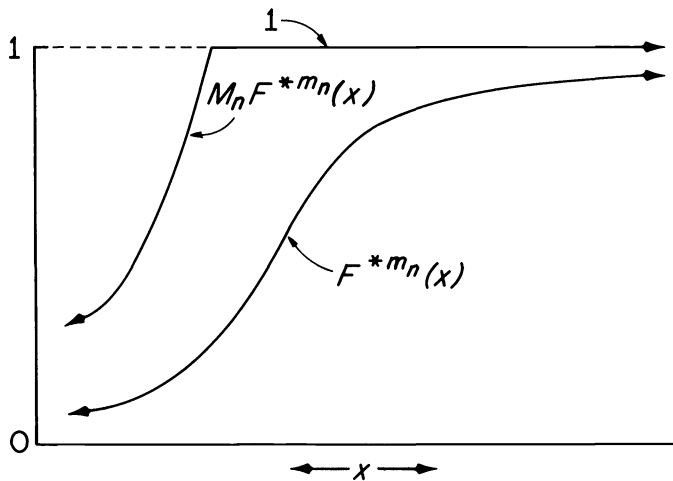


FIG. 2. Illustration for Corollary 1 to Lemma 1. The lower curve gives the distribution of weights of a fixed subgraph in S_n . The upper curve shows the bound, which follows from Lemma 1, on the true optimum. The expected value of the variable described by the upper curve is precisely $B(m_n, M_n^{-1}, F)$.

If F is a normal distribution, it is especially easy to apply this bound.

COROLLARY 2. *If F is unit normal, and M_n approaches infinity, then*

$$E[W_{\min}(\mathcal{G}_n)] \geq -(2m_n \log M_n)^{1/2}.$$

Proof. This follows immediately from the previous corollary and (5). \square

COROLLARY 3. *If F is unit normal, and M_n approaches infinity, then*

$$W_{\min}(\mathcal{G}_n) \geq -(2m_n \log M_n)^{1/2} \quad (\text{pr.}).$$

Moreover, if M_n^{-1} swallows polynomials, then this bound holds almost surely.

Proof. Using Lemma 1, we know that the probability of

$$W_{\min}(\mathcal{G}_n) \leq -(1 + \varepsilon)(2m_n \log M_n)^{1/2}$$

is bounded by

$$\begin{aligned} M_n F^{*m_n}(-(1 + \varepsilon)(2m_n \log M_n)^{1/2}) \\ &= M_n F(-(1 + \varepsilon)(2 \log M_n)^{1/2}) \quad (\text{by (4)}) \\ &= O(M_n^{-2\varepsilon}) \quad (\text{by Fact 2c}). \end{aligned}$$

Clearly this goes to zero if M_n goes to infinity; moreover, if M_n^{-1} swallows polynomials, the sum of this probability must converge, so by the Borel-Cantelli lemma the almost sure convergence is established. \square

4. An upper bound. In this section we obtain an upper bound on the expected behavior of $W_{\min}(\mathcal{G}_n)$. We will use some results about subgraph existence problems on random graphs. Define a random structure \mathcal{G}_{n,p_n} to be a graph on n vertices, where each edge is present with probability p_n , independently of the others. Then, for example, it is known [27] that if we choose c large enough and let $p_n = (c \log n)/n$, then the probability that \mathcal{G}_{n,p_n} will have a hamiltonian cycle approaches one as n approaches infinity. In fact, for any i , we can choose a c large enough so that \mathcal{G}_{n,p_n} has a hamiltonian cycle except with probability $O(n^{-i})$ [1].

In this section, we discuss a simple lemma which uses the notion of thresholds to relate results of this form to the optimization problems we are considering. (This notion has been used by a number of researchers. For example, Garfinkel and Gilbert [18] used it in connection with the bottleneck traveling salesman problem. Walkup [30] used an argument involving thresholds in establishing an upper bound on the assignment problem with uniform edge weights. I used it in the case of the normal distribution in [25]. Weide stated it in a general form in [31].)

LEMMA 2 [18], [25], [31]. *Let p_n be a sequence of reals in $[0, 1]$ and let q_n be the probability that \mathcal{G}_{n,p_n} fails to contain an element of S_n . Then*

$$(6) \quad W_{\min}(\mathcal{G}_n) \leq m_n F^{-1}(p_n)$$

except with a probability of at most q_n .

Proof. Consider the following algorithm for choosing an element H of S_n .

1. Let $a = F^{-1}(p_n)$, and let H_0 be some fixed element of S_n .
2. Let E be the set of edges in G whose weight is less than a ; call these *light* edges.
3. Let H be any element of S_n all of whose edges are light, and stop. If no such H can be found, go on to step 4.
4. Let $H = H_0$.

Note that, except with probability of at most q_n , this algorithm returns a subgraph whose weight satisfies the desired inequality. \square

COROLLARY 1 [25]. *Suppose that F is unit normal, and that q_n goes to zero rapidly enough that*

$$q_n(m_n \log q_n^{-1})^{1/2} = o(m_n(\log p_n^{-1})^{1/2}).$$

Then

$$E[W_{\min}(\mathcal{G}_n)] \leq -m_n(2 \log p_n^{-1})^{1/2}.$$

Proof. Consider again the algorithm in the proof of the lemma. Note that with probability approaching 1, the algorithm will return a subgraph H whose weight is at most

$$m_n F^{-1}(p_n) \sim -m_n(2 \log p_n^{-1})^{1/2}.$$

Let FAIL be the event that we fail to find an element of S_n among the light edges, and must therefore set H to H_0 ; the probability of FAIL is just q_n . By Fact 1, and the fact that $W(\mathcal{G}_n, H_0)$ is normally distributed with variance m_n , we may conclude that the expected weight of H_0 in the event FAIL is $O((m_n \log q_n^{-1})^{1/2})$. Then by the hypotheses of the corollary, the error we commit by ignoring the possibility of event FAIL is negligible. \square

COROLLARY 2 [31]. *If the sum of the q_n in the lemma converges, then*

$$W_{\min}(\mathcal{G}_n) \leq m_n F^{-1}(p_n) \quad (\text{a.s.}).$$

5. Some examples. In this section we show some applications of the methods discussed so far. As mentioned earlier, we will assume that edge weights are unit normal variables. The assignment problem for the normal distribution (and others) was analyzed by Borovkov [4]. He observed that a lower bound for this problem may be obtained by taking the sum of the minimum element in each row of the input matrix; similarly, he observed that a simple greedy algorithm yields a fairly good upper bound. Using these results he showed that

$$W_{\min}(\mathcal{G}_n) \sim -n(2 \log n)^{1/2} \quad (\text{pr.}).$$

His method could also establish a similar result for the traveling salesman problem.

Weide [31] has used results about the probability of finding a Hamiltonian circuit in \mathcal{G}_{n,p_n} [1], [27], to show that for the traveling salesman problem with unit normal edge weights,

$$W_{\min}(\mathcal{G}_n) \leq -n(2 \log n)^{1/2} \quad (\text{a.s.}).$$

Using Corollary 3 to Lemma 1 we can easily extend this to also be a lower bound. A very similar analysis holds for the assignment problem, so we obtain the following.

THEOREM 1. *For the traveling salesman problem or the assignment problem, with unit normal edge weights,*

$$W_{\min}(\mathcal{G}_n) \sim -n(2 \log n)^{1/2} \quad (\text{a.s.}).$$

Of course, these examples do not provide much evidence for the power of the methods discussed here, since we have only slightly extended a long-known result. The bounds achieved in the next example, however, do not appear to be obtainable by simple greedy arguments. Consider the problem of finding the weight of the lightest k -clique in a graph G . (By a k -clique we mean a subgraph on k vertices, all of which are adjacent. In the asymptotic statements which follow, we assume that k is fixed and n goes to infinity.) I am not able to devise a greedy algorithm which gives good bounds for this problem; in fact, in the next section we will see that the natural greedy algorithm, in

probability, fails to produce a good bound. The results of the previous sections, however, easily lead to a tight description of the behavior of this problem.

THEOREM 2. *For the problem of finding the lightest k -clique in an n -vertex graph with unit normal edge weights,*

- (a) $W_{\min}(\mathcal{G}_n) \leq -k((k-1) \log n)^{1/2}$ (a.s.),
- (b) $W_{\min}(\mathcal{G}_n) \sim -k((k-1) \log n)^{1/2}$ (pr. but not a.s.),
- (c) $E[W_{\min}(\mathcal{G}_n)] \sim -k((k-1) \log n)^{1/2}$.

Proof. First note that if we let $p_n = n^{-2/(k-1)+\varepsilon}$, then the probability that \mathcal{G}_{n,p_n} fails to contain a k -clique is $O(\exp(-n^{\varepsilon/3}))$, which swallows polynomials. (This can be seen by breaking the vertex set of \mathcal{G}_{n,p_n} into $n^{\varepsilon/3}$ sets each of size about $n^{1-\varepsilon/3}$. With the p_n we have given, using [10, Corollary 4, Theorem 1], we know that for large enough n the probability of failing to find a clique on any of these vertex subsets is less than e^{-1} , independently of the other subsets. For a more clever way of decomposing a graph to obtain such bounds, involving the notion of a projective geometry, see [3, proof of Theorem 2(ii)].) Then since the number of edges in a k -clique is $C(k, 2)$, where $C(i, j)$ denotes the number of combinations of i things taken j at a time, Corollary 1 to Lemma 2 gives

$$\begin{aligned} E[W_{\min}(\mathcal{G}_n)] &\leq -C(k, 2)(2 \log n^{2/(k-1)-\varepsilon})^{1/2} \\ &\sim -k((k-1) \log n)^{1/2} \left(1 - \frac{\varepsilon(k-1)}{2}\right)^{1/2}. \end{aligned}$$

Since this holds for arbitrarily small ε , we may conclude that

$$E[W_{\min}(\mathcal{G}_n)] \leq -k((k-1) \log n)^{1/2},$$

and, by Corollary 2, part (a) holds almost surely.

Clearly, the number of distinct k -cliques over n vertices is $C(n, k)$. Thus, by Lemma 1 and its corollaries, we see that

$$\begin{aligned} E[W_{\min}(\mathcal{G}_n)] &\geq -(2C(k, 2) \log C(n, k))^{1/2} \\ &\sim -k((k-1) \log n)^{1/2}, \end{aligned}$$

and

$$(7) \quad W_{\min}(\mathcal{G}_n) \geq -k((k-1) \log n)^{1/2} \quad (\text{pr.}).$$

Note that M_n does not become infinite fast enough to guarantee almost sure behavior of Corollary 3. We now sketch a proof that the bound in (7) does *not* hold almost surely. Choose δ small enough so that

$$(8) \quad (2\delta + \delta^2) \left(\frac{2}{k-1}\right) < 1.$$

Now let

$$(9) \quad p_n = n^{-2/(k-1)+\varepsilon},$$

as above. Then if we pick out the light edges of \mathcal{G}_n as in the proof of Lemma 2, we can almost surely construct a k -clique using only light edges. But by (8), (9) and Fact 3, an arbitrarily chosen one of the light edges used in the clique will be less than $F^{-1}(p_n)$ by a factor of $(1 + \delta)$, with a probability whose sum does not converge as n goes to infinity. Since this remains true even if we make ε arbitrarily close to zero, and since the number of edges in a k -clique is independent of n , this likelihood of a single excessively light edge must prevent (7) from holding almost surely. \square

The results obtained so far demonstrate that the bounds discussed in the previous section can give tight descriptions for some interesting problems. However, these bounds are not always tight, even when the edge weights have unit normal distributions. Say a graph H has property $X(k)$ if

- (a) H contains a clique of size k , and
- (b) H has k^2 edges.

Let S_n be the set of all n vertex graphs with property $X(k)$. It is not difficult to show that for this choice of S_n neither the lower nor the upper bound on the asymptotic expected behavior is tight; see [25] for details.

6. Regular greedy algorithms. It is easy to devise greedy algorithms for subgraph optimization problems. For example, to find the lightest Hamiltonian path in a graph, one can start at an arbitrary vertex and iteratively walk to the nearest unused vertex. A greedy algorithm was used by Borovkov [4] in his analysis of the assignment problem. For the lightest clique problem, one can start with the cheapest edge and iteratively add the vertex which increases the weight of the clique by the smallest amount.

Such algorithms can be viewed in the following way. The desired output is a list L of the edges in the subgraph found; by a slight abuse of notation, we will say that a vertex is in L if it is an endpoint of an edge in L . Initially L is the null list. Each partial list L will somehow determine a family CHOICE (L) of sets of edges in G ; each set in CHOICE (L) must be disjoint from L . At each iteration, we choose the set of edges in CHOICE (L) of smallest total weight, and append it to L .

For example, in the k -clique algorithm discussed above, after $r > 0$ iterations L would be a clique on $r + 1$ vertices. If L is the empty list, CHOICE (L) would be a family of singleton sets whose elements were the edges of the graph; for nonempty L , CHOICE (L) would contain one set for each vertex v not in L , namely, the set of $r + 1$ edges which join v to vertices in L .

A pidgin-Algol specification of the algorithm appears below; here cost (E), where E is a set of edges, denotes the total weight of the edges in E .

```

begin
   $L \leftarrow$  the empty list;
  for  $r \leftarrow 1$  until  $t$  do
    begin
      let  $E$  be the set in CHOICE ( $L$ ) which minimizes cost ( $E$ );
      append the elements of  $E$  to the end of  $L$ ;
    end;
  end;

```

Let \mathcal{A} be such an algorithm. If the length of L determines the cardinality of CHOICE (L) and of each element of CHOICE (L), we will say the algorithm is *regular*; henceforth we assume the algorithm is regular. This means that we know, for each r , how many choices are possible at iteration r (call this number $c(r)$) and how many edges will be added during iteration r (call this number $e(r)$). It is tempting at this point to use the following argument, which we shall call the *naive analysis*. At the r th iteration, we choose the minimum of $c(r)$ variables each of which is the sum of $e(r)$ random variables chosen according to CHOICE; thus the amount we add to the cost of L is $X_{1:c(r)}^{*e(r)}$. (In notation like $X_{b:c}^*a$, the superscript is considered to have higher precedence; thus this would mean to choose the b th smallest of c independent observations, each of which was the sum of a observations of X .) Let $\hat{X}_{\mathcal{A}}$ be the random variable corresponding to the sum of these variables over all iterations; i.e.,

$$\hat{X}_{\mathcal{A}} = \sum_{r=1}^t X_{1:c(r)}^{*e(r)}$$

Let $\hat{F}_{\mathcal{A}}$ be the corresponding PDF. Let $F_{\mathcal{A}}$ be the PDF which describes the true distribution of outputs of \mathcal{A} , and let $X_{\mathcal{A}}$ be the corresponding random variable.

Now $\hat{F}_{\mathcal{A}}$ and $F_{\mathcal{A}}$ may be different; the flaw in the above analysis is twofold:

- (a) once any iterations have occurred, the edge weights have been conditioned by previous choices made during the algorithm, and
- (b) the sets in CHOICE (L), for some L , may overlap, and we are thus not choosing the minimum of *independent* variables.

If we rule out such problems, the analysis becomes much easier. Define a *basic* greedy algorithm to be a regular greedy algorithm for which, at each iteration, none of the edges in CHOICE (L) can have appeared in a set in CHOICE (L) at some previous iteration, and the sets in CHOICE (L) are disjoint from each other. Then if \mathcal{A} is a basic greedy algorithm, we have $F_{\mathcal{A}} = \hat{F}_{\mathcal{A}}$.

Note that the natural greedy algorithm for the assignment problem is basic, so the analysis of this algorithm is easily carried out [4]. See [31] for a more difficult analysis of a greedy algorithm (which meets the condition for being a basic greedy algorithm except for the point at which the last edge is added) for the traveling salesman problem under a distribution called a *fixed cost* distribution. The k -clique algorithm described above, however, is not even close to being basic; an edge may be considered at many different iterations. We shall, in the remainder of this paper, undertake the analysis of regular greedy algorithms which are not basic.

THEOREM 3. *If \mathcal{A} is a regular greedy algorithm, then for all x*

$$F_{\mathcal{A}}(x) \leq \hat{F}_{\mathcal{A}}(x).$$

Hence $E[X_{\mathcal{A}}] \geq E[\hat{X}_{\mathcal{A}}]$.

Intuitively, this is because the conditioning on the edges remaining at any step tends to make them larger, since these edges have been rejected whenever they were considered in choosing a minimum. This intuition can fairly easily be formalized into a proof. For this proof we will need a lemma, whose proof is uninteresting and deferred to the Appendix.

LEMMA 3. *Let w be a column vector of m independent real random variables chosen with a distribution function G . Let g be a real-valued function of m -vectors which is monotonic nondecreasing, in the sense that*

$$w \leq w' \Rightarrow g(w) \leq g(w').$$

(Here w is said to be less than or equal to w' if the inequality holds in each component.) Finally, let B be an $r \times m$ matrix of nonnegative reals, and b be a column vector of r reals. Then

$$P\{g(w) \leq x \mid Bw \geq b\} \leq P\{g(w) \leq x\}.$$

Proof of theorem. Suppose we are at the beginning of iteration r . Let L_0 be some possible value for L at this point, and let A_0 be the event that $L = L_0$. Consider the random variable

$$(10) \quad \min_{E \in \text{CHOICE}(L_0)} \text{cost}(E).$$

Were it not for the conditioning on the probabilities of the edge weights due to previous iterations, the PDF for this minimum cost would be less than or equal that for $X_{1:c(r)}^{*e(r)}$. (The inequality is necessary because the sets in CHOICE (L_0) may not be disjoint.) We now show that this statement remains true even when we bear in mind that the edge weights are conditioned. Note that (10) depends only on edges which have not yet been

chosen, and is monotonic increasing in these edges. Now since the choice of edges to add to L is determined by comparisons of sums of edge weights, the event A_0 can be phrased as a set of inequalities on the edge weights; each inequality expresses that fact that the selected set of edges was less than or equal to some other set allowed by CHOICE. Note that each edge not yet chosen must appear only on the greater side of these inequalities. Thus by the lemma, the true PDF for the variable which describes the total weight of edges added to L during this iteration can only be decreased by this conditioning. Summing over all possible L_0 , and integrating over all values of the variables chosen so far, we obtain the theorem. \square

To illustrate the application of this theorem, consider the greedy k -clique algorithm mentioned above, with unit normal edge weights. The naive analysis says that the algorithm returns a clique of weight

$$\begin{aligned} \hat{X}_{\mathcal{A}} &= X_{1:C(n,2)} + \sum_{i=2}^{k-1} X_{1:n-i}^{*i} \\ &\sim -s_k(\log n)^{1/2} \quad (\text{pr.}) \end{aligned}$$

where

$$s_k = 2 + \sum_{i=2}^{k-1} (2i)^{1/2}.$$

LEMMA 4.

(a) $X_{\mathcal{A}} \geq -s_k(\log n)^{1/2}$ (pr.)

and

(b) $E[X_{\mathcal{A}}] \geq -s_k(\log n)^{1/2}$.

Proof. $\hat{X}_{\mathcal{A}}$ satisfies the indicated bounds, and hence so does $X_{\mathcal{A}}$ by the previous theorem. \square

In order to complete our analysis of the behavior of the greedy algorithm for k -cliques, it would be desirable to have an upper bound on the behavior of the solution it obtains. The past theorem gives us little help in this direction, but we may nonetheless establish the desired bound.

LEMMA 5.

(a) $X_{\mathcal{A}} \leq -s_k(\log n)^{1/2}$ (a.s.)

and

(b) $E[X_{\mathcal{A}}] \leq -s_k(\log n)^{1/2}$.

Proof. An idea similar to that used in [13], [20] is useful here—we can simply eliminate all cases in which things do not work out as we like. Choose any $\epsilon > 0$. Note that the probability that the first edge selected is above $-2(1-\epsilon)(\log n)^{1/2}$ goes to zero fast enough to swallow polynomials. Next consider the probability that for some set C of vertices, $|C| < k$,

$$\min_{v \notin C} \sum_{w \in C} d(v, w) \geq -(1-\epsilon)(2|C| \log(n-|C|))^{1/2}.$$

Using Fact 2b, we see that for any fixed choice of C , this probability goes to zero fast enough to swallow polynomials. But, for fixed k , there are only polynomially many choices for C , so the sum of this probability, over all possible C with $|C| < k$, must go to zero fast enough to swallow polynomials. We may conclude that the algorithm produces a clique of weight less than $(1-\epsilon)$ times the expected value predicted by the naive analysis except with a probability which swallows polynomials. Thus the sum of this probability over all n must converge, so we have part (a). Part (b) is then easily obtained using Fact 1. \square

THEOREM 4. *For the greedy k -clique algorithm with unit normal edge weights,*

- (a) $X_{\mathcal{A}} \leq -s_k (\log n)^{1/2}$ (a.s.),
 (b) $X_{\mathcal{A}} \sim -s_k (\log n)^{1/2}$ (pr. but not a.s.)

and

- (c) $E[X_{\mathcal{A}}] \sim -s_k (\log n)^{1/2}$.

Proof. Most of the theorem follows directly from Lemmas 4 and 5. To show that the asymptotic behavior does not hold almost surely, we may use an argument similar to that used in the proof of Theorem 2. \square

Combining Theorems 2 and 4 and Weide's "relative error lemma" [31], we see that for $k \geq 3$

$$(11) \quad \frac{X_{\mathcal{A}}}{W_{\min}} \sim \frac{s_k}{k(k-1)^{1/2}} \quad (\text{pr.}).$$

For $k = 2$, the algorithm is of course exact, since it merely chooses the cheapest edge; as k approaches infinity the ratio on the right of (11) approaches $(\frac{8}{9})^{1/2}$. (Recall that since the quantities we are minimizing tend to be negative, the ratio of the result of the approximate algorithm to the true minimum will tend to be *less* than one.)

7. Conclusions. We have demonstrated the use of some basic methods for analysis of the expected behavior of subgraph optimization problems. These methods have enabled us not only to determine the expected behavior of the optimum, but also to demonstrate that the asymptotic behavior held in probability, and to determine whether or not it held almost surely. In addition, we have demonstrated how to analyze the behavior of a greedy algorithm, even when edge weights were conditioned as the algorithm proceeded, and when the algorithm was provably suboptimal. (The reader is cautioned that n may have to be quite large before the asymptotic behavior becomes apparent. As mentioned earlier, it is well known that even the simple maximum of n unit normal variables approaches its asymptotic behavior slowly as n becomes large [7], [16].)

Although many of the techniques discussed here are of fairly general applicability, we have demonstrated them only in the case where edge weights are chosen from a unit normal distribution. It would be easy to state the results in the case where an arbitrary mean and variance were stated for the normal distribution. We plan to investigate these same problems under distributions other than normal. While the bounds of Lemmas 1 and 2 will still be valid, some complications arise. For one thing, it is more difficult to calculate the distribution of the sum of several variables. Also, it appears that for uniform distributions, the bounds do not tend to be as tight. Thus more complicated techniques are likely to be needed. (See Walkup [30] for an example of a clever analysis of the true optimum of the assignment problem under the uniform distribution.)

Appendix. Proof of Lemma 3.

LEMMA 3. *Let w be a column vector of m independent real random variables chosen with a continuous distribution function G . Let g be a real-valued function of m -vectors which is monotonic nondecreasing, in the sense that*

$$w \leq w' \Rightarrow g(w) \leq g(w').$$

Finally, let B be an $r \times m$ matrix of nonnegative reals, and b be a column vector of r reals. Then

$$P\{g(w) \leq x | Bw \geq b\} \leq P\{g(w) \leq x\}.$$

Proof. We prove the lemma by induction on m . For $m = 1$ it is easy. Suppose it holds for $m = k - 1$. We may decompose w as

$$w = (w^*, w_k),$$

where w^* is the first $k - 1$ components of w , and w_k is the last component of w . Then

$$(12) \quad P\{g(w) \leq x | Bw \geq b\} = P\{g(w^*, w_k) \leq x | B_1 w^* + B_2 w_k \geq b\},$$

where B_1 and B_2 are appropriate submatrices of B . We may write the right-hand side of (12) as

$$\frac{\int dG(\xi) h(\xi) P\{g(w^*, \xi) \leq x | B_1 w^* \geq b - B_2 \xi\}}{\int dG(\xi) h(\xi)},$$

where

$$h(\xi) = P\{B_1 w^* \geq b - B_2 \xi\}.$$

Now by the inductive hypothesis, for any ξ ,

$$P\{g(w^*, \xi) \leq x | B_1 w^* \geq b - B_2 \xi\} \leq P\{g(w^*, \xi) \leq x\}.$$

Thus an upper bound is

$$\frac{\int dG(\xi) h(\xi) P\{g(w^*, \xi) \leq x\}}{\int dG(\xi) h(\xi)}.$$

But since $h(\xi)$ is easily seen to be monotonic increasing, while $P\{g(w^*, \xi) \leq x\}$ is monotonic decreasing in ξ , this ratio is bounded above by

$$\int dG(\xi) P\{g(w^*, \xi) \leq x\},$$

which is precisely $P\{g(w) \leq x\}$. This completes the induction. \square

Acknowledgments. The referees provided some extremely useful suggestions on an early version of this paper. In particular, they suggested that I consider arbitrary distributions rather than only normal distributions, and suggested that I examine the question of stochastic convergence rather than only asymptotic behavior of expected values. Bruce Weide's thesis [31] helped me understand stochastic convergence. Howard Tucker had a very helpful discussion with me about the proof of Lemma 3. Arvind, Dov Harel, Richard Karp, and David Walkup also provided useful suggestions and pointers to relevant literature.

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] J. BEARDWOOD, J. H. HALTON AND J. M. HAMMERSLEY, *The shortest path through many points*, Proc. Camb. Phil. Soc., 55 (1959), pp. 299–327.
- [3] B. BOLLABAS AND P. ERDÖS, *Cliques in random graphs*, Math. Proc. Camb. Phil. Soc., 80 (1976), pp. 419–427.
- [4] A. A. BOROVKOV, *A probabilistic formulation of two economic problems*, Soviet Mathematics, 3 (1962), pp. 1403–1406.
- [5] K. L. CHUNG, *A Course in Probability Theory*, 2nd ed., Academic Press, New York, 1976.
- [6] F. N. DAVID AND D. E. BARTON, *Combinatorial Chance*, Charles Griffin, London, 1962.
- [7] H. A. DAVID, *Order Statistics*, John Wiley, New York, 1970.
- [8] W. E. DONATH, *Algorithm and average-value bounds for assignment problems*, IBM J. Res. Dev., 13 (1969), pp. 380–386.

- [9] P. ERDŐS AND A. RÉNYI, *On random graphs I*, Publ. Mathematicae, 6 (1959), pp. 290–297.
- [10] ———, *On the evolution of random graphs*, Publ. Math. Inst. Hung. Acad. Sci., 5A (1960), pp. 17–61.
- [11] ———, *On Random matrices*, Publ. Math. Inst. Hung. Acad. Sci., 8A (1963), pp. 455–461.
- [12] ———, *On the existence of a factor of degree one of a connected random graph*, Acta Math. Acad. Sci. Hung., 17 (1966), pp. 359–368.
- [13] P. ERDŐS AND J. SPENCER, *Probabilistic Methods in Combinatorics*, Academic Press, New York, 1974.
- [14] W. FELLER, *An Introduction to Probability Theory and Its Applications*, vol. 1, 3rd edition, John Wiley, New York, 1968.
- [15] M. L. FISHER AND D. S. HOCHBAUM, *Probabilistic analysis of the Euclidean K -median problem*, Technical report 78-06-03, Wharton Department of Decision Sciences, University of Pennsylvania, May, 1978.
- [16] R. A. FISHER AND L. H. C. TIPPETT, *Limiting forms of the frequency distribution of the largest or the smallest member in a sample*, Proc. Camb. Phil. Soc., 24 (1928), pp. 180–190.
- [17] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
- [18] R. S. GARFINKEL AND K. C. GILBERT, *The bottleneck traveling salesman problem: algorithms and probabilistic analysis*, J. Assoc. Comput. Mach., 25 (1978), pp. 435–448.
- [19] G. R. GRIMMETT AND C. J. H. MCDIARMID, *On coloring random graphs*, Math. Proc. Camb. Phil. Soc., 77 (1975), pp. 313–324.
- [20] L. J. GUIBAS AND E. SZEMEREDI, *The analysis of double hashing*, Proc. 8th Annual ACM Symposium on Theory of Computing, 1976, pp. 187–191.
- [21] J. H. HALTON AND R. TERADA, *An almost surely optimal algorithm for the Euclidean traveling salesman problem*, Computer Sciences Technical Report #335, University of Wisconsin-Madison, October, 1978.
- [22] 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. 85–104.
- [23] ———, *Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane*, Math. Oper. Res., 2 (1977), pp. 209–224.
- [24] ———, *A patching algorithm for the nonsymmetric traveling-salesman problem*, this Journal, 8 (1979), pp. 561–573.
- [25] G. S. LUEKER, *Maximization problems on graphs with edge weights chosen from a normal distribution*, Technical Report 115, University of California at Irvine, March 1978. A condensed version appears in Proc. 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 13–18.
- [26] D. W. MATULA, *On the complete subgraphs of a random graph*, Proc. 2nd Chapel Hill Conference on Combinatorial Math. and its Applications, University of North Carolina, Chapel Hill, May, 1970, pp. 356–369.
- [27] L. POSA, *Hamiltonian circuits in random graphs*, Discrete Math., 14 (1976), pp. 359–364.
- [28] P. K. SEN, *On stochastic convergence of the sample extreme values from distributions with infinite extremities*, J. Ind. Soc. Agric. Statist., 16 (1964), pp. 189–201.
- [29] D. W. WALKUP, *Matchings in random regular bipartite graphs*, Discrete Math., 31 (1980), pp. 59–64.
- [30] ———, *On the expected value of a random assignment problem*, this Journal, 8 (1979), pp. 440–442.
- [31] B. W. WEIDE, *Statistical methods in algorithm design and analysis*, Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University; available as technical report CMU-CS-78-142, August, 1978.

EQUIVALENCE OF RELATIONAL DATABASE SCHEMES*

CATRIEL BEERI,[†] ALBERTO O. MENDELZON,[‡] YEHOShUA SAGIV[§]
AND JEFFREY D. ULLMAN[¶]

Abstract. We investigate the question of when two database schemes embody the same information. We argue that this question reduces to the equivalence of the sets of fixed points of the project-join mappings associated with the two database schemes in question. When data dependencies are given, we need only consider those fixed points that satisfy the dependencies. A polynomial algorithm to test the equivalence of database schemes, when there are no dependencies, is given. We also provide an exponential algorithm to handle the case where there are functional and/or multivalued dependencies. Furthermore, we give a polynomial time test to determine whether a project-join mapping preserves a set of functional dependencies, and a polynomial time algorithm for equivalence of database schemes whose project-join mappings do preserve the given set of functional dependencies. Lastly, we introduce the “update sets” approach to database design as an application of these results.

Key words. database scheme equivalence, project-join mapping, functional dependency, multivalued dependency, join dependency, tableau, relational database, database design

1. Introduction. A central problem in the design of a relational database is the selection of a *database scheme*, that is, a set of relation formats (or *relation schemes*) into which the information is to be structured. A poor choice of the database scheme may lead to undesirable anomalies in the operation of the database system. The elimination of these anomalies is the purpose of the normalization processes proposed in the literature [Codd1], [Codd2], [Fagin]. We believe that a precise treatment of these issues, with an ultimate goal of automating the database design process, requires a well-defined notion of when a database scheme can be replaced by another without losing any of the information representation capabilities of the database.

To this end, we propose to apply the “universal relation” model of [Arm], [Bern], [ABU]. Under this model, the “universe” that the database is supposed to represent is visualized as a finite table, called an *instance* of the universe, with one column for each attribute of the database. The relationship between the instance that a database represents and the actual relations in the database is given by the projection mapping. That is, each relation is assumed to be a projection of the universal relation on some subset of the attributes.

The assumption of a universal relation is sometimes criticized as unrealistic, usually because updates to the actual relations in the database cannot always be reflected in the universal relation. However, if one reflects on the matter it appears that there is no other way to justify even posing questions like “does one database scheme represent the same information as another?” Similarly, the validity of database design algorithms that purport to produce database schemes representing certain relationships have no known theoretical basis save for the universal relation assumption. It is our belief that the universal relation must be assumed, and issues such as which updates can be permitted

* Received by the editors January 9, 1980, and in revised form June 23, 1980. This work was supported in part by the National Science Foundation under grant MCS-76-15255. Portions of this paper appeared in the Proceedings of the 11th Annual ACM Symposium on Theory of Computing.

[†] Department of Computer Science, The Hebrew University, Jerusalem, Israel.

[‡] Computer Systems Research Group, University of Toronto, Toronto, Ontario, Canada M5S 1A1. The work of this author was supported in part by an IBM Fellowship.

[§] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. The work of this author was supported in part by a grant from Bell Laboratories.

[¶] Department of Computer Science, Stanford University, Stanford, California 94305.

dealt with as well as possible. In the large, problems raised by the universal model, such as the ill-definedness of certain updates, are not solved by rejecting the universal relation assumption.

The instance of the universe should be recoverable from the database by taking the (natural) join of all the relations. By [Ris1], when instances are decomposed into two projections the join is the only operation that can be used to recover an instance from the projections. (That is, an instance that has been decomposed into two projections is recoverable if and only if certain conditions are satisfied, and if an instance I is recoverable then I is the natural join of its two projections.) We believe that even when instances are decomposed into more than two projections, the join is still the only operation that might recover an instance from its projections. However, it is well-known that, given an instance of the universe I and a database scheme \mathbf{R} , it is possible that the join of the projections of I on the relation schemes of \mathbf{R} will not be equal to I ; we may get spurious tuples that were not originally contained in I , although we cannot lose any tuple in I . Let us say that an instance I is a *fixed point* of a database scheme \mathbf{R} if the join of the projections of I on the relation schemes of \mathbf{R} is exactly I .

An instance I is *representable* in a database scheme \mathbf{R} if there are relations r_1, r_2, \dots, r_k for the relation schemes of \mathbf{R} that have I as a natural join. We claim that the following principle holds. An instance I is representable in a database scheme \mathbf{R} if and only if I is a fixed point of \mathbf{R} . Thus the set of fixed points is a measure of the representation power of a database scheme. In proof, if I is a fixed point, then its projection onto the relation schemes in \mathbf{R} can be used to reconstruct I by the natural join. Conversely, if a set of relations r_1, r_2, \dots, r_k in the database have I as a natural join, then I projected onto \mathbf{R} gives r_1, r_2, \dots, r_k again, so I is a fixed point of \mathbf{R} .

Accordingly, we shall consider two database schemes “data equivalent” if they have the same set of fixed points. Note that this notion of equivalence relates only to the ability of the schemes to represent the same set of instances, that is, the same collections of data. There is another aspect of equivalence of database schemes, namely, that they have the same capability to enforce integrity constraints and thereby prevent incorrect updates to the database. We do not treat this aspect of equivalence in this paper; see [BBG] for further details. From now on we shall use the word “equivalence” to mean “data equivalence.” A syntactic characterization of equivalence of database schemes appears in § 3.

It is usually the case that the database designer is not willing to consider all instances of the universe as equally meaningful, since the semantics of the data impose a number of restrictions on the instances. In such a case, the data representation power of a database scheme is measured by the set of meaningful instances in its set of fixed points. Accordingly, we shall amend our definition of database scheme equivalence to say that two database schemes are equivalent with respect to a given set of instances \mathbf{P} if they have the same set of fixed points in the set \mathbf{P} . In § 4 we give some general results on these types of equivalence.

A common way of restricting the set of meaningful instances is by specifying *constraints* that the admissible instances must satisfy. Several kinds of constraints have been studied in the literature; in particular, functional dependencies [Arm], [Codd1] and multivalued dependencies [Del], [Fagin], [Zan1]. In § 5 we consider the case where all the constraints are functional and multivalued dependencies and give an (exponential) algorithm to test for equivalence of database schemes under this type of constraint.

We also examine the case where only functional dependencies (fd's) are given. We show that if the project-join mapping associated with the database scheme preserves

the given fd's, then equivalence can be tested in polynomial time. Furthermore, the question whether the projection mapping preserves the set of fd's can also be decided in polynomial time.

Finally, in § 6 we present the “update sets” approach to database design. Recently, much research effort was expended to investigate the structure of the conceptual scheme [ANSI]. It is generally agreed that the conceptual scheme consists of basic units of information and that transactions to be effected against the database are expressed in terms of these units. We model these ideas by assuming that the design starts with a collection $U = \{U_1, U_2, \dots, U_k\}$ of *update sets*, each U_i being a set of attributes. Each insert, delete or update operation changes a finite set of tuples in a relation u_i on the set U_i . We postulate that the instance I represented by such a database $\{u_1, \dots, u_k\}$ is the largest I whose projection onto each U_i is u_i . We then deduce from our results of §§ 3, 4 and 5 what database schemes can be used to replace the update sets without loss of representation power. A case of special interest is that of fd's. We have considered fd's as constraints in § 5. However, an fd can naturally be thought of as an information unit. This leads us to consider update sets defined by sets of fd's, in the sense that the fd $X \rightarrow Y$ defines the update set that is the union of X and Y . We show, in the second part of § 6, that any set of fd's defines a certain project-join mapping whose behavior on instances that satisfy the fd's is independent of the particular representation chosen for the set of fd's. We propose that the set of fixed points of this project-join mapping that satisfy the given fd's be taken as the set of meaningful instances. In other words, when fd's are given one should look for database schemes that can represent this set of instances. A characterization of such schemes is presented in § 6.2.

We note that there are certain sets of u_i 's that are not the projection of any instance, for example, if one u_i is empty and others are not. We can only suggest that this situation be handled by introducing null symbols into the tuples of the instance, as discussed in [Codd3], [Zan2], [LP]. We feel that this approach needs further research, since none of the published solutions appears to be entirely satisfactory.

2. Basic definitions. We present in this section the basic relational model concepts and the universal relation model, following the terminology of [ABU], [ASU1].

A *universe* is a finite set U of symbols called *attributes*, each with an associated *domain* of values. We shall assume for simplicity that all attributes have the integers as their domain. An *element* of the universe is a function μ mapping each attribute to a value in its domain. Since we shall often assume an ordering A_1, \dots, A_k for the attributes, we can represent an element μ as a tuple in the Cartesian product of the domains of the A_i 's. Thus, elements are really tuples of relations over the attribute set U , in the usual sense of relational database theory. An *instance* of the universe is a finite set of elements. An instance can be visualized as a table with one column for each attribute and one row for each element.

A *database scheme* is a finite set of attributes, called *relation schemes*, such that the union of these sets is the universe U . Each relation scheme can be viewed as defining a table format. A *relation* is the “current value” of a relation scheme, that is, a finite set of *tuples*, each a mapping from the attributes of the relation scheme to their domains. A *database* is an assignment of a relation to each relation scheme of some database scheme.

Often, a relation scheme is considered to consist of a set of attributes and some additional information such as dependencies. For our purposes, however, it suffices to think of a relation scheme merely as a statement of the domain of definition of the mappings that constitute a relation, that is, as a set. In the following, we use the terms relation scheme and set of attributes interchangeably.

Given a relation r on a set X and a subset Y of X , we define $\pi_Y(r)$, the *projection* of r onto Y , to be the set of maps μ from Y to the corresponding domains, such that for some ν in r , ν agrees with μ on Y . In other words, the projection of r onto Y is obtained by deleting from the table represented by r all columns except those labeled by attributes in Y , and then identifying repeated rows.

The (natural) *join* of relations r_1, r_2, \dots, r_k on schemes R_1, R_2, \dots, R_k , denoted $\bowtie_{i=1}^k r_i$, is the set of mappings μ on $\bigcup_{i=1}^k R_i$ such that for each $1 \leq i \leq k$ there is a ν_i in r_i that agrees with μ on R_i . We note that the join operator is commutative and associative. For $k = 2$, we also write $r_1 \bowtie r_2$.

Given a database scheme $\mathbf{R} = \{R_1, \dots, R_k\}$, we define the *project-join mapping associated with \mathbf{R}* , denoted by $m_{\mathbf{R}}$, as a mapping on instances given by

$$\begin{aligned} m_{\mathbf{R}}(I) &= \bowtie_{i=1}^k \pi_{R_i}(I) \\ &= \{\mu \mid \mu \text{ is a map on } U, \text{ and there are maps } \nu_1, \dots, \nu_k \\ &\quad \text{in } I \text{ such that } \mu \text{ and } \nu_i \text{ agree on } R_i \text{ for every } i\}. \end{aligned}$$

It is easy to show the following basic properties of $m_{\mathbf{R}}$.

(J1) $I \subseteq m_{\mathbf{R}}(I)$ for all I .

(J2) $m_{\mathbf{R}}(m_{\mathbf{R}}(I)) = m_{\mathbf{R}}(I)$ (idempotence).

(J3) If $I \subseteq J$, then $m_{\mathbf{R}}(I) \subseteq m_{\mathbf{R}}(J)$ (monotonicity).

Property J1 tells us that if we take any instance I , project it onto a set of schemes and then join the relations obtained, we shall always recover all the tuples that were originally in I . However, it is easy to construct examples where we also obtain some new tuples that were not in I . We say that I is a *fixed point* of \mathbf{R} if $m_{\mathbf{R}}(I) = I$. We denote by $\text{FIXPT}(\mathbf{R})$ the set of all fixed points of \mathbf{R} . A database scheme \mathbf{R} is said to have a *lossless join with respect to a set of instances \mathbf{P}* if $\mathbf{P} \subseteq \text{FIXPT}(\mathbf{R})$.

3. Scheme equivalence. In this section we present our notion of database scheme equivalence and give a necessary and sufficient syntactic condition for two database schemes to be equivalent. (We assume that there are no constraints, that is, all instances have to be considered.)

Let \mathbf{R} and \mathbf{S} be two database schemes. We say that \mathbf{R} and \mathbf{S} are *equivalent* if $\text{FIXPT}(\mathbf{R}) = \text{FIXPT}(\mathbf{S})$. By the principle enunciated in the introduction, this notion of equivalence reflects the equality of the database schemes with regard to their ability to represent instances. We say \mathbf{R} *covers* \mathbf{S} , written $\mathbf{R} \geq \mathbf{S}$, if for all relation schemes S in \mathbf{S} there exists a relation scheme R in \mathbf{R} such that $S \subseteq R$. We write $\mathbf{R} = \mathbf{S}$ if $\mathbf{R} \geq \mathbf{S}$ and $\mathbf{S} \geq \mathbf{R}$.

LEMMA 1. $\mathbf{S} \geq \mathbf{R}$ if and only if $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$ for all instances I .

Proof. The lemma follows as a special case of [ASU1, Thm. 3]. However, we present here a direct proof.

(if). Suppose $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$ for all I . Construct an instance $I_{\mathbf{S}}$ as follows. For each S_j in \mathbf{S} , the instance $I_{\mathbf{S}}$ contains the tuple μ_j , where $\mu_j(A) = 0$ if A is in S_j and $\mu_j(A) = 1$ otherwise. Clearly, $\bar{0}$ is in $m_{\mathbf{S}}(I_{\mathbf{S}})$, where $\bar{0}$ maps all the attributes of U to 0, i.e., $\bar{0}$ is the all-zero tuple. It follows that $\bar{0}$ is in $m_{\mathbf{R}}(I_{\mathbf{S}})$. By the definition of $m_{\mathbf{R}}$, for each R_i in \mathbf{R} , there is a tuple μ_j in $I_{\mathbf{S}}$ such that μ_j agrees with $\bar{0}$ on R_i , that is, $R_i \subseteq S_j$. The claim follows.

(only if). For an instance I and for μ in $m_{\mathbf{S}}(I)$, let ν_1, \dots, ν_k be in I such that ν_j agrees with μ on S_j . If R_i is a subset of S_j , then ν_j agrees with μ on R_i . It follows that if $\mathbf{S} \geq \mathbf{R}$ then μ is also in $m_{\mathbf{R}}(I)$. \square

LEMMA 2. If $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$ for all I , then $\text{FIXPT}(\mathbf{R}) \subseteq \text{FIXPT}(\mathbf{S})$.

Proof. Let I be in $\text{FIXPT}(\mathbf{R})$; that is, $m_{\mathbf{R}}(I) = I$. Then by hypothesis $m_{\mathbf{S}}(I) \subseteq I$. Since $m_{\mathbf{S}}(I)$ contains I by J1, it follows that I is in $\text{FIXPT}(\mathbf{S})$. \square

LEMMA 3. *If $\text{FIXPT}(\mathbf{R}) \subseteq \text{FIXPT}(\mathbf{S})$, then $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$ for all instances I .*

Proof. Let I be any instance. Since $m_{\mathbf{R}}$ is idempotent (property J2), it follows that $m_{\mathbf{R}}(I)$ is in $\text{FIXPT}(\mathbf{R})$, and therefore in $\text{FIXPT}(\mathbf{S})$, that is,

$$m_{\mathbf{S}}(m_{\mathbf{R}}(I)) = m_{\mathbf{R}}(I).$$

Since $I \subseteq m_{\mathbf{R}}(I)$ by J1, it follows by monotonicity of $m_{\mathbf{S}}$ (property J3) that

$$m_{\mathbf{S}}(I) \subseteq m_{\mathbf{S}}(m_{\mathbf{R}}(I))$$

and hence $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$. \square

THEOREM 1. *The following are equivalent:*

- (1) $\mathbf{R} \leq \mathbf{S}$.
- (2) $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$ for all I .
- (3) $\text{FIXPT}(\mathbf{R}) \subseteq \text{FIXPT}(\mathbf{S})$.

Proof. Immediate from Lemmas 1, 2 and 3. \square

COROLLARY 1. *The following are equivalent:*

- (1) $\mathbf{R} \simeq \mathbf{S}$.
- (2) $m_{\mathbf{R}}(I) = m_{\mathbf{S}}(I)$ for all I .
- (3) $\text{FIXPT}(\mathbf{R}) = \text{FIXPT}(\mathbf{S})$.

By Corollary 1, if we add to a database scheme \mathbf{R} or delete from \mathbf{R} a relation scheme R that is contained in another relation scheme of \mathbf{R} , then the resulting database scheme is equivalent to \mathbf{R} . Therefore a database scheme \mathbf{R} can be transformed to a minimal unique form by taking only the maximal relation schemes of \mathbf{R} . (A relation scheme of \mathbf{R} is *maximal* if it is not contained in any other relation scheme of \mathbf{R} .)

4. Equivalence under constraints. It is generally agreed that it is very important in the design of a relational database to specify the various constraints that the data must satisfy in order to model correctly the user's view of the world. In our model, these constraints define a subset of all possible instances of the universe, and instances outside this subset are regarded as meaningless or incorrect.

Let \mathbf{P} be any set of instances. We say that two database schemes \mathbf{R} and \mathbf{S} are *equivalent with respect to \mathbf{P}* if

$$\text{FIXPT}(\mathbf{R}) \cap \mathbf{P} = \text{FIXPT}(\mathbf{S}) \cap \mathbf{P}.$$

From now on, we shall use the abbreviation $\text{FIXPT}_{\mathbf{P}}(\mathbf{R})$ for $\text{FIXPT}(\mathbf{R}) \cap \mathbf{P}$. Given a database scheme \mathbf{R} , we shall be interested in the question of how the membership in \mathbf{P} of an instance is affected by applying the mapping $m_{\mathbf{R}}$ to it. We define $\text{PRESERVED}(\mathbf{R}, \mathbf{P})$ as the set of instances in \mathbf{P} such that their image under $m_{\mathbf{R}}$ is also in \mathbf{P} , that is

$$\text{PRESERVED}(\mathbf{R}, \mathbf{P}) = \{I \mid I \in \mathbf{P} \text{ and } m_{\mathbf{R}}(I) \in \mathbf{P}\}.$$

We say that \mathbf{R} *preserves \mathbf{P}* if $\text{PRESERVED}(\mathbf{R}, \mathbf{P}) = \mathbf{P}$.

Example 1. Consider the database scheme $\mathbf{R} = \{AB, BC\}$, and let \mathbf{P} be the set of all instances I , such that if any two elements of I have the same value for A then they also have the same value for B . (This constraint can be described by a functional dependency.) The instance $I = \{101, 202\}$ is in \mathbf{P} , and so is $m_{\mathbf{R}}(I) = \{101, 102, 201, 202\}$, but I is not a fixed point of \mathbf{R} .

Let \mathbf{P} a set of instances defined by a given set of constraints. One possible approach is to choose a database scheme \mathbf{R} such that $\mathbf{P} \subseteq \text{FIXPT}(\mathbf{R})$ [ABU], [Ris1] (that is, a database scheme that has the lossless join property). A different view is that the constraints alone describe all the relevant properties of the data and, therefore, it is

sufficient to choose a database scheme \mathbf{R} that allows enforcement of all the constraints [Bern]. Under the universal instance assumption, this approach amounts to selecting a database scheme \mathbf{R} that preserves \mathbf{P} . When the only constraints are functional dependencies (which are defined in the next section), the results proved in this section imply that if two database schemes \mathbf{R} and \mathbf{S} have the same set of fixed points in \mathbf{P} then they preserve the same set of instances in \mathbf{P} . In § 6.2 we show that if one accepts the view that functional dependencies describe all the relevant properties of the data, then the set of meaningful instances is a subset of \mathbf{P} . Furthermore, if a database scheme \mathbf{R} preserves \mathbf{P} , then every meaningful instance is a fixed point of \mathbf{R} .

We now give a theorem that characterizes containment of fixed point sets over a set of instances.

THEOREM 2. *Let \mathbf{Q} be any set of instances such that*

$$\text{FIXPT}_{\mathbf{P}}(\mathbf{R}) \subseteq \mathbf{Q} \subseteq \text{PRESERVED}(\mathbf{R}, \mathbf{P}).$$

Then $\text{FIXPT}_{\mathbf{P}}(\mathbf{R}) \subseteq \text{FIXPT}(\mathbf{S})$ if and only if $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$ for all instances I in \mathbf{Q} . (Note that $\text{FIXPT}_{\mathbf{P}}(\mathbf{R}) \subseteq \text{FIXPT}(\mathbf{S})$ if and only if $\text{FIXPT}_{\mathbf{P}}(\mathbf{R}) \subseteq \text{FIXPT}_{\mathbf{P}}(\mathbf{S})$.)

Proof. (only if). Given I in \mathbf{Q} , we know that I is in $\text{PRESERVED}(\mathbf{R}, \mathbf{P})$, and hence $m_{\mathbf{R}}(I)$ is in \mathbf{P} . Also, by J2, $m_{\mathbf{R}}(I)$ is in $\text{FIXPT}(\mathbf{R})$. Thus, by assumption, $m_{\mathbf{R}}(I)$ is in $\text{FIXPT}(\mathbf{S})$. Since $I \subseteq m_{\mathbf{R}}(I)$ by J1, it follows that $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{S}}(m_{\mathbf{R}}(I))$ by J3, that is, $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$.

(if). Let I be in $\text{FIXPT}_{\mathbf{P}}(\mathbf{R})$. Then I is in \mathbf{Q} , so $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I) = I$. Hence, by J1, the instance I is in $\text{FIXPT}(\mathbf{S})$. \square

The following two corollaries are immediate consequences of the above.

COROLLARY 2. *The following are equivalent:*

- (1) $\text{FIXPT}_{\mathbf{P}}(\mathbf{R}) \subseteq \text{FIXPT}(\mathbf{S})$.
- (2) $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$ for all I in $\text{FIXPT}_{\mathbf{P}}(\mathbf{R})$.
- (3) $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$ for all I in $\text{PRESERVED}(\mathbf{R}, \mathbf{P})$.

Proof. For (1) equivalent to (2), let $\mathbf{Q} = \text{FIXPT}_{\mathbf{P}}(\mathbf{R})$ in Theorem 2. For (1) equivalent to (3), let $\mathbf{Q} = \text{PRESERVED}(\mathbf{R}, \mathbf{P})$. \square

COROLLARY 3. *If \mathbf{R} preserves \mathbf{P} , then $\text{FIXPT}_{\mathbf{P}}(\mathbf{R}) \subseteq \text{FIXPT}(\mathbf{S})$ if and only if $m_{\mathbf{S}}(I) \subseteq m_{\mathbf{R}}(I)$ for all instances I in \mathbf{P} .*

5. Equivalence under dependencies. The subset \mathbf{P} of acceptable instances will usually not be given explicitly by the user. Instead, the user will specify in an appropriate language a set of *constraints* to be satisfied by the data, and \mathbf{P} will be taken to be the set of all instances that satisfy these constraints. A useful constraint language should probably allow the user to specify functional dependencies, multivalued dependencies, and other structural properties of the data. In this section we shall deal with the case where functional and multivalued dependencies are allowed.

5.1. Functional and multivalued dependencies. A *functional dependency* (fd) is a constraint of the form $X \rightarrow Y$, where X and Y are sets of attributes. We say that an instance I satisfies $X \rightarrow Y$ if whenever two tuples in I agree in their X -columns, they also agree in their Y -columns. We shall assume without loss of generality [Arm] that every fd is of the form $X \rightarrow A$, where A is a single attribute.

A *multivalued dependency* (mvd) is a constraint of the form $X \twoheadrightarrow Y$, where X and Y are sets of attributes. An instance I satisfies $X \twoheadrightarrow Y$ if the following condition holds. Let $Z = U - X - Y$. Whenever there are two tuples w_1, w_2 in I that agree in the X -columns, then there must also be a tuple v in I such that

$$v[X \cup Y] = w_1[X \cup Y] \quad \text{but} \quad v[Z] = w_2[Z],$$

where $x[W]$ means the components of tuple x corresponding to those attributes in the set of attributes W . In other words, $X \rightarrow Y$ means that the set of Y -values associated with a particular X -value must be independent of the rest of the attributes in the universe.

Given a set of fd's and mvd's D , we define $SAT(D)$ as the set of all instances that satisfy all the dependencies in D , and we shall use the abbreviations $PRESERVED(\mathbf{R}, D)$ for $PRESERVED(\mathbf{R}, SAT(D))$ and $FIXPT_D(\mathbf{R})$ for $FIXPT(\mathbf{R}) \cap SAT(D)$. We also say that \mathbf{R} preserves D if $SAT(D) = PRESERVED(\mathbf{R}, D)$.

5.2. Tableaux. In [ASU1], [ASU2] it is shown how certain matrices called *tableaux* can be used to represent a class of relational expressions. In particular, we shall use a simplified version of this construct, originally introduced in [ABU], to represent the project-join mappings associated with database schemes.

A *tableau* is a matrix consisting of a set of *rows*. Each column of a tableau corresponds to an attribute of the universe. Each row may contain *distinguished* and *nondistinguished* variables. We shall denote the distinguished variables by subscripted a 's and the nondistinguished variables by subscripted b 's. No variable may appear in more than one column, and no column may have two (or more) distinct distinguished variables. We shall assume in this paper that every column contains at least one occurrence of a distinguished variable. Thus, every column has a unique distinguished variable associated with it. (Usually, the distinguished variable in the i th column is denoted by a_i .)

Let T be a tableau and S the set of all the variables that appear in T . A *valuation* ρ for T is a mapping from S into the integers. If the rows of T are w_1, \dots, w_n , then $\rho(w_i)$ is the result of substituting $\rho(v)$ for every variable v that appears in w_i . A tableau T defines a mapping from instances to instances as follows. Let a_i be the distinguished variable appearing in the i th column, and suppose T has m columns. Then, given an instance I , we define

$$T(I) = \{ \rho(a_1, \dots, a_m) \mid \rho \text{ is a valuation for } T, \text{ and } \rho(w_i) \in I \text{ for all rows } w_i \}.$$

In Codd's relational calculus, the mapping associated with the tableau T can be defined by the following expression over a single universal relation I :

$$(a_1, \dots, a_m): \exists b_1 \dots \exists b_k [I(w_1) \wedge \dots \wedge I(w_n)].$$

The distinguished variables are those that appear in the target list, while the nondistinguished variables are all the other variables of the expression.

Example 2. Let T be the tableau

$$\begin{array}{|ccc|} \hline a_1 & a_2 & b_1 \\ \hline b_2 & a_2 & a_3 \\ \hline \end{array},$$

and consider the instance $I = \{121, 322\}$. If we assign 2 to a_2 and 1 to all other variables, each row becomes 121, which belongs to I . Therefore, 121 is in $T(I)$. If we assign 1 to a_1 and b_1 , 3 to b_2 and 2 to a_2 and a_3 , the first row of T is mapped into 121 and the second row into 322. Both these tuples are in I , hence 122 belongs to $T(I)$. Similarly one can show that 322 and 321, and no other tuples, are in $T(I)$. Thus

$$T(I) = \{121, 322, 122, 321\}.$$

Given a database scheme \mathbf{R} , we can construct a tableau $T_{\mathbf{R}}$ such that the mapping defined by $T_{\mathbf{R}}$ is $m_{\mathbf{R}}$, that is, $T_{\mathbf{R}}(I) = m_{\mathbf{R}}(I)$ for all instances I . The tableau $T_{\mathbf{R}}$ has one row for each relation scheme in \mathbf{R} . If the relation scheme is $\{A_1, \dots, A_k\}$, the corresponding row will contain distinguished variables in the columns corresponding to attributes A_1, \dots, A_k . All entries of the tableau not defined by this rule will be assigned distinct nondistinguished variables. The correctness of this construction is obvious when one considers the relational calculus expression that corresponds to this tableau.

Example 3. The tableau of Example 2,

A	B	C
a_1	a_2	b_1
b_2	a_2	a_3

is obtained from the database scheme $\{AB, BC\}$ on universe $\{A, B, C\}$.

Given two tableaux T_1 and T_2 , defined on the same set of attributes, we say T_2 is contained in T_1 , written $T_2 \subseteq T_1$, if $T_2(I) \subseteq T_1(I)$ for all instances I . The following fact [CM] will be useful in proving containment of tableaux:

FACT 1. $T_2 \subseteq T_1$ if and only if there is a mapping (called a containment mapping) from the set of variables of T_1 to the set of variables of T_2 such that:

- (a) Each distinguished variable is mapped to itself.
- (b) The image of each row of T_1 is a row of T_2 .

Example 4. Consider the following two tableaux:

$$T_1 = \begin{array}{|c|c|c|c|} \hline a_1 & a_2 & b_1 & b_2 \\ \hline a_1 & b_3 & a_3 & b_4 \\ \hline b_5 & a_2 & b_6 & a_4 \\ \hline \end{array},$$

$$T_2 = \begin{array}{|c|c|c|c|} \hline a_1 & a_2 & a_3 & b_1 \\ \hline b_2 & a_2 & a_3 & a_4 \\ \hline \end{array}.$$

The mapping that sends the first and second rows of T_1 to the first row of T_2 and the third row of T_1 to the second row of T_2 is a containment mapping, showing that T_2 is contained in T_1 .

So far we have treated tableaux essentially as nonprocedural expressions denoting mappings. However, a tableau can also be viewed as a set of tuples, that is, an instance. This instance has the property that when the corresponding mapping is applied to it, then the tuple (a_1, \dots, a_m) is in the image. This is a canonical instance for the mapping associated with the tableau, in the sense that properties of the mapping can be investigated by examining the instance. We make use of this fact below when we discuss the effect of dependencies on expressions.

5.3. Testing equivalence under dependencies. In this section we return to the problem of testing, given two database schemes \mathbf{R} and \mathbf{S} and a set of functional and multivalued dependencies D , whether every fixed point of \mathbf{S} that satisfies D is also a fixed point of \mathbf{R} .

The algorithm we shall present is an application of the generalized “chase” process described in [MMS]. By Corollary 2, our problem is equivalent to testing whether $m_{\mathbf{R}}(I) \subseteq m_{\mathbf{S}}(I)$ for all I in $\text{FIXPT}_D(\mathbf{S})$. Let a *full join dependency* [Ris2] be a statement of the form $\bowtie[\mathbf{R}]$, where \mathbf{R} is a database scheme. We say that an instance I satisfies the dependency $\bowtie[\mathbf{R}]$ if I is in $\text{FIXPT}(\mathbf{R})$. In other words, $\text{SAT}(\bowtie[\mathbf{R}]) = \text{FIXPT}(\mathbf{R})$. We may now rephrase our problem as follows. Given a database scheme \mathbf{R} , and a set D' of fd's, mvd's, and a full join dependency $\bowtie[\mathbf{S}]$, determine whether $m_{\mathbf{R}}(I) \subseteq m_{\mathbf{S}}(I)$ for all instances I satisfying the dependencies of D' . But $m_{\mathbf{S}}(I) = I$ for every I that satisfies the full join dependency $\bowtie[\mathbf{S}]$. Therefore the problem is to determine whether $m_{\mathbf{R}}(I) \subseteq I$ for all instances I in $\text{SAT}(D')$.

Let $T_{\mathbf{R}}$ be the tableau for the database scheme \mathbf{R} , and let T_0 be the tableau containing a single row with only distinguished variables (i.e., it corresponds to the database scheme $\{U\}$, where U is the set of all the attributes). Tableau T_0 represents the identity mapping, i.e., $T_0(I) = I$ for all instances I . We have to determine whether $T_{\mathbf{R}}(I) \subseteq T_0(I)$ for all instances I in $\text{SAT}(D')$. Since we have a situation where instances are required to satisfy a set D' of fd's, mvd's, and full join dependencies, Fact 1 is no longer true. However, it would be true if the canonical instance for $T_{\mathbf{R}}$ (i.e., $T_{\mathbf{R}}$ viewed as an instance) satisfied the dependencies. Using the *chase* process of [MMS], we can transform $T_{\mathbf{R}}$ to another tableau T' such that $T_{\mathbf{R}}(I) = T'(I)$ for all instances I in $\text{SAT}(D')$, and T' satisfies D' . The chase process is carried out by applying the dependencies of D' to $T_{\mathbf{R}}$. Formally, we associate with each dependency in D' a rule for modifying an arbitrary tableau T , and these rules are repeatedly applied to $T_{\mathbf{R}}$ until no longer possible. The rules for each type of dependency in D' (i.e., fd's, mvd's, and full join dependencies) are defined as follows.

(r1) If $X \rightarrow A$ is one of the fd's, and rows i and j of tableau T have identical variables in all the X -columns but different variables in the A -column, then change T by making the variables found in rows i and j of column A identical wherever they appear in T . If one of the equated variables is a distinguished variable, make the resulting variable equal to that variable. Identify rows that become identical.

(r2) If $X \twoheadrightarrow Y$ is one of the mvd's, and rows i and j of tableau T agree on the X -columns but disagree on some of the Y -columns and some of the remaining columns, then change T by adding two new rows i' and j' such that i' agrees with i on $X \cup Y$ and with j on the rest of the columns, and j' agrees with j on $X \cup Y$ and with i on the rest of the columns. Identify identical rows in the result.

(r3) If $\bowtie[\mathbf{R}]$ is one of the full join dependencies, then replace T with $m_{\mathbf{R}}(T)$ (i.e., apply the mapping $m_{\mathbf{R}}$ to T).

Example 5. Let T be

A	B	C
a_1	b_1	a_3
a_1	a_2	b_2
b_3	a_2	b_4

and $D = \{A \twoheadrightarrow B, B \rightarrow C, \bowtie[\mathbf{R}]\}$, where $\mathbf{R} = \{AC, BC\}$. By applying $A \twoheadrightarrow B$ to the first

and second rows of T , we obtain

A	B	C
a_1	b_1	a_3
a_1	a_2	b_2
a_1	b_1	b_2
a_1	a_2	a_3
b_3	a_2	b_4

Now applying $B \rightarrow C$ to rows 1 and 3, and then to rows 4 and 5 yields, after identifying common rows,

A	B	C
a_1	b_1	a_3
a_1	a_2	a_3
b_3	a_2	a_3

Finally, applying rule r3 for $\bowtie[\mathbf{R}]$ yields

A	B	C
a_1	b_1	a_3
a_1	a_2	a_3
b_3	a_2	a_3
b_3	b_1	a_3

It is shown in [MMS] that any sequence of applications of the rules terminates after a finite (exponential) number of steps, that is, a tableau is reached that cannot be changed by applying any of the rules. (It is also shown there that the final tableau is unique, regardless of the order of application of the rules, but for our purposes here uniqueness does not matter.) For a given set of dependencies C , let $\text{chase}_C(T)$ denote a tableau obtained from T by exhaustive application of the rules.

Let I be in $\text{SAT}(C)$ and suppose a tableau T is changed to a tableau T' by applying one of the rules, using a dependency from C . It can be easily shown that any valuation of T into I is also a valuation of T' into I and vice versa. Therefore, $T'(I) = T(I)$. By induction, we obtain the following.

FACT 2. *Let C be a set of constraints containing fd's, mvd's, and full join dependencies, and let T be a tableau. Then $\text{chase}_C(T)(I) = T(I)$ for all instances I in $\text{SAT}(C)$.*

From the above it follows that $T_2(I) \subseteq T_1(I)$ for all I in $\text{SAT}(C)$ if and only if $\text{chase}_C(T_2)(I) \subseteq T_1(I)$ for all I in $\text{SAT}(C)$. Notice, however, that the canonical instance for $\text{chase}_C(T_2)$ satisfies C . Therefore it can be shown (as in [ASU1], [MMS]) that the last containment holds if and only if $\text{chase}_C(T_2)(I) \subseteq T_1(I)$ for all I . In summary:

FACT 3. *$T_2(I) \subseteq T_1(I)$ for all I in $\text{SAT}(C)$ if and only if $\text{chase}_C(T_2) \subseteq T_1$.*

Returning now to the problem at hand, we have to determine whether $T_{\mathbf{R}}(I) \subseteq T_0(I)$ for all instances I in $\text{SAT}(D')$. (Recall that D' is the original set of fd's and mvd's along with the full join dependency $\bowtie[\mathbf{S}]$, and T_0 is the tableau containing the single row

$a_1 \cdots a_m$ and it represents the identity mapping.) By Facts 2 and 3, we know that $T_{\mathbf{R}}(I) \subseteq T_0(I)$ for all I in $\text{SAT}(D')$ if and only if $\text{chase}_{D'}(T_{\mathbf{R}}) \subseteq T_0$; that is, (by Fact 1) if and only if there is a containment mapping from T_0 to $\text{chase}_{D'}(T_{\mathbf{R}})$. But such a containment mapping exists if and only if $\text{chase}_{D'}(T_{\mathbf{R}})$ contains a row with a distinguished variable in every column. In summary, given two database schemes \mathbf{R} and \mathbf{S} and a set of functional and multivalued dependencies D , we have an algorithm for determining whether every fixed point of \mathbf{S} that satisfies D is also a fixed point of \mathbf{R} . Let D' be D along with the full join dependency $\bowtie[\mathbf{S}]$. The algorithm consists of computing $\text{chase}_{D'}(T_{\mathbf{R}})$ and checking whether $\text{chase}_{D'}(T_{\mathbf{R}})$ has a row containing only distinguished variables. This is summarized in the following theorem.

THEOREM 3. *There is an algorithm of complexity $O(n^{kn})$ to test containment of fixed point sets of two database schemes under a set of fd's and mvd's, where n is the total space required to write down the relation schemes and the dependencies and k is a constant.*

Proof. By the above remarks. \square

5.4. Schemes that preserve functional dependencies. We shall now consider the special case where D consists only of fd's, and furthermore, the database scheme \mathbf{S} preserves the set D , that is, $\text{PRESERVED}(\mathbf{S}, D) = \text{SAT}(D)$. In this case we can derive a polynomial algorithm for testing containment of fixed point sets in the following way.

THEOREM 4. *If database scheme \mathbf{S} preserves a set of fd's D , then $\text{FIXPT}_D(\mathbf{S}) \subseteq \text{FIXPT}(\mathbf{R})$ if and only if $\text{chase}_D(T_{\mathbf{R}}) \subseteq T_{\mathbf{S}}$.*

Proof. By Corollary 3, we know that if \mathbf{S} preserves D , then $\text{FIXPT}_D(\mathbf{S}) \subseteq \text{FIXPT}(\mathbf{R})$ if and only if $m_{\mathbf{R}}(I) \subseteq m_{\mathbf{S}}(I)$ for all instances I in $\text{SAT}(D)$. By Fact 3, the latter condition is equivalent to $\text{chase}_D(T_{\mathbf{R}}) \subseteq T_{\mathbf{S}}$. \square

COROLLARY 4. *Under the conditions of Theorem 4, containment of fixed point sets can be tested in time polynomial in the size of the relation schemes and the dependencies.*

Proof. As shown in [ABU], if D contains only fd's, the chase of $T_{\mathbf{R}}$ under D can be computed in time $O(n^4)$, where n is the total space required to write down $T_{\mathbf{R}}$ and D . It remains to test whether there exists a containment mapping from $T_{\mathbf{S}}$ to $\text{chase}_D(T_{\mathbf{R}})$. Since $T_{\mathbf{S}}$ contains no repeated nondistinguished variables, this can be done simply by checking each row s in $T_{\mathbf{S}}$ against all rows of $\text{chase}_D(T_{\mathbf{R}})$ until we find one that s can be mapped to. That is, until we find a row w of $\text{chase}_D(T_{\mathbf{R}})$ such that w has a distinguished variable in every column in which s has a distinguished variable. This can clearly be done in time $O(p^2q)$ for two tableaux with no more than q columns and p rows. \square

Our next theorem will show that the class of database schemes that preserves a given set of fd's can be characterized by a simple tableau-based condition that can also be tested in polynomial time. We shall need some definitions and preliminary results.

Given a set of fd's, some other fd's will be implied by them: for example, $X \rightarrow Y$ and $Y \rightarrow Z$ imply $X \rightarrow Z$. The *closure* D^+ of D is the set of fd's that must be satisfied by any relation that satisfies D . A set of fd's F covers an fd f if f is in F^+ . In particular, every fd of the form $X \rightarrow Y$, with $Y \subseteq X$, is covered by \emptyset . A *cover* for D is any set of fd's E such that $E^+ = D^+$. A cover for D is *nonredundant* if no proper subset of it is a cover for D . Given a set of attributes X , and a set of fd's D , the *closure of X under D* , denoted $\text{Cl}(X)$, is the set of all attributes A such that $X \rightarrow A$ is in D^+ . A set of fd's D is *embedded* in a database scheme \mathbf{R} if for every dependency $X \rightarrow A$ in D there is a set R in \mathbf{R} such that $X \cup \{A\} \subseteq R$. Similarly, we say that a set of fd's D is embedded in a tableau T if for every dependency $X \rightarrow A$ in D , there is a row in the tableau containing distinguished variables in all columns corresponding to attributes in $X \cup \{A\}$.

PROPOSITION 1. *Suppose \mathbf{R} preserves D and $T_{\mathbf{R}}$ is the tableau for $m_{\mathbf{R}}$. For a row r of $T_{\mathbf{R}}$, let $X(r)$ be the set of attributes such that their columns have distinguished variables in*

row r . Then $\text{chase}_D(T_{\mathbf{R}})$ is obtained from $T_{\mathbf{R}}$ by replacing each row r by the row r^+ , where r^+ has distinguished variables exactly in the $\text{Cl}(X(r))$ -columns and the same nondistinguished variables as r in the other columns, and then identifying identical rows.

Proof. Let $X(r')$ denote the set of attributes whose columns have distinguished variables in row r' of $\text{chase}_D(T_{\mathbf{R}})$. We first prove that $\text{Cl}(X(r')) = X(r')$. Indeed, suppose not. Then there is some attribute B such that $X(r') \rightarrow B$ is in D^+ , but the B -column in row r' contains a nondistinguished variable. By the definition of the chase, $\text{chase}_D(T_{\mathbf{R}})$ (considered as an instance) satisfies D and, since \mathbf{R} preserves D , so does $m_{\mathbf{R}}(\text{chase}_D(T_{\mathbf{R}}))$. However, $m_{\mathbf{R}}(\text{chase}_D(T_{\mathbf{R}}))$ contains the tuple (a_1, \dots, a_m) that agrees with r' on the $X(r')$ -columns but not on the B -column—a contradiction.

Now, each r of $T_{\mathbf{R}}$ is transformed by the chase process into a row r' of $\text{chase}_D(T_{\mathbf{R}})$. (Note that several rows of $T_{\mathbf{R}}$ may be transformed into the same row of $\text{chase}_D(T_{\mathbf{R}})$, because identical rows are identified in the process.) Since the process preserves the distinguished variables in the tableau, $X(r) \subseteq X(r')$, hence $\text{Cl}(X(r)) \subseteq \text{Cl}(X(r')) = X(r')$.

To conclude the proof, note that if the fd $Y \rightarrow B$ is the first fd applied to rows i and j of a tableau T , then $Y \subseteq X(i) \cap X(j)$, so $B \in \text{Cl}(X(i) \cap X(j)) \subseteq \text{Cl}(X(i))$. It follows easily by induction that the set of columns in row r of $T_{\mathbf{R}}$ that are affected by the chase process is contained in $\text{Cl}(X(r))$. Hence $X(r') = \text{Cl}(X(r))$ and the variables in the columns not in $\text{Cl}(X(r))$ remain as in $T_{\mathbf{R}}$. \square

COROLLARY 5. *If \mathbf{R} preserves D , then the application of the chase process to $T_{\mathbf{R}}$ produces the same tableau regardless of which cover of D^+ was used in the process and regardless of the order of application of the fd's in the process.*

Proof. For any set of attributes X , the set $\text{Cl}(X)$ does not depend upon the particular cover chosen to represent D . The tableau $\text{chase}_D(T_{\mathbf{R}})$ is defined in terms of closures, hence is independent of the cover used and the order of application of the fd's. \square

Note that uniqueness of the result of the chase holds even when \mathbf{R} does not preserve D and even when other types of dependencies such as mvd's are present [MMS].

LEMMA 4. *Suppose \mathbf{R} preserves D , the tableau of $m_{\mathbf{R}}$ is $T_{\mathbf{R}}$, and X is a set of attributes. Let T' be the tableau obtained from $\text{chase}_D(T_{\mathbf{R}})$ by adding to it an additional row with distinguished variables in the X -columns and new, nondistinguished variables everywhere else. Then $\text{chase}_D(T')$ is $\text{chase}_D(T_{\mathbf{R}})$ with the additional row modified only by replacing in some columns the nondistinguished variables with the distinguished variables of those columns (and, perhaps, identifying it with a row of $\text{chase}_D(T_{\mathbf{R}})$). Each variable so changed in the additional row is in $\text{Cl}(X)$.*

Proof. The proof is by induction on the number of applications of fd's to compute $\text{chase}_D(T')$. As long as $\text{chase}_D(T_{\mathbf{R}})$ is unchanged, the next application of an fd must necessarily involve the additional row. Say we apply $Y \rightarrow B$ to the new row and to row r of $\text{chase}_D(T_{\mathbf{R}})$. Then all the Y -columns in row r contain distinguished variables. By Proposition 1, the B -column of r must also contain a distinguished variable. Hence row r is not modified and the only change is that the nondistinguished variable in the B -column in the new row becomes distinguished. By the induction hypothesis, $Y \subseteq \text{Cl}(X)$, so $B \in \text{Cl}(X)$. \square

LEMMA 5. *Under the assumptions of the previous lemma, if the last row of $\text{chase}_D(T')$ contains a distinguished variable in its A -column for any attribute A , then there must be a subset of D that covers the dependency $X \rightarrow A$ and is embedded in $\text{chase}_D(T_{\mathbf{R}})$.*

Proof. The proof is again by induction on the number of applications of fd's to compute $\text{chase}_D(T')$. Initially, only variables in the X -columns of the last row are distinguished. For each $A \in X$, the fd $X \rightarrow A$ is covered by the empty set of fd's. Suppose now that, after $n \geq 1$ applications of fd's, the next application uses the fd $Y \rightarrow B$ of D . Let $Y = \{C_1, \dots, C_k\}$. By the proof of Lemma 4, the last row must be involved in this application and all the Y -columns in the last row already have distinguished variables. By the induction hypothesis there are subsets D_1, \dots, D_k of D that cover the fd's $X \rightarrow C_1, \dots, X \rightarrow C_k$, and are embedded in $\text{chase}_D(T_{\mathbf{R}})$. That $Y \rightarrow B$ is embedded in $\text{chase}_D(T_{\mathbf{R}})$ follows from the fact that it is now applicable. The set $(\cup_{i=1}^k D_i) \cup \{Y \rightarrow B\}$ is the required set. \square

THEOREM 5. *Let \mathbf{R} be a database scheme, let D be a set of fd's and let D^* denote the union of all nonredundant covers of D . Then the following are equivalent:*

- (1) \mathbf{R} preserves D .
- (2) Some cover of D is embedded in $\text{chase}_D(T_{\mathbf{R}})$.
- (3) The set D^* is embedded in $\text{chase}_D(T_{\mathbf{R}})$. (That is, every nonredundant cover of D is embedded in $\text{chase}_D(T_{\mathbf{R}})$.)

Proof. (1) implies (2). Let $X \rightarrow A$ be in D . Let T' be the augmented tableau constructed as in Lemmas 4 and 5 for the set X . By definition of the chase, $\text{chase}_D(T')$ satisfies D and, since \mathbf{R} preserves D , so does $m_{\mathbf{R}}(\text{chase}_D(T'))$. This tableau contains a tuple (a_1, \dots, a_n) which agrees with the last row of $\text{chase}_D(T')$ on the X -columns and, hence, also on the A -column. It follows that the A -column in the last row of $\text{chase}_D(T')$ contains a distinguished variable and, by Lemma 5, there exists a subset of D that covers $X \rightarrow A$ and is embedded in $\text{chase}_D(T_{\mathbf{R}})$. Since $X \rightarrow A$ was an arbitrary fd in D , there is the subset of D that covers D and is embedded in $\text{chase}_D(T_{\mathbf{R}})$.

(2) implies (1). Suppose some cover E of D is embedded in $\text{chase}_D(T_{\mathbf{R}})$. (We do not need here the uniqueness of $\text{chase}_D(T_{\mathbf{R}})$. Rather, let $\text{chase}_D(T_{\mathbf{R}})$ be any tableau that is obtained by applying the chase process to $T_{\mathbf{R}}$ using fd's from D .) Let I be an instance in $\text{SAT}(D)$. Note that I also satisfies E . Now, $m_{\mathbf{R}}, T_{\mathbf{R}}$ and $\text{chase}_D(T_{\mathbf{R}})$ define the same mapping on $\text{SAT}(D)$ (Fact 2). For every dependency $X \rightarrow A$ in E , the tableau $\text{chase}_D(T_{\mathbf{R}})$ contains a row where the columns for $X \cup \{A\}$ have distinguished variables. By the definition of the mapping associated with $\text{chase}_D(T_{\mathbf{R}})$, every tuple in $m_{\mathbf{R}}(I) = \text{chase}_D(T_{\mathbf{R}})(I)$ must agree on $X \cup \{A\}$ with some tuple of I .

Suppose that tuples r_1 and r_2 of $\text{chase}_D(T_{\mathbf{R}})(I)$ agree on all the X -columns. There are rows s_1 and s_2 of I such that r_1 and r_2 agree with s_1 and s_2 , respectively, on the columns for $X \cup \{A\}$. Since I satisfies E , rows s_1 and s_2 agree also on their A -column and, hence, so do rows r_1 and r_2 . It follows that $\text{chase}_D(T_{\mathbf{R}})(I)$ satisfies every fd in E , hence it satisfies D and \mathbf{R} preserves D .

(1) implies (3). We have already seen in the proof of "(1) implies (2)" that if E is a set of fd's and \mathbf{R} preserves E , then $\text{chase}_E(T_{\mathbf{R}})$ embeds a subset of E that covers E . In particular, if E is nonredundant, then E itself is embedded in $\text{chase}_E(T_{\mathbf{R}})$. This holds for any E that is a nonredundant cover of D . By Corollary 5, if \mathbf{R} preserves D then $\text{chase}_D(T_{\mathbf{R}})$ is independent of the cover used in the chase. Hence this unique tableau embeds every fd that belongs to some nonredundant cover of D , that is, it embeds D^* .

(3) implies (2). Obvious. \square

COROLLARY 6. *It can be decided in polynomial time whether \mathbf{R} preserves D .*

Proof. Given D , first compute a nonredundant cover E of D using, say, the method of [BB], which takes time $O(|D|^2)$. Then compute $\text{chase}_D(T_{\mathbf{R}})$ by the method of [ABU] in quartic time. Now, check if the cover E is embedded in $\text{chase}_D(T_{\mathbf{R}})$ in $O(|E| \cdot |T_{\mathbf{R}}|)$ time. \square

This algorithm is an extension of the lossless join algorithm of [ABU]. There the problem was to decide whether a given set of fd's D implies a lossless join $\bowtie[\mathbf{R}]$. In other terms, the problem was to decide if $\text{SAT}(D) \subseteq \text{FIXPT}(\mathbf{R})$, which is stronger than \mathbf{R} , preserves D . In both cases, the algorithm starts by computing $\text{chase}_D(T_{\mathbf{R}})$. Then, if a cover of D is embedded in $\text{chase}_D(T_{\mathbf{R}})$ then \mathbf{R} preserves D , while if $\text{chase}_D(T_{\mathbf{R}})$ contains a row of only distinguished variables then $\text{SAT}(D) \subseteq \text{FIXPT}(\mathbf{R})$.

The reader should observe that Theorem 5 is less restrictive than the notion of [Bern] that a cover for D^+ be embedded in $T_{\mathbf{R}}$ (as opposed to $\text{chase}_D(T_{\mathbf{R}})$) in order for a database scheme to be an adequate representation of D . We shall deal with this notion of representation in the next section. The examples in the next section can be used to show that there are schemes that have a lossless join, but do not embed a cover of D , there are schemes that embed a cover of D but have a lossy join and there are schemes that preserve D but have a lossy join and do not embed a cover of D .

6. Update sets.

6.1. Update sets and the conceptual schema. In this section we apply the concepts that we have developed to examine how a universal relation can be used as a conceptual database. Recently, attention has focused on the problem of which structures should be used in the conceptual schema (e.g., [Nijs], [HOT]). It seems to be agreed that the conceptual schema for an application should be constructed of basic, irreducible units of information (called irreducible sentences in [Nijs], irreducible relations in [HOT]). These irreducible information units serve as a complete description of the database structure. Every transaction to be effected against the database is expressed in terms of these units.

In our model, we can take these irreducible units to be simply sets of attributes. The conceptual schema then consists of a collection $\mathbf{U} = \{U_1, \dots, U_k\}$, of sets of attributes, and the corresponding database is viewed as a set of relations $\{u_1, \dots, u_k\}$ where u_i is a relation on the set U_i . Every transaction against the database is expressed in terms of updates to these relations. Therefore, we call the given sets U_1, \dots, U_k *update sets*.

Obviously, an instance I represents the same database as the u_i 's only if its projection onto each U_i is u_i . However, many different instances may have the same projections onto the U_i 's. To avoid ambiguities and, more importantly, to establish in the conceptual database all the relationships that are deducible from the relations u_1, \dots, u_k by using the join operation, we restrict I to be maximal, that is, to be the largest instance that projects to the u_i 's. Put another way, given u_1, \dots, u_k we select their join as the universal instance representing the same database. It follows that the instance selected will be a fixed point of the project-join mapping defined by the U_i 's.

The assumption that we are only interested in instances that are fixed points of \mathbf{U} will permit us to infer, using Theorem 1, that certain database schemes are guaranteed to have a lossless join over the same set of instances as the collection \mathbf{U} . Under our interpretation, this means that each of these schemes is as powerful as \mathbf{U} , that is, the database can be represented by any of these schemes without losing the ability to represent any collection of information pertaining to the user world. Thus, if there are no dependencies, we may summarize this as follows:

COROLLARY 7. *If \mathbf{U} is a collection of update sets, and \mathbf{R} is a database scheme, then the join of the relation schemes in \mathbf{R} is lossless with respect to all the fixed points of $m_{\mathbf{U}}$ if and only if each update set is contained in some relation scheme of \mathbf{R} . \square*

When the instances are constrained, Corollary 7 is unnecessarily restrictive. Suppose we have a set D of functional and multivalued dependencies that every instance must satisfy. Then there may be database schemes that can be used to replace

U without loss of potential information even if they do not meet the conditions of the corollary, since we are only considering instances in $\text{FIXPT}(U) \cap \text{SAT}(D)$. In this case, the algorithm of § 5.3 provides a better criterion, although at a large price in computation time. Since the algorithm is used only at the scheme construction stage, it may well be beneficial to use it despite the high price.

6.2. Functional dependencies as update sets. To illustrate the concept of update sets we consider a special case. Suppose we have a set D of fd's. (The arguments we are going to present apply to mvd's as well but, for simplicity, we restrict ourselves to fd's.) So far, we have taken into account only the role of fd's as constraints on the database. This was effected by restricting our attention to $\text{SAT}(D)$. However, fd's have also a natural interpretation as information units. Fd's such as $\text{EMPLOYEE} \rightarrow \text{SALARY}$ or $\text{DEPARTMENT} \rightarrow \text{MANAGER}$ represent (aside from the functionality constraint) associations between the attributes that appear in their left and right sides. Such associations are, in a sense, more primitive than the relations in the database (see, e.g., [Bern]). They are, therefore, prime candidates for the irreducible relations of the conceptual schema.

For an fd $f = X \rightarrow Y$, let $\text{ATTR}(f)$ denote the set $X \cup Y$; for a set D of fd's let $\text{ATTR}(D)$ be $\{\text{ATTR}(f) \mid f \in D\}$. Given a set D , we would like to choose a set of update sets related to D . We have to be careful, though, since sets of fd's may contain redundancy in various forms. As an example, if an fd $X \rightarrow Y$ is in D^+ then so is the fd $(U - X - Y) \cup X \rightarrow Y$. But the set of attributes of the latter fd is U , so if we choose $\text{ATTR}(D^+)$ to be the collection of update sets, it will always contain U and its set of fixed points is the set of all instances. As another example, suppose we choose $\text{ATTR}(\{A \rightarrow B, B \rightarrow C, A \rightarrow C\})$ as the collection of update sets. Obviously, given relations on the sets AB and BC , the relation on AC can be obtained by join and projection from the first two relations, so there seems to be a good reason not to include AC in the collection.

Let us then restrict our attention to collections of the form $\text{ATTR}(E)$, where E is a nonredundant cover of D . We note that D may have several nonredundant covers. Let us denote by T_E the tableau of $\text{ATTR}(E)$. By Fact 2, the tableaux T_E and $\text{chase}_D(T_E)$ define the same mapping on $\text{SAT}(D)$. By Theorem 5, for each cover E of D , the database scheme $\text{ATTR}(E)$ preserves $\text{SAT}(D)$, hence we can apply Proposition 1 to compute $\text{chase}_D(T_E)$. Let $E = \{f_1, \dots, f_k\}$. So the tableau $\text{chase}_D(T_E)$ is the tableau of the collection $\{\text{Cl}(\text{ATTR}(f_1)), \dots, \text{Cl}(\text{ATTR}(f_k))\}$. However, for an fd $X \rightarrow Y$, $\text{Cl}(X \cup Y) = \text{Cl}(X)$. Furthermore, as shown in [Bern], if E_1 and E_2 are two nonredundant covers of a set of fd's D , then for each $X \rightarrow W$ in E_1 there exists an fd $Y \rightarrow Z$ in E_2 such that $X \rightarrow Y$ and $Y \rightarrow X$ are both in D^+ , and hence $\text{Cl}(X) = \text{Cl}(Y)$. Therefore, the collection

$$D = \{\text{Cl}(X) \mid X \rightarrow Y \text{ is in the nonredundant cover } E \text{ of } D\}$$

is the same for all nonredundant covers of D and is independent of the particular cover E . Hence $\text{chase}_D(T_E)$ is the same tableau for all nonredundant covers of D . We have proved the following:

THEOREM 6. *Let D be a set of fd's. For every pair E_1 and E_2 of nonredundant covers of D , the mappings defined by the collection $\text{ATTR}(E_1)$ and $\text{ATTR}(E_2)$ are the same on $\text{SAT}(D)$. \square*

COROLLARY 8. *For a set of fd's D , the set of instances $\text{FIXPT}(\text{ATTR}(E)) \cap \text{SAT}(D)$ is the same for all nonredundant covers E of D .*

In view of the corollary, for the set of fd's D there exists a set of fixed points in $\text{SAT}(D)$ which is obtained by considering the fd's as information units and is independent of the particular representation E chosen for D , provided that E is nonredundant.

Let us denote this set by

$$\text{INTENDED}(D) = \text{FIXPT}(\text{ATTR}(E)) \cap \text{SAT}(D).$$

We propose that $\text{INTENDED}(D)$ is the set of instances that are of interest for an application described by a set of fd's D . In other words, when looking for a database scheme to represent such an application, one should select one whose set of fixed points contains $\text{INTENDED}(D)$.

The next theorem characterizes all database schemes that have at least $\text{INTENDED}(D)$ as their fixed point set, given some D . These can be thought of as the schemes that may be used instead of any non-redundant cover of D without losing representation power. It turns out that these are exactly the schemes that preserve D .

THEOREM 7. $\text{INTENDED}(D) \subseteq \text{FIXPT}(\mathbf{R}) \cap \text{SAT}(D)$ if and only if \mathbf{R} preserves D .

Proof. (if) Let I be an instance in $\text{INTENDED}(D)$. The database scheme \mathbf{R} preserves D , hence by Proposition 1, the tableau chase $_D(T_{\mathbf{R}})$ corresponds to some database scheme \mathbf{R}' . Since \mathbf{R}' embeds a cover of D by Theorem 5, the instance I must be in $\text{FIXPT}(\mathbf{R}') \cap \text{SAT}(D)$. Since $m_{\mathbf{R}'}$ agrees with $m_{\mathbf{R}}$ on $\text{SAT}(D)$, it follows that I is in $\text{FIXPT}(\mathbf{R}) \cap \text{SAT}(D)$.

(only if) Let \mathbf{S} be a database scheme corresponding to some nonredundant cover of D . Note that $\text{INTENDED}(D) = \text{FIXPT}(\mathbf{S}) \cap \text{SAT}(D)$, and \mathbf{S} preserves D ; i.e., $\text{PRESERVED}(\mathbf{S}, D) = \text{SAT}(D)$. By Corollary 2, $\text{FIXPT}(\mathbf{S}) \cap \text{SAT}(D) \subseteq \text{FIXPT}(\mathbf{R})$ implies $m_{\mathbf{R}}(I) \subseteq m_{\mathbf{S}}(I)$ for all I in $\text{PRESERVED}(\mathbf{S}, D)$. Hence, $m_{\mathbf{R}}(I) \subseteq m_{\mathbf{S}}(I)$ for all I in $\text{SAT}(D)$, which implies that \mathbf{R} preserves D . \square

COROLLARY 9. There is a polynomial time algorithm to determine, given a set of fd's D and a database scheme \mathbf{R} , whether $\text{FIXPT}(\mathbf{R})$ contains $\text{INTENDED}(D)$.

Proof. By Theorem 7 and Corollary 6. \square

By Theorem 7, $\text{INTENDED}(D)$ is the minimal set of fixed points of database schemes that preserve D . However, it is possible that the set of fixed points of \mathbf{R} in $\text{SAT}(D)$ properly contains $\text{INTENDED}(D)$, yet there is no subset of \mathbf{R} that is a database scheme, as in Example 6 below. If we want to have a database scheme whose set of fixed points in $\text{SAT}(D)$ is exactly $\text{INTENDED}(D)$, then the prime candidates are the schemes that embed nonredundant covers of D . An efficient algorithm for synthesizing such schemes is presented in [Bern].

We conclude with a few examples that differentiate between the various concepts that were discussed in §§ 5 and 6.

Example 6.

$$D = \{A \rightarrow C, B \rightarrow C\}$$

$$\mathbf{R} = \{AC, AB\}$$

$$T_{\mathbf{R}} = \begin{array}{c} \begin{array}{ccc} A & B & C \\ a_1 & b_1 & a_3 \\ a_1 & a_2 & b_2 \end{array} \end{array}$$

$$\text{chase}_D(T_{\mathbf{R}}) = \begin{array}{c} \begin{array}{ccc} A & B & C \\ a_1 & b_1 & a_3 \\ a_1 & a_2 & a_3 \end{array} \end{array}$$

Since $\text{chase}_D(T_{\mathbf{R}})$ contains a row of distinguished variables, \mathbf{R} has a lossless join so $\text{INTENDED}(D) \subseteq \text{FIXPT}(\mathbf{R}) \cap \text{SAT}(D) = \text{SAT}(D)$.

Example 7.

$$D = \{A \rightarrow B, A \rightarrow C, B \rightarrow D, C \rightarrow D\}$$

$$\mathbf{R} = \{AB, AC, BD\}$$

	A	B	C	D
$T_{\mathbf{R}} =$	a_1	a_2	b_1	b_2
	a_1	b_3	a_3	b_4
	b_5	a_2	b_6	a_4

	A	B	C	D
$\text{chase}_D(T_{\mathbf{R}}) =$	a_1	a_2	a_3	a_4
	b_5	a_2	b_6	a_4

Here again \mathbf{R} has a lossless join. Note that \mathbf{R} is a proper subset of a nonredundant cover of D .

In the preceding examples we had the somewhat peculiar situation of an attribute that appears on the right side of two fd's. This is not a necessary condition.

Example 8.

$$D = \{AB \rightarrow C, C \rightarrow D, D \rightarrow A, ADE \rightarrow F\}$$

$$\mathbf{R} = \{ABC, CD, DEF\}$$

	A	B	C	D	E	F
$T_{\mathbf{R}} =$	a_1	a_2	a_3	b_1	b_2	b_3
	b_4	b_5	a_3	a_4	b_6	b_7
	b_8	b_9	b_0	a_4	a_5	a_6

	A	B	C	D	E	F
$\text{chase}_D(T_{\mathbf{R}}) =$	a_1	a_2	a_3	a_4	b_2	b_3
	a_1	b_3	a_3	a_4	b_6	b_7
	a_1	b_9	b_0	a_4	a_5	a_6

In this case the join is lossy, since there is no row of distinguished variables in $\text{chase}_D(T_{\mathbf{R}})$. However, the rows of $\text{chase}_D(T_{\mathbf{R}})$ represent the database scheme $\{ABCD, ACD, ADEF\}$, and by Corollary 1, this database scheme is equivalent to the database scheme $\mathbf{D} = \{ABCD, ACD, AD, ADEF\}$ (see the definition of \mathbf{D} just before Theorem 6). Therefore, the set of fixed points of \mathbf{R} in $\text{SAT}(D)$ is exactly $\text{INTENDED}(D)$.

7. Conclusions. In this paper we introduced and investigated a formalization of the concept of data equivalence of database schemes. Algorithms for testing equivalence of schemes under dependencies were presented and cases where poly-

nomial time algorithms exist were distinguished. The problem of whether more efficient algorithms exist is open.

Of particular importance is the concept of update sets introduced in § 6. There are currently several approaches to relational database scheme design, in the presence of fd's. Bernstein [Bern] advocates schemes that embed a nonredundant cover of the given fd's; another approach is to look for schemes that enjoy the lossless join property [ABU], [Ris1]. These approaches do not always yield the same results. Example 6 in § 6.2 is probably the smallest case where they disagree. The scheme $\mathbf{R} = \{AC, AB\}$ has the lossless join property with respect to the given fd's, but does not embed a cover of them. The scheme $\mathbf{S} = \{AC, BC\}$ embeds a cover but has a lossy join.

Our approach to this apparent contradiction has been to restrict the requirement of the lossless join to some set of meaningful instances. When fd's are given, it seems natural in the light of Theorem 7 to take INTENDED (D) as this set. It is then trivially true that every database scheme that embeds a cover of the fd's has the lossless join property with respect to this set.

Note that we deal here only with data equivalence. As stated in the introduction, there is another aspect to equivalence, the ability of two database schemes to enforce the same sets of dependencies. A treatment of this aspect is outside the scope of this paper; however, when this aspect is also taken into consideration, the class of database schemes that have the lossless join property with respect to INTENDED (D) will be probably found to be a proper subset of the set of database schemes that preserve the fd's.

8. Acknowledgment. The authors thank Ron Fagin for helpful comments.

REFERENCES

- [ABU] A. V. AHO, C. BEERI AND J. D. ULLMAN, *The theory of joins in relational databases*, ACM Trans. Database Systems, 4 (1979), pp. 297–314.
- [ANSI] ANSI/X3/SPARC, *The ANSI/X3/SPARC framework*, D. Tsichritzis and A. Klug, eds., AFIPS Press, 1978.
- [Arm] W. W. ARMSTRONG, *Dependency structures of data base relationships*, Proc. IFIP '74, North-Holland, Amsterdam, 1974, pp. 580–583.
- [ASU1] A. V. AHO, Y. SAGIV AND J. D. ULLMAN, *Equivalences among relational expressions*, this Journal, 8 (1979), pp. 218–246.
- [ASU2] ———, *Efficient optimization of a class of relational expressions*, ACM Trans. Database Systems, 4 (1979), pp. 435–454.
- [BB] C. BEERI AND P. A. BERNSTEIN, *Computational problems related to the design of normal form relational schemas*, ACM Trans. Database Systems, 4 (1979), pp. 30–59.
- [BBG] C. BEERI, P. BERNSTEIN AND N. GOODMAN, *A sophisticate's introduction to database normalization theory*, Proc. 4th Int. Conf. on Very Large Data Bases, West Berlin, 1978, pp. 113–124.
- [Bern] P. A. BERNSTEIN, *Synthesizing third normal form relations from functional dependencies*, ACM Trans. Database Systems, 1 (1976), pp. 277–298.
- [CM] A. K. CHANDRA AND P. M. MERLIN, *Optimal implementation of conjunctive queries in relational databases*, Proc. 9th Annual ACM Symposium on Theory of Computing, 1976, pp. 77–90.
- [Codd1] E. F. CODD, *A relational model for large shared data banks*, Comm. ACM, 13 (1970), pp. 377–387.
- [Codd2] ———, *Further normalization of the data base relational model*, in Data Base Systems, Courant Computer Science Symp. 6, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 33–64.
- [Codd3] ———, *Understanding relations*, FDT, 7 (1975), pp. 23–28.
- [Del] C. DELOBEL, *Contributions théorétiques à la conception et à l'évaluation d'un système d'informations appliqué à la gestion*, Thèse d'état, Univ. de Grenoble, October, 1973.
- [Fagin] R. FAGIN, *Multivalued dependencies and a new normal form for relational databases*, ACM Trans. Database Systems, 2 (1977), pp. 262–278.
- [HOT] P. HALL, T. OWLETT AND S. TODD, *Relations and entities*, in Modelling in Data Base Management Systems, G. M. Nijssen, ed., North-Holland, Amsterdam, 1976.

- [LP] M. LACROIX AND A. PIROTTE, *Generalized joins*, SIGMOD Record, 8 (1976), pp. 14–15.
- [MMS] D. MAIER, A. O. MENDELZON AND Y. SAGIV, *Testing implications of data dependencies*, ACM Trans. Database Systems, 4 (1979), pp. 455–469.
- [Nijs] G. M. NIJSSEN, *On the gross architecture for the next generation database management systems*, Information Processing '77, B. Gilchrist, ed., North-Holland, Amsterdam, 1977.
- [Ris1] J. RISSANEN, *Independent components of relations*, ACM Trans. Database Systems, 2 (1977), pp. 317–325.
- [Ris2] ———, *Theory of relations for databases—a tutorial survey*, in Mathematical Foundations of Computer Science 1978, Springer-Verlag, New York, 1978, pp. 536–551.
- [Zan1] C. ZANIOLO, *Analysis and design of relational schemata for database systems*, TR UCLA-ENG-7669, Ph.D. Thesis, Computer Science Dept., University of California at Los Angeles, July 1976.
- [Zan2] ———, *Relational views in data base systems: support for queries*, Proc. COMPSAC 77, November 8–11, 1977, Chicago, IL.

AN EXTENSION OF STRASSEN'S DEGREE BOUND*

C. P. SCHNORR†

Abstract. With every set P_1, \dots, P_m of multivariate polynomials we associate in a natural way several algebraic varieties, i.e., irreducible Zariski-closed sets. The degree of each of these closed sets can be nicely bounded in terms of the number $L_{ns}(P_1, \dots, P_m)$ of nonscalar operations which are necessary to evaluate P_1, \dots, P_m . We establish lower bounds $L_{ns}(P) \cong \Omega(k \lg n)$ for single specific polynomials P of degree n , depending on $O(k)$ variables with 0, 1-coefficients. Typical examples are $L_{ns}(\sum_{i=1}^k x_i^n y^i) \cong \frac{1}{2}k \lg n$, $L_{ns}(\sum_{i=1}^k (x_1 + x_2 + \dots + x_i)^n y_i) > \frac{1}{2}k \lg n$, provided $k < n^{1/2}$. By our method and evaluating the degree of closed sets one obtains lower bounds $L_{ns}(P) > \Omega(k \lg n)$ for "almost all" polynomials P of degree n depending on k variables, $k \ll n$. These lower bounds hold for any field characteristic.

Key words. arithmetic complexity, Bezout's theorem, Strassen's degree bound, computationally hard polynomials

1. Introduction and preliminaries. It was Strassen who first recognized the importance of algebraic geometry for the complexity of polynomial evaluation. Strassen [10] proved that every set of multivariate polynomials which can be evaluated with $\leq v$ nonscalar operations defines an algebraic variety of degree $\leq 2^v$. This yields examples of sets of k polynomials of degree n depending on k variables that cannot be computed with less than $k \lg n$ nonscalar operations. We are now able to prove such $k \lg n$ -lower bounds for single polynomials of degree n depending on k variables. The lower bounds of this paper apply to specific polynomials with small integer coefficients and even 0, 1-coefficients, as well as to those with algebraic coefficients. Observe that specific polynomials that are hard to compute and which have algebraic or large integer coefficients are known from Strassen [11]; see also Schnorr [6], Schnorr and van de Wiele [8] and Heintz and Sieveking [33].

In order to make the paper understandable for readers without prior knowledge of algebraic geometry, the basic definitions and facts are collected in § 2. For convenience we shall work with affine varieties. Kendig [5] and Hartshorne [1] contain an introduction to affine and projective varieties.

A basic tool for the application of algebraic geometry to the complexity of polynomials is the Bezout inequality for the degree of the affine closed sets. See Heintz [2], Heintz and Sieveking [3] and Heintz and Schnorr [4] for further applications of the Bezout inequality. For completeness and for lack of a suitable reference we give a proof of the Bezout inequality in the Appendix. We reduce the Bezout inequality for affine closed sets to the well-known Bezout equality for projective varieties which is due to van de Waerden [12]. A complete proof for Bezout's inequality based on commutative algebra has been given by Heintz [2].

In § 3 we reprove Strassen's degree bound and extend it to computations with arbitrary rational operations. We give a direct proof based on Bezout's inequality, which unlike Strassen's original proof does not proceed by induction on the length of the computation. Observe that the elementary proof of Schönhage [9] only yields a weak version of Strassen's degree bound. Our main new results are contained in §§ 4 and 5. These results follow from the obvious extension of Strassen's degree bound in Theorem 1 without further application of algebraic geometry. We implicitly use a

* Received by the editors June 7, 1979, and in revised form January 15, 1980. The results of this paper were obtained in the spring of 1979 during a stay in Nice, supported by the DAAD.

† Fachbereich Mathematik, Universität Frankfurt, West Germany.

technique of Schnorr [6] for representing the coefficients of all polynomials that can be computed with $\leq v$ multiplications and divisions. This is the background of our key Lemma 1.

Throughout the paper \mathbb{K}_0 is an algebraically closed field of any characteristic; $x_1, \dots, x_n, y, y_1, \dots, y_m$ are indeterminates over \mathbb{K}_0 . $\mathbb{K}_0[x_1, \dots, x_n]$ is the ring of multivariate polynomials in the indeterminates x_1, \dots, x_n with coefficients in \mathbb{K}_0 . $\mathbb{K}_0(x_1, \dots, x_n)$ is the field of rational functions in the determinates x_1, \dots, x_n with coefficients in \mathbb{K}_0 . Tuples are underlined, e.g., $\underline{x} = (x_1, x_2, \dots, x_n)$. \mathbb{N} is the set of natural numbers, 0 included, and \mathbb{Z} is the set of integers. \lg is the logarithm to base 2. Given a field \mathbb{K} and a subset $F \subset \mathbb{K}$, an *arithmetical computation* β over F is a sequence R_1, \dots, R_v of elements in \mathbb{K} such that for $i = 1, \dots, v$ either (1) $R_i \in F$ or (2) $R_i = R_j \circ R_k$ with $j, k < i$ and $\circ \in \{+, -, *, /\}$. R_1, \dots, R_v are the *results* of β . We say “ β computes P_1, \dots, P_m ” if $\{P_1, \dots, P_m\} \subset \{R_1, \dots, R_v\}$. Particularly important is the case that $F = \mathbb{K}_0 \cup \{x_1, \dots, x_n\} \subset \mathbb{K} = \mathbb{K}_0(x_1, \dots, x_n)$ where $\mathbb{K}_0 \subset \mathbb{K}$ is a subfield of *constants*. In this case a “computation step” $R_i := R_j \circ R_k$ is called *nonscalar* provided (1) \circ is $*$ and $R_j, R_k \notin \mathbb{K}_0$ or (2) \circ is $/$ and $R_k \notin \mathbb{K}_0$. For $P_1, \dots, P_m \in \mathbb{K}_0(x_1, \dots, x_n)$ let $L_{ns}(P_1, \dots, P_m)$ be the minimal number of nonscalar steps in any computation of P_1, \dots, P_m over $\mathbb{K}_0 \cup \{x_1, \dots, x_n\}$.

2. Requisites from algebraic geometry. Given an algebraic closed field \mathbb{K}_0 , a subset $E \subset \mathbb{K}_0^n$ is called (*Zariski*) *closed* if it is definable as the set of common zeros of some set of polynomials $B \subset \mathbb{K}_0[x_1, \dots, x_n]$; i.e.,

$$E = \{a \in \mathbb{K}_0^n \mid P \in B : P(a) = 0\}.$$

E is a *hypersurface* or *hyperplane* if B consists of a single polynomial or single linear polynomial, respectively. Note that an arbitrary intersection and a finite union of closed sets is closed. These closed sets define the Zariski topology on \mathbb{K}_0^n . The *closure* \bar{A} of a set $A \subset \mathbb{K}_0^n$ is the intersection of all closed sets E that contain A , or equivalently, \bar{A} is the smallest closed set containing A .

A closed set E is called *irreducible* (E is then called an *affine variety*) if there do not exist closed sets E_1 and E_2 such that $E = E_1 \cup E_2$ and $E_1, E_2 \neq E$. The irreducible closed sets $E \subset \mathbb{K}_0^n$ are exactly those sets $E \subset \mathbb{K}_0^n$ which are definable as the sets of zeros of a prime ideal $P \subseteq \mathbb{K}_0[x_1, \dots, x_n]$. Each closed set is a finite union of irreducible closed sets. Such a representation of E as a finite union of irreducible closed sets is unique if it is not redundant. Therefore the irreducible closed sets appearing in this representation of E are called *components* of E . The *dimension* $\dim E$ of a closed set $E \subset \mathbb{K}_0^n, E \neq \emptyset$ is the maximal integer m such that there exist distinct irreducible closed sets Z_1, \dots, Z_m such that $\emptyset \neq Z_1 \subset \dots \subset Z_m \subset E$. We have $\dim \mathbb{K}_0^n = n$. The zero-dimensional subsets of \mathbb{K}_0^n are exactly the finite, nonempty sets. The dimension of a hypersurface $H \subset \mathbb{K}_0^n$ is $n - 1$. Our definition immediately implies the following fact.

FACT 1. *Let E, D be closed sets, E irreducible and $E \not\subset D$. Then $\dim(E \cap D) < \dim E$.*

The *degree* $\deg E$ of an irreducible closed set $E \subset \mathbb{K}_0^n$ is the maximal cardinality of a finite intersection $E \cap L$ with a linear affine subspace L :

$$\deg E := \max \{ \#(E \cap L) < \infty \mid L \subset \mathbb{K}_0^n \text{ linear affine subspace} \}.$$

Following Heintz [2] we extend this definition to reducible closed sets $E \subset \mathbb{K}_0^n$ as

$$\deg E := \sum_{C \text{ component of } E} \deg C.$$

Every closed set has finite degree. The degree of a linear affine subspace is 1. Let $P = \prod_{i=1}^r P_i^{v_i}$ be a polynomial with a pairwise distinct, irreducible factors P_i , and let H_P ,

H_{P_i} be the hypersurface defined by P, P_i . Then H_{P_i} is irreducible, $\deg H_{P_i} = \deg P_i$, $H_P = \bigcup_{i=1}^r H_{P_i}$ and we have $\deg H_P = \sum_{i=1}^r \deg H_{P_i} = \sum_{i=1}^r \deg P_i \leq \deg P$ with $\deg H_P = \deg P$ if and only if P is squarefree, i.e., $\nu_i = 1$ for $i = 1, \dots, r$.

The degree of $E \subset \mathbb{K}_0^n$ can be characterized in a particularly useful way if all components of E have the same dimension d (E is then called of *pure dimension* d). With $\underline{a} \in \mathbb{K}_0^{n+1}, (a_1, \dots, a_n) \neq 0^n$ we associate the hyperplane $H(\underline{a})$ defined by the equation $\sum_{i=1}^n a_i x_i = a_{n+1}$. Let $E \subset \mathbb{K}_0^n$ be of pure dimension d ; then "for almost all $\underline{a}^1, \dots, \underline{a}^d \in \mathbb{K}_0^{n+1} : \#(E \cap \bigcap_{i=1}^d H(\underline{a}^i)) = \deg E$ " holds in the following sense:

FACT 2. ([5], p. 196, Thm. 6.2). *Let $E \subset \mathbb{K}_0^n$ be of pure dimension d . Then $\{(\underline{a}^1, \dots, \underline{a}^d) \in \mathbb{K}_0^{d(n+1)} \mid \#(E \cap \bigcap_{i=1}^d H(\underline{a}^i)) \neq \deg E\}$ is contained in some proper closed subset of $\mathbb{K}_0^{d(n+1)}$.*

Our main tool in applying algebraic geometry to the complexity of polynomials is Bezout's inequality for the degree of affine closed sets. For completeness and for lack of a suitable reference we give a proof in the Appendix.

BEZOUT'S INEQUALITY. *Let $E, D \subset \mathbb{K}_0^n$ be closed sets. Then $\deg(E \cap D) \leq \deg E \cdot \deg D$.*

For instance, let the closed set E be defined by the polynomials P_1, \dots, P_m , i.e., $E = \bigcap_{i=1}^m H_{P_i}$. Then Bezout's inequality implies $\deg E \leq \prod_{i=1}^m \deg H_{P_i} \leq \prod_{i=1}^m \deg P_i$.

Let $E \subset \mathbb{K}_0^n$; then $U \subset E$ is called *open in E* if there is a closed $D \subset \mathbb{K}_0^n$ such that $U = E - D$. Next we define the *degree of sets U open in E* with $E \subset \mathbb{K}_0^n$ irreducible and $U \neq \emptyset$. Let $U = E - D$ with $D \subset \mathbb{K}_0^n$ closed; then by Fact 1 $\dim(E \cap D) < \dim E$ provided $U \neq E$. Hence Fact 2 implies that for almost all hyperplanes $H_1, \dots, H_d \subset \mathbb{K}_0^n, d := \dim E$ we have $\deg E = \#(E \cap H_1 \cap \dots \cap H_d)$ and $E \cap D \cap H_1 \cap \dots \cap H_d = \emptyset$. Hence there exists an integer k such that for almost all hyperplanes $H_1, \dots, H_d \subset \mathbb{K}_0^n: \#(U \cap H_1 \cap \dots \cap H_d) = k$. Moreover, $k = \max \{ \#(U \cap L) < \infty \mid L \subset \mathbb{K}_0^n \text{ linear affine subspace} \}$. We define $\deg U := k$. This clearly implies $\deg U = \deg E$ but the definition of $\deg U$ is independent from E .

In particular, this applies to the graph of a (partial) *rational map* $R : \mathbb{K}_0^n \rightarrow \mathbb{K}_0^m$; i.e., R is given as $R(\underline{a}) = (R_1(\underline{a}), \dots, R_m(\underline{a}))$ with rational functions $R_i \in \mathbb{K}_0(x_1, \dots, x_n), i = 1, \dots, m$ and $\text{graph } R = \{(\underline{a}, R(\underline{a})) \in \mathbb{K}_0^{n+m} \mid R(\underline{a}) \text{ defined}\}$.

FACT 3. *Let $R : \mathbb{K}_0^n \rightarrow \mathbb{K}_0^m$ be a rational map. Then $\text{graph } \overline{R} \subset \mathbb{K}_0^{n+m}$ is irreducible, $\dim \text{graph } \overline{R} = n$ and $\text{graph } R$ is open in $\text{graph } \overline{R}$.*

Proof. $\text{graph } R$ is open in $\text{graph } \overline{R}$: Let $R_i = S_i/T_i$ with $S_i, T_i \in \mathbb{K}_0[x_1, \dots, x_n], \text{gcd}(S_i, T_i) = 1, i = 1, \dots, m$ and let $x_1, \dots, x_n, z_1, \dots, z_m$ be the coordinates of \mathbb{K}_0^{n+m} . Then the equations

$$z_i T_i(x_1, \dots, x_n) = S_i(x_1, \dots, x_n), \quad i = 1, \dots, m$$

define a closed set $E \subset \mathbb{K}_0^{n+m}$ with $\text{graph } R \subset E$. Hence $\text{graph } \overline{R} \subset E$. Let $D \subset \mathbb{K}_0^{n+m}$ be the hypersurface associated with $\prod_{i=1}^m T_i$; then $E - D = \text{graph } R$ and therefore $\text{graph } \overline{R} - D = \text{graph } R$. This proves that $\text{graph } R$ is open in $\text{graph } \overline{R}$.

$\dim \text{graph } \overline{R} = n$ and $\text{graph } \overline{R}$ is irreducible since $\text{graph } R$ and \mathbb{K}_0^n are "birational equivalent" (compare [1, p. 24 ff]). This birational equivalence is given by the rational map R and the projection $\pi : \text{graph } R \rightarrow \mathbb{K}_0^n$. Clearly $\dim \mathbb{K}_0^n = n$ and \mathbb{K}_0^n is irreducible; moreover, dimension and irreducibility are invariants with respect to birational equivalence. \square

3. Strassen's degree bound. m rational functions $P_1, \dots, P_m \in \mathbb{K} := \mathbb{K}_0(x_1, \dots, x_n)$ give rise to a rational map $\varphi_P : \mathbb{K}_0^n \rightarrow \mathbb{K}_0^m$ as $\varphi_P(\underline{a}) = (P_1(\underline{a}), \dots, P_m(\underline{a}))$. It was Strassen [10] who first established a lower bound on the multiplicative complexity of $\{P_1, \dots, P_m\}$ in terms of the degree of the closure of the graph of the map φ_P :

STRASSEN'S DEGREE BOUND. Suppose $\{P_1, \dots, P_m\} \in \mathbb{K}$ can be computed over $\mathbb{K}_0 \cup \{x_1, \dots, x_n\}$ with v nonscalar steps. Then $\text{deg graph } \overline{\varphi_P} \leq 2^v$.

We know from Fact 3 that $\text{graph } \overline{\varphi_P}$ is open in $\text{graph } \varphi_P$ and $\text{graph } \overline{\varphi_P}$ is irreducible; hence $\text{deg graph } \overline{\varphi_P} = \text{deg graph } \varphi_P$. Therefore in Strassen's degree bound we can replace $\text{graph } \overline{\varphi_P}$ by $\text{graph } \varphi_P$. Observe that $\text{deg graph } \varphi_P$ is the maximal cardinality of a finite set which is obtained by intersecting $\text{graph } \varphi_P$ with a linear affine subspace. This shows that there is no further need of algebraic geometry in applying Strassen's degree bound.

We need a more general version of Strassen's degree bound. Instead of allowing only arithmetic operations in computations, we shall permit the application of arbitrary rational operations. Our main notion is the degree of such rational computations.

Let y_1, y_2, \dots, y_k be indeterminates over $\mathbb{K} := \mathbb{K}_0(x_1, \dots, x_n)$ and $F \subset \mathbb{K}$. A rational computation β over F is a sequence $R_1, \dots, R_v \in \mathbb{K}$ together with a sequence of rational operations $w_1, \dots, w_v \in \mathbb{K}_0(y_1, \dots, y_k)$ for some k such that for $i = 1, \dots, v$, $R_i = w_i(S_{i,1}, \dots, S_{i,k})$ with $\{S_{i,1}, \dots, S_{i,k}\} \subset \{R_1, \dots, R_{i-1}\} \cup F$. R_1, \dots, R_v are the results of β .

Let the degree of k -ary operation w be the degree of the closure of the graph of the rational map $w: \mathbb{K}_0^k \rightarrow \mathbb{K}_0$. In particular, if $w = \delta/\gamma$ with $\delta, \gamma \in \mathbb{K}_0[y_1, \dots, y_k]$, $\text{gcd}(\delta, \gamma) = 1$, then $\text{deg}(w) = \max\{\text{deg } \delta, 1 + \text{deg } \gamma\}$. For instance the degree of a nonscalar multiplication/division is 2, the degree of an addition or a scalar multiplication/division is 1. Then the degree of a rational computation β is by definition the product of the degrees of all operations β ; i.e., $\text{deg } \beta = \prod_{i=1}^v \text{deg } w_i$. For instance, the degree of a computation which only uses arithmetical operations $*, /, +, -$ is 2^u , provided that the number of nonscalar steps in β is u . We are now able to extend Strassen's degree bound in an obvious way.

THEOREM 1. Suppose the rational computation β computes $P_1, \dots, P_m \in \mathbb{K}_0(x_1, \dots, x_n)$ over $\{x_1, \dots, x_n\}$. Then $\text{deg graph } \overline{\varphi_P} \leq \text{deg } \beta$.

Proof. Let R_1, \dots, R_u be the results of β and let $R: \mathbb{K}_0^n \rightarrow \mathbb{K}_0^u$ be the rational map given by $R(\underline{a}) = (R_1(\underline{a}), \dots, R_u(\underline{a}))$. We know from Fact 3 that $\text{graph } R$ is open in $\text{graph } R$ and $\text{graph } \overline{\varphi_P}$ is open in $\text{graph } \varphi_P$. Moreover, there is a projection $\pi: \mathbb{K}_0^{n+u} \rightarrow \mathbb{K}_0^{n+m}$ such that $\text{graph } \overline{\varphi_P} = \pi \text{ graph } R$. Hence $\text{deg graph } \overline{\varphi_P} = \text{deg } \pi \text{ graph } R \leq \text{deg graph } R$, since π is linear. Therefore it will suffice to prove $\text{deg graph } R \leq \text{deg } \beta$.

We introduce indeterminates z_1, \dots, z_u and associate to each computation step of β a polynomial equation in the indeterminates $x_1, \dots, x_n, z_1, \dots, z_u$. Let the i th computation step be

$$R_i := w_i(R_{j_1}, \dots, R_{j_s}, x_{j_{s+1}}, \dots, x_{j_k}), \quad j_1, \dots, j_s < i, \quad 1 \leq j_{s+1}, \dots, j_k \leq n$$

with $w_i = \delta_i/\gamma_i$, $\delta_i, \gamma_i \in \mathbb{K}_0[y_1, \dots, y_k]$, $\text{gcd}(\delta_i, \gamma_i) = 1$.

Let $E \subset \mathbb{K}_0^{n+u}$ be the closed set which is defined by the equations

$$z_i \gamma_i(z_{j_1}, \dots, z_{j_s}, x_{j_{s+1}}, \dots, x_{j_k}) = \delta_i(z_{j_1}, \dots, z_{j_s}, x_{j_{s+1}}, \dots, x_{j_k})$$

for $i = 1, \dots, u$. Clearly $\text{graph } R \subset E$. We have $\overline{\text{graph } R} \subset E$, and since $\overline{\text{graph } R}$ is irreducible the decomposition $\overline{\text{graph } R} = \bigcup_{C \text{ comp. of } E} (\overline{\text{graph } R} \cap C)$ is trivial. Hence there is a component C of E with $\text{graph } R \subset C$. Let $R_i = S_i/T_i$ with $S_i, T_i \in \mathbb{K}_0[x_1, \dots, x_n]$, $\text{gcd}(S_i, T_i) = 1$ for $i = 1, \dots, u$, and let $D \subset \mathbb{K}_0^n$ be the hypersurface defined by $\prod_{i=1}^u T_i = 0$. We have $E - D \times \mathbb{K}_0^u = \text{graph } R = C - D \times \mathbb{K}_0^u$, which shows that $\text{graph } R$ is open in C . Therefore $\text{deg graph } R = \text{deg } C \leq \text{deg } E$ and by Bezout's inequality and the definition of E we have $\text{deg } E \leq \prod_{i=1}^u \max\{\text{deg } \delta_i, 1 + \text{deg } \varphi_i\} = \text{deg } \beta$. \square

4. A new, more powerful degree bound. Strassen's degree bound does not yield any nontrivial lower bound on the multiplicative complexity of single polynomials. Here we establish a more powerful degree bound which is nontrivial even for single polynomials and which contains Strassen's degree bound as a special case. Let $\mathbb{K} = \mathbb{K}_0(x_1, \dots, x_n)$ and $P = \{P_1, \dots, P_m\} \subset \mathbb{K}[y]$, $P_\nu = \sum_{i=0}^k P_{\nu,i} y^i$ with $P_{\nu,i} \in \mathbb{K}$. With any such P and a finite set $I \subset \{1, \dots, m\} \times \mathbb{N}$ we associate the rational map $\varphi_{P,I} : \mathbb{K}_0^n \rightarrow \mathbb{K}_0^{\#I}$ defined as $\varphi_{P,I}(a) = (P_{\nu,i}(a))_{(\nu,i) \in I}$. We abbreviate $I_+ := \{(\nu, i) \in I \mid i \neq 0\}$ and we establish a lower bound on the multiplicative complexity of P_1, \dots, P_m in terms of graph $\varphi_{P,I}$.

THEOREM 2. *Suppose $\{P_1, \dots, P_m\} \subset K[y]$ can be computed over $\mathbb{K}_0 \cup \{x_1, \dots, x_n, y\}$ with v nonscalar operations. Then, for all finite sets $I \subset \{1, \dots, m\} \times \mathbb{N}$, $\text{deg graph } \varphi_{P,I} \leq 2^v \prod_{(\nu,i) \in I_+} 2vi$.*

In the special case $I \subset \{1, \dots, m\} \times \{0\}$ and $P_\nu \in \mathbb{K}$ this means $\text{deg graph } \varphi_{P,I} \leq 2^v$, which is Strassen's degree bound.

Comments on the proof. Suppose $L_{ns}(\sum_{i=0}^k a_i y^i) = v$ with a_i in the field \mathbb{K} . Then, following Schnorr [6, Thm. 2.1], there exist polynomials $Q_i \in \mathbb{Z}[z_1, \dots, z_m]$, $i = 1, \dots, k$, $m = (v+1)(v+2)$ such that $\text{deg } Q_i \leq 2vi$ and $(a_1, \dots, a_k) \in \overline{\text{Im}(Q_1, \dots, Q_k)}$, where $\text{Im}(Q_1, \dots, Q_k)$ is the image of the map $(Q_1, \dots, Q_k) : \mathbb{K}^m \rightarrow \mathbb{K}^k$. Now suppose $(a_1, \dots, a_k) \in \text{Im}(Q_1, \dots, Q_k)$. Then $a_i = Q_i(\gamma_1, \dots, \gamma_m)$ for some $\gamma_i \in \mathbb{K}_0(x_1, \dots, x_n)$. A careful inspection of the proof of Schnorr [6, Thm. 2.1] shows that the γ_i can be chosen such that $\text{deg graph } (\gamma_1, \dots, \gamma_m) \leq 2^v$. Since the a_1, \dots, a_k are obtained by evaluating Q_1, \dots, Q_k at $\gamma_1, \dots, \gamma_m$ this yields $\text{deg graph } (a_1, \dots, a_k) \leq \text{deg graph } (\gamma_1, \dots, \gamma_m) \cdot \text{deg graph } (Q_1, \dots, Q_k) \leq 2^v \prod_{i=1}^k 2vi$, which is the core of the theorem. There are several difficulties in exploiting this proof idea. First, in general (a_1, \dots, a_k) is not in $\text{Im}(Q_1, \dots, Q_k)$ but only in the closure of the image of (Q_1, \dots, Q_k) . We will overcome this difficulty by means of an additional variable z . In order to obtain nice functions $\gamma_1, \dots, \gamma_m$ (these will be $x_1, \dots, x_n, R_{1,0}, \dots, R_{v,0}$ in the proof) the Q_1, \dots, Q_k will not be polynomials but rational functions. This is necessary in order to handle the operation of division. With division not allowed the proof would be somewhat easier. Lemma 1 in the proof rephrases the proof of the corresponding Schnorr [6, Thm. 2.1], adapting this proof to our special intentions.

Proof. Let β be an arithmetic computation for P with $v = L_{ns}(P)$ nonscalar arithmetic operations. In order to prove the theorem we introduce a new indeterminate z and rational functions $P_{\nu,i}^* \in \mathbb{K}_0(x_1, \dots, x_n, z)$, such that $P_{\nu,i} = P_{\nu,i}^*|_{z=0}$ and which can be computed by some rational computation β^* over $\{x_1, \dots, x_n, z\}$ with $\text{deg } \beta^* \leq 2^v \prod_{(\nu,i) \in I_+} 2vi$.

Let $\varphi_{P,I}^* : \mathbb{K}_0^{\#I} \rightarrow \mathbb{K}_0^{\#I}$ be the rational map defined as $\varphi_{P,I}^*(x_1, \dots, x_n, z) = (P_{\nu,i}^*(x, z))_{(\nu,i) \in I}$. Then Theorem 1 implies $\text{deg graph } \varphi_{P,I}^* \leq 2^v \prod_{(\nu,i) \in I_+} 2vi$. Since $\varphi_{P,I} = \varphi_{P,I}^*|_{z=0}$ this yields $\text{deg graph } \varphi_{P,I} \leq 2^v \prod_{(\nu,i) \in I_+} 2vi$.

We now construct the $P_{\nu,i}^*$ and β^* which are closely related to β . After collecting scalar steps, β can be written as a recursion scheme with parameters $a_{\mu,i}, b_{\mu,i}, c_{\nu,i} \in \mathbb{K}_0$ and linear polynomials $U_\mu, V_\mu, W_\nu \in \mathbb{K}_0[x_1, \dots, x_n]$:

$$\begin{aligned}
 R_0 &:= y \quad \text{for } \mu = 1, \dots, v; \\
 (1) \quad R_\mu &:= \left(U_\mu + \sum_{i=0}^{\mu-1} a_{\mu,i} R_i \right) * \left/ \left(V_\mu + \sum_{i=0}^{\mu-1} b_{\mu,i} R_i \right) \right. \quad \text{for } \nu = 1, \dots, m; \\
 P_\nu &= \left(W_\nu + \sum_{\mu=0}^v c_{\nu,\mu} R_\mu \right).
 \end{aligned}$$

R_1, \dots, R_v are the results of the nonscalar steps in β . There exist rational functions $R_{\mu,i} \in \mathbb{K}_0(x_1, \dots, x_n, z)$ such that for all but finitely many $\eta \in \mathbb{K}_0$

$$R_\mu = \sum_{i \geq 0} R_{\mu,i}(x_1, \dots, x_n, \eta)(y - \eta)^i, \quad \mu = 1, \dots, v.$$

In particular, we have $R_{0,0} \equiv \eta, R_{0,1} \equiv 1$ and $R_{0,i} \equiv 0$ for $i > 1$. We can now define the $P_{\nu,i}^*$ with the above-mentioned properties:

$$(2) \quad P_{\nu,i}^* = \begin{cases} \sum_{\mu=0}^v c_{\nu,\mu} R_{\mu,i}, & i \neq 0, \\ W_\nu + \sum_{\mu=0}^v c_{\nu,\mu} R_{\mu,0}, & i = 0. \end{cases}$$

By definition $P_{\nu,i} = P_{\nu,i}^*|z=0$ and we shall construct β^* .

As a first part of the computation β^* for the $P_{\nu,i}^*$ with $(\nu, i) \in I$, we compute $R_{\mu,0}, \mu = 1, \dots, v$ with v nonscalar operations over \mathbb{K}_0 :

$$(3) \quad R_{\mu,0} := \left(U_\mu + \sum_{i=1}^{\mu-1} a_{\mu,i} R_{i,0} \right) * / \left(V_\mu + \sum_{i=1}^{\mu-1} b_{\mu,i} R_{i,0} \right) \text{ for } \mu = 1, \dots, v.$$

Observe that there is not always a similar computation for $R_{\mu,0}|z=0, \mu = 1, \dots, v$ since some of the $R_{\mu,0}$ may be undefined at $z=0$. However $R_{\mu,0}$ always exists as a rational function in z and this is the reason for introducing z .

The remaining part of the computation β^* , which computes $P_{\nu,i}^*$ for $(\nu, i) \in I$ over $\mathbb{K}_0 \cup \{R_{1,0}, \dots, R_{v,0}, x_1, \dots, x_n\}$ and which has degree $\leq \prod_{(\nu,i) \in I} 2\nu i$, will be prepared by Lemma 1 below.

Using U_μ and V_μ as in the recursion (1), we abbreviate:

$$(4) \quad S_\mu := U_\mu + \sum_{i=1}^{\mu-1} a_{\mu,i} R_{i,0}, \quad T_\mu := V_\mu + \sum_{i=1}^{\mu-1} b_{\mu,i} R_{i,0}.$$

LEMMA 1. *There exist polynomials $Q_{\mu,i}$ with $v+n$ indeterminates and coefficients in \mathbb{K}_0 such that, for $\mu = 0, \dots, v$ and $i = 1, 2, \dots$,*

$$(a) \quad R_{\mu,i} = Q_{\mu,i}(R_{1,0}, \dots, R_{v,0}, x_1, \dots, x_n) / \prod_{j=1}^v (T_j)^i,$$

$$(b) \quad \deg Q_{\mu,i} \leq 2\nu i.$$

Proof. We first show that it is sufficient to construct polynomials $\bar{Q}_{\mu,i} \in \mathbb{K}_0[r_1, \dots, r_v, y_1, \dots, y_n, z_1, \dots, z_v]$ with $2v+n$ indeterminates and coefficients in \mathbb{K}_0 such that, for $\mu = 0, \dots, v$ and $i = 1, 2, \dots$,

$$(a^*) \quad R_{\mu,i} = \bar{Q}_{\mu,i}(R_{1,0}, \dots, R_{v,0}, x_1, \dots, x_n, (1/T_1), \dots, (1/T_v)),$$

$$(b^*) \quad \deg_{z_j} \bar{Q}_{\mu,i} \leq \begin{cases} i & \text{for } j \leq \mu, \\ 0 & \text{for } j > \mu. \end{cases}$$

Here \deg_* denotes the degree with respect to $r_1, \dots, r_v, y_1, \dots, y_n$. Since S_μ and T_μ are linear combinations of $R_{1,0}, \dots, R_{v,0}, x_1, \dots, x_n$ over \mathbb{K}_0 , there exist linear polynomials $\tilde{S}_\mu, \tilde{T}_\mu \in \mathbb{K}_0[r_1, \dots, r_v, y_1, \dots, y_n]$ such that

$$(5) \quad \begin{aligned} \tilde{S}_\mu(R_{1,0}, \dots, R_{v,0}, x_1, \dots, x_n) &= S_\mu, \\ \tilde{T}_\mu(R_{1,0}, \dots, R_{v,0}, x_1, \dots, x_n) &= T_\mu. \end{aligned}$$

Given the $\bar{Q}_{\mu,i}$ we define $Q_{\mu,i} \in \mathbb{K}_0[r_1, \dots, r_\nu, y_1, \dots, y_n]$ as

$$Q_{\mu,i} := \bar{Q}_{\mu,i}(r_1, \dots, r_\nu, y_1, \dots, y_n, (1/\tilde{T}_1), \dots, (1/\tilde{T}_\nu))^* \prod_{j=1}^{\nu} (\tilde{T}_j)^i.$$

Now (a*), (b*) imply (a), (b).

We define the $\bar{Q}_{\mu,i}$ by induction on μ following the recursion (1). The induction hypothesis holds for $\mu = 0$ and

$$\bar{Q}_{0,i} \equiv \begin{cases} 1, & i = 1, \\ 0, & i \neq 1. \end{cases}$$

For the induction step we distinguish whether the μ th nonscalar step in (1) is a multiplication or a division. In case of a multiplication we define the $\bar{Q}_{\mu,i}$ as

$$(6) \quad \sum_{i \geq 0} \bar{Q}_{\mu,i} y^i = \left(\tilde{S}_\mu + \sum_{\nu=0}^{\mu-1} a_{\mu,\nu} \sum_{j \geq 1} \bar{Q}_{\nu,j} y^j \right) * \left(\tilde{T}_\mu + \sum_{\nu=0}^{\mu-1} b_{\mu,\nu} \sum_{j \geq 1} \bar{Q}_{\nu,j} y^j \right).$$

By (1), (4), (5) and the induction hypothesis we conclude that (a*) holds for μ . It can easily be seen that (6) implies

$$\text{deg}_* \bar{Q}_{\mu,i} \leq \max \left\{ \begin{array}{l} \text{deg}_* \bar{Q}_{\nu,j} + \text{deg}_* \bar{Q}_{\bar{\nu},\bar{j}} \Big| \nu, \bar{\nu} = 0, \dots, \mu - 1 \\ \text{deg}_* \bar{Q}_{\nu,i} + 1 \qquad \qquad \qquad j + \bar{j} = i \end{array} \right\}.$$

Application of the induction hypothesis $\text{deg}_* \bar{Q}_{\nu,j} \leq \nu j$ for $\nu < \mu$ yields $\text{deg}_* \bar{Q}_{\mu,i} \leq (\mu - 1)i + 1 \leq \mu i$.

The induction hypothesis also implies

$$\text{deg}_{z_j} \bar{Q}_{\mu,i} \leq \begin{cases} i & \text{for } j < \mu, \\ 0 & \text{for } j \geq \mu. \end{cases}$$

In case of a division, (1) can be rewritten as

$$\begin{aligned} \sum_{i \geq 0} R_{\mu,i} y^i &= \left(S_\mu + \sum_{\nu=0}^{\mu-1} a_{\mu,\nu} \sum_{j \geq 1} R_{\nu,j} y^j \right) * (1/T_\mu) \\ &\quad * \sum_{\sigma \geq 0} \left(-(1/T_\mu) \sum_{\nu=0}^{\mu-1} b_{\mu,\nu} \sum_{j \geq 1} R_{\nu,j} y^j \right)^\sigma \\ &= \left(R_{\mu,0} + (1/T_\mu) \sum_{\nu=0}^{\mu-1} a_{\mu,\nu} \sum_{j \geq 1} R_{\nu,1} y^j \right), \\ &\quad * \sum_{\sigma \geq 0} \left(-(1/T_\mu) \sum_{\nu=0}^{\mu-1} b_{\mu,\nu} \sum_{j \geq 1} R_{\nu,j} y^j \right)^\sigma, \end{aligned}$$

since $R_{\mu,0} = S_\mu/T_\mu$. Thus we can define the $\bar{Q}_{\mu,i}$ by

$$\sum_{i \geq 0} \bar{Q}_{\mu,i} y^i = \left(r_\mu + z_\mu \sum_{\nu=0}^{\mu-1} a_{\mu,\nu} \sum_{j \geq 1} \bar{Q}_{\nu,j} y^j \right) * \sum_{\sigma \geq 0} \left(-z_\mu \sum_{\nu=0}^{\mu-1} b_{\mu,\nu} \sum_{j \geq 1} \bar{Q}_{\nu,j} y^j \right)^\sigma.$$

This implies

$$\deg_{\mathbb{K}^*} \bar{Q}_{\mu,i} \leq \max \left\{ 1 + \sum_k \deg_{\mathbb{K}^*} \bar{Q}_{\nu_k, j_k} \mid \begin{array}{l} \sum_k j_k = i \\ j_k \geq 1, 0 \leq \nu_k < \mu \end{array} \right\}.$$

Application of the induction hypothesis $\deg_{\mathbb{K}^*} \bar{Q}_{\nu,j} \leq \nu j$ for $\nu < \mu$ implies $\deg_{\mathbb{K}^*} \bar{Q}_{\mu,i} \leq 1 + (\mu - 1)i \leq \mu i$. It can also easily be seen that the induction hypothesis implies

$$\deg_{z_i} \bar{Q}_{\mu,i} \leq \begin{cases} i & \text{for } j < \mu, \\ 0 & \text{for } j \geq \mu. \end{cases}$$

This finishes the proof of Lemma 1. \square

Following the recursion (1) and by Lemma 1, we can now give a rational computation for $P_{\nu,i}^*$ with $(\nu, i) \in I$ over $\{R_{1,0}, \dots, R_{v,0}, x_1, \dots, x_n\}$ as follows:

$$(7) \quad T_j := V_j + \sum_{\mu=1}^{j-1} b_{j,\mu} R_{\mu,0} \quad \text{for } j = 1, \dots, v;$$

$$(8) \quad P_{\nu,i}^* := \left[\sum_{\mu=0}^v c_{\nu,\mu} Q_{\mu,i}(R_{1,0}, \dots, R_{v,0}, x_1, \dots, x_n) \right] / \prod_{j=1}^v (T_j)^i \quad \text{for } (\nu, i) \in I, \quad i \neq 0;$$

$$(9) \quad P_{\nu,0}^* := W_\nu + \sum_{\mu=0}^v c_{\nu,\mu} R_{\mu,0} \quad \text{for } (\nu, 0) \in I.$$

Each operation in (7) has degree 1, each instance of (8) represents an operation of degree $\leq 2\nu i$, and each instance of (9) represents an operation of degree 1. Therefore the computation (7), (8), (9) has degree $\leq \prod_{(\nu,i) \in I_+} 2\nu i$.

The whole computation β^* for $(P_{\nu,i}^* | (\nu, i) \in I)$ consisting of (3), (7), (8), (9) has degree $\leq 2^v \prod_{(\nu,i) \in I_+} 2\nu i$.

Therefore Theorem 1 implies that the degree of the rational map $\varphi_{P,I}^* : \mathbb{K}_0^{n+1} \rightarrow \mathbb{K}_0^{\#I}$ which is induced by $(P_{\nu,i}^* | (\nu, i) \in I)$, is bounded as $\deg \text{graph } \varphi_{P,I}^* \leq 2^n \prod_{(\nu,i) \in I_+} 2\nu i$. Since $\varphi_{P,I} = \varphi_{P,I}^* | z = 0$, it follows immediately that $\text{graph } \varphi_{P,I} = \text{graph } \varphi_{P,I}^* \cap H$, where H is the hyperplane defined by $z = 0$. This clearly implies $\deg \text{graph } \varphi_{P,I} \leq \deg \text{graph } \varphi_{P,I}^*$ which finally proves $\deg \text{graph } \varphi_{P,I} \leq 2^v \prod_{(\nu,i) \in I_+} 2\nu i$. \square

5. Applications of the new degree bound.

COROLLARY. *Let $P_i \in \mathbb{K}_0[x_i]$, $\deg P_i \geq n$ with distinct indeterminates $x_i, i = 1, \dots, k$. Then $L_{ns}(\sum_{i=1}^k P_i y^i) \geq \delta k \lg n$, provided $\delta \in \mathbb{Q}$ satisfies $k \leq (n^{1-\delta} / (2 \lg n))^{1/2}$.*

Proof. Let $\text{graph } P_i = \{(b, P_i(b)) | b \in \mathbb{K}_0\} \subset \mathbb{K}_0^2$. We prove $\deg \text{graph } P_i = \deg P_i$. Let z_1, z_2 be the coordinates of \mathbb{K}_0^2 and let $H_{c,d} \subset \mathbb{K}_0^2$ be the hyperplane defined by $cz_1 + dz_2 = 0$ for $c, d \in \mathbb{K}_0 - 0$. Then

$$H_{c,d} \cap \text{graph } P_i = \{(b, P_i(b)) | P_i(b) = cb/d\}.$$

b is a multiple zero of $P_i(x) - cx/d$ iff b is a common zero of $P_i(x) - cx/d$ and $P_i'(x) - c/d$ (here P_i' is the formal derivation of P_i). Clearly this is the case iff $P_i(b) = P_i'(b) \cdot b$. Since $P_i(x) - P_i'(x)x$ has only finitely many zeros, it follows that, for ‘‘almost all’’ $(c, d) \in \mathbb{K}_0^2$, $P_i(x) - cx/d$ has no multiple zeros; hence $\#(H_{c,d} \cap \text{graph } P_i) = \deg P_i$. This proves $\deg \text{graph } P_i = \deg P_i$.

Let $\varphi_P : \mathbb{K}_0^k \rightarrow \mathbb{K}_0^k$ be defined by $\varphi_P(x) = (P_1(x_1), \dots, P_k(x_k))$. Since $\text{graph } \varphi_P = \text{graph } P_1 \times \dots \times \text{graph } P_k$, we have $\deg \text{graph } \varphi_P = \prod_{i=1}^k \deg P_i \geq n^k$. On the other hand Theorem 2 implies

$$\deg \text{graph } \varphi_P \leq 2^v (2vk)^k \quad \text{for } v := L_{ns} \left(\sum_{i=1}^k P_i y^i \right).$$

Hence $n^k \leq 2^v (2vk)^k$, which yields $k \lg(n/2vk) \leq v$. Suppose $v < \delta k \lg n$; then by $k \leq (n^{1-\delta}/(2 \lg n))^{1/2}$ we have

$$2vk < 2\delta k^2 \lg n \leq \delta n^{1-\delta}.$$

Hence $v \geq k \lg(n/2vk) > k \lg(n^\delta/\delta) \geq \delta k \lg n$, which contradicts our assumption $v < \delta k \lg n$. This proves $v \geq \delta k \lg n$. Observe that the proof holds for any field characteristic. \square

In the special case $P_i = x_i^n$ the bound of Corollary 1 is sharp up to the factor δ , since $L_{ns}(\sum_{i=1}^k x_i^n y^i) = O(k \lg n)$. The restriction $k \leq (n^{1-\delta}/(2 \lg n))^{1/2}$ in the above lower bounds is rather strange, and it is an open problem whether this restriction can be removed.

COROLLARY 2. *Let $P_i \in \mathbb{K}_0[x_i]$, $\deg P_i = n$ for $i = 1, \dots, k$ and let the (k, k) -matrix $(c_{i,j})_{i,j \leq k}$ with $c_{i,j} \in \mathbb{K}_0$ be regular. Then every computation of $\sum_{i=1}^k \sum_{j=1}^k c_{i,j} P_i y^j$ over $\mathbb{K}_0 \cup \{x_1, \dots, x_k, y\}$ requires $\geq \delta k \lg n$ nonscalar steps, provided $k \leq (n^{1-\delta}/(2 \lg n))^{1/2}$.*

Proof. Let $\varphi_P: \mathbb{K}_0^k \rightarrow \mathbb{K}_0^k$ and $\varphi_R: \mathbb{K}_0^k \rightarrow \mathbb{K}_0^k$ be defined as $\varphi_P(\underline{a}) = (P_1(\underline{a}), \dots, P_k(\underline{a}))$, $\varphi_R(\underline{a}) = (\sum_{j=1}^k c_{i,j} P_j(\underline{a}) \mid i = 1, \dots, k)$. Since the matrix $(c_{i,j})_{i,j \leq k}$ is regular we have $\deg \text{graph } \varphi_R = \deg \text{graph } \varphi_P$. Therefore the assertion follows as in the proof of Corollary 1. \square

COROLLARY 3. *Let $(c_{i,j})_{i,j \leq k}$ be a regular (k, k) -matrix with $c_{i,j} \in \mathbb{K}_0$. Then $k \leq (n^{1-\delta}/(2 \lg n))^{1/2}$ implies $L_{ns}(\sum_{i=1}^k (\sum_{j=1}^k c_{i,j} x_j)^n y^i) \geq \delta k \lg n$.*

Proof. Let $z_i := \sum_{j=1}^k c_{i,j} x_j$. Since $(c_{i,j})_{i,j \leq k}$ is regular, z_1, \dots, z_k are indeterminates over \mathbb{K}_0 . But we already know from Corollary 1 that $L_{ns}(\sum_{i=1}^k z_i^n y^i) \geq \delta k \lg n$ with respect to computations over $\mathbb{K}_0 \cup \{z_1, \dots, z_k\}$. This clearly implies $L_{ns}(\sum_{i=1}^k z_i^n y^i) \geq \delta k \lg n$ with respect to computations over $\mathbb{K}_0 \cup \{x_1, \dots, x_k\}$. \square

Observe that our method distinguishes well between hard polynomials as above and extremely easy polynomials such as

$$(x_1 + \dots + x_k + y)^n = \sum_{i=0}^n \binom{n}{i} (x_1 + \dots + x_k)^{n-i} y^i$$

and

$$[1 - (x_1 + \dots + x_k)^{n+1} y^{n+1}] / [1 - (x_1 + \dots + x_k) y] = \sum_{i=0}^n (x_1 + \dots + x_k)^i y^i,$$

which can be computed with $O(\lg n)$ nonscalar steps. Indeed, let $\varphi_P: \mathbb{K}_0^k \rightarrow \mathbb{K}_0^n$ be defined as $\varphi_P(x_1, \dots, x_k) = ((\sum_{j=1}^k x_j)^i \mid i = 1, \dots, n)$, then $\deg \text{graph } \varphi_P = n$.

Theorem 2 can be extended in a straightforward way from one indeterminate y to several indeterminates y_1, \dots, y_k . Let $\underline{i} = (i_1, \dots, i_k) \in \mathbb{N}^k$ be a multiindex, $|\underline{i}| := \sum_{\nu=1}^k i_\nu$ and $y^{\underline{i}} := \prod_{\nu=1}^k y_\nu^{i_\nu}$. We consider a set

$$P = \{P_1, \dots, P_m\} \subset \mathbb{K}[y_1, \dots, y_m],$$

$$P_\nu = \sum_{\underline{i}} P_{\nu, \underline{i}} y^{\underline{i}} \quad \text{with } P_{\nu, \underline{i}} \in \mathbb{K}.$$

With each finite set $I \subset \{1, \dots, m\} \times \mathbb{N}^k$ we associate the rational map $\varphi_{P,I}: \mathbb{K}_0^n \rightarrow \mathbb{K}_0^{\#I}$ defined as

$$\varphi_{P,I}(x_1, \dots, x_n) = (P_{\nu, \underline{i}}(\underline{x}) \mid (\nu, \underline{i}) \in I).$$

Let $I_+ := \{(\nu, \underline{i}) \in I \mid |\underline{i}| \neq 0\}$. Then the obvious extension of the proof of Theorem 2 yields

THEOREM 3. *Suppose $P = \{P_1, \dots, P_m\} \subset \mathbb{K}[y_1, \dots, y_k]$ can be computed over $\mathbb{K}_0 \cup \{x_1, \dots, x_m, y_1, \dots, y_k\}$ with v nonscalar operations. Then for all finite sets $I \subset \{1, \dots, m\} \times \mathbb{N}^k$, $\deg \text{graph } \varphi_{P,I} \leq 2^v \prod_{(\nu, \underline{i}) \in I_+} 2v|\underline{i}|$.*

Also, the above corollaries can easily be extended to the case of several variables y_1, \dots, y_k . In this way one obtains lower bounds on $L_{ns}(P) \geq \delta k \lg n$ for single polynomials P with $2k$ variables and degree n , provided $k \leq n^{1-\delta}/(2 \lg n)$. As an example we extend Corollary 1.

COROLLARY 4. *Let $P_i \in \mathbb{K}_0[x_i]$, $\deg P_i \geq n$ for $i = 1, \dots, k$. Then every computation of $\sum_{i=1}^k P_i y_i$ over $\mathbb{K}_0 \cup \{x_1, \dots, x_k, y_1, \dots, y_k\}$ requires $\geq \delta k \lg n$ nonscalar steps provided $k \leq n^{1-\delta}/(2 \lg n)$.*

Proof. Let $\varphi_P: \mathbb{K}_0^k \rightarrow \mathbb{K}_0^k$ be defined as $\varphi_P(\underline{a}) = (P_1(\underline{a}), \dots, P_k(\underline{a}))$. Let $v = L_{ns}(\sum_{i=1}^k P_i y_i)$; then Theorem 3 implies $\deg \text{graph } \varphi_P \leq 2^v (2v)^k$. On the other hand, $\deg \text{graph } \varphi_P = n^k$. It follows that $k \lg(n/2v) \leq v$. Suppose $v < \delta k \lg n$; then $2v < 2\delta k \lg n \leq \delta n^{1-\delta}$. It follows that

$$v \geq k \lg(n/2v) > k \lg(n^\delta/\delta) \geq \delta k \lg n.$$

This contradicts our assumption that $v < \delta k \lg n$. Hence $v \geq k \lg n$. \square

It is perhaps an interesting observation that all our lower bounds still hold if multiplications by y, y_1, \dots, y_k are counted as scalar multiplications.

THEOREM 4. *Suppose $P = \{P_1, \dots, P_m\} \subset \mathbb{K}[y]$ can be computed over $\mathbb{K}_0 \cup \{x_1, \dots, x_m, y\}$ with v nonscalar operations without counting multiplications by y . Then for all finite sets $I \subset \{1, \dots, m\} \times \mathbb{N}$, $\deg \text{graph } \varphi_{P,I} \leq 2^v \prod_{(v,i) \in I} 2vi$.*

Proof. Consider the proof of Theorem 2. In case where the μ th nonscalar step in the computation β is a multiplication by y , then (6) looks like

$$\sum_{i \geq 0} \bar{Q}_{\mu,i} y^i = \left(\tilde{S}_\mu + \sum_{\nu=0}^{\mu-1} a_{\mu,\nu} \sum_{j \geq 1} \bar{Q}_{\nu,j} y^j \right) * y.$$

This clearly implies $\deg_* \bar{Q}_{\mu,i} \leq \max\{1, \deg_* \bar{Q}_{\nu,i-1} | \nu < \mu\}$ and $\deg_{z_i} \bar{Q}_{\mu,i} \leq \max\{\deg_{z_i} \bar{Q}_{\nu,i-1} | \nu < \mu\}$. Therefore Lemma 1 holds even if v does not contain the multiplications by y . Then, however, Theorem 4 follows in the same way as Theorem 2. \square

It is obvious by Theorem 4 that Theorem 3 and Corollaries 1–4 still hold if multiplications by y, y_1, \dots, y_k are considered as scalar steps.

We finally observe that by our method and by evaluating the degree of closed sets one obtains lower bounds $L_{ns}(P) = \Omega(k \lg n)$ for “almost all” polynomials P of degree n depending on $k+1$ variables, $k \ll n$. We identify $P \in \mathbb{K}_0[x_1, \dots, x_k, y]$ with its vector of coefficients in \mathbb{K}_0 . Then, for each $n \in \mathbb{N}$,

$$V_{n,k} := \left\{ \sum_{i=1}^k P_i y^i \mid P_i \in \mathbb{K}_0[x_1, \dots, x_k], \deg P_i \leq n, i = 1, \dots, k \right\}$$

is a closed set.

Let $\varphi_P: \mathbb{K}_0^k \rightarrow \mathbb{K}_0^k$ be defined by $\varphi_P(x_1, \dots, x_k) = (P_1(\underline{x}), \dots, P_k(\underline{x}))$. Then

$$V_{n,k}^* := \{P \in V_{n,k} \mid \deg \text{graph } \varphi_P < n^k\}$$

is contained in a proper closed subset of $V_{n,k}$. Hence $\deg \text{graph } \varphi_P = n^k$ for “almost all” $P \in V_{n,k}$. Using similar calculations as in the proof of Corollary 4 we obtain $L_{ns}(P) \leq 2^{-r-1} k \lg n$ for all $P \in V_{n,k} - V_{n,k}^*$ provided $k \leq n^{1-2^{-r}}/(2 \lg n)$ with $r \in \mathbb{N}$.

Appendix. A proof of Bezout’s inequality. We reduce Bezout’s inequality to the well-known Bezout equality for the degree of projective closed sets which was first proved by van der Waerden [12] (see also Kendig [5, Thm. 7.1, p. 207]. Heintz [2] has given a direct proof of Bezout’s inequality based on commutative algebra.

The projective n -space over \mathbb{K}_0 , denoted $\mathbb{P}^n(\mathbb{K}_0)$, is the set of equivalence classes (called projective points) of $(n + 1)$ -tuples $(a_0, \dots, a_n) \in \mathbb{K}_0^{n+1} - O^{n+1}$ under the equivalence relation $(a_0, \dots, a_n) \sim (\lambda a_0, \dots, \lambda a_n)$ for all $\lambda \in \mathbb{K}_0 - O$. Thus the equivalence class $[(a_0, \dots, a_n)]$ of (a_0, \dots, a_n) is the line in \mathbb{K}_0^{n+1} through (a_0, \dots, a_n) and the origin O^{n+1} . Since the zero sets of homogeneous polynomials $P \in \mathbb{K}_0[x_0, \dots, x_n]$ are closed with respect to this equivalence relation one defines $V \subset \mathbb{P}^n(\mathbb{K}_0)$ to be (Zariski) closed if V is the set of projective points which consist entirely of common zeros of some set of homogeneous polynomials $B \subset \mathbb{K}_0[x_0, \dots, x_n]$. Dimension, irreducibility and components of projective closed sets $V \subset \mathbb{P}^n(\mathbb{K}_0)$ are defined as in the affine case. A linear subspace $L \subset \mathbb{P}^n(\mathbb{K}_0)$ is a closed set which is defined by homogeneous linear equations. Then the degree of an irreducible closed set $V \subset \mathbb{P}^n(\mathbb{K}_0)$ is $\max \#(V \cap L)$ over all linear subspaces $L \subset \mathbb{P}^n(\mathbb{K}_0)$ such that $V \cap L$ is finite. For reducible closed sets $E \subset \mathbb{P}^n(\mathbb{K}_0)$ we set $\deg E = \sum_{C \text{ comp. of } E} \deg C$. Facts 1, 2 also hold for closed sets $E, D \subset \mathbb{P}^n(\mathbb{K}_0)$. Our remark on the degree of sets U, U open in E with E irreducible closed, applies to closed sets $E \subset \mathbb{P}^n(\mathbb{K}_0)$ as well; i.e., $\deg U = \deg E$ in this case. We use the embedding $\rho: \mathbb{K}_0^n \rightarrow \mathbb{P}^n(\mathbb{K}_0)$ defined as $\rho(a_1, \dots, a_n) = [(1, a_1, \dots, a_n)]$. $\rho(\mathbb{K}_0^n)$ is open in $\mathbb{P}^n(\mathbb{K}_0)$ since $\mathbb{P}^n(\mathbb{K}_0) - H_0 = \rho(\mathbb{K}_0^n)$ where H_0 is the hyperplane defined by $x_0 = O$. Also, for every closed set $E \subset \mathbb{K}_0^n$, $\rho(E)$ is open in $\overline{\rho(E)}$: Let $E \subset \mathbb{K}_0^n$ be defined by the polynomials $P_1, \dots, P_\nu \in \mathbb{K}_0[x_1, \dots, x_n]$; P_1, \dots, P_ν correspond to homogeneous polynomials $\tilde{P}_1, \dots, \tilde{P}_\nu \in \mathbb{K}_0[x_0, x_1, \dots, x_n]$ which define a closed set $\tilde{E} \subset \mathbb{P}^n(\mathbb{K}_0)$ with $\tilde{E} - H_0 = \rho(E)$. It can easily be seen that $E \subset \mathbb{K}_0^n$ is irreducible iff $\overline{\rho(E)}$ is irreducible. $L \subset \mathbb{K}_0^n$ is a linear affine subspace iff $\overline{\rho(L)} \subset \mathbb{P}^n(\mathbb{K}_0)$ is a linear projective subspace. This taken together implies $\deg E = \deg \rho(E) = \deg \overline{\rho(E)}$ for every closed $E \subset \mathbb{K}_0^n$. Therefore it remains to prove $\deg(E \cap D) \leq \deg E \cdot \deg D$ for closed sets $E, D \subset \mathbb{P}^n(\mathbb{K}_0)$. Moreover, it is obvious that we only need this statement for irreducible closed sets $E, D \subset \mathbb{P}^n(\mathbb{K}_0)$. We use the

BEZOUT EQUALITY. (Kendig [5, Thm. 7.1, p. 207], van der Waerden [12].) *Let $V, W \subset \mathbb{P}^n(\mathbb{K}_0)$ be irreducible and closed such that V and W intersect properly (which means $\dim(V \cap W) = \dim V + \dim W - n$). Then*

$$\sum_{C \text{ comp. of } V \cap W} i(V, W, C) \deg C = \deg V \cdot \deg W,$$

where $i(V, W; C) \in \mathbb{N} - 0$ is the "intersection multiplicity" of V and W along C .

Since $i(V, W; C) \geq 1$, Bezout's equality implies $\deg(V \cap W) \leq \deg V \cdot \deg W$, provided V and W intersect properly. This latter condition is needed in Bezout's equality for the definition of the intersection multiplicities, but it is unnecessary for the conclusion $\deg(V \cap W) \leq \deg V \cdot \deg W$. We finally show how to eliminate this condition.

For $A, B \subset \mathbb{P}^n(\mathbb{K}_0)$ closed we define

$$A \otimes B := \{[(a_0, \dots, a_n, b_0, \dots, b_n)] \mid [(a_0, \dots, a_n)] \in A, [(b_0, \dots, b_n)] \in B\}.$$

$A \otimes B \subset \mathbb{P}^{2n+1}(\mathbb{K}_0)$ is closed: it is defined by the polynomials that define A (working on the first $n + 1$ coordinates of $\mathbb{P}^{2n+1}(\mathbb{K}_0)$) and the polynomials that define B (working on the last $n + 1$ coordinates of $\mathbb{P}^{2n+1}(\mathbb{K}_0)$). We have $\dim(A \otimes B) = \dim A + \dim B + 1$. For our given irreducible closed sets $E, D \subset \mathbb{P}^n(\mathbb{K}_0)$, we set $\tilde{E} := E \otimes \mathbb{P}^n(\mathbb{K}_0)$, $\tilde{D} := \mathbb{P}^n(\mathbb{K}_0) \otimes D$, $\tilde{E}, \tilde{D} \subset \mathbb{P}^{2n+1}(\mathbb{K}_0)$. It can easily be seen that \tilde{E}, \tilde{D} are irreducible and that $\deg E = \deg \tilde{E}$, $\deg D = \deg \tilde{D}$. We have $\tilde{E} \cap \tilde{D} = E \otimes D$, $\dim(\tilde{E} \cap \tilde{D}) = \dim(E \otimes D) = \dim E + \dim D + 1 = \dim \tilde{E} + \dim \tilde{D} - (2n + 1)$, which shows that \tilde{E}, \tilde{D} intersect properly. Therefore Bezout's equality implies $\deg(E \otimes D) = \deg(\tilde{E} \cap \tilde{D}) \leq \deg \tilde{E} \cdot \deg \tilde{D} = \deg E \cdot \deg D$.

It remains to prove $\deg(E \cap D) \leq \deg(E \otimes D)$. Let $x_0, \dots, x_n, z_0, \dots, z_n$ be the coordinates of $\mathbb{P}^{2n+1}(\mathbb{K}_0)$ and let $H_i \subset \mathbb{P}^{2n+1}(\mathbb{K}_0)$ be the hyperplane defined by $x_i = z_i$. Let $\Delta: \mathbb{P}^n(\mathbb{K}_0) \rightarrow \mathbb{P}^{2n+1}(\mathbb{K}_0)$ be the linear map $\Delta[(a_0, \dots, a_n)] = [(a_0, \dots, a_n, a_0, \dots, a_n)]$. Clearly Δ yields a linear isomorphism $\Delta: E \cap D \rightarrow E \otimes D \cap \bigcap_{i=0}^n H_i$. Hence $\deg(E \cap D) = \deg(E \otimes D \cap \bigcap_{i=0}^n H_i)$. Now we can inductively apply Bezout's equality to the components V of $E \otimes D \cap \bigcap_{i \leq j} H_i$ and $W := H_{j+1}$, observe that V and H_{j+1} intersect properly provided $V \not\subset H_{j+1}$ (see, e.g., Kendig [5, p. 183 ff]). This inductive application of Bezout's equality yields

$$\deg\left(E \otimes D \cap \bigcap_{i=0}^n H_i\right) \leq \deg(E \otimes D).$$

Thus we have proved $\deg(E \cap D) \leq \deg(E \otimes D) \leq \deg E \cdot \deg D$ for any irreducible, closed sets $E, D \subset \mathbb{P}^n(\mathbb{K}_0)$, and this immediately implies the same assertion for any closed sets $E, D \subset \mathbb{P}^n(\mathbb{K}_0)$. \square

Acknowledgment. I am greatly indebted to Joos Heintz for thoroughly reading and criticizing the manuscript, and for indicating improvements both in style and in proofs.

REFERENCES

- [1] R. HARTSHORNE, *Algebraic Geometry*, Springer-Verlag, New York, 1977.
- [2] J. HEINTZ, *Definability bounds of first order theories of algebraically closed fields*, extended abstract in *Fundamentals of Computation Theory, FCT '79*, L. Budach, ed., Akademie-Verlag, Berlin, 1979, pp. 160–166; full version: preprint, Universität Frankfurt.
- [3] J. HEINTZ AND M. SIEVEKING, *Lower bounds for polynomials with algebraic coefficients*, preprint Universität Frankfurt, 1979; *Theoret. Comput. Sci.*, to appear.
- [4] J. HEINTZ AND C. P. SCHNORR, *Testing polynomials which are easy to compute*, Proc. of 12th ACM Symposium on Theory of Computing, Los Angeles 1980, pp. 262–272.
- [5] K. KENDIG, *Elementary Algebraic Geometry*, Springer-Verlag, New York, 1977.
- [6] C. P. SCHNORR, *Improved lower bounds on the number of multiplications/divisions which are necessary to evaluate polynomials*, *Theoret. Comput. Sci.*, 7 (1978), pp. 251–261.
- [7] ———, *An extension of Strassen's degree bound*, *Fundamentals of Computation Theory, FCT '79*, L. Budach, ed., Akademie-Verlag, Berlin, 1979, pp. 404–416.
- [8] C. P. SCHNORR AND J. P. VAN DE WIELE, *On the additive complexity of polynomials*, *Theoret. Comput. Sci.*, 10 (1980), pp. 1–18.
- [9] A. SCHÖNHAGE, *An elementary proof of Strassen's degree bound*, *Theoret. Comput. Sci.*, 3 (1976), pp. 276–272.
- [10] V. STRASSEN, *Die Berechnungskomplexität der elementarsymmetrischen Funktionen und von Interpolationskoeffizienten*, *Numer. Math.*, 20 (1972), pp. 238–251.
- [11] ———, *Polynomials with rational coefficients which are hard to compute*, this Journal, 3 (1974), pp. 128–149.
- [12] B. L. VAN DER WAERDEN, *Zur algebraischen Geometries XIV*, *Math. Anna.*, 115, 4 (1938), pp. 619–642.
- [13] ———, *Einführung in die algebraische Geometrie*. Zweite Auflage, Springer-Verlag, New York, 1973.

LANGUAGES SIMULTANEOUSLY COMPLETE FOR ONE-WAY AND TWO-WAY LOG-TAPE AUTOMATA*

J. HARTMANIS† AND S. MAHANEY†

Abstract. In this paper we study languages accepted by nondeterministic log n -tape automata that scan their input only once and relate their computational power to two-way log n -tape automata. We show that for the one-way log n -tape automata the nondeterministic model (1-NL) is computationally much more powerful than the deterministic model (1-L), that under one-way log n -tape reductions there exist natural complete languages for these automata and that the complete languages cannot be sparse. Furthermore, we show that any language complete for nondeterministic one-way log n -tape automata (under 1-L reductions) is also complete for the computationally more powerful nondeterministic two-way log n -tape automata (NL) under two-way log n -tape reductions. Therefore, for all bounds $T(n)$, $T(n) \geq \log n$, the deterministic and nondeterministic $T(n)$ -tape bounded computations collapse iff the nondeterministic one-way log n -tape computations can be carried out by two-way deterministic log n -tape automata.

Key words. deterministic languages, nondeterministic languages, log n -tape automata, one-way automata, two-way automata, complete languages, context-sensitive languages

1. Introduction. Work in computational complexity theory has been strongly influenced by the study of the families of languages accepted by Turing machines in polynomial tape, nondeterministic polynomial time, deterministic polynomial time, nondeterministic logarithmic tape and deterministic logarithmic tape, denoted by

PTAPE, NP, P, NL and L

respectively [1], [5].

Though the proper containment relations between these families of languages are not yet known, they form a natural hierarchy to classify the complexity of many practical computational problems by their membership in these families. Furthermore, each of these families contains many natural complete problems which, in a sense, characterize and represent their computational complexity. Currently, among the most challenging problems in the theory of computation is to establish the proper containment relations between these families of languages and to gain a better understanding of their structure.

In this paper we show that the above described "hierarchy" of families of languages, PTAPE, NP, P, NL and L, can be naturally extended downward to one-way log n -tape deterministic and nondeterministic Turing machines. These are (nondeterministic) two-tape Turing machines with a one-way input tape (with end marker) and for an input of length n a two-way, read-write work tape of length $\lceil \log_2 n \rceil$ (with end markers). We denote the families of languages accepted by the nondeterministic and deterministic models by 1-NL and 1-L, respectively.

The results in this paper show that these automata exhibit some very interesting properties and that they capture our intuitive idea of what can be achieved *by guessing and polynomial bounded counting in one scan of the input*.

Their importance is further enhanced by their relations to nondeterministic two-way log n -tape automata. We prove that every complete 1-NL language under one-way log n -tape reductions, is also complete for NL languages under two-way log n -tape reductions. Thus, these two different families of languages share many

* Received by the editors March 5, 1980, and in final form June 24, 1980. This research was supported in part by the National Science Foundation under grant MSC 78-00418.

† Department of Computer Science, Cornell University, Ithaca, New York 14853.

complete sets. From this follows that for all tape bounds $T(n)$, $T(n) \geq \log n$, the deterministic and nondeterministic $T(n)$ -tape bounded computations collapse iff the nondeterministic one-way $\log n$ -tape computations can be performed by two-way deterministic $\log n$ -tape machines. Thus, we see that the basic question about eliminating nondeterminism in tape-bounded computations is equivalent to the question whether we can replace nondeterminism in one-way $\log n$ -tape computations by deterministic two-way computations.

Finally, we show that if $NL \neq L$ then there exist natural incomplete languages in $1-NL - 1-L$. Recall that the incomplete languages in $NP - P$, under the assumption that $P \neq NP$, are obtained by diagonalization and that no natural incomplete languages are known for NP [9].

2. 1-NL languages. First we observe that, though we do not yet know whether

$$P \neq NP \quad \text{and} \quad L \neq NL,$$

we can easily show that for one-way $\log n$ -tape computations the nondeterministic computations are more powerful than the deterministic ones; i.e.,

$$1-L \subsetneq 1-NL.$$

Actually, we prove that the gap between deterministic and nondeterministic one-way $\log n$ -tape computations is exponential. This shows that one-way computations with small amounts of work tape behave differently from two-way tape-bounded computations (with $\log n$ or more tape), which going from nondeterministic to deterministic computations need no more than to square the amount of tape used [11].

Let $1\text{-TAPE}[T(n)]$ denote the family of languages accepted by deterministic one-way Turing machines with $T(n)$ work tape.

THEOREM 1. *If $T(n)$ is such that*

$$\lim_{n \rightarrow \infty} \frac{T(n)}{n} = 0$$

then

$$1-NL \not\subseteq 1\text{-TAPE}[T(n)].$$

Proof. The language

$$A = \{u \# v \mid u, v \in \Sigma^*, \# \notin \Sigma \text{ and } u \neq v\}$$

is in $1-NL$ but not in $1\text{-TAPE}[T(n)]$, if the above limit condition holds for $T(n)$. To see this, note that A can be recognized by a nondeterministic, one-way $\log n$ -tape machine which guesses whether $|u| \neq |v|$ or $|u| = |v|$ and $u \neq v$; in the first case the guess is easily verified on $\log n$ -tape, in the second case the machine guesses in which digit the two strings differ and then counts up to the position in u and v , respectively, using its work tape to verify its guess.

On the other hand, if

$$\lim_{n \rightarrow \infty} \frac{T(n)}{n} = 0,$$

then for any $q, q > 0$, and sufficiently large n ,

$$q^{T(n)} < 2^n.$$

Therefore for some sufficiently long sequences u_1 and u_2 , $u_1 \neq u_2$, the machine must be

in the same total configuration after scanning

$$u_1 \# \quad \text{and} \quad u_2 \#.$$

But then

$$u_1 \# u_1 \quad \text{and} \quad u_2 \# u_1$$

are either both accepted or both rejected, showing that A is not in $1\text{-TAPE}[T(n)]$. \square

Next we show that there exist natural complete problems for the 1-NL family of languages. To do this we use deterministic one-way log n -tape reductions that were introduced and studied in [6]. A 1-L *transducer* is a deterministic one-way log n -tape machine with a one-way output tape. From Theorem 1 we know that the 1-L transducers are not computationally strong enough to recognize the languages in 1-NL.

Recall that it is not yet known whether $P \neq NP$ and/or $L \neq NL$, and therefore we do not know whether the polynomial time reductions used to study NP problems are not sufficiently powerful computationally to recognize directly all the languages in NP; the related question about two-way log n -tape reductions also remains open. On the other hand, from [6] we know that the well-known “classic” complete problems for PTAPE, NP, P, NL and L, respectively, all remain complete for these families under one-way log n -tape reductions and that these transducers are not capable of recognizing the languages in L. At the same time, it was also shown that there are complete problems, say, in NP under polynomial time reductions, which are not complete under one-way log n -tape reductions [6].

We recall that there is a nice hierarchy of “graph connectivity problems” which form complete sets for PTAPE, P, NL and L. The complete problem that we will define below for 1-NL fits in naturally in this hierarchy, extending it downward. For the sake of comparison we describe the other complete “graph connectivity” problems.

A complete language for PTAPE is formed by all directed graphs with an IN and OUT node for which there exists a winning strategy for the first player of the game of hex on this graph [1].

A complete language for P is formed by all directed graphs whose nodes are labeled with either AND or OR and are such that the IN and OUT nodes are connected by a path system which must take all possible edges out of an AND node and some edge out of an OR node.

A complete language for NL is formed by the set of all directed graphs with an IN and OUT node, which have a directed path from the IN to the OUT node [7].

A complete language for L is formed by all directed graphs of outdegree one for which there is a path from the IN to the OUT node [6], [7].

A directed graph is represented by a sequence of pairs of nodes (a, b) indicating that there is an edge leading from node a to node b . We will say that the representation of a directed, acyclic graph, G is *topologically sorted* if for any pair of edges (a, b) and (b, c) in G , edge (a, b) is listed before (b, c) .

Next we show that the problem of determining whether the topologically sorted representations of acyclic graphs have a directed path from the IN to an OUT node is a complete 1-NL problem. Denote the set of these graph representations by TAGAP. It is seen that TAGAP fits in naturally in the hierarchy of the other complete “graph connectivity problems” for PTAPE, P, NL and L, where the acyclic nature of the graphs directly mirrors the limited ability of reading the input only once. Similarly, the difference between the complete graph problems for NL and L is captured in the difference between the outdegree of the graphs: for NL we can use graphs with outdegree $k \geq 1$, whereas for L we are restricted to outdegree one.

THEOREM 2. *The set of topologically sorted representations of directed, acyclic graphs with a directed path from the IN to an OUT node is a complete language for 1-NL.*

Proof. TAGAP is contained in 1-NL since a one-way nondeterministic device can successively guess on its $\log n$ tape the sequence of nodes which form a path from IN to OUT. Since the representation of the graph is topologically sorted, the correctness of these guesses can be verified in one scan of the input.

To see that TAGAP is complete for 1-NL, we show that for every one-way $\log n$ -tape nondeterministic machine M , we can reduce every input string w (by a 1-L transducer) to an acyclic graph in which IN is connected to OUT iff w is accepted by M . The nodes of the graph G_w , corresponding to the input w , consists of triplets

$$(n_1, n_2, x),$$

where n_1 , $1 \leq n_1 \leq |w| = n$, indicates the head position of M on the input tape, n_2 , $0 \leq n_2 \leq q^n$, indicates the number of operations performed since the last head move on the input and x describes the configuration of M (input symbol scanned, state, worktape content, and head position on work tape).

For any fixed M we can construct a one-way deterministic $\log n$ -tape transducer which for input w enumerates in topological order on its output tape all pairs of nodes

$$((n_1, n_2, x), (n'_1, n'_2, x'))$$

such that in one move the machine M goes from the total configuration described by (n_1, n_2, x) to the one described by (n'_1, n'_2, x') . If the input head is moved, then

$$n'_1 = n_1 + 1 \quad \text{and} \quad n'_2 = 0;$$

otherwise, if the machine performs an operation and $n_2 < q^n$,

$$n'_1 = n_1 \quad \text{and} \quad n'_2 = n_2 + 1.$$

We denote the pair

$$((n_1, n_2, x), (n'_1, n'_2, x')) \quad \text{by} \quad ((n_1, n_2, x), \text{Succ}(n_1, n_2, x)).$$

The enumeration is described by the following program:

```

begin
  For each  $n_1 = 1, 2, \dots, |w|$  do
    begin
      For each  $n_2 = 0, 1, 2, \dots, q^{|w|}$  do
        begin
          For each  $x, |x| \leq [\log |w|]$  do
            begin
              print all pairs  $((n_1, n_2, x), \text{Succ}(n_1, n_2, x))$ 
            end
          end
        end
      end
    end
  end

```

The resulting graph is topologically sorted and the IN node, the initial configuration, is connected by a directed path to an OUT node, a node whose configuration contains the halt state, iff w is accepted by M .

Thus we see that TAGAP is complete for 1-NL. \square

3. Relations between one-way and two-way computations. We know that the nondeterministic two-way $\log n$ -tape computations are computationally much more

powerful than the nondeterministic one-way log n -tape computations; i.e., $1\text{-NL} \neq \text{NL}$. Nevertheless, we will show in this section that there are sets which are simultaneously complete for both families of languages. More precisely, we know that TAGAP is complete for 1-NL under one-way log n -tape reductions and we prove below that TAGAP is also complete for NL under two-way log n -tape reductions. Later we will see that this result has some very interesting implications.

THEOREM 3. TAGAP is a complete language in NL under two-way log n -tape reductions.

Proof. Since TAGAP is in 1-NL it is also contained in NL. To show that TAGAP is complete for NL, let A be a language in NL. We know that A is accepted by a nondeterministic two-way log n -tape automaton M . This automaton can be easily converted to an oblivious automaton, M' , that moves its input head on complete sweeps from (left) end marker to (right) end marker (and back). In other words, the head movements on the input tape are oblivious to the input and machine configuration. Furthermore, for any accepting computation of M' there is an accepting computation which does not repeat any of the log n long work tape patterns, and therefore we know that for M' there exists a k such that if any input w is accepted then w is accepted by M' within $k + n^k$ operations, $n = |w|$.

We will now show that a deterministic two-way log n -tape transducer can reduce any set A in NL, using an oblivious acceptor M' of A to topologically sorted representations of acyclic graphs, G_w , such that an OUT node of G_w can be reached from the IN node iff w is in A .

For input w , $|w| = n$, the graph G_w has nodes of the form

$$(n_1, n_2, n_3, x),$$

where $n_1, 1 \leq n_1 \leq k + n^k$, counts the sweeps on input w ;

$n_2, 1 \leq n_2 \leq n$, indicates the head position on the input tape;

$n_3, 0 \leq n_3 \leq k + n^k$, counts the operations performed since the last head motion on the input tape; and

$x, x \in \Gamma^*, |x| \leq \log n$, represents a configuration of M' .

The graph G_w has an edge of the form

$$((n_1, n_2, n_3, x), (n'_1, n'_2, n'_3, x))$$

iff in one possible operation M' transforms the state described by (n_1, n_2, n_3, x) to (n'_1, n'_2, n'_3, x') . The IN node is again the initial configuration of M' on w and the OUT nodes are the nodes with an accepting state in x .

For input w the two-way log n -tape transducer enumerates all the nodes of G_w in sequence for sweeps, head position, how long it has computed without moving the head and the machine configuration, and lists the possible edges of G_w if there is a one-step transition

$$((n_1, n_2, n_3, x), (n'_1, n'_2, n'_3, x')).$$

This can be done by an algorithm similar to the one used in the proof of Theorem 2.

It is easily seen that G_w has the desired property that IN is connected to an OUT iff $w \in A$ and that G_w is an acyclic graph, printed in a topologically sorted form. Thus TAGAP is complete for NL, as was to be shown. \square

From the previous result we can see that for all tape bounds $T(n)$, $T(n) \geq \log n$, the deterministic and nondeterministic $T(n)$ -tape bounded computations collapse, i.e.,

$$\text{TAPE}[T(n)] = \text{NTAPE}[T(n)],$$

if and only if nondeterminism in one-way log n -tape computations can be eliminated by using deterministic two-way computations.

COROLLARY 4. $1\text{-NL} \subseteq L$ iff $\text{NL} = L$.

Proof. $\text{NL} = L$ implies that $1\text{-NL} \subseteq L$. Conversely, $1\text{-NL} \subseteq L$ implies that TAGAP is in L . Since TAGAP is complete for NL we see that $L = \text{NL}$. \square

The above corollary puts a premium on understanding the power of two-way computations and their relation to one-way nondeterministic computations. It is interesting to recall that the related question about one-way nondeterministic push-down automata and two-way deterministic pushdown automata also remains unsolved: we know that deterministic two-way pushdown automata can accept languages which are not context-free, but we do not know whether they can accept all context-free languages.

From the study of NP-complete problems we know that if $P \neq \text{NP}$ then there exist incomplete languages in $\text{NP} - P$ [9]. On the other hand, the known incomplete languages are constructed by delayed diagonalization, and so far no natural languages are known to be incomplete.

For 1-NL languages the situation is quite different. Let A be given by

$$A = \{u \# v \mid u, v \in \Sigma^*, \# \notin \Sigma \text{ and } u \neq v\}.$$

THEOREM 5. *If $L \neq \text{NL}$, then the language A is not complete for 1-NL and is in 1-NL but not in 1-L .*

Proof. The language A is seen to be in 1-NL and also in L but, by Theorem 1, A is not in 1-L . If A is complete for 1-NL then, by Theorems 2 and 3, it is also complete for NL ; but then $\text{NL} = L$. \square

4. Nonexistence of sparse complete sets. In this section we show that there cannot exist sparse complete sets for 1-NL , and compare this result with the not yet completely resolved question for NP-complete languages. In the last section we discuss the corresponding problem for complete NL problems under two-way log n -tape reductions.

A set B , $B \subseteq \Sigma^*$, is said to be *sparse* iff there exists a k such that for all n

$$|B \cap \Sigma^n| \leq k + n^k.$$

It has been shown [2], [5] that the *known* complete sets in NP (and similarly in PTAPE) are isomorphic under polynomial time mappings, and therefore the *known* complete languages are similar in a very strong technical sense. The existence of sparse complete languages for NP would prove that not all complete languages for NP are polynomial time isomorphic, and therefore there would exist (as yet undiscovered) radically different types of complete languages. Furthermore, it would prove that the necessary information to solve NP-complete problems can be condensed into a sparse oracle tape (which could be computed once up to sufficiently long strips and then used to solve NP problems in deterministic polynomial time) [2], [5].

At present we do not know if there exist sparse complete sets in NP under polynomial time reductions. Recently it has been shown that, if a language A over a single letter alphabet $A \subseteq a^*$ is complete for NP, then $P = \text{NP}$ [3]. Similarly, if there exists a sparse complete language for co-NP, then $P = \text{NP}$ [4]. Furthermore, the existence of a sparse complete language in PTAPE implies that $P = \text{PTAPE}$ and therefore $P = \text{NP} = \text{PTAPE}$ [10]. Unfortunately, the main problem, whether there can exist sparse NP-complete sets under polynomial time reductions without forcing $P = \text{NP}$, remains unsolved.

The situation is different for one-way log n -tape reductions, for which we show next that there cannot exist sparse complete sets in 1-NL , L , NL , P , NP and PTAPE.

THEOREM 6. *There are no sparse complete sets in 1-NL, L, NL, P, NP and PTAPE under 1-L reductions.*

Proof. We will prove that there exist sets in 1-NL which cannot be reduced to any sparse sets by 1-L reductions. Suppose, to the contrary, that TAGAP is reduced to a sparse set A .

For a 1-L transducer let CONFIG be the total configuration consisting of the input tape square scanned, the machine state, work tape content and head position, and output tape content.

Consider as inputs topologically sorted representations of dags which consist of nodes labeled IN, 1, 2, \dots , n , and OUT. The edges will be (IN, i) for all i in some sets and (j, OUT) for some j .

After the prefix $T = \{(IN, i) : i \in REACH\}$ is read, the set REACH is exactly the set of nodes that can connect IN to OUT. If $REACH \neq \emptyset$, then a suitable choice of (j, OUT) can put the graph into TAGAP. Thus, after reading a string in prefix T , the 1-L reducer for TAGAP to A must print a string from prefixes of A on its output tape.

Since A and PREFIX(A) are sparse sets, we see that any time during the reduction of a directed tree of size n the reducer is in one of polynomially many different CONFIG (including the content of the output tape). On the other hand, there are exponentially many graphs with descriptions of length n and different REACH sets. Therefore, there exist two graphs T_1 and T_2 , with $REACH[T_1] \neq REACH[T_2]$, which are mapped by the 1-L transducer onto the same total configuration. But then, by adjoining the same appropriate edge to both of them, connecting some j to the OUT node in T_1 and T_2 , we place T_1 in TAGAP and T_2 not in TAGAP. Since the 1-L transducer maps T_1 and T_2 onto the same element we see that TAGAP cannot be reduced to a sparse set. \square

5. Two-way reductions and an open problem. It is interesting to note that for 1-L reductions we cannot have sparse complete sets in 1-NL, N, NL, etc., nor can we have a complete set A over a single letter, $A \subseteq a^*$ for NP under polynomial reductions, if $NP \neq P$. On the other hand, so far we have not been able to show that the existence of a complete set over a single letter alphabet for NL would imply that

$$L = NL.$$

The assumption that there exists a complete set A , $A \subseteq a^*$, for NL leads to some strange implications.

We know that if $L = NL$ then the deterministic and nondeterministic computations for larger amounts of tape are also equal. In particular, $NL = L$ implies that the deterministic and nondeterministic context-sensitive languages are the same, DCSL = NDCSL. On the other hand, we get the following result.

THEOREM 7. *If A , $A \subseteq a^*$, is complete for NL, then DCSL = NDCSL implies that $L = NL$.*

Proof. Let M be a nondeterministic log n -tape acceptor for the complete language A , $A \subseteq a^*$. Then we can convert M to an equivalent nondeterministic log n -tape automaton, which for any input a^n first deterministically represents n in binary on its $\lceil \log n \rceil$ work tape and then, using only the available work tape, simulates M on a^n . Clearly, the last operation can be performed by a nondeterministic linearly bounded automaton and, therefore, because we assume that DCSL = NDCSL, by a deterministic lba working on the $\lceil \log n \rceil$ work tape. Thus, A is in L and since A is complete for NL we have $NL = L$, as was to be shown. \square

Furthermore, if A , $A \subseteq a^*$, is complete for NL, then every language in NL can be recognized by a log n -tape automaton which uses its nondeterminism only after it has scanned the input tape.

We refer to a two-way $\log n$ -tape automaton which operates deterministically while it scans its input tape and only uses nondeterministic operations (on its $\log n$ work tape) after the input has been completely scanned, as a *restricted* nondeterministic, $\log n$ -tape automaton, and designate the family of languages accepted by these automata by RNL.

THEOREM 8. *There exists a complete NL language A , $A \subseteq a^*$, iff $\text{RNL} = \text{NL}$.*

Proof. If $\text{RNL} = \text{NL}$ then we can construct from any complete language A for NL a complete language A' for NL such that $A' \subseteq a^*$. To do this, let M be the restricted recognizer for A . For each input w let w' be the content of the work tape after M has finished scanning the input w in a deterministic mode. Let the complete set A' , $A' \subseteq a^*$, be the set obtained from A by expressing in unary form all the deterministically computed work tape contents, w' , which we can view as binary representation of integers.

Clearly, there is an L reducer of A to A' , and since A is a complete set for NL, it is easily seen that A' , $A' \subseteq a^*$, is a complete set for NL.

Conversely, let A , $A \subseteq a^*$, be a complete set for NL. Then any other set B in NL can be reduced to A and the resulting element a^k can be represented on the transducer's $\log n$ tape as a binary number. After that we simulate on a separate track of the $\log n$ tape the acceptor of A (the input a^k is simulated by counting up to k) and accepts iff the simulated acceptor accepts. Since the above device uses nondeterminism only after it has finished scanning the input (computing a^k), we see that any set in NL is accepted by a restricted machine. Thus

$$\text{RNL} = \text{NL},$$

as was to be shown. \square

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] L. BERMAN AND J. HARTMANIS, *On isomorphisms and density of NP and other complete sets*, this Journal, 6 (1977), pp. 305–323.
- [3] P. BERMAN, *Relationships between density and deterministic complexity on NP complete languages*, Fifth Int. Symp. on Automata, Languages, and Programming, Udine, Italy; Lecture Notes in Computer Science 62, Springer-Verlag, New York, 1978, pp. 63–71.
- [4] S. FORTUNE, *A note on sparse complete sets*, this Journal, 8 (1979), pp. 431–433.
- [5] J. HARTMANIS, *Feasible Computations and Provable Complexity Properties*, CBMS–NSF Regional Conference Series in Applied Mathematics, Vol. 30, Society for Industrial and Applied Mathematics, Philadelphia, 1978.
- [6] J. HARTMANIS, N. IMMERMANN AND S. MAHANEY, *One-way log-tape reductions*, 19th Annual IEEE Symposium on Foundations of Computer Science, October 16–18, 1978, pp. 65–72.
- [7] N. D. JONES, *Space-bounded reducibility among combinatorial problems*, J. Comput. System Sci., 11 (1975), pp. 68–85.
- [8] N. D. JONES AND W. T. LAASER, *Problems complete for deterministic polynomial time*, Theoret. Comput. Sci., 3 (1977), pp. 105–117.
- [9] R. E. LADNER, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach., 22 (1975), pp. 155–171.
- [10] A. R. MEYER AND M. S. PATERSON, *With what frequency are apparently intractable problems difficult?*, to be published.
- [11] W. J. SAVITCH, *Relations between nondeterministic and deterministic tape complexities*, J. Comput. Systems Sci., 4 (1970), pp. 177–192.

RECURSIVE GRAPHS, RECURSIVE LABELINGS AND SHORTEST PATHS*

ANDRZEJ PROSKUROWSKI†

Abstract. We consider classes of undirected, not weighted graphs which have recursive representations; these include trees, maximal outplanar graphs, k -trees, chordal graphs, and minimally two-connected graphs. We investigate invariants of recursive labelings of some of these graphs. One consequence of the existence of such invariant relation is that we can describe a single-source, shortest-paths spanning tree in terms of the recursive representation. We also discuss reasons why we cannot do this as well for other types of recursive graphs.

Key words. recursive graphs, chordal graphs, data structures, shortest path

Introduction. We are interested in graphs which may be obtained in a recursive construction process [9]: trees, maximal outplanar graphs, k -trees, chordal graphs and minimally two-connected graphs are examples of such graphs [7]. Very often combinatorial problems “hard” for general graphs are efficiently solvable for some of the above classes [1], [11], [13]. The resulting efficient algorithms frequently use a particular data structure—recursive representation of the graphs—implied by the recursive construction process for the given class of graphs. This process involves a labeling of the graph’s vertices (“recursive labeling”) which is not necessarily unique. We would like to pinpoint reasons for which different classes of recursive graphs have different degree of difficulty for certain problems. We approach this question by investigating ordering relations invariant under all recursive labelings of a graph. The existence of such relations has been studied (“monotone ordering” in [10] and [12]) in connection with numerical solutions of sparse systems of linear equations.

As an application example of this approach we solve the shortest-path, single-source problem for some classes of recursive graphs. For a rooted tree T , a recursive representation gives explicitly the shortest path from any vertex to the root of T . This property that a recursive representation explicitly gives shortest paths from all vertices to the root applies to some other classes as well. In particular we prove this for chordal graphs, which contain mops and k -trees. However, we show that this is not the case for several other classes of recursive graphs. The shortest-path problem has been discussed in many papers; bibliography of recent works may be found, e.g., in [8].

1. Recursive representation. Generally speaking, a class of graphs is recursive if each graph in the class can be constructed from an initial element (vertex, edge, triangle) by a finite number of applications of the following operation: to a graph G already constructed add a new vertex and join it to some set of vertices which define a fixed subgraph of G . A *recursive labeling* of such graph with n vertices uses integers 1 through k to label the *base vertices* and $k + 1$ through n to label the remaining vertices in the order they are added in a given recursive construction process.

A *recursive representation* of a graph G with k distinguished *base vertices* labeled $1, \dots, k$ is an array, GRAPH [$k + 1 \dots n$], such that the i th entry contains labels of the vertices to which the vertex labeled i was joined in the process of recursively constructing G , i.e., $j < i$ for all j in GRAPH [i].

* Received by the editors July 13, 1978, and in revised form May 1, 1980.

† Department of Computer Science, University of Oregon, Eugene, Oregon 97403.

For trees, the base vertex is the root labeled 1. The notion of a k -tree generalizes that of a tree: k -trees are generated recursively by joining a new vertex to a complete subgraph of order k (K_k) in the graph already constructed; in this case the base vertices form a complete graph of order k .

Maximal outerplanar graphs (mops) are an interesting subclass of 2-trees. During the recursive construction of a mop, a new vertex is made adjacent to two vertices adjacent along the Hamiltonian cycle of the already constructed mop. We distinguish here the base triangle, labeled 1, 2, 3.

There is another class of graphs seemingly “somewhere between” mops and k -trees, called *internally maximal planar graphs* (imps): here, a new vertex is joined to a number of vertices on a path along the outer contour of the existing structure [9].

A generalization of k -trees in terms of changing the size of the complete subgraph to which a new vertex is joined is the class of *chordal graphs* (see, for instance, [5]). The base vertex is labeled 1 and the i th entry of a recursive representation of a chordal graph L consists of the mutually adjacent vertices to which the vertex labeled i was joined during the recursive construction of L .

The class of *minimally two-connected graphs* (mtcs) is yet another class of graphs which may be constructed recursively [6]. The generation process, however, differs from those previously described because it calls for a possible modification of the existing structure when adding a new vertex. The new vertex is always joined to two vertices; if these are adjacent, then the existing edge between them is removed.

In general there may be more than one recursive process for constructing a given (unlabeled) graph. We will discuss this in the following section. Figs. 1, 2 and 3 illustrate the above definitions.

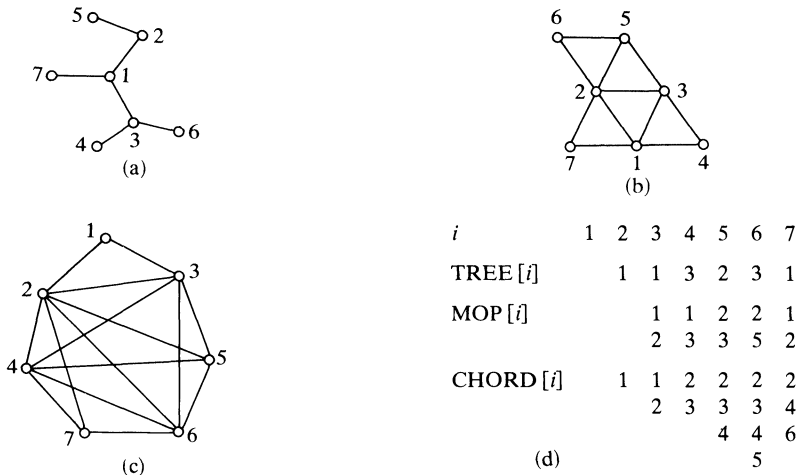


FIG. 1. A tree (a), a mop (b), a chordal graph (c) and their recursive representations (d).

2. Invariants of recursive labelings. The following question is important in studies of recursive graphs: What is invariant among all recursive labelings of a given graph with a specified root and the initial (“base”) structure? We will answer this question for the above defined classes of graphs and point out implications of this answer for the shortest-path problem.

Let us assume that a labeling of the base structure of the recursive graph is given (say, 1 through k). The following fact was discovered in [4] for chordal graphs and gave basis for the recursive definition of these graphs.

FACT 2.1. For a given recursive graph G with a distinguished base structure, there exists a removal sequence of vertices $v_n, v_{n-1}, \dots, v_{k+1}$ such that consecutive deletion of vertices $v_i (i = n, \dots, k+1)$ preserves the following property R : the graph $G - \{v_n, \dots, v_i\}$ is recursive (of the same type as G) and contains the same base structure as G .

The subscripts i used as labels together with the given base labeling give a recursive labeling of G . An invariant of recursive labelings is thus an invariant of removal sequences.

For rooted trees we have an intuitive notion of *end-vertices* (leaves): a vertex of degree 1 in a tree is called a leaf if it is not the root. In order to satisfy property R (Fact 2.1) for trees, the operation of removing vertices can only be applied to leaves. Therefore, the following fact gives a relation on the vertices of a rooted tree T which remains invariant under all possible recursive labelings of T .

FACT 2.2. Labels i and j of two vertices $u \neq w$ in a rooted tree T must be related, $i < j$, in all recursive labelings of T if and only if u is contained in the path from w to the root ($u < w$). The relation $<$ partially orders the set of vertices of T .

Recursive labelings of k -trees have a similar property. Defining a nonbasic vertex of degree k as a k -leaf, we have the following procedure for labeling vertices of a k -tree.

FACT 2.3. For a given k -tree Q with base structure (K_k) , the removal of a vertex v results in a k -tree with the same base structure if and only if v is a k -leaf.

The above fact gives us a relation which is invariant under all recursive labelings of a k -tree with a given base structure.

THEOREM 2.4. Any two adjacent, nonbasic vertices u and w of a k -tree Q with a given base structure must be labeled in the same manner with respect to each other in all recursive labelings of Q (say, the label of u is always greater than the label of w). If all such ordered vertex pairs define a relation R then the transitive closure R^* (denoted $<$) is invariant in all recursive labelings of Q , i.e., if $u < w$ then $\text{label}(u) < \text{label}(w)$.

Proof. Consider a pair of adjacent, nonbasic vertices u and w . Let us assume, without loss of generality, that there exists a recursive removal sequence of vertices v_n, \dots, v_{k+1} such that $u = v_i, w = v_j$ and $n \geq i > j > k$. If u is a k -leaf of Q , then our thesis follows from Fact 2.3, since w cannot be a k -leaf. Otherwise, u is a k -leaf of the k -tree $Q - \{v_n, \dots, v_{i+1}\}$ which also contains w . Let us now assume that there exists another removal sequence v'_n, \dots, v'_{k+1} in which $u = v'_p, w = v'_r$ and $p < r$. Then the removal of all vertices $\{v_n, \dots, v_{i+1}\} \cup \{v'_n, \dots, v'_{r+1}\}$ from Q would result in a k -tree in which both u and w are k -leaves, by repeated applications of Fact 2.3. But this contradicts the adjacency of u and w . Thus in all recursive labelings of Q , labels of u and w are ordered uniformly. \square

This characteristic of k -trees does not apply to *imps*. Two adjacent vertices of an *imp* in Fig. 2, u and w , may be labeled in different ways, thus preventing a uniform ordering.

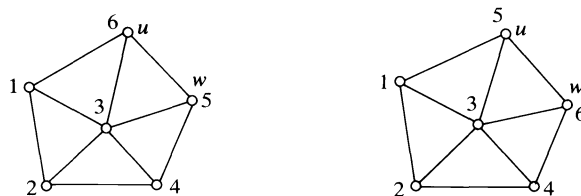


FIG. 2. Two different labelings of an *imp*.

This lack of uniformity in labeling is also characteristic of *mtcs* (cf. Fig. 3).

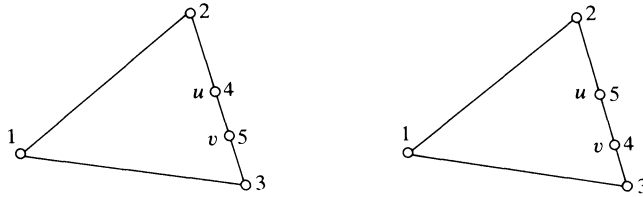


FIG. 3. An mtc with two labelings switching order of vertices u and w .

Every chordal graph L has a nonbasic vertex, called a *simplicial vertex*, all of whose neighboring vertices form a complete subgraph of L [3]. The removal of a simplicial vertex from a chordal graph results in another chordal graph. (A procedure based on this fact has been used as a test of chordality in [4].) In contrast to the case of k -trees, however, two simplicial vertices (which are a generalization of k -leaves) may be adjacent. Thus, for a chordal graph L , the relation R as defined in Theorem 2.4 is not necessarily preserved under different recursive labelings of L . The constraints of the relation R must be slightly relaxed to reflect the weakening of graph's structure by the generalization of k -trees to chordal graphs.

THEOREM 2.5. *In a chordal graph L , any two adjacent vertices u and v must be labeled in the same order in every recursive labeling of L (say, label $(u) < \text{label}(v)$) unless there exists a partial removal sequence w_n, \dots, w_{i+1} of vertices simplicial in $L_n = L$, $L = L_{n-1} - \{w_n\}$, etc., resulting in a chordal graph L_i in which both u and v are simplicial.*

Proof. Obviously, removal of vertices $w_n, w_{n-1}, \dots, w_{i+1}$ defines a recursive labeling of L based on a recursive labeling of L_i . u and v are both simplicial in L_i if and only if they form, together with all their neighbors, a complete subgraph of L_i . In that case they could be removed (or recursively labeled) in arbitrary order. We will show that in the absence of such a sequence w no removal sequence making v a simplicial vertex contains u . For purposes of a contradiction, let us assume that a removal sequence x_n, \dots, x_{j+1} exists such that u is a simplicial vertex of L_j and v has not been removed. Besides, there is no additional sequence y_j, \dots, y_{i+1} of vertices from L which could make u and v both simplicial (otherwise $x_n, \dots, x_{j+1}, y_j, \dots, y_{i+1}$ would play the role of the sequence w above). Let us now consider the vertices z_{k+1}, \dots, z_n with labels larger than $k = \text{label}(v)$ in that particular labeling in which $\text{label}(u) < \text{label}(v)$. Let us further remove from L_j all vertices z . As a result, v would become a simplicial vertex and u would remain one. This, however, contradicts our assumption. \square

3. A shortest-path spanning tree. For a single-source, shortest-path problem in a given graph, we can always find a spanning tree of the graph in which the unique path from any vertex v to the source r is a shortest path from v to r . If this rooted tree is represented recursively such that its vertices are labeled in a breadth-first order, then for any vertex v , the sequence l_0, l_1, \dots, l_p such that $l_0 = \text{label}(v)$, $l_j = \text{TREE}[l_{j-1}]$ for $j = 1, 2, \dots, p$ and $l_p = 1$ is a shortest path from v to the source labeled 1. Actually, this ordering (modeled on Dijkstra's "greedy" approach [2]) may be relaxed to involve only vertices with a common shortest path, i.e., $v < w$ if v lies on the path from w to the source (Fact 2.2). This ordering may be obtained by the labeling procedure in the recursive generation of trees.

FACT 3.1. *For any tree T rooted at a source, the recursive representation, TREE, of T represents explicitly shortest paths from any vertex to the root.*

We will call a solution to a single-source, shortest-path problem a *minimum distance spanning tree*. The invariant of recursive labelings of a chordal graph with a given root, stated in Theorem 2.5, suggests a solution to the shortest-path problem. A

minimum distance spanning tree of such a graph will be given by the array of smallest entries in the recursive representation of the chordal graph. To prove it we will show that any recursive labeling of a chordal graph orders adjacent vertices concordantly with nondecreasing value of the distance to the root.

LEMMA 3.2. *For two adjacent vertices u and v of a chordal graph L , such that $\text{label}(u) < \text{label}(v)$ in a particular recursive labeling of L , their shortest distances to the root r of L satisfy the inequality $d(u, r) \leq d(v, r)$. Furthermore, the addition of a simplicial vertex t does not change the shortest distances to the root of any other vertex.*

Proof. We prove this result by induction on the number of vertices, n . It is certainly true for $n = 2$, where there is only one nonbasic vertex, u , such that $d(r, r) = 0 < 1 = d(u, r)$ and $\text{label}(r) = 1 < 2 = \text{label}(u)$. The addition of another vertex to r and u will not change the value of $d(u, r)$. Let us assume that the theorem is true for all graphs of fewer than n vertices and add a new vertex v according to the recursive construction scheme, creating a chordal graph L of n vertices. Let $u \neq r$ be a vertex of the complete subgraph K of L' to which v has been joined. A shortest path from u to the root leaves K at some of u 's neighbors, or at u itself. Shortest paths from v to the root must pass through one of the vertices of K . Thus $d(u, r) \leq d(v, r)$. By Theorem 2.5, either $\text{label}(u) < \text{label}(v)$ in all recursive labelings of L or u and v are both simplicial in some L'' , a chordal subgraph of L . Then there exists a vertex w of K such that $d(u, r) = 1 + d(w, r)$. Thus $d(v, r) = d(u, r)$ and $\text{label}(x) < \text{label}(y)$ implies $d(x, r) \leq d(y, r)$ for all recursive labelings of L (where $\{x, y\} = \{u, v\}$). The addition of another vertex t to L will not change any minimum distance to the root: t connects (by paths of length 2) only vertices already adjacent. This completes the proof.

From this lemma it follows immediately that the minimum distance spanning tree of a rooted chordal graph may be found trivially from its recursive representation.

THEOREM 3.3. *The array $\text{LOW}[2..n]$ of smallest elements of entries in the recursive representation CHORD of a chordal graph L gives the minimum distance spanning tree of L with respect to the root labeled "1".*

The pointer to the array LOW is enough to indicate the minimum distance spanning tree of a given k -tree, and the theorem gives us, in fact, a "constant time algorithm" to compute it; i.e., the minimum distance spanning tree can be recovered from the recursive representation of the graph in cost proportional to its length. This is qualified by two assumptions: (i) the root of the chordal graph is the source in our problem, and (ii) the smallest label in the set $\text{CHORD}[i]$ is explicitly available. If the latter is not true it will take $O(n)$ operations (constant work per vertex) to recover this information. If the source of our shortest path problem differs from the root of the recursive representation we can "reroot" the graph.

ALGORITHM Rerooting.

- Input:** Old recursive representation of the chordal graph and the new source labeled $m > 1$.
- Output:** New recursive representation rooted at m .
- Method:** Call the vertices labeled lower than a given vertex v "older" than v . Starting at the source m , we proceed towards the root through the vertices older than m relabeling them and arranging elements of the new recursive representation. The vertices not relabeled during this process are already in order.

The above algorithm can be verified by the following observation (supplied by the referee). A recursive labeling of the rerooted graph is given by the removal sequence obtained by first removing simplicial vertices until the new source is simplicial, and then removing simplicial vertices other than the new source until it is the only vertex left. The existence of such a removal sequence follows from the characterization of chordal

graphs given in [4] and from the fact that every chordal graph with at least 2 vertices has more than one simplicial vertex.

An elaboration of the procedure given in the algorithm is given in [11] for the class of 2-trees.

4. Conclusions. In the paper we have investigated different recursive labelings of recursive graphs and established invariant relations under all possible such labelings for k -trees and chordal graphs. As a by-product, we have presented a property of chordal graphs in terms of their recursive representations. This property allows recovering of the solution to the single-source, shortest-path problem for these graphs in cost proportional to its length. In the case when the source differs from the root of a given recursive representation, the graph must be rerooted. For two other classes of recursive graphs, *imps* and *mtcs*, the “recursive removal” operation does not preserve the ordering of adjacent vertices. This makes the proof technique of the “constant time algorithm” inapplicable. It is easy to see that a theorem analogous to Theorem 3.2 does not hold for *imps* or *mtcs*. Fig. 4 presents two counterexamples to this; $\text{dist}(6, 1) < 1 + \text{dist}(4, 1)$ for the *imp* in 4(a) and $\text{dist}(4, 1) = 2 < 1 + \text{dist}(2, 1)$ for the *mtc* in 4(b). For an *mtc*, the array *LOW* does not necessarily represent a tree because of the change of existing structure in the process of recursive construction. This fact singles out the class of *mtcs* from the family of recursive graphs even more.

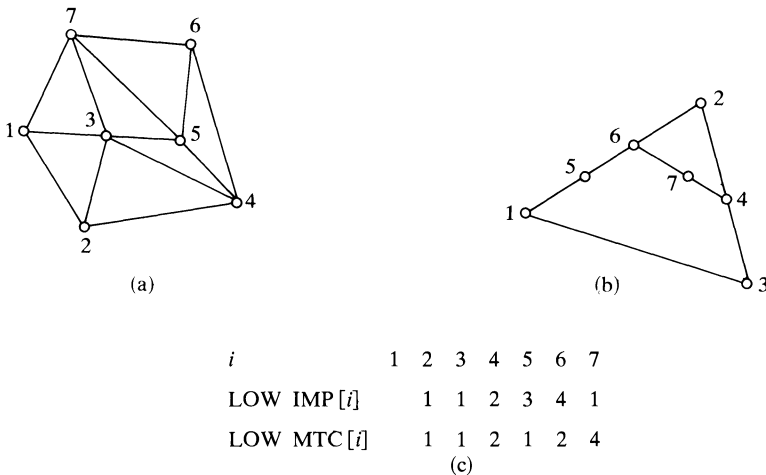


FIG. 4. An *imp* (a) and an *mtc* (b) for which arrays *LOW* (c) do not represent minimum distance spanning trees.

REFERENCES

[1] T. BAYER, A. PROSKUROWSKI, S. HEDETNIEMI AND S. MITCHELL, *Independent domination in trees*, Proc. 8th Southeastern Conf. on Combinatorics, Graph Theory and Computing, Utilitas Mathematica, Winnipeg, 1977, pp. 321–328.
 [2] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math., I (1959), pp. 269–271.
 [3] G. A. DIRAC, *On rigid circuit graphs*, Abh. Math. Sem. Univ. Hamburg, 25 (1961), pp. 71–76.
 [4] D. R. FULKERSON AND O. A. GROSS, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835–855.
 [5] 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.

- [6] S. T. HEDETNIEMI, *Characterizations and constructions of minimally 2-connected graphs and minimally strong digraphs*, Proc. 2nd Louisiana Conf. on Combinatorics, Graph Theory and Computing, Utilitas Mathematica, Winnipeg, 1971, pp. 257–282.
- [7] S. T. HEDETNIEMI, E. J. COCKAYNE AND S. L. MITCHELL, *Linear algorithms on recursive representations of trees*, J. Comput. System Sci., 18 (1979), pp. 76–85.
- [8] D. C. JOHNSON, *Efficient algorithms for shortest paths in sparse networks*, J. Assoc. Comput. Mach., 24 (1977), pp. 1–13.
- [9] S. MITCHELL, *Algorithms on trees and maximal outplanar graphs: design, complexity analysis, and data structures studies*, Ph.D. Thesis, University of Virginia, 1976.
- [10] S. PARTER, *The use of linear graphs in Gauss elimination*, SIAM Rev., 3 (1961), pp. 119–130.
- [11] A. PROSKUROWSKI, *Minimum dominating cycles in two-trees*, Int. J. Comp. Info. Sci., 8 (1979), pp. 405–417.
- [12] D. J. ROSE, *Triangulated graphs and the elimination process*, J. Math. Anal. Appl., 32 (1970), pp. 597–609.
- [13] P. J. SLATER, E. J. COCKAYNE AND S. T. HEDETNIEMI, *Information dissemination in trees*, CS-TR-78-11, Dept. of Computer Science, University of Oregon.

AN ANALYSIS OF A MEMORY ALLOCATION SCHEME FOR IMPLEMENTING STACKS*

ANDREW C. YAO†

Abstract. Consider the implementation of two stacks by letting them grow towards each other in a table of size m . Suppose a random sequence of *insertions* and *deletions* is executed, with each instruction having a fixed probability p ($0 < p < \frac{1}{2}$) to be a deletion. Let $A_p(m)$ denote the expected value of $\max\{x, y\}$, where x and y are the stack heights when the table first becomes full. We shall prove that, as $m \rightarrow \infty$, $A_p(m) = m/2 + \sqrt{m/(2\pi(1-2p))} + O((\log m)^2/\sqrt{m})$. This gives a solution to an open problem in Knuth [*The Art of Computer Programming*, Vol. 1, Exercise 2.2.2-13].

Key words. deletion, insertion, memory allocation, memory utilization, stack

1. Introduction. The purpose of this paper is to give a solution to an open problem of Knuth [2, Exercise 2.2.2-13], regarding the effectiveness of implementing two stacks by letting them grow towards each other.

Consider a contiguous block of m locations, which we use to implement two stacks. One stack grows from the left end of the block and the other from the right end; we denote the heights of the stacks by x and y . One measure¹ of the effectiveness of the memory utilization for this scheme is the expected value of $\max\{x, y\}$ when the two stacks first meet, i.e., when $x + y = m$. For example, suppose the value of $\max\{x, y\}$ is $2m/3$. If we had used one block for each stack, then we should have reserved at least $4m/3$ locations instead of the present m locations. The following model was proposed in Knuth [2], with p ($0 \leq p < 1$) as a parameter. Consider a sequence of stack operations to be carried out until the two stacks meet. Each instruction acts either on the left stack or on the right stack with equal probability; for each choice, there is a probability p for it to be a *deletion* and probability $1 - p$ to be an *insertion*. A deletion on an empty stack will not have any effect. Let $A_p(m)$ denote the expected value of $\max\{x, y\}$ when the two stacks first meet. It was shown in Knuth [2, Exercise 2.2.2-12] that $A_0(m) = m/2 + \sqrt{m/(2\pi)} + O(m^{-1/2})$. It was also stated [2, Exercise 2.2.2-13] that $\lim_{p \rightarrow 1} A_p(m) = 3m/4$ for fixed m . Thus, in this model, there is little gain in memory utilization for large m when only insertions are present, whereas substantial gain results when deletions are overwhelmingly dominant. The question asked was the behavior of $A_p(m)$ for fixed p and large m .

In this paper we prove the following result.

THEOREM 1. *Let $p \in (0, \frac{1}{2})$ be a fixed number. Then²*

$$A_p(m) = \frac{m}{2} + \sqrt{\frac{m}{2\pi(1-2p)}} + O\left(\frac{(\log m)^2}{\sqrt{m}}\right).$$

Thus, for such p , there is no substantial gain in memory utilization asymptotically. Note that the formula is also true for $p = 0$, as mentioned earlier.

We leave open the question of the asymptotic behavior of $A_p(m)$ when $p \cong \frac{1}{2}$.

* Received by the editors January 18, 1979, and in revised form July 8, 1980. This research was supported in part by the National Science Foundation under grant MCS77-05313. Part of this paper was prepared while the author was visiting Bell Laboratories, Murray Hill, New Jersey 07974.

† Computer Science Department, Stanford University, Stanford, California 94305.

¹ This measure is somewhat conservative. An alternative measure might be the expected value of $\max\{x, y\}$ at any time before the two stacks meet.

² Here and throughout this paper, p is assumed to be fixed and the constants in the O -notation may depend on p . Logarithms are the natural logarithms (i.e., with base e).

2. Random walks. The main idea of the proof is very simple. With the insertion probability $1 - p$ greater than $\frac{1}{2}$, the length of each stack behaves like a one-dimensional random walk with an average net gain of $(1 - 2p)/2$ per step. At time t , each stack is almost certain to have a length $(1 - 2p)t/2 + O(\sqrt{t})$. As a result, the two stacks are likely to meet at a point $O(\sqrt{m/(1 - 2p)})$ distance away from the midpoint. A rigorous proof is complicated by the fact that a deletion has no effect on an empty stack, making the problem not exactly a free random walk problem. To overcome this difficulty, the technique is to give a rough estimate of the stack lengths after $\approx \log m$ steps and treat the process after that as a free random walk, taking advantage of the fact that the probability of further encountering an empty stack will be very small.

We begin the formal proof by casting the problem in random walk terminology (see Feller [1] for backgrounds on random walks). Let I_L, I_R denote an *insertion* instruction for the left and the right stack, respectively, and D_L, D_R a respective *deletion* instruction. We can regard the execution of a sequence of such instructions as a “particle” performing a “walk” on the integer lattice points in the plane, with the coordinates (x, y) being the current heights of the stacks. For example, an instruction I_L causes the particle to move from its current position (x, y) to $(x + 1, y)$. An instruction D_L will cause the particle to move from (x, y) to $(x - 1, y)$, unless $x = 0$ (i.e., an empty left stack), in which case the position does not change. We shall call the line $x = 0$ a *reflecting barrier*, the line $y = 0$ being also a reflecting barrier. The line $x + y = m$ will be referred to as the *absorbing barrier*. The two-stack problem with deletion probability p defines a random walk that starts at $(0, 0)$, moving stochastically according to the above interpretation, and stops when any point on the absorbing barrier is reached (the point is called the *absorption point*). Let us call this a $(p, m; 0, 0)$ -random walk.

In general, a $(p, m; a, b)$ -random walk is defined in the same way, except it starts at the point (a, b) . We will assume hereafter that $0 < p < \frac{1}{2}$, $m > 0$, $a \geq 0$, $b \geq 0$, and $a + b \leq m$. Let $(X_{a,b}, Y_{a,b})$ denote the pair of random variables that have as their values the coordinates (x, y) of the absorption point if the walk ends on the absorbing barrier, and have values $(0, 0)$ if the walk never reaches the absorbing barrier. The value $(0, 0)$ in this latter assignment is not important, as we shall see later (see the remark at the end of this section) that it occurs only with probability 0. Let $Z_{a,b} = \max\{X_{a,b}, Y_{a,b}\}$. The quantity of interest, $A_p(m)$, is clearly equal to $\overline{Z_{0,0}}$.

We now consider a “free” random walk that is easier to analyze. In a $(p, m; a, b)$ -random walk, a particle starts at the point (a, b) , moves according to the transition rule

$$(x, y) \rightarrow \begin{cases} (x + 1, y) & \text{with probability } (1 - p)/2, \\ (x, y + 1) & \text{with probability } (1 - p)/2, \\ (x - 1, y) & \text{with probability } p/2, \\ (x, y - 1) & \text{with probability } p/2, \end{cases}$$

and stops when it hits the absorbing barrier $x + y = m$. We use $X'_{a,b}, Y'_{a,b}, Z'_{a,b}$ for the random variables defined in the same way as $X_{a,b}, Y_{a,b}, Z_{a,b}$. Again, we shall see later that the particle will eventually hit the absorbing barrier with probability 1.

The value of $\overline{Z'_{a,b}}$ can be evaluated rather precisely. In particular, we have the following result when (a, b) is close to the origin.

LEMMA 1. *If $a + b = O(\log m)$, then*

$$\overline{Z'_{a,b}} = \frac{m}{2} + \sqrt{\frac{m}{2\pi(1 - 2p)}} + O\left(\frac{(\log m)^2}{\sqrt{m}}\right).$$

Proof. See § 3. \square

We also have the following result.

LEMMA 2. *If $a, b \geq 10/(\log((1-p)/p)) \log m$, then*

$$\overline{Z_{a,b}} = \overline{Z'_{a,b}} + O(m^{-9}).$$

Proof. See § 3. \square

Let

$$\begin{aligned} \varepsilon_p &= \min \{(1-2p)/8, p/8\}, \\ \lambda_p &= \max \left\{ \left\lfloor \frac{10}{\varepsilon_p^2} \right\rfloor, \frac{4}{1-2p} \frac{10}{\log((1-p)/p)} \right\}, \\ \lambda'_p &= \frac{1-2p}{4} \lambda_p. \end{aligned}$$

Clearly, $\lambda'_p \geq 10/\log((1-p)/p)$. Define $R = [\lceil \lambda'_p \log m \rceil, \lceil \lambda_p \log m \rceil + 1]^2$. Lemmas 1 and 2 combine to give the following formula:

$$(1) \quad \overline{Z_{a,b}} = \frac{m}{2} + \sqrt{\frac{m}{2\pi(1-2p)}} + O\left(\frac{(\log m)^2}{\sqrt{m}}\right) \quad \text{for } (a, b) \in R.$$

We shall now use (1) to evaluate $\overline{Z_{0,0}}$.

Let $t = \lceil \lambda_p \log m \rceil + 1$ and S be the set of all sequences of length t in $\{I_L, I_R, D_L, D_R\}$. For each $s = s_1 s_2 \cdots s_t \in S$, let $r(s) = \prod_{1 \leq i \leq t} r_0(s_i)$, where $r_0(s_i) = (1-p)/2$ if $s_i \in \{I_L, I_R\}$ and $r_0(s_i) = p/2$ if $s_i \in \{D_L, D_R\}$. For each $s \in S$, let $(f_1(s), f_2(s))$ be the position of the particle in a $(p, m; 0, 0)$ -random walk after the sequence s have been executed. Clearly, for each k ,

$$\Pr(Z_{0,0} = k) = \sum_{s \in S} r(s) \Pr(Z_{f_1(s), f_2(s)} = k).$$

As a result, we have

$$(2) \quad \overline{Z_{0,0}} = \sum_{s \in S} r(s) \overline{Z_{f_1(s), f_2(s)}}.$$

Now, let M_p be any integer such that, if $m \geq M_p$, then $t < m$.

LEMMA 3. *Suppose $m \geq M_p$. Let $S_0 = \{s \in S; (f_1(s), f_2(s)) \notin R\}$. Then $\sum_{s \in S_0} r(s) \leq 8m^{-10}$.*

Proof. We need the following fact (see Rényi [3, p. 200]). If the toss of a certain coin has a probability v ($0 < v < 1$) to result in a ‘‘head’’, then after tossing the coin N times we have, for any

$$(3) \quad 0 < \delta < \left(2 \max \left\{ \frac{1-v}{v}, \frac{v}{1-v} \right\} \right)^{-1},$$

$$\Pr(|\# \text{ of ‘‘heads’’} - vN| > \delta N) \leq 2 e^{-N\delta^2/(4v(1-v))}.$$

For each $s \in S$, let $\#I_L(s), \#I_R(s), \#D_L(s), \#D_R(s)$ denote the number of appearances of I_L, I_R, D_L, D_R in s , respectively. It follows from (3) and the fact $4v(1-v) < 1$ that, for a random $s \in S$ (weighted by $r(s)$, of course),

$$(4) \quad \Pr(|\#i(s) - r_0(i)t| > \varepsilon_p t) \leq 2 \exp(-\varepsilon_p^2 t),$$

for $i \in \{I_L, I_R, D_L, D_R\}$.

As $m \geq M_p$, the particle will not be absorbed in t steps. Since $f_j(s) \leq t$ for $j \in \{1, 2\}$, it follows that $s \in S_0$ only if $f_j(s) \leq \lceil \lambda'_p \log m \rceil$ for some $j \in \{1, 2\}$. Observe that

$f_1(s) \cong \#I_L(s) - \#D_L(s)$ and $f_2(s) \cong \#I_R(s) - \#D_R(s)$. It is straightforward to verify that $f_j(s) \leq \lceil \lambda'_p \log m \rceil$ for some $j \in \{1, 2\}$ only if at least one of the conditions $|\#i(s) - r_0(i)t| > \varepsilon_p t$, where $i \in \{I_L, I_R, D_L, D_R\}$ is satisfied. It follows then from (4) that

$$\sum_{s \in S_0} r(s) \leq 4 \cdot 2 e^{-\varepsilon_p^2 t} \leq 8m^{-10}. \quad \square$$

From (1), (2) and Lemma 3 we obtain that, for $m \geq M_p$,

$$\begin{aligned} \overline{Z_{0,0}} &= \sum_{s \in S_0} r(s) \overline{Z_{f_1(s), f_2(s)}} + \sum_{s \in S_0} r(s) \overline{Z_{f_1(s), f_2(s)}} \\ &= \left(\frac{m}{2} + \sqrt{\frac{m}{2\pi(1-2p)}} + O\left(\frac{(\log m)^2}{\sqrt{m}}\right) \right) (1 - O(m^{-10})) + O(m^{-10}) \cdot O(m) \\ &= \frac{m}{2} + \sqrt{\frac{m}{2\pi(1-2p)}} + O\left(\frac{(\log m)^2}{\sqrt{m}}\right). \end{aligned}$$

This proves Theorem 1. \square

Remark. Let N be any large integer such that $((1-2p)/8)N > m$. Similar to the proof of Lemma 3, one can show that, with probability $1 - O(e^{-\varepsilon_p^2 N})$, the particle must have been absorbed in the first N steps in a $(p, m; a, b)$ -random walk (or a $(p, m; a, b)'$ -random walk). Let $N \rightarrow \infty$. This shows that the particle will be absorbed with probability 1.

3. Proofs of Lemma 1 and Lemma 2. We need some basic facts about 1-dimensional random walks (see Feller [1]). Consider a random walk in one dimension that starts at 0, and at each step moves to the left with probability p ($0 < p < \frac{1}{2}$) and to the right with probability $1 - p$. Let $u_{m,n}(p)$ be the probability that position m ($m > 0$) is reached for the first time at exactly the n th step. It is known (see Feller [1, Chap. 14, formula (4.14)]) that

$$(5) \quad u_{m,n}(p) = \frac{m}{n} \binom{n}{(n+m)/2} (1-p)^{(n+m)/2} p^{(n-m)/2}$$

if $n \geq m$ and n, m are of the same parity. All other $u_{m,n}(p) = 0$. Clearly,

$$(6) \quad \sum_n u_{m,n}(p) = 1.$$

FACT 1. Let $n_0 = m/(1-2p)$ and $c_p = 4p(1-p)/(1-2p)^2$. Then

$$\sum_n u_{m,n}(p)n = n_0, \quad \sum_n u_{m,n}(p)(n - n_0)^2 = c_p n_0.$$

Proof. The generating function $U_m(z) = \sum_{n \geq 0} u_{m,n} z^n$ is equal to $(G(z))^m$, where

$$G(z) = (1 - \sqrt{1 - 4p(1-p)z^2}) / (2pz),$$

as can be directly verified. The first sum is given by

$$\sum_n u_{m,n}(p)n = U'_m(1) = mG'(1) = n_0.$$

The second sum is then the variance of the sequence $u_{m,n}(p)$, $n = 0, 1, 2, \dots$, regarded

as a probability distribution. Thus, after some calculations, we find

$$\begin{aligned} \sum_n u_{m,n}(p)(n - n_0)^2 &= U''_m(1) + U'_m(1) - (U'_m(1))^2 \\ &= m(G''(1) + G'(1) - (G'(1))^2) \\ &= c_p n_0. \end{aligned} \quad \square$$

We also need the following result (see Feller [1, Chap. 14, formula (2.8)]).

FACT 2. *The probability that the above random walk ever reaches $-z$ (where $z > 0$) is equal to $(p/(1-p))^z$.*

We state one more fact. Let l be any number. For each $s \in \{\alpha, \beta\}^n$, let $W_n^{(l)}(s)$ denote the quantity $|\# \text{ of } \beta - \# \text{ of } \alpha - l|$. Let $w_n^{(l)}$ be the average value of $W_n^{(l)}(s)$, assuming that all 2^n sequences are equally likely. We omit the proof, which involves only standard manipulations.

FACT 3. $w_n^{(l)} = \sqrt{2n/\pi} + O((|l|^2 + 1)/\sqrt{n})$.

Proof of Lemma 1. Let $m' = m - (a + b)$ and $l = a - b$. A $(p, m; a, b)$ -random walk can be generated in the following way. First generate a sequence $\xi \in \{I, D\}^*$ one symbol at a time, each with a probability p to be a D and probability $1 - p$ to be an I , until $(\#I - \#D) = m'$ for the first time.³ Then convert ξ into a sequence $s \in \{I_x, I_y, D_x, D_y\}^*$ probabilistically by attaching with equal probability a suffix x or y to each symbol in ξ . We now associate with s a walk starting from the point (a, b) to an absorption point on $x + y = m$, by interpreting each I_x, I_y, D_x, D_y as a step moving from position (x, y) to $(x + 1, y), (x, y + 1), (x - 1, y), (x, y - 1)$, respectively. It is easy to verify that this procedure indeed generates a $(p, m; a, b)$ -random walk. It is also not difficult to see that, for each such s generated, the value of $Z'_{a,b}$ is given by

$$Z'_{a,b}(s) = \frac{m}{2} + \frac{h(s)}{2},$$

where $h(s) = |(\# \text{ of } I_y + \# \text{ of } D_x) - (\# \text{ of } I_x + \# \text{ of } D_y) - l|$.

Note that, for each sequence ξ of n symbols, the average value of $h(s)$ for s derived from ξ is in fact equal to $w_n^{(l)}$. Thus, we have

$$\overline{Z'_{a,b}} = \frac{m}{2} + \frac{1}{2} \sum_n (\text{probability that } |\xi| = n) \cdot w_n^{(l)}.$$

It is easy to see that the quantity (probability that $|\xi| = n$) is exactly $u_{m',n}(p)$. Hence

$$\overline{Z'_{a,b}} = \frac{m}{2} + \frac{1}{2} \sum_n u_{m',n}(p) w_n^{(l)}.$$

Using Fact 3 and the fact $l = O(\log m)$, we have

$$(7) \quad \overline{Z'_{a,b}} = \frac{m}{2} + \sqrt{\frac{1}{2\pi}} \sum_n u_{m',n}(p) \left(\sqrt{n} + O\left(\frac{(\log m)^2}{\sqrt{n}}\right) \right).$$

Write \sqrt{n} and $1/\sqrt{n}$ as

$$\sqrt{n} = \sqrt{n'_0} + \frac{1}{2}(n - n'_0)(n'_0)^{-1/2} + O((n - n'_0)^2(n'_0)^{-3/2}),$$

³We have ignored here the possibility that ξ may be infinite. However, our discussion is valid as the probability is zero for ξ to be infinite (see the remark at the end of § 2).

and

$$\begin{aligned} \frac{1}{\sqrt{n}} &= \frac{1}{\sqrt{n'_0}} + O(|n - n'_0|(n'_0)^{-3/2}) \\ &= \frac{1}{\sqrt{n'_0}} + O((n - n'_0)^2(n'_0)^{-3/2}) \quad \text{for all } n \geq 1, \end{aligned}$$

where $n'_0 = m'/(1 - 2p)$. Substituting these expressions into (7) and making use of Fact 1, we obtain

$$\begin{aligned} \overline{Z}_{a,b} &= \frac{m}{2} + \sqrt{\frac{1}{2\pi}} \sum_n u_{m',n}(p) \left(\sqrt{n'_0} + \frac{1}{2\sqrt{n'_0}}(n - n'_0) + O((\log m)^2) \cdot \left(\frac{1}{\sqrt{n'_0}} + \frac{(n - n'_0)^2}{n'_0\sqrt{n'_0}} \right) \right) \\ &= \frac{m}{2} + \sqrt{\frac{n'_0}{2\pi}} + \frac{(\log m)^2}{\sqrt{n'_0}} O\left(1 + \frac{1}{n'_0} \sum_n u_{m',n}(p)(n - n'_0)^2\right) \\ &= \frac{m}{2} + \sqrt{\frac{m'}{2\pi(1 - 2p)}} + O\left(\frac{(\log m)^2}{\sqrt{m'}}\right). \end{aligned}$$

Since $m' = m - O(\log m)$, the lemma follows. \square

Proof of Lemma 2. Consider a $(p, \infty; a, b)$ -random walk, and let $\Delta(a, b)$ be the probability that the particle will ever touch the reflecting boundaries ($x = 0$ or $y = 0$). By Fact 2, the probability for it to touch $x = 0$ is $(p/(1 - p))^a$ and for it to touch $y = 0$ is $(p/(1 - p))^b$. This implies that $\Delta(a, b) \leq (p/(1 - p))^a + (p/(1 - p))^b \leq 2m^{-10}$.

Since any walk that does not touch the reflecting barriers occurs with the same probability in both the $(p, m; a, b)$ -random walk and the $(p, m; a, b)'$ -random walk, we conclude that

$$|\overline{Z}_{a,b} - \overline{Z}'_{a,b}| \leq m \cdot \Delta(a, b) \leq 2m^{-9}.$$

This completes the proof of Lemma 2. \square

REFERENCES

[1] W. FELLER, *An Introduction to Probability Theory and Its Applications*, vol. I, 3rd ed., John Wiley, New York, 1968.
 [2] D. E. KNUTH, *The Art of Computer Programming*, vol. 1, 2nd ed., Addison-Wesley, Reading, MA, 1975.
 [3] A. RÉNYI, *Foundations of Probability*, Holden-Day, San Francisco, 1970.

INFERRING A TREE FROM LOWEST COMMON ANCESTORS WITH AN APPLICATION TO THE OPTIMIZATION OF RELATIONAL EXPRESSIONS*

A. V. AHO†, Y. SAGIV‡, T. G. SZYMANSKI† AND J. D. ULLMAN§

Abstract. We present an algorithm for constructing a tree to satisfy a set of lineage constraints on common ancestors. We then apply this algorithm to synthesize a relational algebra expression from a simple tableau, a problem arising in the theory of relational databases.

Key words. tree algorithms, lowest common ancestors, relational databases, relational algebra, tableaux, query optimization, join minimization

1. A tree discovery problem. In a rooted tree, the lowest common ancestor of two nodes x and y , denoted (x, y) , is the node a that is an ancestor of both x and y such that no proper descendant of a is also an ancestor of both x and y . Suppose we are told a tree T has leaves numbered $1, 2, \dots, n$, and we are given a set of constraints on the lowest common ancestors of certain pairs of leaves. In particular, suppose the constraints are of the form $(i, j) < (k, l)$ where $i \neq j$ and $k \neq l$, meaning that the lowest common ancestor of i and j is a proper descendant of the lowest common ancestor of k and l . Note that the order of i and j in (i, j) and of k and l in (k, l) are irrelevant. From a set of constraints of this form, can we reconstruct T , or determine that no such tree exists?

Example 1. Suppose we are given the constraints

$$(1.1) \quad \begin{aligned} (1, 2) &< (1, 3), \\ (3, 4) &< (1, 5), \\ (3, 5) &< (2, 4). \end{aligned}$$

One possible tree T consonant with these constraints is shown in Fig. 1. However, if we add the constraint $(4, 5) < (1, 2)$, then it can be shown that no tree can simultaneously satisfy all these constraints.

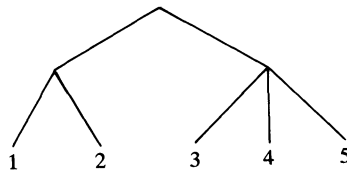


FIG. 1. A solution to constraints (1.1).

In this paper we give an efficient algorithm for solving this problem. We then present an application of this algorithm to the problem of synthesizing a relational algebra expression from a simple tableau.

2. Finding the tree. The central idea behind the solution is to determine for a potential tree T the sets of leaves that are descendants of each child of the root of T . Call these sets S_1, S_2, \dots, S_r . We may assume $r \geq 2$, since if a tree satisfying a set of

* Received by the editors July 10, 1979, and in revised form June 3, 1980.

† Bell Laboratories, Murray Hill, New Jersey 07974.

‡ Department of Computer Science, University of Illinois, at Urbana-Champaign, Urbana, Illinois 61801.

§ Department of Computer Science, Stanford University, Stanford, California. The work of this author was supported in part by the National Science Foundation under grant MCS-76-15255.

constraints exists, we can find another tree satisfying the same set of constraints if we merge each node having one child with that child. In Example 1, we have $S_1 = \{1, 2\}$ and $S_2 = \{3, 4, 5\}$.

There are two conditions that these sets must satisfy for each constraint $(i, j) < (k, l)$.

(1) i and j must be in the same set. Otherwise (i, j) is the root of T , and the root cannot be a proper descendant of (k, l) .

(2) Either k and l are in different sets, or i, j, k and l are all together in one set. Otherwise (i, j) cannot be a proper descendant of (k, l) .

We shall also show that conditions (1) and (2) are sufficient. Thus, if we can partition the nodes into two or more sets satisfying (1) and (2), and if we can recursively build trees for each set, then a tree exists; otherwise, one does not.

Given a set of constraints C , we define a partition π_C on the leaves $1, 2, \dots, n$ using the following rules:

(1) If $(i, j) < (k, l)$ is a constraint, then i and j are in one block of π_C .

(2) If $(i, j) < (k, l)$ is a constraint, and k and l are in one block, then i, j, k and l are all in the same block.

(3) No two leaves are in the same block of π_C unless it follows from (1) and (2).

Example 2. π_C for the constraints given in Example 1 is $\{1, 2\}, \{3, 4, 5\}$. Note that no instance of rule (2) is used. If we add to (1.1) the constraint $(4, 5) < (1, 2)$, to obtain the set

$$\begin{aligned} (1, 2) < (1, 3), \\ (3, 4) < (1, 5), \\ (3, 5) < (2, 4), \\ (4, 5) < (1, 2), \end{aligned}$$

then by rule (2) the two blocks must be merged, yielding a trivial partition. Since a necessary condition for the existence of a tree is that π_C have more than one block, we can immediately infer that this new set of constraints is not satisfied by any tree.

In Fig. 2 we present a recursive algorithm to build a tree T satisfying a set of constraints C on a nonempty set of nodes S . It returns the null tree if no tree exists. The basic idea is to compute the partition π_C , check that it has at least two blocks $S_1, S_2, \dots, S_r, r \geq 2$, and construct the sets of constraints $C_m, 1 \leq m \leq r$, such that C_m is C restricted to those constraints that involve members of S_m only.

THEOREM 1. *If BUILD(S, C) returns a nonnull tree T , then T satisfies the constraint set C .*

Proof. We proceed by induction on the size of S . The basis, one node, is trivial. Suppose then that the theorem is true for all sets smaller than S . Because every recursive invocation of BUILD is applied to a set of strictly smaller size than S , we can assume that all recursive calls of BUILD obey the inductive hypothesis.

We must show that all constraints in C are satisfied by T . Accordingly, let $(i, j) < (k, l)$ be an arbitrary member of C . Two cases arise depending on whether k and l are in the same or in different blocks.

Case 1. If k and l are in the same block, then all of i, j, k and l are in some set S_m by rule (2) in the definition of π_C . Also, $(i, j) < (k, l)$ is in C_m . By the inductive hypothesis, BUILD(S_m, C_m) produces a tree that satisfies $(i, j) < (k, l)$, so the final tree satisfies that constraint also.

Case 2. If k and l are in different blocks, then (k, l) is the root of T . By rule (1) in the definition of π_C , i and j are in the same block, and therefore (i, j) is not the root. Thus $(i, j) < (k, l)$ is surely satisfied. \square


```

procedure BUILD( $S, C$ );
if  $S$  consists of a single node  $i$  then
    return the tree consisting of node  $i$  alone
else
    begin
        compute  $\pi_C = S_1, S_2, \dots, S_r$ ;
        if  $r = 1$  then
            return the null tree
        else
            for  $m := 1$  to  $r$  do
                begin
                     $C_m := \{(i, j) < (k, l) \text{ in } C \mid i, j, k, l \text{ are in } S_m\}$ ;
                     $T_m := \text{BUILD}(S_m, C_m)$ ;
                    if  $T_m =$  the null tree then
                        return the null tree
                    end;
                /* if we reach here a tree exists */
                let  $T$  be the tree with a new node for its root and
                    whose children are the roots of  $T_m, 1 \leq m \leq r$ ;
                return  $T$ 
            end
        end
    end BUILD

```

FIG. 2. The procedure BUILD.

LEMMA 1. Let T be any tree satisfying constraint set C . Then each block of π_C is wholly contained within a subtree rooted at some child of the root of T .

Proof. Let us consider any fixed order in which we may apply rules (1) and (2) to construct π_C , starting with each leaf in a block by itself. We shall show, by induction on the number of applications of the rules, that each block of the partition being formed always lies wholly within the subtree of some child of the root of T .

Basis. Zero applications. Each leaf is in a block by itself, so the result is trivial.

Induction. Case 1. Rule (1) is applied to a constraint $(i, j) < (k, l)$ causing the blocks containing i and j , say B_1 and B_2 , to be merged. By induction, all the nodes in B_1 and B_2 are contained, respectively, in the subtrees rooted at n_1 and n_2 , two children of the root. If $n_1 \neq n_2$, then (i, j) is the root of T . But then no constraint of the form $(i, j) < (k, l)$ can be satisfied by T , violating our assumption that T satisfied C . We must therefore have $n_1 = n_2$ and all the nodes in $B_1 \cup B_2$ are descendants of the same child of the root.

Case 2. Rule (2) is applied to constraint $(i, j) < (k, l)$ because k and l are in the same block. Let B_1 and B_2 be the blocks containing i and j , and let B_3 be the block containing k and l . By induction, there exist nodes n_1, n_2 and n_3 , children of the root of T , that are the roots of subtrees containing all the nodes of B_1, B_2 and B_3 respectively. Clearly, $n_1 = n_2$, for otherwise (i, j) is the root of T and cannot possibly be a proper descendant of (k, l) as required by the constraint. If $n_1 \neq n_3$, then (i, j) and (k, l) are descendants of different children of the root, again violating the constraint. We must therefore have $n_1 = n_2 = n_3$ and all the nodes in $B_1 \cup B_2 \cup B_3$ are descendants of the same child of the root. \square

THEOREM 2. If there is a tree satisfying the constraint set C , then BUILD(S, C) returns some nonnull tree.

Proof. Suppose not, and assume that T with the constraint set C is as small a counterexample as there is. Suppose $\text{BUILD}(S, C)$ returns the null tree. Then either π_C has only one block, or $\text{BUILD}(S_m, C_m)$ returns the null tree for some m . In the first case, it follows from Lemma 1 that the root of T has only one child. But then T' , which is T with the root removed, provides a smaller counterexample to the theorem.

In the second case, where $\text{BUILD}(S_m, C_m)$ returns the null tree, let S_m be contained in the subtree rooted at child n of the root of T . Define T' to be the subtree of T with root n , after deleting all leaves not in S_m and any interior nodes none of whose descendant leaves are in S_m . Then T' satisfies C_m , T' is smaller than T , yet $\text{BUILD}(S_m, C_m)$ does not construct a nonnull tree. Thus T together with C was not the smallest counterexample to the theorem, as supposed.

As a single node cannot be counterexample to the theorem, we conclude that the theorem has no smallest counterexample. Since a counterexample, if one exists, is finite, there can be no counterexample at all, from which we conclude the theorem. \square

Example 3. Suppose we are given the constraints

$$\begin{aligned} (1, 3) < (2, 5) & \quad (4, 5) < (1, 9) \\ (1, 4) < (3, 7) & \quad (7, 8) < (2, 10) \\ (2, 6) < (4, 8) & \quad (7, 8) < (7, 10) \\ (3, 4) < (2, 6) & \quad (8, 10) < (5, 9). \end{aligned}$$

Rule (1) produces partition $\{1, 3, 4, 5\}, \{2, 6\}, \{7, 8, 10\}, \{9\}$. By Rule (2), we must merge the first two blocks because of constraint $(3, 4) < (2, 6)$. Thus the tree constructed top down begins as in Fig. 3(a). The constraints germane to $\{1, 2, \dots, 6\}$ are $(1, 3) < (2, 5)$ and $(3, 4) < (2, 6)$, while only $(7, 8) < (7, 10)$ is germane to $\{7, 8, 10\}$. The partition for the former is $\{1, 3, 4\}, \{2\}, \{5\}, \{6\}$, and for the latter $\{7, 8\}, \{10\}$. The second level of tree construction is shown in Fig. 3(b). No constraints are applicable to any of the remaining blocks, so at the next level, each block of more than one leaf is partitioned into singletons. The final tree is shown in Fig. 3(c). \square

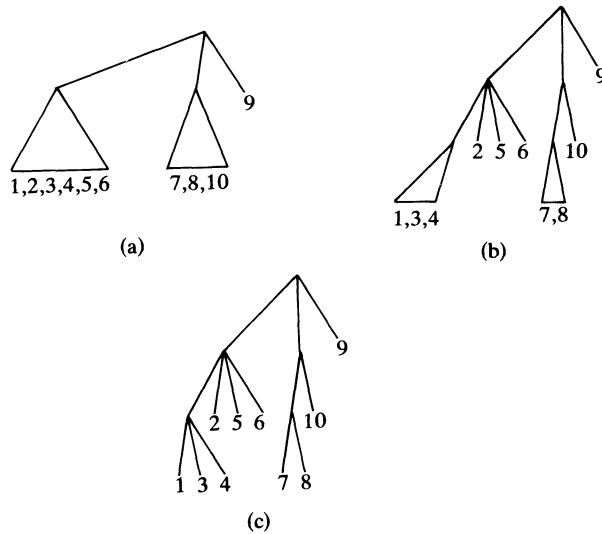


FIG. 3. Construction of a tree.

3. Complexity of the tree synthesis algorithm. The running time of the tree synthesis algorithm of § 2 is highly dependent on the method used to partition the set of constraints. In this section, we shall first analyze the running time of BUILD as a

function of n , the number of constraints, and f , a function specifying the time needed to partition a set of constraints. We shall see that the use of the naive partitioning algorithm leads to an $O(n^3)$ implementation of BUILD. By using a more sophisticated partitioning algorithm, the overall running time can be reduced to $O(n^2 \log n)$. Moreover, by imposing the restriction that all constraints be of the form $(i, j) < (i, k)$, we can further reduce the running time to $O(n^2)$. Section 4 of this paper presents a database application in which problems of this restricted form arise naturally.

LEMMA 2. *Let $f(n)$ be the time needed to partition a set of n constraints subject to rules (1) and (2) of the previous section. If f is monotonically nondecreasing, then the time consumed by BUILD when applied to a set of n constraints is $O(nf(n))$.*

Proof. The worst-case running time occurs when the algorithm succeeds in synthesizing a nonnull tree, so we shall restrict our attention to analyzing this case. We shall proceed by assigning a cost to each node of the tree and summing the costs.

Since the top level call on BUILD involves n constraints, each of which can introduce at most $4n$ leaves into the tree, the synthesized tree has at most $4n$ leaves. (Leaves not mentioned in any constraint can be made children of the root of the final tree.) Because the branching factor at each internal node is at least 2, the number of internal nodes cannot exceed $4n - 1$.

At each internal node, BUILD partitions some subset of the original constraints and then spends a constant amount of time (exclusive of recursive calls) processing each block of the partition. Since f is monotonic, $f(n)$ is certainly an upper bound on the cost of performing each partition. Moreover, since f must grow at least linearly with its argument, we can neglect the cost of distributing the constraints over the blocks of the partition when compared with $f(n)$ and charge each internal node $O(f(n))$ units of time. Thus the time charged to all internal nodes is $O((4n - 1)f(n)) = O(nf(n))$. Since BUILD spends a constant amount of time at each leaf, the contribution of the leaves to the running time is $O(n)$ for a total over all nodes of $O(nf(n))$. \square

The problem of producing a fast implementation of BUILD thus reduces to one of producing a fast partitioning algorithm. Let us first address this problem for the special case mentioned earlier.

THEOREM 3. *If all constraints are of the form $(i, j) < (i, k)$, then BUILD can be implemented to run in $O(n^2)$ time, where n is the number of constraints.*

Proof. By Lemma 2, it suffices to show how to partition a set of n constraints in $O(n)$ time. When all constraints are of the form $(i, j) < (i, k)$, rule (2) of the partitioning algorithm of § 2 need never be applied explicitly. To see this, suppose that i and k are in the same block. By rule (1), i and j must also be in the same block, and thus by transitivity, i , j , and k are all in the same block. Accordingly, it suffices to implement rule (1) alone.

If we take each leaf appearing in one or more constraints to be a node, and each constraint $(i, j) < (i, k)$ to represent an edge between i and j , then we shall have a graph (actually, a multigraph) whose connected components represent the blocks of the partition we are looking for. Since it is easy to find connected components in time proportional to the number of edges [1], and the graph described above has at most n edges, we conclude that partitioning can be done in $O(n)$ time. \square

THEOREM 4. *If no restrictions are placed on the form of the constraints, then BUILD can be implemented to run in $O(n^2 \log n)$ time.*

Proof. By Lemma 2, it suffices to show how to partition a set of n constraints in $O(n \log n)$ time. Blocks of the partition will be maintained using a set merging algorithm that supports the following operations:

- (1) Given an element i , find the set that i currently belongs to.
- (2) Given two sets, merge them together and assign them a common name.

The particular set merging algorithm we need was originally described in [7] and operates as follows. An array S is maintained so that at all times $S[i]$ gives the name of the set currently containing element i . This allows each find operation to be performed in constant time. Each set is represented by a linked list of the current members of the set and an integer specifying the cardinality of the set. Now suppose we want to merge sets i and j . Without loss of generality, let set i have more members than set j . Change $S[k]$ to i for each element k in set j . Then append the list of elements of set j to the list of elements for set i . Finally, update the length of the new set i . It is easy to see that a merge operation can be performed in time proportional to the size of the smaller of the two sets being merged. Thus, starting with a collection of singleton sets, a sequence of n merges can be performed in time $O(n \log n)$.

Returning to the constraint partitioning algorithm, we can use the rules defining the partition π_C to transform a given set of constraints into a set of *commands* of the form $i \equiv j$, which means that i and j must appear in the same block, and a set of *implications* of the form $p \equiv q \Rightarrow r \equiv s$, which means that if p is in the same block as q , then r must be in the same block as s . In the algorithm, each block B of the partition will be associated with a list L_B consisting of all those implications of the form $p \equiv q \Rightarrow r \equiv s$ for which either p or q is currently in B . We assume that the length of each list is stored with it and is available in constant time. We shall also employ a queue Q of commands that remain to be processed. A command is processed by checking that the nodes involved are in the same block of the partition. If not, a merge operation is performed on the appropriate blocks and the algorithm then examines (by traversing the appropriate list L_B) all implications, whose left side involves nodes in the blocks being merged. When an implication is examined, it can cause another command (i.e., its right side) to be generated and placed on Q . The algorithm is shown in Fig. 4.

Before proceeding, notice that each constraint gives rise to one command, which is immediately placed on Q in step (2), and two copies of an implication, say $p \equiv q \Rightarrow r \equiv s$, which are placed on different lists. At most one of these copies will eventually cause the command $r \equiv s$ to be placed on Q , and this will happen when a merge operation first causes p and q to become elements of the same block. Thus the total number of commands enqueued is at most $2n$ (n in step (2) and n in step (3)).

The time spent in steps (1) and (2) is clearly $O(n)$. The time expended by the inner **for** loop of Step (3) is $O(n \log n)$ because each copy of an implication is only considered in this loop when it is being moved from a shorter to a longer list. Since no more than $n - 1$ merges can be performed, no implication can be moved more than $\log_2 n - 1$ times. Thus the amount of work done in the inner **for** loop is $O(n \log n)$. The time spent in the rest of step (3), exclusive of the inner **for** loop, is expended in removing at most $2n$ commands from Q , doing at most $O(n)$ finds and comparisons, and performing at most $n - 1$ merges. This clearly requires $O(n \log n)$ time, and so the total time used by the algorithm in $O(n \log n)$. \square

Extensions. The tree synthesis algorithm of §§ 2 and 3 can be extended to handle a more general set of constraints than the ones considered so far. More specifically, we can handle any collection of constraints of the following types:

- (1) $(i, j) < (k, l)$,
- (2) $(i, j) \leq (k, l)$,
- (3) $(i, j) = (k, l)$
- (4) (i, j) is comparable to (k, l) ,
- (5) (i, j) is incomparable to (k, l) .

In the above constraints, two nodes of a tree are said to be *comparable* if one is the ancestor of the other, and *incomparable* otherwise.

```

(1) for each leaf  $l$  mentioned in a constraint do
    begin
        set  $L_l$  to the empty list;
        set  $S[l]$  to  $l$ ;
    end;
(2) for each constraint  $(i, j) < (k, l)$  do
    begin
        let  $c$  be the implication  $k \equiv l \Rightarrow i \equiv j$ ;
        add  $c$  to  $L_{S[k]}$ ;
        add  $c$  to  $L_{S[l]}$ ;
        add the command  $i \equiv j$  to  $Q$ ;
    end;
(3) while  $Q$  is not empty do
    begin
        remove a command  $p \equiv q$  from  $Q$ ;
        if  $S[p] \neq S[q]$  then
            begin
                let  $L$  be the shorter of  $L_{S[p]}$  and  $L_{S[q]}$ ;
                for each implication  $u \equiv v \Rightarrow x \equiv y$  on  $L$  do
                    if one of  $u$  and  $v$  is in  $S[p]$ 
                        and the other is in  $S[q]$  then
                            add the command  $x \equiv y$  to  $Q$ ;
                append  $L_{S[p]}$  to  $L_{S[q]}$ ;
                merge  $S[p]$  and  $S[q]$ ;
            end;
        end;
    end;

```

FIG. 4. The general case partitioning algorithm.

As before, there exists a set of rules for partitioning the leaves of the tree into disjoint sets so that the leaves in each set are descendants of a different child of the root. The appropriate set of rules are:

- (1) If $(i, j) < (k, l)$ is a constraint, then i and j are in the same block of π_C .
- (2) If $(i, j) < (k, l)$ is a constraint, and k and l are in the same block, then i, j, k and l are all in the same block.
- (3) If $(i, j) \equiv (k, l)$ is a constraint, and k and l are in the same block, then i, j, k and l all in the same block.
- (4) If $(i, j) = (k, l)$ is a constraint, and i and j are in the same block (or k and l are in the same block), then i, j, k and l are all in the same block.
- (5) If (i, j) is comparable to (k, l) is a constraint, i and j are all in the same block, and k and l are in the same block, then i, j, k and l are all in the same block.
- (6) If (i, j) is incomparable to (k, l) is a constraint, then i is in the same block as j , and k is in the same block as l .
- (7) No two leaves are in the same block of π_C unless it follows from (1) through (6).

These rules can show to be both necessary and sufficient conditions for constructing a tree obeying a set of constraints (if one exists) using the procedure BUILD. Each rule involves adding one case to each of Theorem 1 and Lemma 1. A straightforward modification of the partitioning algorithm in § 3 allows us to handle this wider class of constraints without increasing the running time claimed in Theorem 3. For a further generalization of this problem, see [6].

4. An application to relational database queries. In [5], Codd introduced relational algebra as a notation for expressing database queries. In [2], [3] a class of relational expressions called SPJ-expressions was investigated in which the operands of an expression are relations and the operators are the relational algebra operations select, project and natural join.

In [3] it was shown that the value of an SPJ-expression can be represented in terms of a two-dimensional matrix called a *tableau*. A join-minimization procedure for SPJ-expressions was outlined in which a tableau is constructed from a given SPJ-expression E . The tableau is then transformed into an equivalent minimum row tableau. From this minimum row tableau we can then construct an SPJ-expression that has the fewest joins of any relational expression equivalent to the original expression E .

Unfortunately, this optimization procedure may be computationally expensive in that it is NP-complete to minimize the number of rows in a tableau. However, for the important special case of SPJ-expressions with simple tableaux this optimization method can be carried out efficiently. A simple tableau can be minimized in polynomial time [3]. In the remainder of this paper we shall complete the details of the optimization procedure by showing how to use the tree discovery algorithm of § 2 to construct an SPJ-expression from a simple tableau in polynomial time. By way of contrast, Yannakakis and Papadimitriou have shown that it is NP-complete to determine whether an arbitrary tableau has an equivalent SPJ-expression [11].

SPJ-expressions. We assume a data base consisting of a set of two-dimensional tables called *relations*. The columns of a table are labeled by distinct *attributes* and the entries in each column are drawn from a fixed *domain* for that column's attribute. The ordering of the columns of a table is unimportant. Each row of a table is a mapping from the table's attributes to their respective domains. A row is often called a *tuple* or *record*. If r is a relation that is defined on a set of attributes that includes A , and if μ is a tuple of r , then $\mu(A)$ is the value of the A -component of μ .

A *relation scheme* is the set of attributes labeling the columns of a table. When there is no ambiguity, we shall use the relation scheme itself as the name of the table. A relation is just the "current value" of a relation scheme. The relation is said to be *defined on* the set of attributes of the relation scheme. The operators select, project and join are defined as follows.

(1) *Select*. Let r be a relation on a set of attributes X , A an attribute in X , and c a value from the domain of A . Then the *selection* $A = c$ applied to r , written $\sigma_{A=c}(r)$, is the subset of r having value c for attribute A .

(2) *Project*. Let r be a relation on a set of attributes X . Let Y be a subset of X . We define $\pi_Y(r)$, the *projection* of r onto Y , to be the relation obtained by removing all the components of the tuples of r that do not belong to Y and removing duplicate tuples.

(3) *Join*. The join operator, denoted by \bowtie , permits two relations to be combined into a single relation whose attributes are the union of the attributes of the two argument relations. Let R_1 and R_2 be two relation schemes with values r_1 and r_2 . Then

$$r_1 \bowtie r_2 = \{\mu \mid \mu \text{ is a tuple defined on the attributes in } R_1 \cup R_2, \text{ and there exist tuples } v_1 \text{ in } r_1 \text{ and } v_2 \text{ in } r_2, \text{ such that } v_1(A) = \mu(A) \text{ for all } A \text{ in } R_1 \text{ and } v_2(A) = \mu(A) \text{ for all } A \text{ in } R_2\}.$$

An *SPJ-expression* is an expression in which the operands are relation schemes and the operators are select, project and join. An SPJ-expression with operands R_1, R_2, \dots, R_n evaluates to a single relation when relations are assigned to R_1, R_2, \dots, R_n .

Definition of a tableau. A tableau is a matrix in which the columns correspond to the attributes of the universe in a fixed order. The first row of the matrix is called the *summary* of the tableau. The remaining rows are exclusively called *rows*. The general idea is that a tableau is a shorthand for a set former $\{a_1 \cdots a_n | \psi(a_1, \dots, a_n)\}$ that defines the value of a relational expression. In the set former ψ does not have any a_i 's as bound variables. To simplify later discussion we shall adopt the following conventions regarding tableaux. The symbols appearing in a tableau are chosen from:

- (1) Distinguished variables, for which we use a 's, possibly with subscripts. These correspond to the symbols to the left of the bar in ψ .
- (2) Nondistinguished variables, for which we generally use b 's. These are the bound variables appearing in the set former.
- (3) Constants, for which we use c 's or nonnegative integers.
- (4) Blank.

The summary of a tableau may contain only distinguished variables, constants, and blanks. The rows of a tableau may contain variables (distinguished and nondistinguished) and constants. When tableaux represent SPJ-expressions, we can assume that the same variable does not appear in two different columns of a tableau, and that a distinguished variable does not appear in a column unless it also appears in the summary of that column.

Let T be a tableau and let S be the set of all symbols appearing in T (i.e., variables and constants). A *valuation* ρ for T associates with each symbol of S a constant, such that if c is a constant in S , then $\rho(c) = c$. We extend ρ to the summary and rows of T as follows. Let w_0 be the summary of T , and w_1, w_2, \dots, w_n the rows. Then $\rho(w_i)$ is the tuple obtained by substituting $\rho(v)$ for every variable v that appears in w_i .

A tableau defines a mapping from *universal instances* (relations over the set of all attributes) to relations over a certain subset of the attributes, called the *target relation scheme*, in the following way. If T is a tableau and I an instance, then $T(I)$ is the relation on the attributes whose columns are nonblank in the summary, such that

$$T(I) = \{\rho(w_0) | \text{for some valuation } \rho \text{ we have } \rho(w_i) \text{ in } I \text{ for } 1 \leq i \leq n\}.$$

Example 4. Let T be the tableau

A	B	C
a_1	a_2	
a_1	b_1	b_3
b_2	a_2	1
b_2	b_1	b_4

We conventionally show the summary first, with a line below it. We can interpret this tableau as defining the relation on AB

$$T(I) = \{a_1 a_2 | (\exists b_1)(\exists b_2)(\exists b_3)(\exists b_4) \text{ such that } a_1 b_1 b_3 \text{ is in } I \text{ and } b_2 a_2 1 \text{ is in } I \text{ and } b_2 b_1 b_4 \text{ is in } I\}$$

given any universal instance I .

Equivalence. Two SPJ-expressions $E_1(R_1, \dots, R_m)$ and $E_2(S_1, \dots, S_n)$ are said to be *equivalent* if, for all universal instances I , $E_1(r_1, \dots, r_m) = E_2(s_1, \dots, s_n)$, where $r_i = \pi_{R_i}(I)$, $1 \leq i \leq m$, and $s_i = \pi_{S_i}(I)$, $1 \leq i \leq n$. In words, E_1 is equivalent to E_2 if they represent the same mapping on universal instances.

Similarly, two tableaux T_1 and T_2 are equivalent if, for all I , $T_1(I) = T_2(I)$. Likewise, a tableau T is equivalent to an expression $E(R_1, \dots, R_n)$ if, for all I , $T(I) = E(r_1, \dots, r_n)$ where $r_i = \pi_{R_i}(I)$ for $1 \leq i \leq n$.

Representation of SPJ-expressions by tableaux. Given an SPJ-expression E , we can construct a tableau T to represent the expression in the following manner. The construction proceeds inductively on the form of E .

- (1) If E is a single relation scheme R , then the tableau T for E has one row and a summary such that:
 - (i) If A is an attribute in R , then in the column for A tableau T has the same distinguished variable in the summary and row.
 - (ii) If A is not in R , then its column has a blank in the summary and a new nondistinguished variable in the row.
- (2a) Suppose E is of the form $\sigma_{A=c}(E_1)$, and we have constructed T_1 , the tableau for E_1 .
 - (i) If the summary for E_1 has blank in the column for A , then $T = T_\emptyset$, where T_\emptyset is a tableau that maps any universal instance into the empty set.
 - (ii) If there is a constant $c' \neq c$ in the summary column for A , then $T = T_\emptyset$. If $c = c'$, then $T = T_1$.
 - (iii) If T_1 has a distinguished variable a in the summary column for A , the tableau T for E is constructed by replacing a by c wherever it appears in T_1 .
- (2b) Suppose E is of the form $\pi_X(E_1)$, and T_1 is the tableau for E_1 . The tableau T for E is constructed by replacing nonblank symbols by blanks in the summary of T_1 for those columns whose attributes are not in X . Distinguished variables in the rows of those columns are consistently replaced with new nondistinguished variables.
- (2c) Suppose E is of the form $E_1 \bowtie E_2$ and T_1 and T_2 are the tableaux for E_1 and E_2 , respectively. Let S_1 and S_2 be the symbols of T_1 and T_2 , respectively. Without loss of generality, we may take S_1 and S_2 to have disjoint sets of nondistinguished variables, but identical distinguished variables in corresponding columns.
 - (i) If T_1 and T_2 have some column in which their summaries have distinct constants, then $T = T_\emptyset$.
 - (ii) If no corresponding positions in the summaries have distinct constants, the set of rows of the tableau T for E consists of the union of all the rows of T_1 and T_2 . The summary of T has in a given column:
 - (a) The constant c if one or both of T_1 and T_2 have c in that column's summary. In this case we also replace any distinguished variable in that column by c .
 - (b) The distinguished variable a if (a) does not apply, but one or both of T_1 and T_2 have a in that column's summary.
 - (c) Blank, otherwise.

It is not hard to show that the tableau constructed by this procedure is equivalent to the given expression. Note that the number of rows in the resulting tableau is one more than the number of join operators in the original expression.

Example 5. Let A, B and C be the attributes, in that order, and suppose we are given the expression $\pi_A(\sigma_{B=0}(AB \bowtie BC))$. By Rule (1), the tableaux for AB and BC are

A	B	C		A	B	C
a_1	a_2		and	a_2	a_3	
a_1	a_2	b_1		b_2	a_2	a_3

By Rule (2c), the tableau for $AB \bowtie BC$ is

A	B	C
a_1	a_2	a_3
a_1	a_2	b_1
b_2	a_2	a_3

By Rule (2a), the tableau for $\sigma_{B=0}(AB \bowtie BC)$ is

A	B	C
a_1	0	a_3
a_1	0	b_1
b_2	0	a_3

Finally, by Rule (2b), the tableau for $\pi_A(\sigma_{B=0}(AB \bowtie BC))$ is

A	B	C
a_1		
a_1	0	b_1
b_2	0	b_3

Simple tableaux. A tableau is *simple* if in any column with a repeated nondistinguished variable there is no other symbol that appears in more than one row. For simple tableaux there exists a polynomial-time equivalence algorithm, whereas for general tableaux the equivalence problem is NP-complete [3]. In practice, it is not easy to find an SPJ-expression with a nonsimple tableau. The expression $\pi_{AC}(AB \bowtie BC) \bowtie (AB \bowtie BD)$ is a minimal expression that gives rise to a nonsimple tableau. The tableau is shown in Fig. 5. The rows in the column for B have repeated nondistinguished and distinguished variables.

A	B	C	D
a_1	a_2	a_3	a_4
a_1	b_1	b_2	b_3
b_4	b_1	a_3	b_5
a_1	a_2	b_6	b_7
b_8	a_2	b_9	a_4

FIG. 5. A nonsimple tableau.

5. Synthesis of relational expressions from tableaux. In this section we shall present an algorithm for constructing a relational expression from a simple tableau. To help clarify the presentation, we shall portray a relational expression by its parse tree.

Although many relational expressions can be synthesized from the same tableau, the relational expression produced by our algorithm has a parse tree with the following properties:

- (1) Each project operation is done as soon as possible (i.e., is as low in the parse tree as possible).
- (2) Each select operation is applied to a leaf.

These two points are motivated by efficiency considerations; performing projections and selections as early as possible can significantly reduce the size of intermediate relations computed in the evaluation of a relational expression. We should point out, however, that to evaluate a relational expression efficiently in practice, one must take into account many parameters of the database environment such as the costs of the various data access methods that are available and the nature of the data structures used to store the relations. See [8], [9], [10], [12] for more discussion of query evaluation strategies.

We shall assume the leaves of a parse tree are labeled by relation schemes. In the absence of other information, we can choose as leaf labels relation schemes having as few attributes as possible. For example, the expressions $\pi_A(A)$, $\pi_A(AB)$, $\pi_A(AC)$, and $\pi_A(ABC)$ all have the same tableau. Of these expressions, $\pi_A(A)$ is minimal in that the other expressions can be produced from it by simply adding one or more attributes to the relation scheme which is the operand of the expression.

For the application considered here, however, the problem of deciding which relation scheme corresponds to which leaf of the tree vanishes. Here we are interested in minimizing the number of joins in a given SPJ-expression. One way to perform this optimization is to construct a tableau for the given expression, minimize the number of rows in the tableau using the procedure in [2], and then convert the resulting tableau back to an SPJ-expression. The number of rows in the tableau is one more than the number of joins in the associated expression. In the process, it is easy to keep track of which rows correspond to which operands of the original expression. Since the tableau minimization algorithm of [2] can only delete rows but never change one, we can associate each leaf of the tree eventually produced with a relation scheme appearing in the original expression. Accordingly, we shall henceforth assume that each row of the tableaux under consideration is identified with some given relation scheme.

Let us initially consider a simple tableau T with no constants in the summary, although constants may appear in the rows. Let A be a column in which rows i and j have the same nondistinguished variable, and k a row with a distinguished variable in column A ¹. If T comes from an expression, consider the parse tree P of that expression. Each leaf of P corresponds to an operand relation scheme associated with some row of the tableau.

Suppose that in P (i, j) is not a proper descendant of (i, k) . Since (i, j) and (i, k) cannot be independent, it follows that (i, k) must be a descendant of (i, j) . Then consider the tableau constructed for the subexpression rooted at the node (i, k) of P . In particular, which rows have the distinguished variable in column A ? Surely k does, as the distinguished variable appears there in tableau T , and in the tableau construction algorithm a nondistinguished variable is never changed into a distinguished variable. If row i does not also have the distinguished variable in column A , then the variables in

¹ In a simple tableau, k will be unique, but we can relax the "simple" constraint to the point that a column with a repeated nondistinguished variable can have a repeated distinguished variable or constant, but not another repeated nondistinguished variable.

rows i and j cannot be equated when we create the tableau for the expression represented by node (i, j) . However, if both rows i and k have the distinguished variable, then they must have the same variable in T , a contradiction. We can therefore infer that $(i, j) < (i, k)$. Arguing similarly, we can show that $(i, j) < (j, k)$. From these two observations we are led to the conclusion:

- (*) Suppose P is the parse tree of an expression yielding tableau T . If rows i and j of T have the same nondistinguished variable in column A , and row k has a distinguished variable in column A , then we must have $(i, j) < (i, k)$ and hence $(i, j) < (j, k)$ in P .

That this is sufficient for simple tableaux is expressed in the next theorem.

THEOREM 5. *Let T be a tableau having no constants in the summary and no column with two or more repeated nondistinguished variables. Then T comes from an expression if and only if the set of constraints defined in (*) determines a tree.*

Proof. Only if. We argued above that if T comes from an expression, the constraints (*) must be satisfied.

If. Using the algorithm of § 2, suppose (*) determines a tree P . We shall show how to convert this tree into an expression for T . This expression may use join as an n -ary operator. But since binary join is associative and commutative, nothing is lost by doing so, and the n -ary joins can be replaced by binary joins in any order.

The construction proceeds by induction on the height of a node in the tree P . For the inductive hypothesis we shall construct for each subtree an expression yielding a tableau whose rows are the rows of T that are leaves of the subtree, but with repeated nondistinguished variables, all of whose occurrences do not appear among these rows, replaced by distinguished variables. The conditions on T guarantee that no conflicts arise, that is, no column needs more than one distinguished variable.

Basis. Height 0. The node is a leaf labeled by a relation scheme R , and identified with some row i . Row i cannot have a constant or distinguished variable in columns other than those for attributes in R . (We assume the relation scheme R containing all attributes in which row i has a distinguished variable, constant, or repeated nondistinguished variable is a legitimate operand. If not all relation schemes are permissible, then it is easy to check for tableaux with illegal rows.)

If row i has constant c in column A , then apply $\sigma_{A=c}$ to R . Then project onto those attributes in whose column row i has either a distinguished variable or a repeated nondistinguished variable. The result is an expression whose tableau has one row, which is row i with repeated nondistinguished variables replaced by distinguished variables.

Induction. We construct the expression for node n from the expressions E_1, E_2, \dots, E_m representing the children of node n . Begin by joining E_1, E_2, \dots, E_m . Then project onto those attributes A in whose columns either

- (1) some descendant leaf of n is a row with a distinguished variable in column A , or
- (2) some, but not all, of the occurrences of the repeated nondistinguished variable in column A are found in rows which are descendants of n .

The result is easily seen to satisfy the inductive hypothesis, as in each column A , variables that appear in two or more of the E_i 's and that must become the repeated nondistinguished variable for column A are distinguished in the tableaux for the E_i 's and are therefore equated in the join.

Finally, the inductive hypothesis applied to the root implies that the constructed expression has tableau T . \square

The proof of Theorem 5 contains the algorithm to construct an expression from a tableau. The following example illustrates the procedure.

Example 6. Consider the tableau

	A	B	C	D
		a_1	a_2	a_3
1	0	b_1	b_2	b_3
2	b_4	b_1	a_2	b_5
3	b_6	b_7	a_2	a_3
4	b_8	a_1	a_2	b_9

We assume that rows 1, 2, 3 and 4 come from relation schemes (AB) , (BC) , (CD) and (BC) , respectively. Since b_1 is the only repeated nondistinguished variable in the tableau, the only constraints are $(1, 2) < (1, 4)$ and $(1, 2) < (2, 4)$. Applying the procedure BUILD to these constraints we obtain the tree shown in Fig. 6.

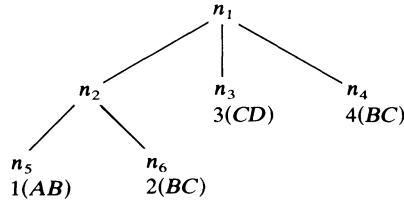


FIG. 6. Initial tree.

Using the construction in the basis for node n_5 we obtain the expression $\pi_B(\sigma_{A=0}(AB))$, while for the leaves n_6, n_3 and n_4 , the expressions (BC) , (CD) and (BC) suffice. Using the construction in the inductive step for node n_2 we obtain the expression $\pi_C(\pi_B(\sigma_{A=0}(AB)) \bowtie (BC))$. We project onto column C because only column C has a distinguished variable in rows 1 or 2, and all occurrences of the repeated nondistinguished variable b_1 are found in these rows. The expression for n_1 is obtained by joining the above expression with (CD) and (BC) in any order. We should then project onto BCD , but this projection is seen to be superfluous, since the final join produces a relation over only those attributes. The parse tree for the final expression is shown in Fig. 7. \square

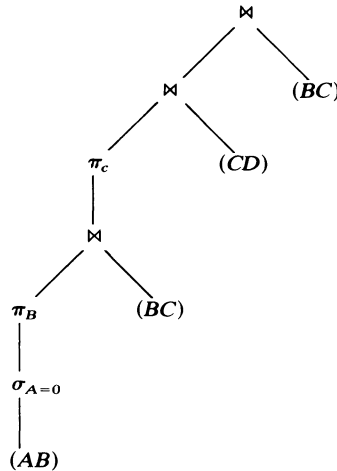


FIG. 7. Synthesized expression.

Extension to the case where constants appear in the summary. If there is a constant c in column A of the summary of a tableau, then c must appear in some row of column A , else the tableau can be shown not to come from an expression. If c appears in only one row k of column A , we may treat c as a distinguished variable, introducing the constraints $(i, j) < (i, k)$ and $(i, j) < (j, k)$ for each pair of rows i and j having a repeated nondistinguished variable in column A .

If c appears in more than one row of column A , we have choices. For all but one of these rows we can select $A = c$ and then project A out. But for one row we must treat c as if it were distinguished, and introduce the appropriate constraints. Since we may have a choice for each column that has a constant in the summary, we apparently have a combinatorial problem. However, we may adopt the simple expedient of permitting a new operator *augment*, defined by $\alpha_{A=c}(r) = \{\mu \mid \mu(A) = c \text{ and there exists } v \text{ in } r \text{ such that for all attributes } B \text{ on which } r \text{ is defined, } \mu(B) = v(B)\}$. Note that we may assume that r is not defined on A ; otherwise *augment* is the same as *select*. Then we may synthesize an expression from a tableau with constants in the summary by:

- (1) deleting constants from the summary,
- (2) synthesizing an expression for the resulting tableau, if it exists, and then
- (3) using the *augment* operator to introduce the constants into the tuples of the relation resulting from application of the expression from (2).

6. Finding a minimal expression equivalent to a given tableau. We should also consider a variant of the problem of finding an expression that yields a given tableau. In most circumstances it will be sufficient to find an expression that yields an equivalent tableau, that is, an expression defining the same mapping from universal instances to target relations as the tableau.

It turns out that this question is no harder than the original problem, as we can show that a tableau comes from an expression only if its minimal equivalent tableau does. Thus, if we minimize the number of rows in a tableau, then we can obtain an expression with the fewest joins equivalent to a given tableau since the number of rows in the tableau is one more than the number of joins in the resulting expression.

THEOREM 6. *If a tableau comes from an expression, then its minimal equivalent tableau comes from an expression.*

Proof. It follows from [4] that any given tableau T has a minimal equivalent tableau T' (one with the fewest number of rows) such that each row of T' is a row of T . Given an expression E yielding tableau T , we can delete the operands of E corresponding to rows of T that are not in T' .

Nodes that apply unary operators (*select* and *project*) to a deleted operand, and nodes that join two deleted operands are themselves deleted. Nodes that join a deleted operand with one that is not deleted are identified with the nondeleted operand. The resulting expression will yield tableau T' . We may prove by an easy induction on the

	A	B	C	D
	a_1	a_2	a_3	
1	a_1	b_1	b_2	b_3
2	b_4	b_1	b_5	b_6
3	a_1	a_2	b_7	b_6
4	b_8	a_2	a_3	b_9

FIG. 8. A tableau.

height of a node remaining in the expression that the tableau for this node is the same as the tableau for the corresponding node in E , with rows that do not eventually become rows in T' deleted. \square

Example 7. The tableau in Fig. 8 implies constraints $(1, 2) < (1, 3)$, $(1, 2) < (2, 3)$, $(1, 2) < (1, 4)$, and $(1, 2) < (2, 4)$, so we may synthesize the expression of Fig. 9. The minimal equivalent tableau for Fig. 8 has only rows 3 and 4. To synthesize an expression for the minimal tableau, we delete nodes n_8 and n_9 , which correspond to rows 1 and 2. This causes nodes n_7 and n_5 to be deleted and node n_4 to be merged with n_6 . The resulting expression is shown in Fig. 10.

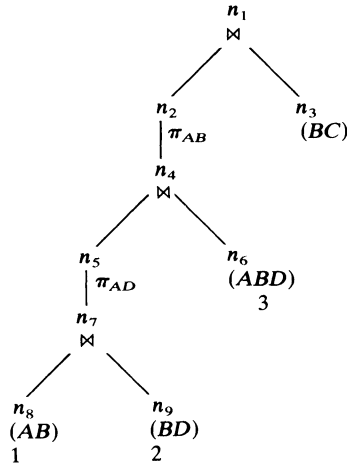


FIG. 9. An expression for the tableau of Fig. 8.

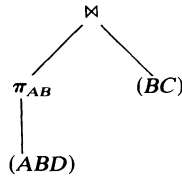


FIG. 10. Expression for minimal tableau.

7. Summary. In this paper we have presented an $O(n^2 \log n)$ algorithm that will construct, whenever possible, a tree to satisfy n given constraints on common ancestors of a set of nodes. The constraints specify lineage restrictions on lowest common ancestors of pairs of nodes. We have also shown that the tree construction algorithm can be used as part of a process to minimize the number of joins in a certain class of relational algebra expressions containing only select, project and join operators.

REFERENCES

- [1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [2] A. V. AHO, Y. SAGIV AND J. D. ULLMAN, *Efficient optimization of a class of relational expressions*, ACM Trans. Database Systems, 4 (1979), pp. 435-454.
- [3] ———, *Equivalences among relational expressions*, this Journal, 8 (1979), pp. 218-246.

- [4] A. K. CHANDRA AND P. M. MERLIN, *Optimal implementation of conjunctive queries in relational databases*, Proc. Ninth Annual ACM Symposium on Theory of Computing 1976, pp. 77–90.
- [5] E. F. CODD, *A relational model for large shared data banks*, Comm. ACM, 13 (1970), pp. 377–387.
- [6] P. J. DOWNEY, R. SETHI AND R. E. TARJAN, *Variations on the common subexpression problem*, J. Assoc. Comput. Mach., 27 (1980), pp. 758–771.
- [7] J. E. HOPCROFT AND J. D. ULLMAN, *Set merging algorithms*, this Journal, 2 (1973), pp. 294–303.
- [8] P. G. SELINGER, M. M. ASTRAHAN, D. D. CHAMBERLIN, R. A. LORIE AND T. G. PRICE, *Access path selection in a relational database management system*, Proc. ACM SIGMOD Conference on Management of Data, Boston, MA, 1979, pp. 23–24.
- [9] J. D. ULLMAN, *Principles of Database Systems*, Computer Science Press, Potomac, MD, 1980.
- [10] E. WONG AND K. YOUSSEFI, *Decomposition—a strategy for query processing*, ACM Trans. Database Systems, 1 (1976), pp. 223–241.
- [11] M. YANNAKAKIS AND C. PAPADIMITRIOU, *Algebraic dependencies*, Proc. 21st Annual IEEE Symposium on Foundations of Computer Science, 1980, pp. 328–332.
- [12] S. B. YAO, *Optimization of query evaluation algorithms*, ACM Trans. Database Systems, 4 (1979), pp. 133–155.

OPTIMAL MULTI-WAY SEARCH TREES*

L. GOTLIEB†

Abstract. Given a set of N weighted keys, $N + 1$ missing-key weights and a page capacity m , we describe an algorithm for constructing a multi-way lexicographic search tree with minimum cost. The program runs in time $O(N^3m)$ and requires $O(N^2m)$ storage locations. If the missing-key weights are zero, the time can be reduced to $O(N^2m)$. A further refinement enables the factor m in the above costs to be replaced by $\log m$.

Key words. algorithms, optimal weighted search trees, dynamic programming

1. Introduction. In this paper, we consider the problem of constructing a search tree for use in secondary storage. Because the access cost of the external memory is high compared to internal processing speeds, keys are grouped for searching into storage units called *pages*. Given a set of N keys with weights $\{p_i\}$, $N + 1$ missing-key weights $\{q_j\}$, and a page capacity m , we show how to construct a multi-way search tree which minimizes the expected number of pages accessed during a search. Section 2 sets up the problem and reviews Knuth's dynamic programming technique for finding optimal binary search trees [1]. Section 3 extends this method to produce an algorithm for constructing optimal multi-way trees. The running time is $O(N^3m)$, and $O(N^2m)$ storage locations required. If the q weights are absent, a "monotonicity" property (expressed by Theorem 2, which is proved in § 6) enables the running time to be reduced to $O(N^2m)$; however a counterexample presented in § 5 shows that this property need not hold if any of the q 's are positive. § 4 outlines a refinement whereby the factor m in the above time and storage costs can be replaced by $\log m$.

Variants of the problem considered here have been addressed by Itai [4] and by Wood et al. [6]. Itai shows how to construct optimal multi-way *code* trees, that is, trees in which the weights are confined to appear at the leaves only. Recently, Wood and his colleagues have shown how to construct optimal multi-way search trees with a height constraint, and also how to optimize multi-way trees when the cost of searching is defined as a combination of page accesses and the time required to search within a page. Their results have been derived independently of this work.

2. The problem. We are given keys $K_1 < K_2 < \dots < K_N$, a page capacity $m \geq 1$, and nonnegative weights $\{p_1, \dots, p_N, q_0, \dots, q_N\}$, where p_i/W is the probability that K_i is the search argument, q_j/W is the probability of a search between K_j and K_{j+1} , $1 \leq j < N$, and W is the total weight, $p_1 + \dots + p_N + q_0 + \dots + q_N$. (q_0/W is the probability that the search argument precedes K_1 , q_N/W that it follows K_N).

The problem is to construct an $m + 1$ -way search tree (an example of which is illustrated in Fig. 1), which minimizes the cost

$$\sum_{i=1}^N p_i \cdot \text{level}(p_i) + \sum_{j=0}^N q_j \cdot (\text{level}(q_j) - 1).$$

Fig. 1 shows an optimal 4-way tree ($m = 3$) for the 15 most common names in the Canadian census. The number accompanying each name (the p -weight) is the number of

* Received by the editors November 24, 1977, and in final revised form June 6, 1980.

† Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A7. Present address, Gellman, Hayward and Partners Ltd., Calgary, Canada T2G 0G3. This research was supported in part by the National Research Council of Canada.

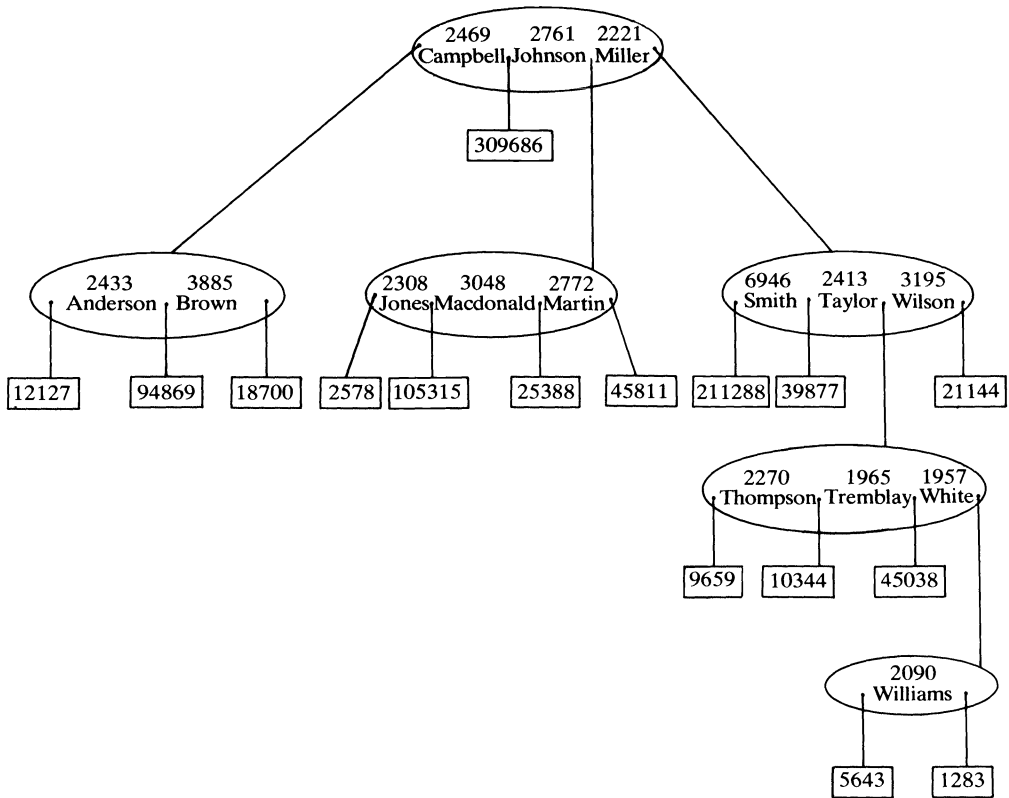


FIG. 1. Optimal 4-way search tree on 15 most common names in Canadian census.

times it occurred in a list of over a million names, while the numbers in the leaves (the q weights) represent those list members falling alphabetically between pairs from the top 15; thus there were 309,686 names (with repetitions) between “Campbell” and “Johnson”, and 12, 127 preceding “Anderson”. Thus Level (p_i) is actually the level of K_i , while level (q_j) is the level of a node representing a missing-key range.

Two points about Fig. 1 should be noted. First, each page which has pages (i.e., nodes with names) as descendants must be filled, or the cost could be reduced by promoting names up to the unfilled pages. In other words, in an optimal tree, only leaf pages may be unfilled. Second, the search cost of a q -weight is not the level of the leaf labeled with that weight, but rather the level of the page which is the father of that weight; for example, the cost of the leaf weighted 309,686 is 309,686 times the level of the root, since the search for a name between “Campbell” and “Johnson” would stop at the root page. Thus one is subtracted from level (q_j) in the expression for the cost of the tree. The level of the root is taken to be one, since we want the level of a page to reflect its access cost.

For $m = 1$ (binary branching), a dynamic programming approach, employing the principle that subtrees of an optimal tree must also be optimal, yields an $O(N^2)$ construction algorithm [1]. Let $t(i, j)$ denote an optimal tree on weights $\langle q_i, p_{i+1}, q_{i+1}, \dots, p_j, q_j \rangle$ (henceforth abbreviated $\langle i, j \rangle$), $c(i, j)$ be the cost of this tree, and $w(i, j)$ be its weight, $p_{i+1} + \dots + p_j + q_i + \dots + q_j$. The idea is to compute $c(0, N)$, starting with

$$c(i, i) = 0, \quad 0 \leq i \leq N,$$

and using

$$(1) \quad c(i, j) = w(i, j) + \min_{i < h \leq j} \{c(i, h - 1) + c(h, j)\}, \quad 0 \leq i < j \leq N.$$

Each time (1) is performed, the key corresponding to an h which gives a minimum is selected as $r(i, j)$, the root of $t(i, j)$. The computational cost is proportional to

$$\sum_{j=1}^N \sum_{i=0}^{j-1} (j - i),$$

which is roughly $N^3/6$, and N^2 storage locations are needed for the $c(i, j)$'s and $r(i, j)$'s. The running time can be further reduced by using Knuth's observation [1] that

$$r(i, j - 1) \leq r(i, j) \leq r(i + 1, j);$$

that is, one need not move left in the key set, looking for a new root when a key is added following the rightmost key in the tree, and vice-versa. This makes it possible to restrict the range of h in (1), thereby reducing the running time to

$$(2) \quad \sum_{j=1}^N \sum_{i=0}^{j-1} (r(i + 1, j) - r(i, j - 1) + 1).$$

(To make this summation correct when $j - i = 1$, we adopt the convention that $r(i, i) = 0$, $0 \leq i \leq N$; in fact, (1) is not needed when $j - i = 1$, since by definition, $c(i, i + 1) = w(i, i + 1)$, and $r(i, i + 1) = i + 1$.) After cancelling terms, the sum reduces to

$$\frac{N(N + 1)}{2} + \sum_{i=1}^N (r(i, N) - r(0, N - i)) \approx N^2,$$

since $r(i, N) \leq N$ and $r(0, N - i) > 0$, so the time is $O(N^2)$.

The algorithm for optimal $m + 1$ -way trees, $m > 1$, is basically an extension of the above technique. Let

$$i < r(i, j, 1) < \dots < r(i, j, m) \leq j$$

be the (indices of the) keys on the root page of the optimal $m + 1$ -way tree $t(i, j)$ with cost $c(i, j)$ and weight $w(i, j)$. We have

$$c(i, i) = 0, \quad 0 \leq i \leq N,$$

$$c(i, j) = w(i, j), \quad 1 \leq j - i \leq m,$$

and for $m < j - i \leq N$, we must find an assignment for $r(i, j, k)$ such that

$$c(i, j) = w(i, j) + c(i, r(i, j, 1) - 1) + \sum_{k=1}^{m-1} c(r(i, j, k), r(i, j, k + 1) - 1) + c(r(i, j, m), j)$$

is a minimum.

The problem is carrying out this minimization. For each $r(i, j)$, there are $\binom{j-i}{m}$ choices for the root page, and when $j - i = d$, there are $N - d + 1$ $c(i, j)$'s and $r(i, j)$'s to determine, namely for $(i, j) = \langle (0, d), (1, d + 1), \dots, (N - d, N) \rangle$. Since $j - i$ ranges from $m + 1$ to N , a brute force approach which considers each possible candidate for the root page of $t(i, j)$ would take on the order of $\binom{N}{m}$ steps. This cost is too large to be practical, so a way of reducing the number of potential root pages per subtree is needed.

3. A dynamic programming solution. Let $\mathbf{T}(i, j, k)$, $1 \leq k \leq m$, denote the set of optimal k -rooted trees on weights $\langle i, j \rangle$, that is, optimal trees in which

i) there are *exactly* k keys on the root page,

and

ii) the $k + 1$ subtrees of the root page are (optimal) $m + 1$ -way trees.

The basis for our construction algorithm is an observation similar to that made by Itai for code trees [4]. We note that the cost of a search tree, expressed iteratively as $\sum_i p_i \cdot \text{level}(p_i) + \sum_j q_j \cdot (\text{level}(q_j) - 1)$, can also be expressed recursively, in a form which makes it clear how to carry out minimization. For example, the cost of the tree in Fig. 1 can be written

$$\begin{aligned} & \text{Cost (2-rooted tree on } \langle \text{"Anderson"}, \dots, \text{"Martin"} \rangle) \\ & + \text{Weight ("Miller")} + \text{Weight (Right Subtree of "Miller")} \\ & + \text{Cost (Right Subtree of "Miller")}. \end{aligned}$$

Now for the tree to be optimal, the right subtree of "Miller" must be optimal, as must the 2-rooted tree to the left; otherwise the total cost could be reduced by finding better trees on the same key sets. Moreover, the choice of "Miller" as the rightmost key on the root page must produce a cost no greater than any other (optimal 2-rooted tree, rightmost root key, optimal 4-way subtree) combination. Therefore, given all optimal 4-way and 2-rooted trees for proper subsequences of $\langle \text{"Anderson"}, \dots, \text{"Williams"} \rangle$, the problem of finding the optimal 4-way tree on the whole sequence is reduced to that of choosing the rightmost key of the root page (i.e., that which gives the smallest cost when substituted into the above formula).

More generally, let $c(i, j, k)$ be the cost of $t(i, j, k) \in \mathbf{T}(i, j, k)$. Then for $j - i \geq m$,

$$c(i, j, m) = \min_{i+m \leq h \leq j} \{c(i, h-1, m-1) + p_h + w(h, j) + c(h, j, m)\}.$$

In other words, $t(i, j, m)$ is the smallest cost concatenation of an optimal $m - 1$ -rooted tree on the left, together with a single root and its (right) $m + 1$ -way subtree. (The weight of the right subtree, $w(h, j)$, must be added in because the tree starts at level two, and therefore its stand-alone cost, $c(h, j, m)$, does not reflect its access cost as a subtree.) Similarly, $t(i, j, m - 1)$ can be determined from optimal $m - 2$ -rooted and m -rooted subtrees, and in general, for $2 \leq k \leq m$, $j - i \geq k$,

$$(3) \quad c(i, j, k) = \min_{i+k \leq h \leq j} \{c(i, h-1, k-1) + p_h + w(h, j) + c(h, j, m)\}.$$

(Note that $i + k$ is the lower bound, since the set of candidates for the rightmost root of a k -rooted tree starts at the k th key from the left.)

The computation of $c(i, j, 1)$ is slightly different since no concatenation of trees is involved; rather a single root is chosen to minimize the cost of left and right subtrees. We have, for $i < j$,

$$(4) \quad c(i, j, 1) = w(i, j) + \min_{i < h \leq j} \{c(i, h-1, m) + c(h, j, m)\}$$

which is similar to (1), except that the subtrees are $m + 1$ -way, rather than binary.

(The equivalence of the recursive formulation of cost, given by (3) and (4), with the iterative expression $\sum_i p_i \cdot \text{level}(p_i) \dots$ etc.) is easy to show. For trees with maximum height one, (4) is trivially correct, and (3) is established by induction on k . Then, assuming the equivalence for trees with maximum height less than h , (4) is proved for trees with maximum height h , and (3) follows, again by induction on k .)

Assuming then, that $c(i, j, k)$ has been computed for $1 \leq k \leq m$ and $j - i < d$, we can compute $c(i, j, k)$ for $1 \leq k \leq m$ and $j - i = d$, starting with (4) to get $c(i, j, 1)$, and followed by successive applications of (3) with $k = 2, 3, \dots, m$. (In the final step when $c(0, N, m)$ is to be determined, it is only necessary to use (3) once, since at this point optimal trees with fewer than m keys on the root page are not needed; however the savings are negligible.) The timing analysis is essentially the same as for the binary algorithm: the cost of (4) is proportional to

$$\sum_{j=1}^N \sum_{i=0}^{j-1} (j-i) \approx N^3/6,$$

while for $2 \leq k \leq m$, (3) takes

$$\sum_{j=k}^N \sum_{i=0}^{j-k} (j-i-k+1) \approx \frac{(N-k+1)^3}{6} \text{ steps.}$$

The total cost is therefore approximately

$$\sum_{k=1}^m \frac{(N-k+1)^3}{6},$$

which is $O(N^3 m)$.

For $2 \leq k \leq m$, let $r(i, j, k)$ be the largest value of h that gives a minimum in (3), and define $r(i, j, 1)$ similarly for (4); $r(i, j, k)$ is simply the largest possible key on a root page of $T(i, j, k)$. In §§ 5 and 6, we show two facts about $r(i, j, k)$:

1) During construction of an optimal k -rooted tree, the rightmost key in the root can move left when a key is added at the right (alphabetically high) end of the key set. In particular, $r(i, j, k)$ may be less than $r(i, j-1, k)$ (§ 5).

2) If the missing-key weights (q 's) are all zero, then the following *monotonicity property* holds (Theorem 2, § 6):

$$r(i, j-1, k) \leq r(i, j, k) \leq r(i+1, j, k).$$

Point (1) means that, in contrast with the situation for binary trees, $r(i, j, k)$ can lie outside $[r(i, j-1, k), r(i+1, j, k)]$ and therefore, when q 's are present, the speed up in running time from N^3 to N^2 cannot be applied. When the q 's are absent, (2) restricts the search for the minimum in (3) and (4), making the total cost proportional to

$$\sum_{k=1}^m \sum_{j=k}^N \sum_{i=0}^{j-k} (r(i+1, j, k) - r(i, j-1, k) + 1).$$

(Again, it is convenient to let $r(i, j, k) = 1$ if $j - i < k$.) For each k , the inner double sum is equivalent to (2) above (in fact, slightly smaller when $k > 1$), which is bounded by N^2 ; hence the running time is $O(N^2 m)$.

The storage requirement is $O(N^2 m)$ locations, since $r(i, j, k)$ and $c(i, j, k)$ are N by N by m arrays. To see how the desired tree, $t(0, N, m)$, is reconstructed from the information in r , note that the keys on the root page of $t(0, N, m)$ are given by the following sequence:

$$\begin{aligned} r_1 &= r(0, N, m) \\ r_k &= r(0, r_{k-1} - 1, m - k + 1), \quad \text{for } 2 \leq k \leq m. \end{aligned}$$

In other words, the rightmost key r_1 is $r(0, N, m)$, the key second from the right is the rightmost root key for the weights $\langle 0, r_1 - 1 \rangle$ and so on. The root pages of the subtrees of the root page, and hence ultimately the whole tree, are similarly determined.

An algorithm which uses (3) and (4) to compute optimal $m + 1$ -way search trees on weights $\langle q_0, p_1, \dots, p_N, q_N \rangle$ is presented in [5].

4. A refinement. The time and space costs of the above algorithm can be reduced using a technique employed by Itai for the construction of optimal multi-way code trees [4, § 7]. He observed that an optimal m -way code tree is the concatenation of an optimal collection of s m -way trees, together with an optimal collection of $m-s$ trees. Here we cannot concatenate trees in the same way code trees are combined (there would be one subtree too many), but we can join them with a middle root; for example the tree of Fig. 1 can be regarded as the join of two optimal 1-rooted trees (with roots "Campbell", "Miller"), together with the key "Johnson". Thus, for $u + v = k - 1$, $k > 2$,

$$(5) \quad c(i, j, k) = \min_{i+u < h \leq j-v} \{c(i, h-1, u) + p_h + c(h, j, v)\}.$$

This makes it possible to determine $c(i, j, m)$ without computing all the intermediate costs, $c(i, j, 1), c(i, j, 2), \dots, c(i, j, m-1)$. For example, suppose $c(i, j, k)$ are available for $k \in \langle 1, 3, 7, 11, 12 \rangle$ and $j - i < d$; then for $j - i = d$, $c(i, j, k)$ can be derived for the same set of k 's by the following steps:

- (1) use (4) to get $c(i, j, 1)$,
- (2) use (5) with $u = v = 1$ to get $c(i, j, 3)$,
- (3) use (5) with $u = v = 3$ to get $c(i, j, 7)$,
- (4) use (5) with $u = 7, v = 3$ to get $c(i, j, 11)$
- (5) use (3) with $k = 12$ to get $c(i, j, 12)$.

More generally, let s_0, s_1, \dots, s_l be any sequence which satisfies

$$(6) \quad s_0 = 0, s_l = m,$$

and

$$(7) \quad s_h = 1 + s_i + s_j, \quad 0 \leq i \leq j \leq h \leq l.$$

For $1 \leq h \leq l$, let $\text{left}(h)$ and $\text{right}(h)$ be, respectively, the i and j for which (7) holds, that is, $s_h = s_{\text{left}(h)} + s_{\text{right}(h)} + 1$. Finally, suppose $c(i, j, s_h)$ has been computed for $h \in \langle 1, 2, \dots, l \rangle$ and $j - i < d$. Then the following rule is used to decide which of (3), (4) or (5) to apply when computing $c(i, j, s_h)$ for $j - i = d$ and $h = 1$ to l :

- if** $\text{left}(h) = 0$ **then** use (4)
else if $\text{right}(h) = 0$ **then** use (3) with $k = s(h)$
else use (5) with $u = \text{left}(h)$ and $v = \text{right}(h)$.

Sequences for which (6) and (7) hold are closely related to addition chains. An *addition chain of length l* for $n > 1$ is a sequence of integers $1 = a_0, a_1, \dots, a_l = n$, such that $a_h = a_i + a_j$, $0 \leq i \leq j < h \leq l$. Given an addition chain for $n = m + 1$, note that the sequence $\langle a_0 - 1, a_1 - 1, \dots, a_l - 1 \rangle$ satisfies (6) and (7), since

$$a_0 - 1 = 0, \quad a_l - 1 = m,$$

and

$$a_h - 1 = (a_i + a_j) - 1 = 1 + (a_i - 1) + (a_j - 1), \quad 1 \leq h \leq l.$$

Let $l(n)$ be the length of the shortest possible addition chain for $n > 1$. Given such a chain, it is possible to construct an optimal $m + 1$ -way tree in time $O(N^3 l(m + 1))$. As it turns out, neither a minimal chain nor $l(n)$ are easy to compute efficiently for n in general [2, § 4.6.3]; however the *binary* addition chain for n is easily derived from the binary representation of that number, and has at most $2 \lfloor \log n \rfloor$ terms $((1, 2, \dots, 2^{\lfloor \log n \rfloor})$ plus a term for each “one” bit except the high order one). Thus the sequence of k 's for which $c(i, j, k)$ must be computed need be no longer than $2 \lfloor \log m + 1 \rfloor$, which makes possible an $O(N^3 \log m)$ construction algorithm (or $O(N^2 \log n)$, when the q 's are absent).

5. Failure of monotonicity property in the general case. In this section we show that, during construction of an optimal multi-way search tree, the rightmost key in the root can move left when a key is added at the right. Consider a set of 8 keys, $\{A, B, \dots, H\}$, each weighted one, together with 9 missing-key weights q , all equal to one except for $q(8)$ (between G and H), which is 4. For convenience, these weights are unnormalized; to get the actual probabilities, it is necessary to divide each weight by the total. Thus the probability of a search for A is $\frac{1}{20}$.

Fig. 2(b) shows a 3-way ($m = 2$) tree which is optimal for these weights. The rightmost key in the root is F ; it is also $r(0, 8, 2)$, the largest possible rightmost root, since no other 3-way tree costs as little for this data. Now the algorithm outlined in § 3 requires an optimal tree on $\langle A, G \rangle$ before it can determine one on $\langle A, H \rangle$, and such a tree is illustrated in Fig. 2(a). This tree is also uniquely optimal for the weights given, which makes G , the rightmost key in the root, equal to $r(0, 7, 2)$. Thus the rightmost root key shifts left from G , on keys $\langle A, G \rangle$, to F , on keys $\langle A, H \rangle$, and in particular, $r(0, 8, 2) \leq r(0, 7, 2)$, which shows that $r(i, j, k)$ may lie outside the interval $[r(i, j - 1, k), r(i + 1, j, k)]$. (The number of keys in the root of the tree in Fig. 2 is not critical to the example, and thus the result is true for optimal k -rooted trees as defined at the beginning of § 3.)

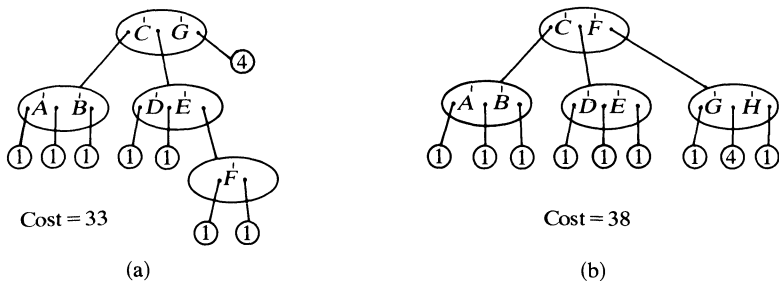


FIG. 2. Optimal trees on $\langle A, G \rangle$ and $\langle A, H \rangle$.

The example just presented affects an assumption made by Itai regarding the time needed to construct optimal multi-way code trees [4, § 7]. To see how, observe that if the internal weights are ignored, the trees in Fig. 2 are 3-way code trees. The one in (a) is the join of two 3-way trees together with the leaf weighted “4”, while (b) is formed by concatenating two 3-way trees with a rightmost tree consisting of weights $(1, 4, 1)$. The last weight before the rightmost tree in the least-cost concatenation is called the *breakpoint*. For code trees, Itai assumed the equivalent of the monotonicity property, namely that when a weight is added at the right, a breakpoint greater than or equal to the previous one can always be found. But Fig. 2 shows that this need not be true, since the breakpoint in (a) is the 7th leaf weight (following F), while it is the 6th (after E) in

(b). Hence the dynamic programming construction procedure which he outlined is $O(N^3 \log m)$, not $O(N^2 \log m)$ as claimed.

In [7] it is shown that, during construction, the rightmost root in a multi-way search tree or code tree can shift the maximum possible number of keys away from the added key. Thus the dependence on key set size of dynamic programming methods for multi-way tree construction presented so far is $O(N^3)$. However our counterexamples to the monotonicity property rely on the existence of nonzero leaf (q) weights; in the next section we show that when these weights are absent, the addition of a key at the right cannot force the rightmost key in the root to move left.

6. Proof of monotonicity property when q 's absent. The purpose of this section is to show that when $q_h = 0$, $0 \leq h \leq N$, $r(i, j, k)$, the largest key on the root page of all trees in $\mathbf{T}(i, j, k)$, satisfies

$$r(i, j-1, k) \leq r(i, j, k) \leq r(i+1, j, k), \quad k < j-i \leq N, \quad 1 \leq k \leq m.$$

The proof is a generalization of the one for binary trees outlined in [3, § 6.2.2]; however it should be noted that for binary trees, the result holds whether the q 's are zero or not. Note also that to stay consistent with the general case, we will use $\langle i, j \rangle$ to denote the weights $\langle p_{i+1}, \dots, p_j \rangle$, even though the absence of the q 's makes $\langle i+1, j \rangle$ more natural. The following definitions are needed:

DEFINITION 1. Define $A \triangleleft B$ to hold between nonempty sets of integers A, B if $a \in A$, $b \in B$ and $b < a \Rightarrow a \in B$ and $b \in A$. Informally, if A , regarded as a sequence, is "left" of B in terms of position, then $A \triangleleft B$ holds if "overlapping" members are common to both sets. The relation has the following property, which we state without proof:

LEMMA 1. \triangleleft is transitive.

DEFINITION 2. $R(i, j, k)$, $1 \leq k \leq m$, is the set of rightmost keys (henceforth called *rightmost roots*) on the root pages of trees in $\mathbf{T}(i, j, k)$; in other words, it is the set of $i+k \leq h \leq j$ for which (3) is minimized if $k > 1$, or (4) is minimized if $k = 1$.

DEFINITION 3. $L(i, j, k)$, $1 \leq k \leq m$, is the set of leftmost keys (*leftmost roots*) on the root pages of trees in $\mathbf{T}(i, j, k)$. We have $L(i, j, 1) = R(i, j, 1)$ for $1 \leq j-i \leq N$, and for $2 \leq k \leq m$, $k < j-i \leq N$,

$$L(i, j, k) = \{i < h \leq j-k+1 | c(i, h-1, m) + w(i, h-1) + p_h + c(h, j, k-1) \text{ is a minimum}\}$$

Proof outline. We first show that except for possible addition of the new key, the set of rightmost roots does not change when the existing key set is augmented at its right (alphabetically high) end by a key with weight zero (Lemma 2). The main theorem (Theorem 1) is proven in two parts. In Lemma 4, weights $\langle i, \dots, j-1 \rangle$ are fixed while p_j is increased from x to y ; in this case it is shown that there will always be a rightmost root \geq any key which is a rightmost root when $p_j = x$. Together with Lemma 2, this shows that $R(i, j-1, k) \triangleleft R(i, j, k)$, or, informally, that we need never move left in the key set to look for a new rightmost root when a key is added at the right. The equivalent result for leftmost roots is then used to show that the rightmost root does not have to move right when a key is added at the left, thus completing the proof. Finally, Theorem 1 is used to prove Theorem 2, the desired result.

LEMMA 2. Let $q_h = 0$, $0 \leq h \leq N$. If $p_i = 0$, then $R(i, j-1, k) = R(i, j, k) - \{j\}$, $k < j-i \leq N$.

Proof. Since $p_i = 0$, adding it to a tree in $\mathbf{T}(i, j - 1, k)$ cannot increase the cost, but cannot lead to a better rearrangement either, or removing it would then leave a better tree than the original, which was optimal. Therefore any tree in $\mathbf{T}(i, j - 1, k)$ can be converted to one in $\mathbf{T}(i, j, k)$ by straight insertion of p_i , which does not change the rightmost key in the root. A similar argument shows that any tree in $\mathbf{T}(i, j, k)$ in which p_i is not the rightmost root must also be in $\mathbf{T}(i, j - 1, k)$ when p_i is removed.

The main theorem to be proved is

THEOREM 1. *If $q_h = 0, 0 < h \leq N$, then*

$$R(i, j - 1, k) \leq R(i, j, k) \leq R(i + 1, j, k),$$

and

$$L(i, j - 1, k) \leq L(i, j, k) \leq L(i + 1, j, k),$$

for

$$k < j - i \leq N, 1 \leq k \leq m.$$

Proof. By induction on $j - i > k$.

Basis. If $j - i = k + 1$, then $R(i, j - 1, k) = \{j - 1\}$, $R(i + 1, j, k) = \{j\}$, and $R(i, j, k) \subseteq \{j - 1, j\}$. Similarly $L(i, j - 1, k) = \{i + 1\}$, $L(i + 1, j, k) = \{i + 2\}$, and $L(i, j, k) \subseteq \{i + 1, i + 2\}$; hence the theorem holds trivially.

Induction. Assuming that the theorem holds for $k < j - i < d < N, 1 \leq k \leq m$, we will show that it remains true for $j - i = d$.

LEMMA 3. *If $u - w < d$, and $u < v < w$, then $R(u, w, k) \leq R(v, w, k)$, and $L(u, v, k) \leq L(u, w, k)$.*

Proof. By the induction hypothesis and Lemma 1.

LEMMA 4. *Let $R_x(i, j, k)$ denote the set of rightmost roots when $p_i = x \geq 0$, and $R_y(i, j, k)$ be the corresponding set when $p_i = y > x$, while weights $\langle i, \dots, j - 1 \rangle$ remain fixed; then $R_x(i, j, k) \leq R_y(i, j, k)$.*

Proof. Let $r \in R_x(i, j, k)$, T_x be an optimal k -rooted tree with rightmost root r and $p_i = x$, and l_x be the level of p_i in T_x ; similarly for $s \in R_y(i, j, k)$, T_y , y and l_y . Suppose $s < r$. The lemma will be proved if we can modify T_x without changing its rightmost root so that it is optimal for y , and similarly for T_y with respect to x .

The cost of T_x can be expressed as a linear function of p_i , namely $a + l_x \cdot p_i$, and similarly, $\text{cost}(T_y) = b + l_y \cdot p_i$. Of the nine possible outcomes for a compared to b , l_x compared to l_y , all contradict the definitions of T_x and T_y except $a = b, l_x = l_y$, in which case the Lemma is proved, and $a < b, l_x > l_y$.

When $a < b$ and $l_x > l_y$, $\text{cost}(T_x)$ meets $\text{cost}(T_y)$ at $z > 0$, and Fig. 3 shows the two basic situations possible at this point. In 3(a), T_x is optimal for $p_i \leq z$, while T_y is optimal for $p_i \geq z$. In 3(b), neither is optimal at z , because for $x < p_i < y$, there are intermediate trees (two in the figure) which are optimal and cost less than either T_x or T_y . We consider the case of (a) first.

In tree T_x , let r_1, r_2, \dots, r_{l_x} be the sequence of rightmost roots along the path from the root page down to the page containing p_i , and let s_1, \dots, s_{l_y} be the corresponding sequence for T_y . Now $s = s_1 < r_1 = r$ by assumption, and $r_{l_y} < s_{l_y} = j$, since $l_x > l_y$, and $r_{l_x} = j$, the index of the last key; thus there must be a level h such that $r_h > s_h$ and $r_{h+1} < s_{h+1}$. Both T_x and T_y are optimal at z , so $r_{h+1} \in R_z(r_h, j, m)$ and $s_{h+1} \in R_z(s_h, j, m)$. Since $s_h < r_h < j$, and $j - s_h < j - i = d$, we can apply Lemma 3 to get $R_z(s_h, j, m) \leq R_z(r_h, j, m)$. But $r_{h+1} < s_{h+1}$, so by definition of \leq , it follows that $s_{h+1} \in R_z(r_h, j, m)$ and $r_{h+1} \in R_z(s_h, j, m)$.

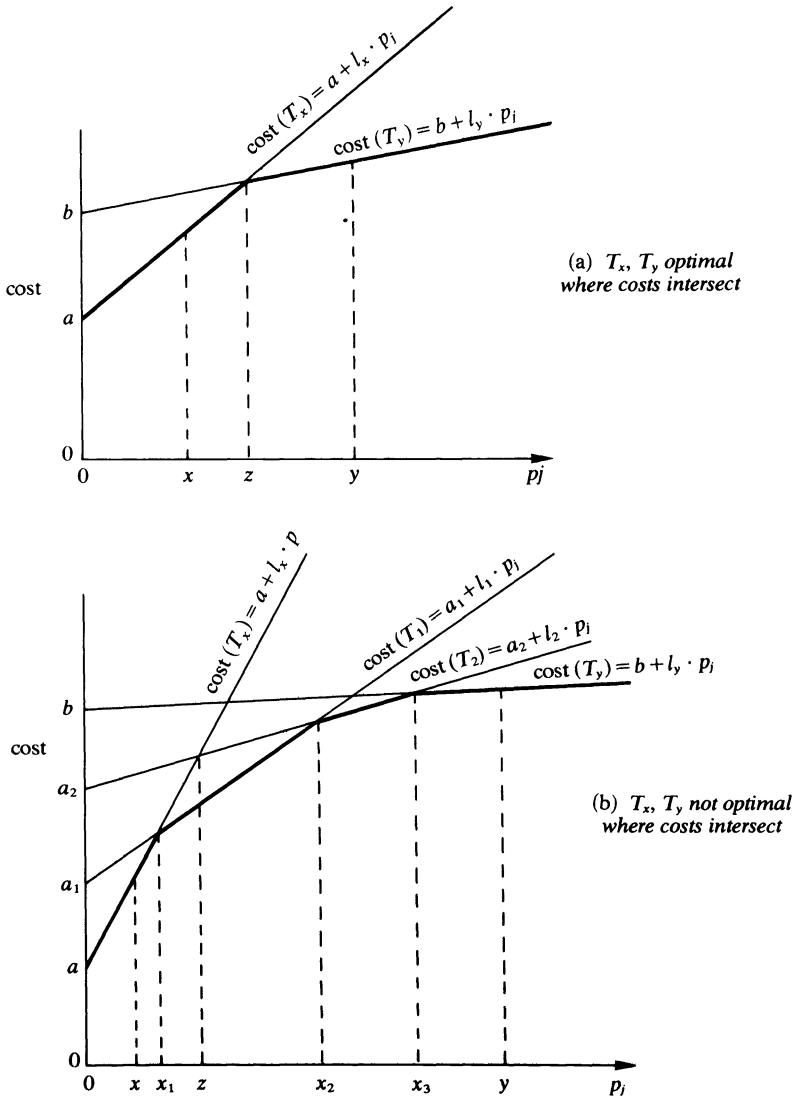


FIG. 3. Cost of T_x, T_y as functions of p_j .

In other words, there is a rearrangement of T_x , call it T'_x , which is optimal for $p_j = z$, and has s_{h+1} (at level $h + 1$) as the rightmost root of the subtree for keys to the right of r_h . Now the subtree to the right of s_{h+1} , containing p_j , can remain unchanged from T_y , since T_y is also optimal at z , and this subtree starts at the same level as it did in T_y . Thus the level of p_j in T'_x is l_y , and $\text{cost}(T'_x) = a' + l_y \cdot p_j$. But $\text{cost}(T_y) = b + l_y \cdot p_j$ and both agree at z ; therefore $a' = b$, and T'_x , with rightmost root r , is optimal for y . Similarly we can rearrange T_y , without changing its rightmost root s , to T'_y , which has the same cost at T_x . Thus $r \in R_y(i, j, k)$ and $s \in R_x(i, j, k)$.

If $\text{cost}(T_x)$ meets $\text{cost}(T_y)$ at a point where neither is optimal, then there is a sequence of trees $T_x = T_0, T_1, \dots, T_l = T_y$, and values $x = x_1, \dots, x_{l+1} = y$, such that T_I is optimal on $[x_I, x_{I+1}]$, $0 \leq I \leq l$. (Fig. 3(b) illustrates the situation for $l = 3$.) Let $rt(I)$ be the rightmost root of T_I ; for the sequence of roots $\langle rt(0), \dots, rt(l) \rangle$, we have $r = rt(0)$ and $rt(l) = s$. If $s < r$, an induction argument shows that the sequence can be transformed

so that s is the first term (i.e., $s \in R_x(i, j, k)$), and r is the last ($r \in R_y(i, j, k)$). The basis for sequences of length two was established above, and the induction step is straightforward. \square

From Lemma 2, $R(i, j-1, k) = R_x(i, j, k) - \{j\}$ if $x = 0$, and from Lemma 4, $R_x(i, j, k) \leq R_y(i, j, k)$ for $x < y$; hence $R(i, j-1, k) \leq R(i, j, k)$, as desired. By symmetry, the equivalent of Lemmas 2 and 4 for leftmost roots can be used to show that $L(i, j, k) \leq L(i+1, j, k)$, $1 \leq k \leq m$. Thus it remains to show

$$R(i, j, k) \leq R(i+1, j, k) \quad \text{and} \quad L(i, j-1, k) \leq L(i, j, k).$$

For $k = 1$ these are immediate, since $R(i, j, 1) = L(i, j, 1)$. We will show $R(i, j, k) \leq R(i+1, j, k)$ for $k > 1$.

Let $T1 \in \mathbf{T}(i, j, k)$ with rightmost root $r1$ and leftmost root $l1$, and similarly for $T2 \in \mathbf{T}(i+1, j, k)$, $r2$ and $l2$. Assume $r2 < r1$; $l1$ may be equal to, less than, or greater than $l2$, and Fig. 4 illustrates the three cases.

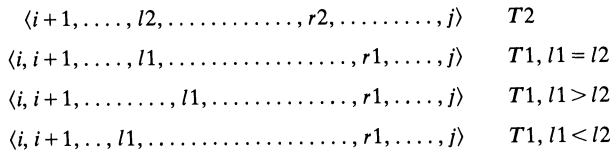


FIG. 4. Proving $R(i, j, k) \leq R(i+1, j, k)$.

If $l1 = l2$, then since $r2$ and $r1$ are both in $R(l1, j, k-1)$, the subtrees on $\langle l1, j \rangle$ in $T1$ and $T2$ can be switched; thus $r2 \in R(i, j, k)$ and $r1 \in R(i+1, j, k)$. Suppose $l1 > l2$. Since from above, $L(i, j, k) \leq L(i+1, j, k)$, there is a re-arrangement of $T2$ which is optimal but has $l1$ as its leftmost root. In this tree, the keys to the right of $l1$ can be arranged as they are in $T1$, giving $r1$ as the rightmost root. Thus $r1 \in R(i+1, j, k)$ and a similar re-arrangement of $T1$ gives $r2 \in R(i, j, k)$. Finally, suppose $l1 < l2$; then $R(l1, j, k-1) \leq R(l2, j, k-1)$ by Lemma 3, and since $r2 < r1$, $r2$ must also be in $R(l1, j, k-1)$ and $r1$ in $R(l2, j, k-1)$ by definition of \leq . Therefore the $k-1$ rooted tree to the right of $l1$, with rightmost root $r1$, can be replaced by one with the same cost and rightmost root $r2$, i.e., $r2 \in R(i, j, k)$; a similar replacement in $T2$ gives $r1 \in R(i+1, j, k)$.

A symmetric argument shows that $L(i, j-1, k) \leq L(i, j, k)$, $k > 1$, and thus completes the proof of the theorem. \square

The needed result is expressed by:

THEOREM 2. *If $q_h = 0$, $0 \leq h \leq N$, then $r(i, j, k)$, the largest possible rightmost root, satisfies*

$$r(i, j-1, k) \leq r(i, j, k) \leq r(i+1, j, k), \quad k < j-i \leq N, \quad 1 \leq k \leq m.$$

Proof. By induction on $j-i > k$.

If $j-i = k+1$, then $r(i, j-1, k) = j-1$, $r(i+1, j, k) = j$, and the only two possibilities for $r(i, j, k)$ are $j-1$ or j , so the theorem holds. Assuming that the theorem holds for $k < j-i < d < N$, $1 \leq k \leq m$, we will show that it remains true for $j-i = d$.

First, observe that from the induction hypothesis, $r(i, j-1, k) \leq r(i+1, j-1, k) < r(i+1, j, k)$. To show the left inequality, suppose $r \in R(i, j, k) < r(i, j-1, k)$. Then since $r(i, j-1, k) \in R(i, j-1, k)$ and $R(i, j-1, k) \leq R(i, j, k)$ by Theorem 1, $r(i, j-1, k) \in R(i, j, k)$, that is, the rightmost possible root in $T(i, j, k)$ is $\geq r(i, j-1, k)$.

To show $r(i, j, k) \leq r(i+1, j, k)$, suppose $r \in R(i, j, k) > r(i+1, j, k) \in R(i+1, j, k)$. By Theorem 1 we have $R(i, j, k) \leq R(i+1, j, k)$, so $r \in R(i+1, j, k)$. But r is greater than $r(i+1, j, k)$, which by definition is the largest rightmost root, a contradiction. Therefore any member of $R(i, j, k)$, and in particular, $r(i, j, k)$, is $\leq r(i+1, j, k)$. \square

7. Conclusion. The algorithm outlined in § 3 employs “double” dynamic programming—building optimal trees on increasingly larger key sets, and at the same time building trees with successively more keys on the root page. The technique is powerful: even when the monotonicity property does not hold, it reduces the number of possible root pages from $\binom{N}{m}$ to N , which is independent of page capacity. In [5], the method is used to construct optimal weighted B-trees. Since these have the property that a page may have anywhere from the minimum number of keys allowable to the maximum, the basic $O(N^3 m)$ algorithm is not completely supplanted by the refinement of § 4.

Acknowledgments. I would like to thank my advisor, Prof. John D. Lipson, for advice and encouragement.

REFERENCES

- [1] D. E. KNUTH, *Optimum binary search trees*, Acta Informat., 1 (1971), pp. 14–25; *Errata*, 1 (1972), p. 270.
- [2] ———, *The Art of Computer Programming*, vol. 2, *Seminumerical Algorithms*, Addison-Wesley, Reading, MA., 1969, Section 4.6.3.
- [3] ———, *The Art of Computer Programming*, vol. 3, *Sorting and Searching*, Addison-Wesley, Reading, MA., 1973.
- [4] ALON ITAI, *Optimal alphabetic trees*, this Journal, 5 (1976), pp. 9–18.
- [5] L. GOTLIEB, *Optimal Multi-way Search Trees*, TR 128/78, Department of Computer Science, University of Toronto, 1978.
- [6] V. K. VAISHNAVI, H. P. KRIEGEL AND D. WOOD, *Optimal Multi-way Search Trees*, TR 78-CS-13, Dept. of Applied Mathematics, McMaster University, Hamilton, Ontario, 1978.
- [7] L. GOTLIEB AND D. WOOD, *The construction of multiway search trees and the monotonicity principle*, Internat. J. Comput. Math., to appear.

PARTIAL AND TOTAL MATRIX MULTIPLICATION*

A. SCHÖNHAGE†

Abstract. In 1979 considerable progress was made in estimating the complexity of matrix multiplication. Here the new techniques and recent results are presented, based upon the notion of approximate rank and the observation that certain patterns of *partial* matrix multiplication (some of the entries of the matrices may be zero) can efficiently be utilized to perform multiplication of large *total* matrices. By combining Pan's trilinear technique with a strong version of our *compression* theorem for the case of several disjoint matrix multiplications it is shown that multiplication of $N \times N$ matrices (over arbitrary fields) is possible in time $O(N^\beta)$, where β is a bit smaller than $3 \ln 52 / \ln 110 \approx 2.522$.

Key words. computational complexity, matrix multiplication, tensor rank, trilinear aggregating, exponent

Introduction. In our model of computation for the multiplication of matrices $A = (a_{\nu,\mu})$, $B = (b_{\mu,\nu})$ we consider the a 's and b 's as indeterminates over some scalar field F . Let $\text{mam}(N|F)$ denote the total number of arithmetic operations $+$, $-$, $*$ in $F[a \cdots, b \cdots]$ needed in a minimal straight-line program for the computation of the elements of AB , where A and B are $N \times N$ matrices. Then

$$(0.1) \quad \omega(F) = \inf \{ \beta \mid \text{mam}(N|F) = O(N^\beta) \}$$

is called the exponent of matrix multiplication over F .

We will also need the noncommutative bilinear model, where $m^*(N|F)$ denotes the minimal number of products [linear combination of the a 's] $*$ [linear combination of the b 's] sufficient for the computation of AB . It is well known that $m^*(n|F) = n^\beta$ for a particular value of n implies $\text{mam}(N|F) = O(N^\beta)$, hence

$$(0.2) \quad \omega(F) \leq \ln m^*(n|F) / \ln n.$$

For nearly ten years all attempts to improve Strassen's famous result $m^*(2|F) \leq 7$, $\omega(F) < 2.808$ (cf. [10]) failed, but recently some progress was made. Pan [7], [8] obtained $m^*(n|F) \leq \frac{1}{3}n^3 + O(n^2)$ with such constants that he could show (using $n = 48$) the new bound $\omega(F) < 2.781$ for any field F . In [1], [2] Bini, Capovani, Lotti and Romani found the slightly better bound $\omega(F) \leq \ln 1000 / \ln 12 < 2.780$, but much more significant is their new approach. They exhibit an *approximating* algorithm which shows that 12×12 matrix multiplication has approximate rank ≤ 1000 , and in [2] Bini shows that this implies $m^*(12^k|F) \leq (1 + 6k) \cdot 1000^k$. We will present these new notions in § 2, where we also give the smaller example of approximate rank ≤ 21 for the case of 3×3 matrices.

Actually, there is a rather striking example of even smaller size. The results in [1] are based upon a clever construction (cf. Example 2.2) which shows how to multiply two 2×2 matrices A and B with one element vanishing, say $a_{2,2} = 0$, approximately by means of only 5 multiplications instead of the trivial number 6.

In §§ 3 and 4 of this paper we describe a general method for exploiting such a small partial matrix multiplication for the construction of algorithms for the total multiplication of large matrices such that full efficiency is maintained: whenever the trivial number f of multiplications in a partial matrix multiplication over some field F can be replaced by l multiplications in an approximating algorithm the exponent of (total) matrix multiplication over F will be bounded (cf. Theorem 4.1) by $3 \ln l / \ln f$, for

* Received by the editors March 3, 1980, and in revised form June 16, 1980.

† Mathematisches Institut, Universität Tübingen, Tübingen, Germany.

instance by $3 \ln 5/\ln 6 < 2.695$. Other more favorable examples of partial matrix multiplication with comparatively few multiplications are given in § 5, for instance with $l = 17$ instead of $f = 26$, which implies $\omega(F) \leq 3 \ln 17/\ln 26 < 2.609$.

So far we don't have any deeper insight why some patterns of partial matrix multiplication lead to small exponents while others do not. The special structure of our examples seems to indicate, however, that this question is closely related to the *additivity conjecture* about tensor ranks (see [11, p. 194]). In § 6 we show that this conjecture cannot hold for approximate ranks, since there exist certain pairs of matrix multiplication problems AB and UV with completely disjoint sets of variables which can be done faster in one compound computation than separately (as far as approximate algorithms are concerned). Such configurations can be used to set up some kind of economical mass production. In pursuing these ideas we have established a strong generalization (Theorem 7.1) of our previous result. Let us demonstrate its effectiveness by an example, where two disjoint matrix multiplications of size $f_1 = 16$ and $f_2 = 9$ can be performed approximately by means of 17 multiplications (see § 6): in this case the previous bound $3 \ln 17/\ln 25 = 2.640$ can be replaced by $\omega(F) \leq 3\alpha < 2.548$, where $16^\alpha + 9^\alpha = 17$.

Again the problem arises of finding more favorable patterns, and in fact, soon after the presentation of our general theorem at the Oberwolfach conference on complexity theory in October 1979, V. Pan found an instance of three disjoint matrix multiplications which lead to the bound $\omega(F) \leq 3 \ln 52/\ln 110 < 2.522$. In § 8 we present a variant of his ingenious design and some tiny improvements.

Finally it must be stressed, however, that so far all these new results are mainly of theoretical interest. The point of intersection with Strassen's method lies beyond any practical matrix size, and with respect to $m^*(n|F)$ Pan's estimates for moderate values of n are still unbeaten. Furthermore, it will be seen that the constants in our estimations may also depend on the size of the field F . On the other hand we will show $\omega(F) = \omega(F_0)$, where F_0 denotes the prime field of F (Theorem 2.8). Thus the exponent of matrix multiplication over F can only depend on the characteristic of F .

1. Tensors. In this section we give an outline of the tensor machinery needed below. It will be sufficient to consider 3-dimensional tensors $t = (t_{i,j,k})$, where (i, j, k) varies over some finite index cube. With multi-indices $i = (i_1, i_2)$, $j = (j_1, j_2)$, $k = (k_1, k_2)$ the *tensorial product* $t = t' \otimes t''$ has the elements

$$(1.1) \quad t_{i,j,k} = t'_{i_1,j_1,k_1} \cdot t''_{i_2,j_2,k_2}$$

and $t^{\otimes s} = t \otimes t \otimes \dots \otimes t$ denotes the s -fold *tensorial power* of t formed correspondingly. Sometimes it will be convenient to describe a tensor $t = (t_{i,j,k})$ by means of its associated trilinear form

$$\psi = \sum_{i,j,k} t_{i,j,k} a_i b_j c_k,$$

where the a 's, b 's and c 's are considered as indeterminates over the underlying field F .

In the case of matrix multiplication we need triples of double indices: the product of a $K \times M$ matrix A with an $M \times N$ matrix B is a $K \times N$ matrix D ; its elements form a set of bilinear forms in the a 's and b 's expressed by the formula

$$(1.2) \quad d_{\alpha,\hat{\nu}} = \sum_{\hat{\mu},\hat{\nu}} t_{\hat{\alpha},\hat{\mu};\hat{\mu},\hat{\nu};\hat{\nu},\alpha} a_{\hat{\alpha},\hat{\mu}} b_{\hat{\mu},\hat{\nu}}$$

thus inducing the tensor elements

$$(1.3) \quad t_{\hat{\alpha}, \mu, \hat{\mu}, \nu, \hat{\nu}, \kappa} = \delta_{\hat{\alpha}, \kappa} \delta_{\hat{\mu}, \mu} \delta_{\hat{\nu}, \nu}.$$

The corresponding trilinear form is

$$(1.4) \quad \psi = \sum_{\alpha, \mu, \nu} a_{\alpha, \mu} b_{\mu, \nu} c_{\nu, \alpha}$$

which equals the trace of the product ABC of three matrices (observe the transposed indices of the c 's). We call this a matrix multiplication of *type* (K, M, N) ; its tensor, as given by (1.3), will be denoted by the special symbol $\langle K, M, N \rangle$. Hence, in particular, $\langle K, M, 1 \rangle$ is a tensor of size $KM \times M \times K$ describing the multiplication of a $K \times M$ matrix A with a column vector B , $\langle 1, M, 1 \rangle$ expresses a scalar product of length M , and $\langle 1, 1, N \rangle$ stands for the product of a single variable $a_{1,1}$ with a row vector B .

Combining (1.1) with (1.3) leads to the well-known process of nested matrix multiplications, namely

$$(1.5) \quad \langle K', M', N' \rangle \otimes \langle K'', M'', N'' \rangle = \langle K'K'', M'M'', N'N'' \rangle$$

and its s -fold version

$$(1.6) \quad \langle K, M, N \rangle^{\otimes s} = \langle K^s, M^s, N^s \rangle.$$

Another special case is that of *symmetrization*,

$$(1.7) \quad \langle K, M, N \rangle \otimes \langle M, N, K \rangle \otimes \langle N, K, M \rangle = \langle P, P, P \rangle,$$

where $P = KMN$.

In the noncommutative model the *rank* of a tensor t denoted by $rk(t)$ is defined as the minimal length r of a decomposition into triads

$$(1.8) \quad t = \sum_{\rho=1}^r (x_{\rho} \otimes y_{\rho} \otimes z_{\rho}),$$

or equivalently,

$$\psi = \sum_{\rho=1}^r \left(\sum_i x_{\rho,i} a_i \right) \left(\sum_j y_{\rho,j} b_j \right) \left(\sum_k z_{\rho,k} c_k \right),$$

$$t_{i,j,k} = \sum_{\rho=1}^r x_{\rho,i} y_{\rho,j} z_{\rho,k} \quad \text{for all } i, j, k,$$

where the $x_{\rho,i}$, $y_{\rho,j}$, $z_{\rho,k}$ are suitable elements of F . By virtue of (1.1) we have the submultiplicativity

$$(1.9) \quad rk(t' \otimes t'') \leq rk(t') \cdot rk(t''), \quad rk(t^{\otimes s}) \leq (rk(t))^s,$$

which will need refined consideration in the case of approximate rank (see § 2).

With regard to the tensors of matrix multiplication we have, first of all, the basic equation (cf. [11])

$$(1.10) \quad m^*(P|F) = rk \langle P, P, P \rangle.$$

Furthermore it is well known [5] (and can immediately be understood from (1.3) or (1.4)) that $rk \langle K, M, N \rangle$ is symmetric in K, M, N . Hence symmetrization and (1.9), (1.10) yield

$$m^*(KMN|F) \leq (rk \langle K, M, N \rangle)^3,$$

and (0.2) gives the important bound

$$(1.11) \quad \omega(F) \leq 3 \cdot \frac{\ln rk(K, M, N)}{\ln(KMN)}.$$

Let us mention here what is known for the small cases: in addition to $rk(2, 2, 2) = 7$ and $rk(4, 4, 4) \leq 49$ from [10] we have $rk(3, 2, 2) = 11$, $rk(3, 2, 3) = 15$ (see [4]) and $rk(3, 3, 3) \leq 23$ (see [6]). Among these, $3 \ln 7 / \ln 8$ is still the best.

Finally, we want to add some explanations about the *disjoint sum* of tensors. For matrices G, H (regarded as 2-dimensional tensors) of arbitrary rectangular size the disjoint sum is simply obtained by forming the block matrix

$$G \oplus H = \begin{pmatrix} G & 0 \\ 0 & H \end{pmatrix}.$$

Hence also $G \oplus G$ will make sense and will, of course, be different from $2G = G + G$. In the same way the disjoint sum of 3-dimensional tensors t', t'' may be formed by packing a copy of t' and a copy of t'' into opposite corners of a cube of appropriate size while the other positions are filled with zeros. As a matter of fact, $t' \otimes t''$ then will usually be different from $t' \oplus t''$, but in the context of this paper it will suffice to have the existence of suitable isomorphic mappings. In such cases we make free use of the equality sign, for instance in stating the *distributivity laws*

$$(1.12) \quad \begin{aligned} t \otimes (t' \oplus t'') &= (t \otimes t') \oplus (t \otimes t''), \\ (t' \oplus t'') \otimes t &= (t' \otimes t) \oplus (t'' \otimes t). \end{aligned}$$

For matrices we have the additivity of rk when applied to disjoint sums: $rk(G \otimes H) = rk(G) + rk(H)$. For 3-dimensional tensors one can easily prove that always

$$(1.13) \quad rk(t' \oplus t'') \leq rk(t') + rk(t''),$$

but Strassen's additivity conjecture (cf. [11]) that \leq can be replaced by $=$ here is still an open problem.

2. Approximate rank. In order to motivate the notion of approximate rank let F be the real number field. Then it can happen that the limit of a converging sequence of tensors has higher rank than all its approximants. Such a phenomenon could be described by means of a variable ϵ representing "small" numbers. We prefer, however, to use purely algebraic notions suitable for any field F (in particular for finite fields). In this setting ϵ is an extra indeterminate over F .

By an *approximate decomposition* of order $h \geq 0$ and length r of a tensor t or its trilinear form ψ —sometimes also called an *approximate algorithm* for ψ —we mean a representation

$$(2.1) \quad \sum_{\rho=1}^r (x_{\rho}(\epsilon) \otimes y_{\rho}(\epsilon) \otimes z_{\rho}(\epsilon)) = \epsilon^h t + O(\epsilon^{h+1}),$$

where $x_{\rho}(\epsilon) = x_{\rho}^0 + \epsilon x_{\rho}^1 + \dots + \epsilon^h x_{\rho}^h$, $y_{\rho}(\epsilon) = \dots$, $z_{\rho}(\epsilon) = \dots$ are of degree $\leq h$ in ϵ , the x_{ρ}^{α} , y_{ρ}^{β} , z_{ρ}^{γ} are vectors over F , and $O(\epsilon^{h+1})$ stands for a multiple of ϵ^{h+1} . The minimal r of this kind denoted by $r_h(t)$ is called the *approximate rank* of order h . Obviously $rk(t) = r_0(t) \geq r_1(t) \geq \dots$; the minimum $rk(t)$ of these numbers is called the *border rank* of t .

The decomposition (2.1) may also be written as

$$(2.2) \quad \epsilon^{-h} \sum_{\rho} (\dots) = t + \epsilon u_1 + \epsilon^2 u_2 + \dots + \epsilon^d u_d;$$

the minimal number $d \geq 0$ of this kind is called the *error degree* of this decomposition (or algorithm). Apparently we always have

$$(2.3) \quad d \leq 2h.$$

Example 2.1. Let t denote the tensor for

$$\psi = a_1 b_1 c_1 + a_1 b_2 c_2 + a_2 b_1 c_2.$$

Then $r_0(t) = 3$, but $r_1(t) = 2$, since

$$a_1 b_1 (-c_2 + \varepsilon c_1) + (a_1 + \varepsilon a_2)(b_1 + \varepsilon b_2)c_2 = \varepsilon \psi + \varepsilon^2 a_2 b_2 c_2;$$

this decomposition is of order $h = 1$ and has the error degree $d = 1$.

Example 2.2. (Bini et al. [1]). With respect to the multiplication of partial 2×2 matrices mentioned in the introduction, let t be the tensor for the trace ψ as given in (1.4) for three 2×2 matrices, where $a_{2,2} = 0$, i.e.,

$$\psi = a_{1,1}(b_{1,1}c_{1,1} + b_{1,2}c_{2,1}) + a_{1,2}(b_{2,1}c_{1,1} + b_{2,2}c_{2,1}) + a_{2,1}(b_{1,1}c_{1,2} + b_{1,2}c_{2,2}).$$

The approximate decomposition

$$\begin{aligned} &(a_{1,2} + \varepsilon a_{1,1})(b_{1,2} + \varepsilon b_{2,2})c_{2,1} + (a_{2,1} + \varepsilon a_{1,1})b_{1,1}(c_{1,1} + \varepsilon c_{1,2}) \\ &\quad - a_{1,2}b_{1,2}(c_{1,1} + c_{2,1} + \varepsilon c_{2,2}) - a_{2,1}(b_{1,1} + b_{1,2} + \varepsilon b_{2,1})c_{1,1} \\ &\quad + (a_{1,2} + a_{2,1})(b_{1,2} + \varepsilon b_{2,1})(c_{1,1} + \varepsilon c_{2,2}) = \varepsilon \psi + O(\varepsilon^2) \end{aligned}$$

shows that $r_1(t) \leq 5$, whereas $rk(t) = 6$ (by [3, Corollary 4.4]); again we have $h = d = 1$.

Example 2.3. For the multiplication of (total) 3×3 matrices we want to show $r_2(3, 3, 3) \leq 21$, which compares rather favorably with the best upper bound 23 known for the exact rank. Here we present the corresponding approximate decomposition of the tensor $\langle 3, 3, 3 \rangle$ more directly as an algorithm for approximate matrix multiplication: Given two 3×3 matrices A, B and a “small” $\varepsilon \neq 0$ first compute, for $i, j \in \{1, 2, 3\}$, the 21 products

$$\begin{aligned} u_{i,i} &= (a_{i,1} + \varepsilon^2 a_{i,2})(\varepsilon^2 b_{1,i} + b_{2,i}), \\ v_{i,i} &= (a_{i,1} + \varepsilon^2 a_{i,3})b_{3,i}, \\ w_i &= a_{i,1}(b_{2,i} + b_{3,i}), \\ u_{i,j} &= (a_{i,1} + \varepsilon^2 a_{i,2})(b_{2,i} - \varepsilon b_{1,j}), \quad (i \neq j), \\ v_{i,j} &= (a_{i,1} + \varepsilon^2 a_{i,3})(b_{3,i} + \varepsilon b_{1,j}), \quad (i \neq j), \end{aligned}$$

then recombine

$$\begin{aligned} d'_{i,i} &= \frac{1}{\varepsilon^2}(u_{i,i} + v_{i,i} - w_i), \\ d'_{i,j} &= \frac{1}{\varepsilon^2}(u_{i,j} + v_{i,j} - w_i) + \frac{1}{\varepsilon}(v_{j,i} - v_{j,j}), \quad (i \neq j). \end{aligned}$$

This constitutes a matrix D' which differs from $D = AB$ by not more than $O(\varepsilon)$. Translation into the shape of (2.1) or (2.2) shows that this decomposition has order $h = 2$ and error degree $d = 2$.

Example 2.4. Any simple transcendental field extension of F can be represented as the field $F(\varepsilon)$ of quonominials over F , which is contained in the field \hat{F} of all formal power series $\varepsilon^{-n}(\gamma_0 + \gamma_1 \varepsilon + \gamma_2 \varepsilon^2 + \dots)$ with arbitrary integers $n \geq 0$ and γ 's in F . Let t

be any tensor over F with border rank $rk(t)$ and let $\hat{r}(t)$ denote its rank with respect to \hat{F} . Then we have $rk(t) \leq \hat{r}(t)$ (the proof is left to the reader).

Frequently we will need a refinement of (1.9) for approximate ranks. Given tensors t', t'' with approximate decompositions of length and order r', h' or r'', h'' , respectively, the tensor product of the corresponding formulae (2.1) is

$$\sum_{\rho_1=1}^{r'} \sum_{\rho_2=1}^{r''} ([x'_{\rho_1}(\varepsilon) \otimes x''_{\rho_2}(\varepsilon)] \otimes [y'_{\rho_1}(\varepsilon) \otimes y''_{\rho_2}(\varepsilon)]) \otimes [z'_{\rho_1}(\varepsilon) \otimes z''_{\rho_2}(\varepsilon)] = \varepsilon^{h'+h''} (t' \otimes t'') + O(\varepsilon^{h'+h''+1}).$$

The elements of these vectors $[\cdot \cdot \cdot]$ are polynomials of degree $\leq h' + h''$ in ε . That proves (see also [2]) the following lemma.

LEMMA 2.5. *Approximate ranks are submultiplicative according to the following rules:*

$$(2.4) \quad r_{h'+h''}(t' \otimes t'') \leq r_{h'}(t') \cdot r_{h''}(t''),$$

$$(2.5) \quad r_{sh}(t^{\otimes s}) \leq (r_h(t))^s.$$

An immediate corollary is the submultiplicativity of the border rank; i.e., $rk(t' \otimes t'') \leq rk(t') \cdot rk(t'')$.

Next we discuss estimations of the exact rank $rk(t)$ by means of approximate ranks. In [2] Bini uses evaluation of the left-hand side of (2.2) at $d + 1$ different values $\varepsilon_j \in F$, $\varepsilon_j \neq 0$, and interpolation to prove the next lemma.

LEMMA 2.6. *If a tensor t has an approximate decomposition of length r with error degree d , then the estimate*

$$(2.6) \quad rk(t) \leq (1 + d)r$$

holds, provided the underlying field F contains at least $d + 2$ elements.

By means of (2.3) this implies (for $\#F \geq 2h + 2$)

$$(2.7) \quad rk(t) \leq (1 + 2h)r_h(t).$$

In order to get results for all fields F we use a different approach, which independently was also applied by S. Winograd for proving an inequality of the same kind. Formula (2.1) becomes an exact identity without error term if it is read as an equation over the extended scalar domain $F' = F[\varepsilon]/(\varepsilon^{h+1})$. With respect to a second quite analogous situation we deal with a slightly more general case.

THEOREM 2.7. *Let $g \in F[\theta]$ be a polynomial of degree d over F , and $0 \leq k < d$. Let rk' denote the rank of tensors with respect to $F' = F[\theta]/(g)$. Then the F -rank of any tensor t (with elements on F) is bounded by*

$$(2.8) \quad rk(t) \leq M(F'/F) \cdot rk'(\theta^k t),$$

where $M(F'/F)$ denotes the number of multiplications (noncommutative) necessary to multiply two polynomials over F modulo g .

Proof. By definition of $rk'(\theta^k t)$ there exists a decomposition

$$(2.9) \quad \theta^k t = \sum_{\rho=1}^r (x_\rho \otimes y_\rho \otimes z_\rho), \quad r = rk'(\theta^k t)$$

into triads of F' -vectors which, by means of suitable F -vectors $x_\rho^\alpha, y_\rho^\beta$, have representations

$$(2.10) \quad x_\rho = \sum_{\alpha=0}^{d-1} \theta^\alpha x_\rho^\alpha, \quad y_\rho = \sum_{\beta=0}^{d-1} \theta^\beta y_\rho^\beta.$$

According to the definition of $M(F'/F)$ there are scalars $\xi_{i,\alpha}, \eta_{i,\beta}, \zeta_{i,\delta} \in F$ ($1 \leq i \leq M(F'/F)$) such that (2.10) gives

$$\begin{aligned} x_\rho \otimes y_\rho \otimes z_\rho &= \left(\sum_{\delta=0}^{d-1} \theta^\delta \left(\sum_i \zeta_{i,\delta} \left(\sum_\alpha \xi_{i,\alpha} x_\rho^\alpha \right) \otimes \left(\sum_\beta \eta_{i,\beta} y_\rho^\beta \right) \right) \right) \otimes z_\rho \\ &= \sum_i \left(\left(\sum_\alpha \xi_{i,\alpha} x_\rho^\alpha \right) \otimes \left(\sum_\beta \eta_{i,\beta} y_\rho^\beta \right) \otimes (\varphi_i z_\rho) \right), \end{aligned}$$

with $\varphi_i = \sum_{\delta=0}^{d-1} \zeta_{i,\delta} \theta^\delta \in F'$.

Similar to (2.10), the F' -vectors $\varphi_i z_\rho$ have representations

$$\varphi_i z_\rho \equiv \sum_{\gamma=0}^{d-1} \theta^\gamma z_{i,\rho}^\gamma \pmod{g}.$$

Since the tensor t on the left-hand side of (2.9) has all its elements in F , we finally get, by comparing coefficients,

$$t = \sum_\rho \sum_i \left(\left(\sum_\alpha \xi_{i,\alpha} x_\rho^\alpha \right) \otimes \left(\sum_\beta \eta_{i,\beta} y_\rho^\beta \right) \otimes z_{i,\rho}^k \right).$$

This decomposition of t proves (2.8).

By replacing θ by ε we now return to the case of approximate ranks. Then we have to choose $g = \varepsilon^{h+1}$, $k = h$, and $rk'(\varepsilon^h t)$ becomes $r_h(t)$. In this way (2.7) is obtained as a corollary of (2.8) with $M(F'/F) = 1 + 2h$ for $\#F \geq 2h$. For smaller fields F we will be content with the crude bound $M(F'/F) \leq (1 + h)^2$, hence:

$$(2.11) \quad rk(t) \leq (1 + h)^2 r_h(t) \quad \text{for all fields.}$$

Now we are ready to give a paradigmatic application of these methods by deriving from our Example 2.3 the bound

$$(2.12) \quad \omega(F) \leq \ln 21 / \ln 3 < 2.772.$$

Proof. By means of (1.6) and (2.5) $r_2(3, 3, 3) \leq 21$ implies $r_{2s}(3^s, 3^s, 3^s) \leq 21^s$. Then (2.11) with $h = 2s$ and (1.10) yield $m^*(3^s|F) \leq (1 + 2s)^2 \cdot 21^s$, hence

$$\omega(F) \leq (2 \ln(1 + 2s) + s \ln 21) / (s \ln 3).$$

By $s \rightarrow \infty$ this finally gives (2.12).

The last step is the crucial point with respect to all practical applications of the method. Of course, we can assume that F is infinite and use Lemma 2.6, which gives the better bound $m^*(3^s|F) \leq (1 + 2s)21^s$, but even then the minimal value of s for which this becomes smaller than the trivial bound 27^s is $s = 14$, and $3^{14} = 4,782,969$. The same will apply for all other bounds given in this paper.

By combining the foregoing method of proof with Example 2.4 we can show that a simple transcendental field extension of F cannot result in a reduction of $\omega(F)$.

Consider any fixed $\beta > \omega(\hat{F})$. Then there is some q with $m^*(q|\hat{F}) \leq q^\beta$, and by the statement in Example 2.4 we get

$$r_h \langle q, q, q \rangle = rk \langle q, q, q \rangle \leq \hat{r} \langle q, q, q \rangle \leq q^\beta$$

for some h sufficiently large. The further steps are

$$\begin{aligned} r_{sh} \langle q^s, q^s, q^s \rangle &\leq q^{\beta s}, \\ m^*(q^s|F) = rk \langle q^s, q^s, q^s \rangle &\leq (1 + 2sh)^2 q^{\beta s}, \\ \omega(F) &\leq (2 \ln(1 + 2sh) + s\beta \ln q) / (s \ln q); \end{aligned}$$

now $s \rightarrow \infty$ gives $\omega(F) \leq \beta$, hence $\omega(F) \leq \omega(\hat{F})$.


Similarly, no simple algebraic field extension $F' = F(\theta)$ can reduce $\omega(F)$. In this case we can use Theorem 2.7 with $k = 0$ and choose g as the minimal polynomial of θ over F .

THEOREM 2.8. *The exponent of matrix multiplication over F can only depend on the characteristic of F ; in other words, if F_0 is the prime field of F , then $\omega(F) = \omega(F_0)$.*

Proof. For arbitrary $\beta > \omega(F)$ there is again some q with $m^*(q|F) \leq q^\beta$ and a corresponding decomposition of the tensor $\langle q, q, q \rangle$ in which only a finite number of elements of F can occur. Hence there is a finite chain of simple field extensions $F_0 \subset F_1 \subset \dots \subset F_k$ such that also $m^*(q|F_k) \leq q^\beta$. That implies $\omega(F_k) \leq \beta$, and from the preceding proofs we already know (by induction) that $\omega(F_0) \leq \omega(F_k)$. Putting all this together we get $\omega(F_0) \leq \omega(F)$, but trivially also $\omega(F) \leq \omega(F_0)$.

3. Partial matrix multiplication. More precisely the heading of this section should be: multiplication of matrices which are partially filled. Since we will study tensorial products of such matrices, this subject requires a careful description. Let us begin with a “small” matrix multiplication of type (k, m, n) : $D^{(1)}$ is the product of a $k \times m$ matrix $A^{(1)}$ with an $m \times n$ matrix $B^{(1)}$. Some of the positions in these factors are occupied by variables $a_{\alpha, \mu}, b_{\mu, \nu}$, while others may be filled with zeros. The variable-positions in $A^{(1)}$ and $B^{(1)}$ constitute subsets of index pairs,

$$\begin{aligned} (3.1) \quad I &\subseteq \{1, \dots, k\} \times \{1, \dots, m\}, \\ J &\subseteq \{1, \dots, m\} \times \{1, \dots, n\}. \end{aligned}$$

In case of the pattern  we have, for instance, $I = \{(1, 1), (1, 2), (2, 1)\}$ (see Example 2.2). Later we will need the characteristic functions of these sets; i.e., $\chi_I(\alpha, \mu) = 1$ iff $(\alpha, \mu) \in I$ ($= 0$ otherwise), and similarly χ_J .

Next we form the s -fold tensorial power of the patterns; i.e., by blockwise nesting we obtain a (k^s, m^s, n^s) type partial matrix multiplication $A^{(s)} B^{(s)} = D^{(s)}$ with its variable-positions belonging to $I^{(s)}$ and $J^{(s)}$, respectively, where

$$\begin{aligned} (3.2) \quad I^{(s)} &= \left\{ (\alpha, \mu) \in \{1, \dots, k\}^s \times \{1, \dots, m\}^s \mid \prod_{\sigma=1}^s \chi_I(\alpha_\sigma, \mu_\sigma) = 1 \right\}, \\ J^{(s)} &= \left\{ (\mu, \nu) \in \{1, \dots, m\}^s \times \{1, \dots, n\}^s \mid \prod_{\sigma=1}^s \chi_J(\mu_\sigma, \nu_\sigma) = 1 \right\}. \end{aligned}$$

In the example mentioned above, $I^{(3)}$, for instance, describes the pattern shown in Fig. 3.1.

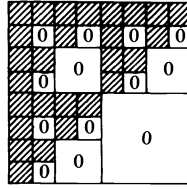



FIG. 3.1. 3-fold tensorial power of .

The multi-indexed elements of the tensor $t^{(s)}$ associated with these partial matrix multiplications are obtained by adequate modification of (1.3); for all $s \geq 1$ we get

$$\begin{aligned}
 (3.3) \quad t_{\hat{\alpha}, \mu; \hat{\beta}, \nu; \hat{\gamma}, \kappa}^{(s)} &= \delta_{\hat{\alpha}, \alpha} \delta_{\hat{\beta}, \beta} \delta_{\hat{\gamma}, \gamma} \prod_{\sigma=1}^s (\chi_I(\alpha_\sigma, \mu_\sigma) \cdot \chi_J(\mu_\sigma, \nu_\sigma)) \\
 &= \prod_{\sigma=1}^s t_{\alpha_\sigma, \mu_\sigma; \beta_\sigma, \nu_\sigma; \gamma_\sigma, \kappa_\sigma}^{(1)}
 \end{aligned}$$

With respect to ranks, Lemma 2.5 has the following corollary.

LEMMA 3.1. *If the tensor $t^{(1)}$ of a partial matrix multiplication $A^{(1)}B^{(1)} = D^{(1)}$ has an (approximate) decomposition of order $h \geq 0$ and length l , then its s -fold tensorial power $t^{(s)}$ has an (approximate) decomposition of order hs and length l^s .*

Given a tensor t or its trilinear form ψ , a tensor t' is called a *subtensor* of t if the corresponding ψ' is obtained from ψ by replacing some of the a 's, b 's or c 's by zero. Application of such a homomorphism to approximate decompositions of t immediately yields the following lemma.

LEMMA 3.2. *If t' is a subtensor of t , then*

$$(3.4) \quad r_h(t') \leq r_h(t) \quad \text{for all } h \geq 0.$$

In the next section we will describe how from any given partial matrix multiplication $A^{(1)}B^{(1)} = D^{(1)}$ with tensor $t^{(1)}$ by compression of the *sparse* matrix multiplications $A^{(s)}B^{(s)} = D^{(s)}$ algorithms for the multiplications of relatively large *total* matrices can be constructed. Their efficiency will decisely depend on the relation between the number of ones in the tensor $t^{(1)}$ and its border rank $rk(t^{(1)}) = \min_h r_h(t^{(1)})$. Therefore let us analyse the subsets I, J of such a partial multiplication more precisely.

There are k_μ variable-positions in the μ th column of $A^{(1)}$, and n_μ variable-positions in the μ th row of $B^{(1)}$, where

$$\begin{aligned}
 (3.5) \quad k_\mu &= \sum_{\alpha} \chi_I(\alpha, \mu), \\
 n_\mu &= \sum_{\nu} \chi_J(\mu, \nu).
 \end{aligned} \quad (1 \leq \mu \leq m).$$

In order to avoid confusing redundancies we will always assume that all these numbers are greater than zero. The number of ones in the corresponding tensor $t^{(1)}$ is given by

$$(3.6) \quad f = k_1 n_1 + k_2 n_2 + \dots + k_m n_m.$$

This quantity can also be interpreted as the trivial number of multiplications needed in a straightforward algorithm.

4. Total matrix multiplication by compression. The bound $\ln 21 / \ln 3$ in (2.12) and the bound $3 \ln 10 / \ln 12$ in [1] are examples of the general fact that the inequality (1.11)

remains true, if rk is replaced by the border rank \underline{rk} . Its generalization to tensors of partial matrix multiplication is one of our main results.

THEOREM 4.1. *Let t be a tensor of partial matrix multiplication which contains f ones and has border rank $\leq l$ over the field F . Then the exponent of matrix multiplication over F is bounded by*

$$(4.1) \quad \omega(F) \leq \lambda(l, f) = 3 \ln l / \ln f.$$

Strassen's exponent is $\lambda(7, 8)$. Example 2.2 yields $\lambda(5, 6) \leq 2.695$. In § 5 we shall derive even better bounds.

The proof of this theorem is rather complicated. It will be given stepwise in the following subsections. In view of Theorem 2.8 it is no restriction to assume the field F to be infinite.

4.1. Opening. By definition of the border rank the given tensor $t^{(1)}$ has some approximate decomposition of length l and order h , which will be held fixed in our further analysis. In addition we choose some value $s > 1$ and a fixed partition

$$(4.2) \quad s = \sigma_1 + \sigma_2 + \dots + \sigma_m, \quad (\sigma_\mu \geq 0).$$

Variation of these parameters will be discussed in § 4.4. Now consider the partial matrix multiplication $A^{(s)}B^{(s)} = D^{(s)}$ associated with $t^{(1)}$ as described in § 3. In view of (3.3) its tensor $t^{(s)}$ will contain exactly

$$(4.3) \quad f^s = (k_1 n_1 + k_2 n_2 + \dots + k_m n_m)^s$$

ones. The multinomial expansion of this expression is closely related to the distribution of the variable-positions in $A^{(s)}$ and $B^{(s)}$. Let \mathfrak{M} denote the set of all multi-indices $\mu = (\mu_1, \dots, \mu_s)$ with exactly σ_i of the entries μ_j being equal to i , for $1 \leq i \leq m$. Then the number of elements in \mathfrak{M} is

$$(4.4) \quad M = \frac{s!}{\sigma_1! \sigma_2! \dots \sigma_m!}.$$

Combining (3.2) with (3.5) we see that for any $\mu \in \mathfrak{M}$ there are exactly K variable-positions in column μ of $A^{(s)}$ and N variable-positions in row μ of $B^{(s)}$, where

$$(4.5) \quad K = k_1^{\sigma_1} k_2^{\sigma_2} \dots k_m^{\sigma_m},$$

$$(4.6) \quad N = n_1^{\sigma_1} n_2^{\sigma_2} \dots n_m^{\sigma_m}.$$

Our goal is to accomplish a total matrix multiplication of type (K, M, N) . Therefore we now cancel all other columns of $A^{(s)}$ and rows of $B^{(s)}$ by setting $a_{\kappa, \mu}^{(s)} = 0, b_{\mu, \nu}^{(s)} = 0$ for all $\mu \notin \mathfrak{M}$ and all κ, ν . This reduces $A^{(s)}B^{(s)} = D^{(s)}$ to a partial matrix multiplication $AB = D$ of type (k^s, M, n^s) . Its tensor t is a subtensor of $t^{(s)}$. Hence Lemma 3.1 and Lemma 3.2 yield the bound

$$(4.7) \quad r_{hs}(t) \leq l^s.$$

4.2. Compression. By construction there are K variables in each column of A and N variables in each row of B . The main idea is now to apply a transformation that converts $AB = D$ into a total matrix multiplication $UV = W$ of type (K, M, N) . This compression is achieved by *sandwiching* with suitable scalar matrices G and Q of size $K \times k^s$ and $n^s \times N$, respectively:

$$(4.8) \quad GA = U, \quad BQ = V, \quad W = GABQ = GDQ.$$

Now the question is how to choose G and Q . For any $j(1 \leq j \leq M)$ let $a_{\cdot j}$ denote the j th column of A , and b_j the j th row of B . Correspondingly we consider the j th column of U and the j th row of V . Then (4.8) becomes

$$(4.9a) \quad Ga_{\cdot j} = u_{\cdot j}, \quad (1 \leq j \leq M).$$

$$(4.9b) \quad b_j Q = v_j,$$

For fixed j we can select K indices $i_1 < i_2 < \dots < i_K$ such that all the pairs (i_r, j) are variable-positions of A . Let us choose G as a scalar $K \times k^s$ matrix with all its $K \times K$ minors different from zero (the details are given below). Then there exists a $K \times K$ inverse H_j such that

$$(4.10) \quad \begin{aligned} (a_{i_1, j} \cdots a_{i_K, j})^T &= H_j u_{\cdot j} \\ a_{i, j} &= 0 \quad \text{for } i \neq i_1, \dots, i_K \end{aligned}$$

is consistent with (4.9a). Similarly we choose Q as a scalar $n^s \times N$ matrix with none of its $N \times N$ minors vanishing. Then there are certain $N \times N$ inverses R_j such that $v_j R_j$ contains the variable b 's that (together with additional zeros) fulfill condition (4.9b).

Let us summarize what we have achieved by this construction: For given matrices U and V the a 's are expressed as linear forms of the u 's by (4.10), and the b 's are linear forms of the v 's. Inequality (4.7) implies that there exists an approximate algorithm of order hs and length $\leq l^s$ for the partial matrix multiplication $AB = D$. Finally the w 's are obtained as linear combinations of the d 's according to $W = GDQ$. Putting all this together we get: *Total matrix multiplication of type (K, M, N) has an approximate algorithm of order hs and length $\leq l^s$.*

We have to supplement the technicalities of choosing G and Q as mentioned above. Here we use that F is infinite and construct a matrix G of the required kind simply by choosing k^s different elements $\alpha_i \in F$. Then

$$(4.11) \quad G = (\gamma_{q,i} = \alpha_i^{q-1} | 1 \leq q \leq K, 1 \leq i \leq k^s)$$

will work. The matrix Q can be constructed in the same way.

4.3. Symmetrization. It is rather unlikely that the preceding constructions will yield equal numbers K, M, N . Therefore we insert a step of balancing for square size by symmetrization (see (1.7)). From (1.3) and the definition of approximate rank it follows that also $r_{hs}(K, M, N)$ is symmetric in K, M and N . By means of the bound $r_{hs}(K, M, N) \leq l^s$ found in § 4.2 and Lemma 2.5 we therefore obtain $r_{3hs}(P, P, P) \leq l^{3s}$, which implies (cf. (2.7)):

$$(4.12) \quad rk(P, P, P) \leq (1 + 6hs)l^{3s}.$$

Here $P = KMN$ can be computed from (4.4)–(4.6) explicitly,

$$(4.13) \quad P = \frac{s!}{\sigma_1! \sigma_2! \cdots \sigma_m!} (k_1 n_1)^{\sigma_1} (k_2 n_2)^{\sigma_2} \cdots (k_m n_m)^{\sigma_m}.$$

4.4. Optimal parameters. Now we are ready to discuss the optimal choice of $\sigma_1, \dots, \sigma_m$ under the constraint (4.2). Apparently, the best we can do is to maximize P as given by (4.13) by choosing a maximal term in the multinomial expansion of (4.3). This determines the σ 's implicitly, and we could now work out a rather precise lower bound for the corresponding value of P by using Stirling's formula. For the present proof, however, a much simpler bound is sufficient. Since the multinomial expansion of

(4.3) contains exactly $\binom{s+m-1}{m-1}$ terms, we get immediately

$$P \geq f^s / \binom{s+m-1}{m-1}.$$

Finally we have to combine this with (4.12). Then

$$\omega(F) \leq \frac{\ln m^*(P|F)}{\ln P} \leq \frac{\ln(1+6hs) + 3s \ln l}{s \ln f - (m-1) \ln(s+m-1)},$$

and $s \rightarrow \infty$ completes our proof of Theorem 4.1.

5. Patterns and exponents. In the vast variety of partial matrix multiplications we have found two general designs which, by proper choice of the parameters, lead to patterns with particularly favorable exponents $\lambda(l, f)$. At first we describe a family of flag-shaped patterns (see Fig. 5.1). Depending on one parameter $q \geq 2$ the data of § 3 are specified in the following way:

$$(5.1) \quad k = n = q + 1, \quad m = 2q,$$

$$(5.2) \quad I = \{(\alpha, \mu) | q + 1 \leq \mu \leq 2q \text{ or } \alpha = k\},$$

$$J = \{(\mu, \nu) | 1 \leq \mu \leq q \text{ or } \nu = k\}.$$

According to (1.4) the tensor $t^{(1)}$ of this partial matrix multiplication induces the trilinear form

$$(5.3) \quad \psi = \sum_{i=1}^q \sum_{j=1}^{q+1} a_{k,i} b_{i,j} c_{j,k} + \sum_{i=1}^{q+1} \sum_{j=1}^q a_{i,j+q} b_{j+q,k} c_{k,i}.$$

It has an approximate decomposition of order 3 and length $l = (q + 1)^2$, namely

$$\begin{aligned} & \sum_{i=1}^q \sum_{j=1}^q (a_{k,i} + \varepsilon^2 a_{i,j+q})(b_{j+q,k} + \varepsilon^2 b_{i,j})(\varepsilon c_{j,k} + \varepsilon c_{k,i} - c_{k,k}) \\ & + \sum_{i=1}^q \left(a_{k,i} \cdot \left(\varepsilon^3 b_{i,k} + \sum_{j=1}^q (b_{j+q,k} + \varepsilon^2 b_{i,j}) \right) (c_{k,k} - \varepsilon c_{k,i}) \right) \\ & + \sum_{j=1}^q \left(\left(\varepsilon^3 a_{k,j+q} + \sum_{i=1}^q (a_{k,i} + \varepsilon^2 a_{i,j+q}) \right) \cdot b_{j+q,k} (c_{k,k} - \varepsilon c_{j,k}) \right) \\ & - \left(\sum_{i=1}^q a_{k,i} \right) \left(\sum_{j=1}^q b_{j+q,k} \right) c_{k,k} = \varepsilon^3 \psi + O(\varepsilon^4). \end{aligned}$$

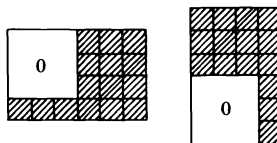


FIG. 5.1. Flag-shaped pattern for $q = 3$.



FIG. 5.2. Pattern from the second design for $k = 4, q = 3$.

From (5.3) we get $f = 2q^2 + 2q$; thus Theorem 4.1 yields the exponents (for $q = 2, 3, 4, 5$):

$$\begin{aligned}
 \lambda(9, 12) &= 2.6526 \dots, \\
 \lambda(16, 24) &= 2.6172 \dots, \\
 \lambda(25, 40) &= 2.6177 \dots, \\
 \lambda(36, 60) &= 2.6257 \dots.
 \end{aligned}
 \tag{5.4}$$

Even without Theorem 4.1 the patterns (5.2) can be utilized to derive quite reasonable bounds for total matrix multiplication: For any $p \leq q$ we can exchange rows p and k of $A^{(1)}$ and, simultaneously, columns p and k of $B^{(1)}$. Then we obtain q other isomorphic patterns. The sum of all such trilinear forms then equals the trace of a product ABC of total matrices. The corresponding total matrix multiplication $AB = D$ is of type $(k, 2k - 2, k)$; it has an approximate algorithm of order 3 and length k^3 . In the case $q = 2$, for instance, we get $r_3(3, 4, 3) \leq 27$, which would still give an exponent of $\lambda(27, 36) < 2.76$.

Slightly better bounds are obtainable by our second design (see Fig. 5.2) depending on two parameters k and q . Here the specifications are

$$m = q + 1, \quad n = 1 + (k - 1)(q - 1), \quad k \text{ as given,}
 \tag{5.5}$$

$$\begin{aligned}
 I &= \{(\kappa, \mu) \mid \kappa = 1 \text{ or } \mu > 1\}, \\
 J &= \{(\mu, \nu) \mid \mu = 1 \text{ or } \nu = 1\}.
 \end{aligned}
 \tag{5.6}$$

The corresponding trilinear form is

$$\psi = \sum_{\nu=1}^n a_{1,1} b_{1,\nu} c_{\nu,1} + \sum_{i=1}^k \sum_{j=2}^m a_{i,j} b_{j,1} c_{1,i}.
 \tag{5.7}$$

It contains $f = 1 + (k - 1)(q - 1) + kq$ terms and has the following approximate decomposition of order 2 and length $l = 1 + kq$, where the summation indices i and j shall control the row index ν such that a 1-1 map $(i, j) \mapsto \nu$,

$$\{2, \dots, k\} \times \{3, \dots, m\} \rightarrow \{2, \dots, n\},$$

is accomplished (e.g., $\nu = i + (j - 3)(k - 1)$ will do). Then

$$\begin{aligned}
 &(a_{1,1} + \varepsilon^2 a_{1,2})(b_{2,1} + \varepsilon^2 b_{1,1})c_{1,1} \\
 &+ \sum_{j=3}^m \left((a_{1,1} + \varepsilon^2 a_{1,j}) b_{j,1} \left(c_{1,1} - \sum_{i=2}^k (\varepsilon c_{\nu,1}) \right) \right) \\
 &+ \sum_{i=2}^k \left((a_{1,1} + \varepsilon^2 a_{i,2}) \left(b_{2,1} - \sum_{j=3}^m (\varepsilon b_{1,\nu}) \right) c_{1,i} \right) \\
 &+ \sum_{i=2}^k \sum_{j=3}^m \left((a_{1,1} + \varepsilon^2 a_{i,j}) (b_{j,1} + \varepsilon b_{1,\nu}) (c_{1,i} + \varepsilon c_{\nu,1}) \right) \\
 &- a_{1,1}(b_{2,1} + b_{3,1} + \dots + b_{m,1})(c_{1,1} + c_{1,2} + \dots + c_{1,k}) = \varepsilon^2 \psi + O(\varepsilon^3).
 \end{aligned}
 \tag{5.8}$$

Here Theorem 4.1 leads to exponents

$$\lambda(l, f) = 3 \ln(1 + kq) / \ln(2 + 2kq - k - q),
 \tag{5.9}$$

symmetric in k and q . The minimal value is attained for $k = q = 4$, namely $\lambda(17, 26) = 2.6087 \dots$.

Also the patterns (5.6) can be used to compose efficient approximate algorithms for total matrix multiplication directly. In fact, our Example 2.3 has been constructed from three copies of such a pattern with $k = 3, q = 2, l = 7, f = 9$. In a similar way (yet more involved) we could obtain $\underline{rk}\langle 4, 4, 4 \rangle \leq 47$. In the case of 2×2 matrices it is still an open problem whether $\underline{rk}\langle 2, 2, 2 \rangle$ equals 6 or 7. Altogether it seems to be rather difficult to get any nontrivial lower bounds for the border rank of a tensor.

6. Disjoint matrix multiplications. A careful analysis of the patterns given in § 5 reveals that they are built up from smaller total matrix multiplications somehow linked together. The example in Fig. 5.2, for instance, contains two total matrix multiplications of types $(4, 3, 1)$ and $(1, 1, 7)$, respectively. Their small overlap can be removed by setting $b_{1,1} = 0$, thus leading to two disjoint problems of types $(4, 3, 1)$, $(1, 1, 6)$, and the same can be done for general parameters. Let us represent the approximate decomposition (5.8) here in a modified way. In particular, we like to apply symmetry for getting a permutation of the parameters such that we are dealing with the disjoint tensor sum $\langle k, 1, n \rangle \oplus \langle 1, m, 1 \rangle$, where $m = (k - 1)(n - 1)$. This tensor describes the problem of evaluating the following two special matrix products

$$(6.1) \quad \begin{pmatrix} a_1 \\ \vdots \\ a_k \end{pmatrix} (b_1, \dots, b_n) \quad \text{and} \quad \sum_{i=1}^{k-1} \sum_{j=1}^{n-1} u_{i,j} v_{i,j}$$

in disjoint sets of variables (the u 's and v 's in the inner product of length m have double indices for technical reasons). The corresponding trilinear form has a rather elegant approximate decomposition of order 2 and length $l = kn + 1$, if we introduce the additional quantities

$$(6.2) \quad \begin{aligned} u_{i,n} &= 0, & v_{i,n} &= -\sum_{j < n} v_{i,j} \quad \text{for all } i \leq k, \\ u_{k,j} &= -\sum_{i < k} u_{i,j}, & v_{k,j} &= 0 \quad \text{for all } j \leq n. \end{aligned}$$

Then we obtain (by cancellation of all ε -terms)

$$(6.3) \quad \begin{aligned} &\sum_{i=1}^k \sum_{j=1}^n (a_i + \varepsilon u_{i,j})(b_j + \varepsilon v_{i,j})(\varepsilon^2 c_{j,i} + w) - \left(\sum_{i=1}^k a_i \right) \left(\sum_{j=1}^n b_j \right) w \\ &= \varepsilon^2 \sum_{i,j} (a_i b_j c_{j,i} + u_{i,j} v_{i,j} w) + O(\varepsilon^3). \end{aligned}$$

Let us summarize and state that this result is sharp.

LEMMA 6.1. *If $k, n \geq 2$, and $m = (k - 1)(n - 1)$, then*

$$r_2(\langle k, 1, n \rangle \oplus \langle 1, m, 1 \rangle) = kn + 1.$$

Proof. By (6.3) and the observation that the border rank of any tensor which describes q linearly independent bilinear forms cannot be less than q ; here we have $q = kn + 1$ by counting the different results in (6.1).

The same argument shows $\underline{rk}\langle k, 1, n \rangle \geq kn$. Since $\underline{rk}\langle k, 1, n \rangle \leq rk\langle k, 1, n \rangle = kn$, we get precisely $\underline{rk}\langle k, 1, n \rangle = kn$, and by means of symmetry also $\underline{rk}\langle 1, m, 1 \rangle = rk\langle 1, 1, m \rangle = m$, hence $\underline{rk}\langle k, 1, n \rangle + \underline{rk}\langle 1, m, 1 \rangle = kn + m$. Now we choose $k = 2, n > 2, m = n - 1$ and obtain

COROLLARY 6.2. *For 3-dimensional tensors the border rank is only subadditive: the difference $rk(t') + rk(t'') - rk(t' \oplus t'')$ can assume any nonnegative integer value.*

How can the bound in Lemma 6.1 be used for speeding up matrix multiplication? Theorem 4.1 gives only $\omega(F) \leq 3 \ln(kn + 1) / \ln(kn + m)$, worse than (5.9), but can we do better by exploiting the additional feature of disjointness given here? An affirmative answer will be given by our main result in Theorem 7.1. In order to motivate the rather intricate analysis of § 7, let us assume for the remainder of this section that the additivity conjecture for the exact rank is true (there is no obvious way to infer something about it from Corollary 6.2). Then we can derive a better bound in the following way.

Since \oplus and \otimes behave distributively (cf. (1.12)), tensorial powers of disjoint sums can be expressed by some kind of binomial expansion. By observing (1.5) we get

$$(6.4) \quad \langle (k, 1, n) \oplus (1, m, 1) \rangle^{\otimes s} = \bigoplus_{\sigma=0}^s \binom{s}{\sigma} \odot \langle k^\sigma, m^{s-\sigma}, n^\sigma \rangle,$$

where \odot denotes multiple use of \oplus . Then Lemma 6.1 and the inequalities (2.5), (2.11) yield

$$rk\left(\bigoplus_{\sigma} \binom{s}{\sigma} \odot \langle k^\sigma, m^{s-\sigma}, n^\sigma \rangle\right) \leq (1 + 2s)^2(kn + 1)^s.$$

At this point we use the additivity of rk , hence

$$\sum_{\sigma=0}^s \binom{s}{\sigma} rk\langle k^\sigma, m^{s-\sigma}, n^\sigma \rangle \leq (1 + 2s)^2(kn + 1)^s.$$

Next we apply (1.11) rewritten as

$$(KMN)^{\omega/3} \leq rk\langle K, M, N \rangle \quad \text{for all } K, M, N$$

and obtain

$$\sum_{\sigma=0}^s \binom{s}{\sigma} (kn)^{\sigma\omega/3} \cdot m^{(s-\sigma)\omega/3} \leq (1 + 2s)^2(kn + 1)^s,$$

$$(kn)^{\omega/3} + m^{\omega/3} \leq (1 + 2s)^{2/s}(kn + 1).$$

Finally $s \rightarrow \infty$ proves the next theorem.

THEOREM 6.3. *Assume that the additivity conjecture for the exact rank is true. Then $\omega(F) \leq 3\alpha$, where*

$$(6.5) \quad (kn)^\alpha + ((k - 1)(n - 1))^\alpha = kn + 1.$$

As announced, the same bound will be obtained in the next section without any unproved hypothesis. It is a remarkable fact that the preceding argument has led to an upper bound for the complexity of matrix multiplication though no algorithm has been specified, not even implicitly.

Table 6.1 shows that $k = n = 4$, i.e., $16^\alpha + 9^\alpha = 17$, gives the best bound $\omega(F) \leq 2.548$. We have also tabulated the case $r_2(\langle 2, 1, 3 \rangle \oplus \langle 1, 2, 1 \rangle) = 7$. Theorem 4.1 would give nothing better than Strassen's exponent $\lambda(7, 8)$, but here we get a considerably smaller value due to disjointness.

TABLE 6.1.
Some exponents obtained from Theorem 6.3

k	n	kn	m	3α
2	3	6	2	2.7341...
2	4	8	3	2.6657...
3	3	9	4	2.5938...
3	4	12	6	2.5653...
3	5	15	8	2.5614...
4	4	16	9	2.5479...
4	5	20	12	2.5486...
4	6	24	15	2.5543...
5	5	25	16	2.5506...
5	6	30	20	2.5568...
6	6	36	25	2.5629...
10	10	100	81	2.6119...

7. Multiple compression. Now we come to the following generalization of Theorem 4.1, which gives better results whenever the partial matrix multiplication splits up into several disjoint pieces, including also those cases where the pieces themselves may be partial again.

THEOREM 7.1. *Let t_1, t_2, \dots, t_p be tensors of partial matrix multiplication with f_i ones in t_i ($1 \leq i \leq p$) and border rank $\underline{rk}(t_1 \oplus t_2 \oplus \dots \oplus t_p) \leq l$ over the field F . Then the exponent of matrix multiplication over F is bounded by $\omega(F) \leq 3\alpha$, where*

$$(7.1) \quad f_1^\alpha + f_2^\alpha + \dots + f_p^\alpha = l.$$

The proof is organized in two stages. Since at present the most favorable applications can be obtained from the special case in which all pieces are total, i.e.,

$$(7.2) \quad t_i = \langle k_i, m_i, n_i \rangle, \quad f_i = k_i m_i n_i,$$

we will first (in subsections 7.1–3) give a self-contained proof for this version. It is only the extension to the case of partial pieces in § 7.4 where we have to refer to the rather involved proof methods of § 4 (and for which again infinite F is assumed).

The special case $p = 1$ is already covered by Theorem 4.1 (or by (1.11) in the total case). Furthermore we can ignore the trivial case $\alpha = 1$. In view of $\underline{rk}(t_1 \oplus \dots \oplus t_p) \geq p$ we therefore may assume

$$(7.3) \quad f_1 + f_2 + \dots + f_p > l \geq p \geq 2.$$

7.1. An iterative procedure. We start by describing a method for the approximate computation of the solution α of the equation (7.1). Consider the function φ defined on $0 \leq \tau \leq 1$ by

$$\varphi(\tau) = \ln \left(\sum_i f_i^\tau \right) / \ln l.$$

Since (7.3) implies $f_i > 1$ for some i , its derivative is greater than some positive constant, say

$$(7.4) \quad \varphi'(\tau) \geq \gamma_0 > 0.$$

In view of $\varphi(0) = \ln p / \ln l \leq 1 < \varphi(1)$ there exists a unique solution $\alpha \in [0, 1)$ such that $\varphi(\alpha) = 1$. Given any $\tau > \alpha$ we have $\sum_i f_i^\tau > l$; let $\theta < 1$ be such that $(\sum_i f_i^\tau)^\theta = l$, i.e.,

$\theta = 1/\varphi(\tau)$. By the general inequality $(\sum_i u_i)^\theta < \sum_i u_i^\theta$ for $\theta < 1$ and $u_i = f_i^\tau \geq 1$ we get $l < \sum_i f_i^{\tau\theta}$, hence also $\tau\theta > \alpha$. This can be used for the construction of an iterative algorithm.

LEMMA 7.2. *The sequence $1 = \tau_0, \tau_1, \tau_2, \dots$, recursively defined by*

$$(7.5) \quad \tau_{j+1} = \tau_j \theta_j, \quad \text{where } \theta_j = 1/\varphi(\tau_j) = \ln l / \ln \left(\sum_i f_i^{\tau_j} \right),$$

is strictly decreasing and converges to the solution α of (7.1).

Proof. Let $\tau_j = \alpha + \delta$ with $\delta > 0$. Then (7.4) yields $\varphi(\tau_j) \geq 1 + \gamma_0 \delta$ and we obtain

$$\tau_{j+1} - \alpha = \tau_j \theta_j - (\tau_j - \delta) = \delta - \tau_j \left(1 - \frac{1}{\varphi(\tau_j)} \right) \leq \delta - (\alpha + \delta) \frac{\gamma_0 \delta}{1 + \gamma_0 \delta}.$$

Hence $\tau_j - \alpha = \delta$ implies

$$(7.6) \quad \tau_{j+1} - \alpha \leq \delta(1 - \gamma_1 \alpha) - \gamma_1 \delta^2,$$

with some constant $\gamma_1 > 0$. Since $\alpha \geq 0$, this proves convergence $\tau_j \rightarrow \alpha$ at least in the qualitative sense which will be sufficient for the final proof of Theorem 7.1 in § 7.3, where we will show $\omega(F) \leq 3\tau_j$ for all j by an induction argument.

Of course, a posteriori we then induce from $\omega(F) \geq 2$ that, in all applications of the theorem, α cannot be smaller than $\frac{2}{3}$, hence (7.6) guarantees linear convergence. Thus the recursion (7.5) represents a simple and numerically stable iterative procedure for the computation of such α 's. In case of the equation $6^\alpha + 2^\alpha = 7$ mentioned at the end of § 6 we have, for instance, $3\tau_0 = 3.0$, $3\tau_1 = 2.8073 \dots$, $3\tau_2 = 2.7555 \dots$, $3\tau_3 = 2.7405 \dots$, etc.

7.2. How to use disjointness. By definition of rk and the assumptions of Theorem 7.1 there exists an integer $c \geq 0$ such that (in the total case, see (7.2)):

$$r_c(\langle k_1, m_1, n_1 \rangle \oplus \dots \oplus \langle k_p, m_p, n_p \rangle) \leq l.$$

The s -fold tensorial power can be expressed by a multinomial expansion similar to (6.4), and Lemma 2.5 then yields

$$r_{cs} \left(\bigoplus_{s_1 + \dots + s_p = s} \frac{s!}{s_1! \dots s_p!} \odot \left\langle \prod_i k_i^{s_i}, \prod_i m_i^{s_i}, \prod_i n_i^{s_i} \right\rangle \right) \leq l^s.$$

This inequality can be read as a statement about economic *mass production*. In order to simplify the further analysis we restrict everything to just one product type by selecting a particular partition $s = s_1 + \dots + s_p$. (An optimal choice of these parameters will be discussed in § 7.3.) Then we are left with E disjoint copies of $\langle K, M, N \rangle$, where

$$(7.7) \quad E = \frac{s!}{s_1! \dots s_p!}, \quad K = \prod_i k_i^{s_i}, \quad M = \prod_i m_i^{s_i}, \quad N = \prod_i n_i^{s_i}.$$

Since t' is always a subtensor of $t' \oplus t''$, Lemma 3.2 shows that the above bound holds also for the restricted disjoint sum, i.e.,

$$(7.8) \quad r_{cs}(E \odot \langle K, M, N \rangle) \leq l^s.$$

By symmetrization and Lemma 2.5 we get

$$r_{3cs}(E^3 \odot \langle P, P, P \rangle) \leq l^{3s},$$

where $P = KMN$ is given explicitly (see (7.2)) as

$$(7.9) \quad P = \prod_i f_i^{s_i}.$$

Finally (2.11) yields

$$(7.10) \quad rk(E^3 \odot \langle P, P, P \rangle) \leq (1 + 3cs)^2 l^{3s}.$$

Now we have arrived at the crucial point of the whole proof: Inequality (7.10) means that it is possible to perform E^3 disjoint multiplications of $P \times P$ matrices by not more than $(1 + 3cs)^2 l^{3s}$ multiplications. Then this number of multiplications will also suffice to carry out any sample of at most E^3 multiplications of $P \times P$ matrices (this could be proved in a formal way by introducing suitable homomorphisms). We could exploit this, for instance, in one big multiplication of two $EP \times EP$ matrices, which certainly does not require more than E^3 block multiplications of size $P \times P$, but as we already know exponents smaller than 3 we should do better. The best we can achieve by such an argument is the following assertion.

LEMMA 7.3. *Let E and P be given by (7.7), (7.9). Then $m^*(q|F) \leq E^3$ implies $m^*(qP|F) \leq (1 + 3cs)^2 l^{3s}$.*

The idea is to use this result for the construction of better and better algorithms recursively.

7.3. The induction argument. Our goal is to prove $\omega(F) \leq 3\tau_j$ by induction on j , where we refer to the sequence of Lemma 7.2; for $j = 0$ this simply means $\omega(F) \leq 3$. An equivalent version of the induction hypothesis is

Prop (j): *For every $\tau > \tau_j$ there is a number $N_0(\tau)$ such that*

$$(7.11) \quad m^*(N|F) \leq N^{3\tau} \text{ for all } N \geq N_0(\tau).$$

In order to prove Prop (j + 1) we consider an arbitrary $\tau' > \tau_{j+1} = \tau_j \theta_j$ and choose $\tau > \tau_j$ such that $\tau \theta_j < \tau'$, say $\tau = \frac{1}{2}(\tau_j + \tau' / \theta_j)$. These numbers are held fixed in the sequel. Now consider any $s \geq 2$. By choosing the partition $s = s_1 + \dots + s_p$ used in § 7.2 such that (see (7.7), (7.9))

$$EP^{\tau_j} = \frac{s!}{s_1! \dots s_p!} \prod_i f_i^{\tau_j s_i}$$

becomes maximal among all terms of the multinomial expansion of $(\sum_i f_i^{\tau_j})^s$ we obtain

$$(7.12) \quad EP^{\tau} \geq EP^{\tau_j} \geq \left(\sum_i f_i^{\tau_j} \right)^s / \binom{s+p-1}{p-1}.$$

With regard to Lemma 7.3 consider the maximal integer q with $m^*(q|F) \leq E^3$. Certainly $m^*(q|F) > \frac{1}{8}E^3$, for otherwise q could be replaced by $2q$. Large values of s will induce large E 's and q 's such that $q \geq N_0(\tau)$ holds, and (7.11) may be used. Thus we get $q^{3\tau} > \frac{1}{8}E^3$, $(qP)^{\tau} > \frac{1}{2}EP^{\tau}$; this can be combined with (7.12) and the bound $m^*(qP|F) \leq (1 + 3cs)^2 l^{3s}$ from Lemma 7.3 such that, after taking logarithms,

$$(7.13) \quad \frac{\ln m^*(qP|F)}{\ln(qP)} \leq \tau \frac{3s \ln l + O(\ln s)}{s \ln(\sum_i f_i^{\tau_j}) - O(\ln s)}$$

is obtained. For $s \rightarrow \infty$ the right-hand side tends to $3\tau \theta_j$ (see (7.5)), and τ was chosen such that $3\tau \theta_j < 3\tau'$. Therefore we can choose some interpolating exponent β , say $\beta = \frac{3}{2}(\tau \theta_j + \tau')$, and specify some s sufficiently large such that (7.13) yields $m^*(qP|F) \leq (qP)^{\beta}$ for the corresponding number qP . This finally implies $m^*(N|F) = O(N^{\beta})$, and in

view of $\beta < 3\tau'$ there is some $N_0(\tau')$ with $m^*(N|F) \leq N^{3\tau'}$ for $N \geq N_0(\tau')$. Hereby we have finished the induction argument which completes the proof of Theorem 7.1 in the case of total pieces.

7.4. Partial pieces. If the tensors t_i describe arbitrary partial matrix multiplications, then we have to study the analogue of (3.6) for each f_i . There are numbers m_i , and $k_{i,\mu}, n_{i,\mu}$ for $1 \leq \mu \leq m_i$ counting variable-positions, such that

$$f_i = \sum_{\mu=1}^{m_i} k_{i,\mu} n_{i,\mu}.$$

The tensorial power $(t_1 \oplus \dots \oplus t_p)^{\otimes s}$ is a huge disjoint sum of partial matrix multiplication tensors $t_\pi = t_{\nu_1} \otimes t_{\nu_2} \otimes \dots \otimes t_{\nu_s}$ with multi-indices $\pi = (\nu_1, \dots, \nu_s) \in \{1, \dots, p\}^s$. Observe that in general such partial tensor factors do not commute; $t_1 \otimes t_2$ and $t_2 \otimes t_1$, for instance, need not show the same pattern of variable-positions. Nevertheless we can again choose one partition $s = s_1 + \dots + s_p$ and then select just those products t_π with multi-indices $\pi = (\nu_1, \dots, \nu_s)$ which, for each i , contain exactly s_i of the ν 's equal to i . The set Γ of such π 's has $E = s! / (s_1! \dots s_p!)$ elements, and each such t_π contains exactly $f_1^{s_1} \dots f_p^{s_p}$ ones.

Instead of the inequality (7.8) obtained in the total case we now get

$$(7.14) \quad r_{cs} \left(\bigoplus_{\pi \in \Gamma} t_\pi \right) \leq l^s.$$

This bounds the number of multiplications necessary to perform (approximately) a collection of E disjoint partial matrix multiplications $A_\pi B_\pi$. The distribution of the variable-positions in these matrices A_π and B_π will usually depend on π , but if we choose (similar to (4.2)) p individual partitions

$$(7.15) \quad s_i = \sigma_{i,1} + \dots + \sigma_{i,m_i}, \quad (1 \leq i \leq p),$$

then for any $\pi \in \Gamma$ there are at least

$$(7.16) \quad M = \prod_{i=1}^p \frac{s_i!}{\sigma_{i,1}! \dots \sigma_{i,m_i}!}$$

columns in A_π each having K variable-positions, and corresponding rows in B_π each having N variable-positions, where

$$(7.17) \quad K = \prod_{i=1}^p \prod_{\mu=1}^{m_i} k_{i,\mu}^{\sigma_{i,\mu}}, \quad N = \prod_{i=1}^p \prod_{\mu=1}^{m_i} n_{i,\mu}^{\sigma_{i,\mu}}.$$

Therefore we can apply the compression argument of § 4.2 to each of these pairs A_π, B_π individually, thereby obtaining E disjoint total matrix multiplications $U_\pi V_\pi$ of size (K, M, N) .

Correspondingly (7.14) yields

$$r_{cs}(E \odot \langle K, M, N \rangle) \leq l^s,$$

which is now the same as (7.8), with other values for K, M, N , however. Hence the only further point where our previous proof of Theorem 7.1 needs modification is the lower bound for $P = KMN$. As in § 4.4 we choose the partitions (7.15) such that maximal terms of the multinomial expansions of

$$f_i^{s_i} = \left(\sum_{\mu} k_{i,\mu} n_{i,\mu} \right)^{s_i}$$

are obtained. Then (7.16) and (7.17) give

$$\begin{aligned}
 P = KMN &= \prod_{i=1}^p \left(\frac{s_i!}{\sigma_{i,1}! \cdots \sigma_{i,m_i}!} \prod_{\mu=1}^{m_i} (k_{i,\mu} n_{i,\mu})^{\sigma_{i,\mu}} \right) \\
 &\cong \prod_{i=1}^p (f_i^{s_i} / (s_i + m_i - 1)) \cong \left(\prod_{i=1}^p f_i^{s_i} \right) / (s + g)^g,
 \end{aligned}$$

where $g = \sum_i (m_i - 1)$ is constant. Comparison with (7.9) shows that this can cause only an extra factor $(s + g)^{g\tau_i}$ in the denominator of the right-hand side of (7.12), which does not affect the further steps of our proof in § 7.3 except for the increase of an O -term in (7.13).

8. Further improvements. The theoretical insight gained so far is incomplete with respect to the following points. We were unable to generalize Theorem 4.1 to the evaluation of $\text{trace}(ABC)$ in the case where A, B and C are partial, or equivalently, to admit that matrix multiplication may be partial also in the sense that only some of the elements of the product matrix are to be computed. Furthermore, there is an odd discontinuity in passing from partialness to disjointness which became obvious in § 6 with the example in Fig. 5.2: border rank 13 for 19 ones in one partial tensor leads to $\lambda(13, 19) = 2.613 \cdots$, while setting $b_{1,1} = 0$ (that is, application of a homomorphism) gives two disjoint pieces with $f_1 = 12, f_2 = 6$ and the bound $\omega(F) \leq 2.565 \cdots$ by Theorem 7.1. This indicates that there must be some interpolating result.

Of course, the most provoking lack of knowledge still is about the true value of $\omega(F)$, say for the field of rationals. As mentioned in the introduction, V. Pan was able to derive the smaller bound $\omega(F) < 2.522$ by combining our Theorem 7.1 with his techniques. In the remainder we present his design (which in particular is based upon [9, Table 11.1]) in a modified version which has the advantage of working for all fields F and which enables us to get some further tiny improvements.

Consider $1 \times k$ matrices $A, B, C, k \times n$ matrices X, Y, Z and $n \times 1$ matrices U, V, W . They can be used to set up three disjoint matrix multiplications as given by means of the trilinear form

$$\begin{aligned}
 (8.1) \quad \psi &= \text{trace}(AYW \oplus UBZ \oplus XVC) \\
 &= \sum_{i,j} (a_i y_{i,j} w_j + u_j b_i z_{i,j} + x_{i,j} v_j c_i),
 \end{aligned}$$

where summation now always runs over $1 \leq i \leq k, 1 \leq j \leq n$. The corresponding tensor is $\langle 1, k, n \rangle \oplus \langle n, 1, k \rangle \oplus \langle k, n, 1 \rangle$. We hope the reader will appreciate the alphabetic heuristics in our choice of letters and the abbreviations

$$a = \sum_i a_i, \quad y_i = \sum_j y_{i,j}, \quad w = \sum_j w_j, \quad \text{etc.},$$

which are used to write down the rather formidable identity

$$\begin{aligned}
 (8.2) \quad &\varepsilon^{12} \sum_{i,j} (a_i + \varepsilon^4 u_j + \varepsilon^6 x_{i,j})(b_i + \varepsilon^2 v_j + \varepsilon^5 y_{i,j})(c_i + \varepsilon^3 w_j + \varepsilon^4 z_{i,j}) \\
 &\quad - \varepsilon^8 \sum_i (a_i + \varepsilon^8 u + \varepsilon^{10} x_i)(b_i + \varepsilon^6 v + \varepsilon^9 y_i)(c_i + \varepsilon^7 w + \varepsilon^8 z_i) \\
 &\quad - \varepsilon^4 \sum_j (a + \varepsilon^8 u_j)(b + \varepsilon^6 v_j)(c + \varepsilon^7 w_j) + (a + \varepsilon^{12} u)(b + \varepsilon^{10} v)(c + \varepsilon^{11} w) \\
 &= \varepsilon^{20} \psi + \varepsilon^{18} \sum_{i,j} a_i v_j z_{i,j} + O(\varepsilon^{21}) + (n\varepsilon^{12} - \varepsilon^8) \sum_i a_i b_i c_i + (1 - n\varepsilon^4) abc.
 \end{aligned}$$

In order to cancel the unwanted ϵ^{18} -terms we set up a second trilinear form $\bar{\psi}$ of the same kind, which involves the same matrices A, V, Z but now combined with new matrices $\bar{B}, \bar{C}, \bar{U}, \bar{W}, \bar{X}, \bar{Y}$ built up from other disjoint sets of variables. For

$$\bar{\psi} = \text{trace} (A \bar{Y} \bar{W} \oplus (-\bar{U}) \bar{B} (-Z) \oplus \bar{X} V \bar{C})$$

the corresponding identity (8.2) will contain the ϵ^{18} -terms with a minus sign, if the signs of $(-\bar{U})$ and $(-Z)$ are properly observed. Thus addition yields a sum of $2(k+1)(n+1)$ triads equal to the sum of the right-hand sides, which is

$$(8.3) \quad \epsilon^{20}(\psi + \bar{\psi}) + O(\epsilon^{21}) + (n\epsilon^{12} - \epsilon^8) \sum_i (a_i b_i c_i + a_i \bar{b}_i \bar{c}_i) + (1 - n\epsilon^4)(abc + a\bar{b}\bar{c}).$$

Subtraction of another $2k+2$ triads finally shows that the trilinear form $\psi + \bar{\psi}$ has an approximate decomposition of order 20 and length $l = 2(k+1)(n+2)$. The corresponding tensor consists of three disjoint pieces

$$(8.4) \quad t_1 = \langle 1, k, 2n \rangle, \quad t_2 = \langle n, 2, k \rangle, \quad t_3 = \langle 2k, n, 1 \rangle$$

all having the same volume $f_1 = f_2 = f_3 = 2kn$.

Application of Theorem 7.1 now yields the equation $3(2kn)^\alpha = 2(k+1)(n+2)$. The smallest α is attained for $k = 5, n = 11$; from $3 \cdot 110^\alpha = 156$ we obtain

$$(8.5) \quad \omega(F) \leq 3 \ln 52 / \ln 110 = 2.5218127 \dots$$

Although this bound happens to have the same shape as bounds derived from Theorem 4.1 we should keep in mind the iterated construction of algorithms in § 7.3. The first step (see (7.5)) would give only $3\tau_1 = 2.612 \dots$

Finally we want to point out that the bound (8.5) is certainly not the end of the story. We give a sketchy argument (also of some interest in itself) which yields another small improvement. Any sum of l triads can be viewed as a homomorphic image of the trilinear form belonging to the tensor $l \odot \langle 1, 1, 1 \rangle$, but in the case of the foregoing construction we have a bit more information. The $2k+2$ triads which had to be subtracted from (8.3) are in the shape of the tensor $(k+1) \odot \langle 1, 1, 2 \rangle$. Given any combination $d \odot \langle 1, 1, 1 \rangle \oplus e \odot \langle 1, 1, 2 \rangle$, symmetrization generates the tensor

$$(8.6) \quad t^* = d^3 \odot \langle 1, 1, 1 \rangle \oplus (d^2 e) \odot (\langle 1, 1, 2 \rangle \oplus \langle 1, 2, 1 \rangle \oplus \langle 2, 1, 1 \rangle) \\ \oplus (de^2) \odot (\langle 1, 2, 2 \rangle \oplus \langle 2, 1, 2 \rangle \oplus \langle 2, 2, 1 \rangle) \oplus e^3 \odot \langle 2, 2, 2 \rangle.$$

Now the point is that Strassen's bound $rk\langle 2, 2, 2 \rangle = 7$ causes a saving of e^3 , hence $rk(t^*) \leq (d+2e)^3 - e^3$. On the other hand, symmetrization applied to $(t_1 \oplus t_2 \oplus t_3)$ will generate a disjoint sum of 27 pieces all having the same volume $(2kn)^3$. In the special case $k = 5, n = 11$ used above we have $d = 144, e = 6$. Thus Theorem 7.1 finally yields $\omega(F) \leq 3\bar{\alpha}$, where $27(110^3)^{\bar{\alpha}} = 156^3 - 6^3$, hence $\bar{\alpha} < \ln 52 / \ln 110$. Numerically we get $\omega(F) \leq 2.5218006 \dots$

Obviously, using higher tensorial powers of the tensor t^* in (8.6) one can similarly derive further improvements.

REFERENCES

- [1] D. BINI, M. CAPOVANI, G. LOTTI AND F. ROMANI, *O(n^{2.7799}) complexity for approximate matrix multiplication*, Information Proc. Letters, 8 (1979), pp. 234-235.
- [2] D. BINI, *Relations between EC-algorithms and APA-algorithms, applications*, Nota Interna, B79/8 (March 1979), I.E.I. Pisa.
- [3] H. F. DE GROOTE, *On varieties of optimal algorithms for the computation of bilinear mappings II*, Theoret. Comput. Sci., 7 (1978), pp. 127-148.

- [4] 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.
- [5] J. E. HOPCROFT AND J. MUSINSKI, *Duality applied to the complexity of matrix multiplication and other bilinear forms*, this Journal, 2 (1973), pp. 159–173.
- [6] J. D. LADERMAN, *A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications*, Bull. Amer. Math. Soc., 82 (1976), pp. 126–128.
- [7] V. YA. PAN, *Strassen's algorithm is not optimal*, Proc. 19th IEEE Symposium on Foundations of Computer Science, 1978, pp. 166–176.
- [8] ———, *New fast algorithms for matrix operations*, IBM Thomas J. Watson Research Center, Report RC 7555, 1979; this Journal, 9 (1980), pp. 321–342.
- [9] ———, *Field extension and trilinear aggregating, uniting and canceling for the acceleration of matrix multiplications*, Proc. 20th IEEE Symposium on Foundations of Computer Science, 1979.
- [10] V. STRASSEN, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.
- [11] ———, *Vermeidung von Divisionen*, J. Reine Angew. Math. 264 (1973), pp. 184–202.

A $T = O(2^{n/2})$, $S = O(2^{n/4})$ ALGORITHM FOR CERTAIN NP-COMPLETE PROBLEMS*

RICHARD SCHROEPPel† AND ADI SHAMIR‡

Abstract. In this paper we develop a general-purpose algorithm that can solve a number of NP-complete problems in time $T = O(2^{n/2})$ and space $S = O(2^{n/4})$. The algorithm can be generalized to a family of algorithms whose time and space complexities are related by $T \cdot S^2 = O(2^n)$. The problems it can handle are characterized by a few decomposition axioms; they include knapsack problems, exact satisfiability problems, set covering problems, etc. The new algorithm has considerable cryptanalytic significance, since it can break knapsack-based cryptosystems with up to $n = 100$ generators.

Key words. NP-complete problem, time/space tradeoff, knapsack problem, Merkle-Hellman cryptosystem

1. Introduction. Every NP-complete problem can be solved in $O(2^n)^1$ time by exhaustive search, but this complexity becomes prohibitive when n exceeds 40 or 50. Assuming that $NP \neq P$, we cannot hope to find algorithms whose worst-case complexity is polynomial, but it is both theoretically interesting and practically important to determine whether substantially faster algorithms exist. Researchers have so far discovered a few special-purpose algorithms (most notably a $T = S = O(2^{n/2})$ algorithm for knapsacks by Horowitz and Sahni [1974] and a $T = O(2^{n/3})$, $S = O(n)$ algorithm for cliques by Tarjan and Trojanowski [1977]), but no comprehensive theory of such subexponential algorithms has been developed. In this paper we describe a general-purpose algorithm which can solve a fair number of NP-complete problems (including knapsack, partition, exact satisfiability, set covering, hitting set, disjoint domination in graphs, etc.) in time and space complexities which are related by the tradeoff curve $T \cdot S^2 = O(2^n)$ for $\Omega(2^{n/2}) \leq T \leq O(2^n)$. The novel properties of this algorithm are:

(i) The time/space complexities of the algorithm are considerably better than those of all the algorithms published so far for these problems. Furthermore, the algorithm is completely practical in the sense that it is easy to program and its overhead is small, and thus it can handle problems which are almost twice as big as those handled by previous algorithms.

(ii) The algorithm demonstrates an interesting tradeoff between time and space—in order to decrease time by a factor c , it is enough to increase space by a factor of \sqrt{c} . Since space is much more expensive than time, this tradeoff has very favorable economics.

(iii) The problems to which the algorithm can be applied are characterized axiomatically by their behavior under composition. This approach introduces a natural subclassification of NP-complete problems and indicates how a problem-independent theory of subexponential algorithms may be constructed.

One of the most important applications of the new algorithm is in cryptanalysis, since many of the new public-key cryptosystems are based on large NP-complete problems (Diffie and Hellman [1976]). With current technology, the practical limit on the number of operations a cryptanalyst can perform is between 2^{50} and 2^{60} (a parallel computer with 1000 processors whose cycle time is one microsecond performs 2^{50}

* Received by the editors January 22, 1980.

† Information International, Culver City, California.

‡ Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. This research was partially supported by the U.S. Office of Naval Research under contract N00014-76-C-0366.

¹ Throughout this paper, we ignore polynomial multiplicative factors in the O -notation of exponential functions. These factors are usually of degree 0 or 1, and their practical effect is very small.

operations in about two weeks), and the practical limit on the number of memory cells he can use is between 2^{25} and 2^{30} . By choosing the point $T = O(2^{n/2})$ and $S = O(2^{n/4})$ on the time/space tradeoff curve, instances with n up to 100 are within reach, and thus the new algorithm can break all the knapsack-based cryptosystems recommended so far in the literature (e.g., Merkle and Hellman [1978]). This cryptanalytic attack can be foiled by increasing the minimum recommended size from $n = 100$ to $n = 200$ (at the expense of tripling the key size and the encryption time), but it is a clear warning against overconfidence and narrow safety margins in public-key cryptosystems.

The problems our algorithm can handle are described in § 2. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm is presented in § 3. In § 4 we generalize this algorithm to a family of algorithms whose time and space complexities are related by the $T \cdot S^2 = O(2^n)$ tradeoff curve.

2. A calculus of problems. To make our basic algorithm as versatile as possible and to expose the minimum conditions that guarantee its correctness, we define the notion of a problem in a fairly abstract way.

DEFINITION. A *problem of size n* is a predicate P over n -bit binary strings. A string x is a *solution* (or a *witness*) of the problem if $P(x)$ is true. The goal is to find one such x , if it exists.

Example. The predicate of the knapsack problem “is there a bit string $x_1x_2x_3x_4x_5$ such that $x_1 \cdot 7 + x_2 \cdot 3 + x_3 \cdot 9 + x_4 \cdot 6 + x_5 \cdot 2 = 11$?” is of size 5, and its solutions are 01011 and 00101.

Remark. The size $|P|$ of a problem P is defined as the number of bits in its solution rather than the number of bits in its description, since the $O(2^n)$ complexity of exhaustive search (upon which we want to improve) is determined primarily by the size of the solution space. However, to make our results strictly correct we have to assume that these two measures are polynomially related, i.e., that we are not given huge descriptions of problems with very few bits of unknowns.

One of the most useful algorithmic techniques for solving problems is divide-and-conquer. Given a problem P , we decompose it into a number of subproblems (usually two of half size each), solve them separately, and then combine their solutions. To simplify the mathematical analysis of this process, we introduce the following operator.

DEFINITION. A binary operator \oplus on problems is a *composition operator* if

- (i) it is *additive*: for all P' and P'' , $|P' \oplus P''| = |P'| + |P''|$;
- (ii) it is *sound*: for any two solutions x' of P' and x'' of P'' , the string concatenation² $x'x''$ is a solution of $P' \oplus P''$;
- (iii) it is *complete*: for any solution x of P and for any representation of x as $x = x'x''$, there are problems P' and P'' such that x' solves P' , x'' solves P'' , and $P = P' \oplus P''$;
- (iv) it is *polynomial*: the problem $P' \oplus P''$ can be calculated in time which is polynomial in the sizes of P' and P'' .

Intuitively, \oplus is sound if any two solutions of the subproblems P' and P'' can be easily combined in order to get a solution for the original problem P , and complete if any solution of P can be obtained in such a way:

$$\{x|P(x)\} = \bigcup_{P' \oplus P'' = P} \{x'x''|P'(x') \text{ and } P''(x'')\}.$$

A pair of problems P', P'' is said to be a *decomposition* of P if $P' \oplus P'' = P$; in general a problem can have many possible decompositions. To solve P , we can try out all its possible P', P'' decompositions until we find a pair of solvable subproblems. If P is

² The string concatenation can be replaced by any other simple operation which is length-additive.

solvable and \oplus is complete, these subproblems must exist, while if P is unsolvable and \oplus is sound, these subproblems cannot exist (otherwise their concatenated solutions would have solved P). There are many NP-complete problems for which composition operators exist; the following examples are typical.

Example. Let (b, a_1, \dots, a_n) be the *knapsack problem* in which the *target value* b is to be represented as the sum of a subset of the *generators* a_i . For any two problems $P' = (b', a'_1, \dots, a'_l)$ and $P'' = (b'', a''_1, \dots, a''_m)$ we define $P' \oplus P'' = (b' + b'', a'_1, \dots, a'_l, a''_1, \dots, a''_m)$ (i.e., we add the b 's and concatenate the a_i 's). We claim that this \oplus is a composition operator:

- (i) \oplus is additive, since the number of generators in $P' \oplus P''$ is by definition the sum of the corresponding numbers in P' and P'' .
- (ii) \oplus is sound, since $\sum_{i=1}^l x'_i a'_i = b'$ and $\sum_{i=1}^m x''_i a''_i = b''$ imply that $\sum_{i=1}^{l+m} x_i a_i = b' + b''$ (where the x_i are the bits of $x'x''$ and the a_i are the generators in $P' \oplus P''$).
- (iii) \oplus is complete, since for each $x = x'x''$ such that $|x'| = l$, $|x''| = m$, and $\sum_{i=1}^{l+m} x_i a_i = b$, there are b' and b'' such that x' satisfies $\sum_{i=1}^l x'_i a'_i = b'$, x'' satisfies $\sum_{i=l+1}^{l+m} x''_i a''_i = b''$ and the sum of these two subproblems is the original problem (b, a_1, \dots, a_{l+m}) .
- (iv) \oplus is polynomial, since the only operations involved are numeric addition and list concatenation.

Example. Let F be a formula in CNF (i.e., a conjunction of clauses which are disjunctions of literals which are variables x_i or their negations \bar{x}_i). The *satisfiability problem* is to find a truth-value assignment to the variables which makes at least one literal in each clause true. To decompose this problem, we can partition the list of variables into two complementary sublists, and try to satisfy by the two partial assignments two sets of clauses whose union contains all the clauses. In this generalized formulation, each subproblem corresponds to a sublist of variables and a subset of clauses in F , and the \oplus operator concatenates the sublists (if they are consecutive) and unions the subsets in P' and P'' . This operator is clearly additive and polynomial. It is sound since by definition $P' \oplus P''$ is satisfied by the concatenation of any pair of assignments that satisfy P' and P'' , and it is complete since for any x that solves P we can use the clauses which are actually satisfied by the prefix and suffix of x to define the appropriate P', P'' decomposition.

The relationship between problems (especially NP-complete problems) and their solutions is often asymmetric, since it may be much harder to find a solution for a given problem than to find a problem which is solved by a given solution. This motivates the following definition.

DEFINITION. A set of problems is *polynomially enumerable* if there is a polynomial time algorithm which finds for each bit string x the subset of problems which are solved by x .

Examples. (i) The set of all knapsack problems is not polynomially enumerable since for each x there are infinitely many knapsack problems which are solved by x .

(ii) The set of knapsack problems with a fixed set of a_i generators (but varying target values b) is polynomially enumerable, since for each x there is exactly one b such that $b = \sum_{i=1}^n x_i a_i$, and this b can be easily calculated.

(iii) The set of (generalized) satisfiability problems with a fixed formula F is polynomially enumerable, since the subset of clauses which are satisfied by the truth-value assignment x is uniquely defined and can be found by simple evaluation.

If a set of problems is polynomially enumerable, then all its solvable instances of size n can be tabulated (as problem/solution pairs) in $O(2^n)$ time and space. Again,

there are many NP-complete problems whose sets of subproblems are polynomially enumerable, and they have the curious property that it is almost as difficult to solve a single instance of size n as it is to solve all the instances of size n —in both cases we have to enumerate all the possible n -bit solutions.

The most restrictive and least intuitive condition we impose on problems is the following.

DEFINITION. A composition operator \oplus is *monotonic* if the problems of each size can be totally ordered in such a way that \oplus behaves monotonically: $|P'| = |P''|$ and $P' < P''$ imply that $P' \oplus P < P'' \oplus P$ and $P \oplus P' < P \oplus P''$.

Using this notion, we can state the main result of this paper (which is proved in the next section).

THEOREM 1. *If a set of problems is polynomially enumerable and has a monotonic composition operator, then its instances of size n can be solved in time $T = O(2^{n/2})$ and space $S = O(2^{n/4})$.*

Example. The \oplus operator on knapsack problems is monotonic if we order them lexicographically, since $(b', a'_1, \dots, a'_i) < (b'', a''_1, \dots, a''_i)$ implies that $b' + b, a'_1, \dots, a'_i, a_1, \dots, a_m < (b'' + b, a''_1, \dots, a''_i, a_1, \dots, a_m)$ and $(b + b', a_1, \dots, a_m, a'_1, \dots, a'_i) < (b + b'', a_1, \dots, a_m, a''_1, \dots, a''_i)$. (Note that this \oplus operator is not monotonic if P' and P'' are allowed to be of different sizes.) Consequently, our algorithm can solve knapsack problems.

Composition operators based on set unions are usually nonmonotonic, but they become monotonic if we replace the set unions by multiset unions:

LEMMA 2. (i) *If $|S| \geq 3$, then the subsets of S cannot be totally ordered in a way that makes the set union operator monotonic.*

(ii) *If S is denumerable, then the multisubsets of S with finite multiplicities can be totally ordered in a way that makes the multiset union operator monotonic.*

Proof. (i) Suppose that such an order exists. Without loss of generality, we can assume that for $a, b, c \in S$, $\{a\} < \{b\} < \{c\}$. By taking the set unions of these singletons with $\{a, c\}$ and by using the monotonicity of \cup , we get

$$\{a\} \cup \{a, c\} < \{b\} \cup \{a, c\} < \{c\} \cup \{a, c\};$$

this evaluates to

$$\{a, c\} < \{a, b, c\} < \{a, c\},$$

which is a contradiction.

(ii) Let S be $\{a_1, a_2, \dots\}$. The multisubsets of S with finite multiplicities can be represented by semi-infinite vectors of multiplicities (n_1, n_2, \dots) in which each n_i represents the number of occurrences of a_i . In this representation, multiset union is simply a componentwise addition of multiplicity vectors, and it is clearly a monotonic operator if we order the vectors lexicographically. \square

Example. By part (i) of the lemma, the (generalized) satisfiability problems cannot be totally ordered in a way that makes the \oplus operator monotonic, and thus our algorithm cannot be used to solve them.

Example. The *exact satisfiability problem* is similar to the satisfiability problem, except that we want to satisfy exactly one literal in each clause. Its subproblems consist of sublists of variables and multisets of clauses, and the multiplicity of each clause indicates how many literals are satisfied in it (e.g., the original problem corresponds to the multiset $(1, 1, \dots, 1, 0, 0, \dots)$). By part (ii) of the lemma, the \oplus composition operator that concatenates the sublists and adds the multiplicities is monotonic, and thus we can apply our algorithm to this variant of the satisfiability problem.

We leave it as an exercise for the reader to verify that all the NP-complete problems listed in the introduction have monotonic composition operators. This list is not exhaustive, and it is easy to come up with additional examples.

3. The algorithm. The algorithm uses the soundness and completeness of \oplus in order to reduce the general problem to the following combinatorial search problem.

DEFINITION. Given k problem/solution tables T_i with $O(2^{n/k})$ solvable problems each, a monotonic composition operator \oplus , and a problem P , the k -table problem is to determine whether there are k representatives $P_i \in T_i$ such that $P = P_1 \oplus P_2 \oplus \cdots \oplus P_k$ (under a given parenthesization).

Example. To reduce a given knapsack problem P with $n = 3m$ generators a_i to the 3-table problem, we

- (i) divide the generators into three sublists (a_1, \dots, a_m) , (a_{m+1}, \dots, a_{2m}) and $(a_{2m+1}, \dots, a_{3m})$;
- (ii) tabulate in T_i ($i = 1, 2, 3$) all the $O(2^{n/3})$ target values b_i which can be generated by summing a subset of the $n/3$ generators in the i th third of the problem;
- (iii) check whether the original target value b can be represented as $b = b_1 + b_2 + b_3$ for some $b_1 \in T_1$, $b_2 \in T_2$, $b_3 \in T_3$;
- (iv) concatenate the three solutions x_i tabulated for these b_i target values (if they exist) in order to get a solution $x = x_1 x_2 x_3$ for the original problem.

Example. To reduce an exact satisfiability problem to the 4-table problem, we divide the variable list into four quarters, enumerate for each quarter all the $O(2^{n/4})$ possible truth-value assignments, tabulate for each assignment the multiset of satisfied clauses, and determine whether there are four multiplicity vectors in the four tables whose sum is $(1, 1, \dots, 1, 0, 0, \dots)$.

This general technique is a mixture of divide-and-conquer and dynamic programming—we repeatedly divide problems into pairs of subproblems until we get k problems of sizes n/k each, and then finish by searching k problem/solution tables. Since we do not assume that \oplus is associative, we must fully parenthesize the $P_1 \oplus P_2 \oplus \cdots \oplus P_k$ sum to make it meaningful, but the completeness of \oplus implies that the solvability of this k -table problem does not depend on the parenthesis structure (i.e., we can choose the order that makes the search most efficient).

The obvious algorithm for the k -table problem is to try out all the $O(2^n)$ combinations of representatives from the k tables, and it is clearly optimal for $k = 1$. However, for $k \geq 2$, better algorithms exist.

THEOREM 3. *The 2-table problem can be solved in $O(2^{n/2})$ time and space.*

Proof. Consider the following algorithm:

- (1) Sort T_1 into increasing problem order;
sort T_2 into decreasing problem order.
- (2) Repeat until either T_1 or T_2 becomes empty (in which case print “unsolvable” and halt):
 $S \leftarrow \text{first}(T_1) \oplus \text{first}(T_2)$;
 if $S = P$ print “solvable” and halt;
 if $S < P$ delete first (T_1) from T_1 ;
 if $S > P$ delete first (T_2) from T_2 .

To prove the correctness of this algorithm, we have to show that whenever a problem is deleted from T_1 or T_2 , it cannot possibly participate in any sum which equals P (and thus the deletion cannot affect the correctness of the rest of the algorithm). Since T_2 is decreasing and \oplus is monotonic, $\text{first}(T_1) \oplus P_2 \leq \text{first}(T_1) \oplus \text{first}(T_2)$ for any

$P_2 \in T_2$. Consequently, the left-hand side cannot be equal to P if the right-hand side is smaller than P , and the deletion of first (T_1) from T_1 is justified. Similarly, $P_1 \oplus \text{first}(T_2) \geq \text{first}(T_1) \oplus \text{first}(T_2) = S > P$ justifies the deletion of first (T_2) from T_2 .

The time complexity of the sorting step is $O(2^{n/2}(n/2)) = O(2^{n/2})$, and the time complexity of the search step is $O(|T_1| + |T_2|) = O(2^{n/2})$ since we delete at least one element at each iteration. \square

Remark. This 2-table problem has been posed and solved by a number of authors under various disguises (e.g., Knuth [1973, p. 9], Horowitz and Sahni [1974]). In the rest of this paper we refer to this algorithm as *the basic algorithm*.

The basic algorithm can be easily extended to other values of k :

LEMMA 4. *The 3-table problem can be solved in $O(2^{2n/3})$ time and $O(2^{n/3})$ space.*

Proof. For each one of the $O(2^{n/3})$ problems $P_1 \in T_1$, use the basic algorithm on the T_2, T_3 tables in order to find a solution for $P = P_1 \oplus (P_2 \oplus P_3)$ in time $O(|T_i|) = O(2^{n/3})$. \square

LEMMA 5. *The 4-table problem with a nonbalanced parenthesis structure $P = P_1 \oplus (P_2 \oplus (P_3 \oplus P_4))$ can be solved in $O(2^{3n/4})$ time and $O(2^{n/4})$ space.*

Proof. For each one of the $O(2^{n/4})$ problems $P_1 \in T_1$, solve the remaining 3-table problem in $O(|T_i|^2) = O(2^{n/2})$ time. \square

All the time and space complexities considered so far satisfy the invariant relation $T \cdot S = O(2^n)$, and thus improvements in the space complexity make the time complexity worse by a similar factor. This trend is broken by the unexpected behavior of the following case.

THEOREM 6. *The 4-table problem with balanced parenthesis structure $P = (P_1 \oplus P_2) \oplus (P_3 \oplus P_4)$ can be solved in $O(2^{n/2})$ time and $O(2^{n/4})$ space.*

A direct application of the basic algorithm to the two $O(2^{n/2})$ supertables generated by the $(P_1 \oplus P_2)$ and $(P_3 \oplus P_4)$ combinations leads to a $T = S = O(2^{n/2})$ algorithm. However, the basic algorithm accesses the elements of the sorted supertables sequentially, and thus there is no need to store all the possible combinations simultaneously in memory—all we need is the ability to generate them quickly (on-line, upon request) in sorted order. To implement this key idea, we use two priority queues:

(i) Q' stores pairs of problems from T_1 and T_2 , enables arbitrary insertions and deletions to be done in logarithmic time, and makes the pair with the *smallest* $P_1 \oplus P_2$ sum accessible in constant time.

(ii) Q'' stores pairs of problems from T_3 and T_4 , enables arbitrary insertions and deletions to be done in logarithmic time, and makes the pairs with the *largest* $P_3 \oplus P_4$ sum accessible in constant time.

Efficient heap implementations of priority queues are described in Aho, Hopcroft, Ullman [1974].

The balanced 4-table algorithm.

- (1) Sort T_2 into increasing problem order;
 sort T_4 into decreasing problem order;
 insert into Q' all the pairs $(P_1, \text{first}(T_2))$ for $P_1 \in T_1$;
 insert into Q'' all the pairs $(P_3, \text{first}(T_4))$ for $P_3 \in T_3$.
- (2) Repeat until either Q' or Q'' becomes empty (in which case print "unsolvable" and halt):
 - $(P_1, P_2) \leftarrow$ pair with smallest $P_1 \oplus P_2$ sum in Q' ;
 - $(P_3, P_4) \leftarrow$ pair with largest $P_3 \oplus P_4$ sum in Q'' ;
 - $S \leftarrow (P_1 \oplus P_2) \oplus (P_3 \oplus P_4)$
 - if $S = P$ print "solvable" and stop;
 - if $S < P$ do

delete (P_1, P_2) from Q' ;
 if the successor P'_2 of P_2 in T_2 is defined,
 insert (P_1, P'_2) into Q' ;
 if $S > P$ do
 delete (P_3, P_4) from Q'' ;
 if the successor P'_4 of P_4 in T_4 is defined,
 insert (P_3, P'_4) into Q'' .

LEMMA 7. *The space complexity of this algorithm is $O(2^{n/4})$.*

Proof. It is easy to show by induction that at each stage a $P_1 \in T_1$ can participate in at most one pair in Q' , and a $P_3 \in T_3$ can participate in at most one pair in Q'' (the number of occurrences of $P_2 \in T_2$ and $P_4 \in T_4$ in Q' and Q'' can be higher). The space complexity of the priority queues is thus bounded by $O(|T_i|) = O(2^{n/4})$. \square

LEMMA 8. *The time complexity of this algorithm is $O(2^{n/2})$.*

Proof. Each (P_1, P_2) pair can be deleted from Q' at most once, since it is never reinserted into Q' . Similarly, each (P_3, P_4) pair can be deleted from Q'' at most once. At each iteration of step 2, one pair is deleted from Q' or Q'' , and thus the number of iterations cannot exceed the number of possible pairs, which is $O(2^{n/2})$. \square

LEMMA 9. *Q' can become empty only after we consider all the possible (P_1, P_2) pairs of problems from T_1, T_2 (similarly for Q'' and T_3, T_4).*

Proof. Initially P_1 shares a pair in Q' with the first element of T_2 . After each deletion of a (P_1, P_2) pair we reinsert P_1 together with the next larger element of T_2 , and thus the only way Q' can become empty is if each P_1 runs out of companions after a complete first-to-last scan of T_2 . \square

LEMMA 10. *The sums of the pairs extracted from Q' are in nondecreasing sorted order, and the sums of the pairs extracted from Q'' are in nonincreasing sorted order.*

Proof. The smallest (P_1, P_2) pair in Q' is replaced by a (P_1, P'_2) pair whose sum is larger (since T_2 is sorted and \oplus is monotonic), and thus the sum of the next pair extracted from Q' cannot be smaller than $P_1 \oplus P_2$. The proof for Q'' is similar. \square

Proof of Theorem 6. Lemmas 7, 8, 9 and 10 reduce the 4-table algorithm to the 2-table algorithm whose correctness was proved in Theorem 3. \square

4. Time/space tradeoffs.

LEMMA 11. *If a set of problems has a monotonic composition operator, then for any P and P' there is at most one P'' such that $P = P' \oplus P''$.*

Proof. If $P''_1 < P''_2$ are two different solutions, we get $P' \oplus P''_1 = P' \oplus P''_2$ which contradicts the monotonicity of \oplus (note that $|P''_1| = |P''_2| = |P| - |P'|$). \square

DEFINITION. The *complementation operator* \ominus is the partial binary operator defined by

$$P'' = P \ominus P' \quad \text{iff} \quad P = P' \oplus P''.$$

The problem P'' is the *complement* of P' with respect to P .

Example. Given two knapsack problems $P = (b, a_1, \dots, a_n)$ and $P' = (b', a'_1, \dots, a'_l)$, $P \ominus P'$ is defined (as $(b - b', a_{l+1}, \dots, a_n)$) if and only if $l < n$ and for all $1 \leq i \leq l$, $a_i = a'_i$.

Given two exact satisfiability problems P and P' , $P \ominus P'$ is defined if and only if they have the same CNF formula, the list of variables in P' is a prefix of the list of variables in P and the componentwise difference between their multiplicity vectors is nonnegative.

In all examples of monotonic \oplus operators considered so far, the \ominus operator is easy to compute in polynomial time (either directly or by a quick binary search on candidate problems).

THEOREM 12. *Let Q be a polynomially enumerable set of problems with a monotonic composition operator and a polynomial complementation operator, and let A be an algorithm that solves these problems in $O(2^{\alpha n})$ time and $O(2^{\beta n})$ space (for some $0 < \alpha, \beta < 1$). Then the problems in Q can be solved in any time/space combination along the tradeoff curve $T \cdot S^{(1-\alpha)/\beta} = O(2^n)$, $\Omega(2^{\alpha n}) \leq t \leq O(2^n)$.*

Proof. For each $0 \leq \gamma \leq 1$, let A_γ be the following algorithm:

- (1) Enumerate all the bit strings x' of size $(1 - \gamma) \cdot n$.
- (2) For each x' enumerate all the problems P' which are solved by x' .
- (3) For each P' , find its complement P'' with respect to P ; if it exists, use algorithm A to solve it; if it is solvable, concatenate x' with its solution x'' , print it out and halt.

Note that for $\gamma = 0$, A_0 reduces to a simple exhaustive search, while for $\gamma = 1$, A_1 reduces to A . A slight technical difficulty is that for each n the set of usable values of γ is discrete. However, for large values of n this set becomes essentially continuous.

The correctness of each A_γ follows from the soundness and completeness of \oplus in the usual way. The only new element is the unbalanced decomposition of P into problems of sizes $(1 - \gamma) \cdot n$ and $\gamma \cdot n$, but our definition of completeness is general enough to handle this case.

Algorithm A is applied at step 3 to a problem of size $\gamma \cdot n$ and thus its time complexity is $O(2^{\alpha \gamma n})$ and its space complexity is $O(2^{\beta \gamma n})$. Step 2 multiplies the time complexity by a polynomial factor, and step 1 multiplies it further by $O(2^{(1-\gamma)n})$. The overall time complexity of A_γ is thus $T_\gamma = O(2^{(\alpha\gamma - \gamma + 1)n})$ and its space complexity is $S_\gamma = O(2^{\beta \gamma n})$.

To find the invariant relation satisfied by the time/space complexities of all the A_γ algorithms, we use linear algebra in order to eliminate γ :

$$\begin{aligned} T \cdot S^{(1-\alpha)/\beta} &= O(2^{(\alpha\gamma - \gamma + 1)n}) \cdot O(2^{\beta \gamma n(1-\alpha)/\beta}) \\ &= O(2^{(\alpha\gamma - \gamma + 1)n} \cdot 2^{(\gamma - \gamma\alpha)n}) \\ &= O(2^n). \end{aligned} \quad \square$$

THEOREM 13. *If a polynomially enumerable set of problems has a monotonic composition operator and a polynomial complementation operator, then its instances of size n can be solved in any time/space combination along the tradeoff curve $T \cdot S^2 = O(2^n)$, $\Omega(2^{n/2}) \leq T \leq O(2^n)$.*

Proof. By Theorem 6, there exists an algorithm A with time complexity $T = O(2^{n/2})$ and space complexity $S = O(2^{n/4})$. By substituting $\alpha = \frac{1}{2}$ and $\beta = \frac{1}{4}$ into the general formula, we get the tradeoff curve $T \cdot S^2 = O(2^n)$. \square

While we conjecture that $T = O(2^{n/2})$ is optimal for all the k -table problems, we do not have any reason to believe that $S = O(2^{n/4})$ is optimal. If S can be reduced to $S = O(2^{n/6})$ or $S = O(2^{n/8})$ without worsening T , we can get even better tradeoff curves such as $T \cdot S^3 = O(2^n)$ or $T \cdot S^4 = O(2^n)$.

5. Open problems for further research.

- (i) Are there other axiomatically characterizable subsets of NP-complete problems which can be solved in $o(2^n)$ time?
- (ii) Can we use other properties of \oplus (besides monotonicity) in order to reduce the complexity of the k -table problem?
- (iii) What are the best search strategies for $k > 4$ tables?
- (iv) Is $T = \Omega(2^{n/2})$ a lower bound for all k ?
- (v) Is there an algorithm with $T = \theta(2^{n/2})$ but $S = o(2^{n/4})$?

(vi) Are there easy ways to determine whether a set of problems has a monotonic composition operator?

Acknowledgments. We would like to thank Martin Hellman and Ron Rivest for many fruitful discussions.

REFERENCES

- [1] A. AHO, J. HOPCROFT AND J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [2] W. DIFFIE AND M. HELLMAN, *New directions in cryptography*, IEEE Trans. Information Theory, IT-22 (1976), pp. 644–654.
- [3] E. HOROWITZ AND S. SAHNI, *Computing Partitions With Applications to the Knapsack Problem*, J. Assoc. Comput. Mach., 21 (1974), pp. 277–292.
- [4] D. KNUTH, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, MA, 1973.
- [5] R. MERKLE AND M. HELLMAN, *Hiding information and receipts in trap door knapsacks*, IEEE Trans. Information Theory, IT-24 (1978), pp. 525–530.
- [6] R. TARJAN AND A. TROJANOWSKI, *Finding a maximum independent set*, this Journal, 6 (1977), pp. 537–546.

PARALLEL SORTING WITH CONSTANT TIME FOR COMPARISONS*

ROLAND HÄGGKVIST† AND PAVOL HELL‡‡

Abstract. We prove that there exist graphs with n vertices and at most $2n^{5/3} \log n$ edges for which every acyclic orientation has in its transitive closure at least $\binom{n}{2} - 10n^{5/3}$ arcs. We conclude that with $2n^{5/3} \log n$ parallel processors n items may be sorted with all comparisons arranged in two time intervals. We also show that $\frac{1}{2}n^{3/2}$ processors are not sufficient to achieve the same end. These results are extended to parallel sorting in k time intervals, and related to other work on parallel sorting. The existence of sorting algorithms achieving the bounds is proved by nonconstructive methods. (The constants quoted in the abstract are somewhat improved in the paper.)

Key words. graph, acyclic orientation, transitive closure, sorting, parallel algorithm

1. Motivation and informal statement of the problem. Consider a game of the following variety. Player 1 chooses a linear order on the set $\{1, 2, \dots, n\}$ and player 2 is to discover this order by making as few binary comparisons as possible, i.e., by asking player 1 the smallest number of questions of the type “is $i < j$?”. (Player 1 must answer truthfully.)

There are several alternate rules for such a game. Player 2 may be required to formulate all the questions in advance. In this case the only strategy player 2 can adopt is to ask all $\binom{n}{2}$ questions. If the comparison “is $i < j$?” has not been explicitly made, player 2 would be unable to decide which of the elements i, j is larger, in case player 1 had chosen a linear order in which i and j are consecutive. In the other extreme, player 2 may be allowed to formulate each question separately, on the basis of all previous answers. This is, of course, precisely the case of sorting by binary comparisons, [1], [4], [5]. Player 1 is “nature” and player 2 is a computer programmed to sort an arbitrary n -element linearly ordered set. In other words, a strategy for player 2 is a sorting algorithm. It is well known [4], [5] that the best strategy player 2 can adopt will guarantee the completion of the sorting after fewer than $n \log n$ comparisons.

We are interested in games that lie between these two extremes. A positive integer k is given, and player 2 must complete the sorting (i.e., discover the linear order chosen by player 1) in k rounds. Within each round a set of questions is formulated on the basis of the answers to questions of all previous rounds; all comparisons within a round are made together. We note that for $k = 1$ this game is identical to our first example, and for $k > n \log n$ it coincides with the second example.

In practice, sorting of a large set on a computer is typically done as in our second example, i.e., formulating each question on the basis of all previous answers. This is impractical when, say, the comparisons are being performed by correspondence. Such a situation arises, for example, in testing consumer preferences [7]. Our game (with a fairly small k) is a better model of the situation. Several questions are formulated simultaneously and the resulting questionnaire is evaluated by the subject; on the basis of the answers a new set of questions is formed.

When a set is being sorted by several parallel processors, the comparisons are also arranged in rounds, consisting of all comparisons performed by the processors during one time interval. Consider the following model of a multiprocessor computer. Within

* Received by the editors March 21, 1980, and in revised form July 28, 1980.

† University College, Department of Mathematics, Rutgers University, New Brunswick, NJ 08903.

‡ The work of this author was supported in part by the National Science Foundation under grant MCS 79-04949.

one time interval, each processor is capable of completing one binary comparison. One item may take part in several comparisons during the same time interval. At the beginning of the interval, comparisons are assigned to processors, and at the end of the interval, the results of all these comparisons are known. (There are no restrictions on communication among the processors.)

Clearly, with $m = \binom{n}{2}$ processors, the sorting can be arranged so that all comparisons are completed in one time interval—simply assign each of the $\binom{n}{2}$ comparisons to a different processor. Moreover, it is easy to see that this cannot be accomplished with $m < \binom{n}{2}$ processors; cf. § 1. Our results may be interpreted in this spirit—we estimate the number of processors necessary to sort n items so that all comparisons are completed in k time intervals. We shall do this in § 6. It should be emphasized that we are only concerned with the cost of making comparisons, and our model ignores many important costs connected with storage, movement of data, deciding on the next comparisons, etc. Nevertheless, the negative results are valid in any model of parallel sorting by binary comparisons, and the positive results, in addition to their theoretical interest, may have practical implications in situations where the cost of performing a comparison dominates all other costs.

We shall call a strategy for player 2 in our game, a k -round sorting algorithm. We shall show that with the best k -round sorting algorithm it may be necessary to ask as many as $C_1 \cdot n^{1+1/k}$ questions. On the other hand, we shall prove that there exist k -round sorting algorithms guaranteed to complete the sorting with $C_2 \cdot n^{\alpha_k} \log n$ questions, where $\alpha_2 = \frac{5}{3}$, $\alpha_3 = \frac{11}{7}$, $\alpha_4 = \frac{23}{15}$, and $\lim \alpha_k = \frac{3}{2}$. In particular, there exists a 2-round sorting algorithm which never makes more than $c(n)$ comparisons, where

$$C_1 \cdot n^{3/2} < c(n) < C_2 \cdot n^{5/3} \log n.$$

(It is worth emphasizing that we do not exhibit such an algorithm.)

2. Definitions and notation. Let D be an acyclic digraph and G its underlying graph; we shall refer to D as an *acyclic orientation* of G . The s, t -closure of D is the symmetric closure (underlying graph) of the transitive closure of D . We denote the s, t -closure of D by D^* , and the complement of D^* by \check{D} . Note that both \check{D} and D^* are (undirected) graphs.

Let V be a set with n elements. Let G be a graph on V and $<$ a linear order on V . We denote by $G^<$ the digraph on V with the arcs $\{(i, j) : [i, j] \text{ is an edge of } G \text{ and } i < j\}$. (If $<$ is the order chosen by player 1, and the edges of G represent the questions asked by player 2, then $G^<$ represents the answers given by player 1.) Note that each $G^<$ is an acyclic digraph.

A *tower* on V is a sequence of acyclic digraphs D_0, D_1, \dots, D_t on V , such that each D_{i-1} is a subgraph of D_i , $i = 1, \dots, t$. (We intend D_i to represent all the answers given by player 1 by the end of the i th round.) A k -round algorithm for sorting V (k -round sorting algorithm) is a mapping ϕ which assigns to each tower (on V) D_0, D_1, \dots, D_n , with $t < k$, a subgraph $G = \phi(D_0, D_1, \dots, D_t)$ of \check{D}_t . (Here G represents the set of questions to be asked in the $(t + 1)$ st round; since G is a subgraph of \check{D}_n , questions whose answers can be deduced by transitivity are not asked.) A k -round algorithm ϕ for sorting V and a linear order $<$ on V together determine a unique tower D_0, D_1, \dots, D_k on V as follows:

$$D_0 = (V, \phi),$$

$$D_i = D_{i-1} \cup (\phi(D_0, \dots, D_{i-1}))^<, \quad i = 1, 2, \dots, k.$$

The k -round sorting algorithm ϕ is legitimate if, for any linear order $<$, the tower

determined by ϕ and $<$ terminates with D_k for which $D_k^* \cong K_n$. (Thus a legitimate k -round sorting algorithm completes the sorting in k rounds; all comparisons have either been made or deduced by transitivity.) It is easy to see that a k -round sorting algorithm ϕ is legitimate if and only if $\phi(D_0, D_1, \dots, D_{k-1}) = \check{D}_{k-1}$ (cf. the argument for $k = 1$ described informally in § 1). We shall assume that every k -round sorting algorithm has this property, and will only define ϕ for towers $D_0 \dots, D_t$ with $t \leq k - 2$. Thus every algorithm is legitimate and we shall not state so explicitly.

The (worst case) complexity $c(\phi)$ of a k -round sorting algorithm ϕ is the maximum, over all linear orders $<$, of the sum $\sum_{i=0}^{k-1} e(\phi(D_0, \dots, D_i))$, where D_0, \dots, D_k is the tower determined by ϕ and $<$, and $e(\phi(D_0, \dots, D_i))$ is the number of edges of the graph $\phi(D_0, \dots, D_i)$. Let $c(k, n)$ be the minimum $c(\phi)$ among all k -round algorithms ϕ for sorting a set of n elements. We have already observed that $c(1, n) = \binom{n}{2}$.

3. Auxiliary results. Throughout this paper $\log n = \log_2 n$. Let α be a real number, $\frac{3}{2} < \alpha < 2$. Let $N = \binom{n}{2}$, $p = \lfloor n^{2-\alpha} \rfloor$, $q = \lfloor n^{4-2\alpha} \rfloor$, and $r = \lfloor 2n^{4\alpha-6} \log n \rfloor$. Note that $p^2 \leq q$, and $pqr \leq 2n^\alpha \log n$.

LEMMA. For each sufficiently large n , there exists a graph G with n vertices and pqr edges, whose complement \bar{G} does not contain $K_{p,q}$.

Proof. We shall show that the number A of (labeled) graphs with n vertices and pqr edges, is greater than number B of (labeled) graphs with n vertices, pqr edges and $K_{p,q}$ in the complement (provided n is large enough).

Clearly, $A = \binom{N}{pqr}$. Moreover, $B \leq \binom{N-pq}{pqr} \cdot C$, where C is the number of different $K_{p,q}$'s in K_n . Therefore,

$$\begin{aligned} \frac{A \cdot C}{B} &\geq \frac{\binom{N}{pqr}}{\binom{N-pq}{pqr}} = \frac{N}{N-pq} \cdot \frac{N-1}{N-pq-1} \cdot \dots \cdot \frac{N-pqr+1}{N-pq-pqr+1} > \left(\frac{N}{N-pq}\right)^{pqr} \\ &= \left[\left(1 + \frac{1}{N/pq-1}\right)^{N/pq} \right]^{p^2 q^2 r / N} \end{aligned}$$

We observe the following facts:

(a) $\lim_{n \rightarrow \infty} \frac{N}{pq} = \infty.$

Indeed,

$$\frac{N}{pq} \geq \frac{n^2 - n}{2n^{6-3\alpha}} = \frac{1}{2} \cdot n^{3\alpha-5} \cdot (n-1) > \frac{1}{2} \cdot \frac{n-1}{\sqrt{n}}.$$

(b) $\lim_{m \rightarrow \infty} \left(1 + \frac{1}{m-1}\right)^m = e > 2.$

(c) $\frac{p^2 \cdot q^2 \cdot r}{N} > (p+q) \log n,$

for all sufficiently large n . We have

$$\begin{aligned} p^2 \cdot q^2 \cdot r &\geq (n^{2-\alpha} - 1)^2 (n^{4-2\alpha} - 1)^2 (2n^{4\alpha-6} \log n - 1) \\ &= 2n^{6-2\alpha} \log n - o(n^{6-2\alpha} \log n) \\ &> n^{6-2\alpha} \log n \end{aligned}$$

for all large n . Hence

$$p^2 \cdot q^2 \cdot r > n^2 q \log n > \frac{n^2 - n}{2} \cdot 2q \log n \cong N \cdot (p + q) \log n .$$

By (a), (b), (c)

$$\frac{A \cdot C}{B} > 2^{p^2 q^2 r / N} > 2^{(p+q) \log n} = n^{p+q} .$$

On the other hand,

$$C = \binom{n}{p+q} \cdot \binom{p+q}{p} = \frac{n \cdot (n-1) \cdot \dots \cdot (n-p-q+1)}{p! q!} < n^{p+q} ,$$

so that $A > B$.

Let P'_n denote the graph on $\{1, \dots, n\}$ in which distinct vertices i, j are adjacent if and only if $|i - j| \leq t$. We note that P'_n has at most tn edges, and that each edge is entirely contained within one of the $\lfloor n/t \rfloor$ sets

$$\{jt + 1, jt + 2, \dots, (j + 2)t\},$$

$$j = 0, 1, \dots, \left\lfloor \frac{n}{t} \right\rfloor - 2 \quad \text{and} \quad \{n - 2t + 1, n - 2t + 2, \dots, n\} .$$

THEOREM 1. *Let $p^2 \leq q$, and let G be a graph with n vertices whose complement \bar{G} does not contain $K_{p,q}$. If D is any acyclic orientation of G , then $\tilde{D} - W$ is a subgraph of P_n^{7q} for some set W of fewer than $2n \cdot p/q$ vertices.*

Proof. Since D is acyclic, we may assume that the vertex-set is $V = \{1, \dots, n\}$ and $i < j$ for any arc (i, j) of D . We shall identify a set W of vertices such that $\tilde{D} - W$ is a subgraph of P_n^{7q} . Let $M_{i+1} = \{iq + 1, iq + 2, \dots, (i + 1)q\}$, $i \geq 0$, and let $z = \lfloor n/q \rfloor$. The sets $M_1, M_2, \dots, M_z, M'_{z+1} = M_{z+1} \cap V$ form a partition of V .

First we shall study the structure of G . Any set of p vertices has (in G) more than $n - q$ neighbors. A vertex in M_i is *good* upwards (downwards), if it has, in G , at least p neighbors in $M_{i+1} \cup M_{i+2}$ (in $M_{i-1} \cup M_{i-2}$). A vertex which is not good upwards (downwards) is *bad* upwards (downwards). For $i \leq z - 2$, the set M_i contains fewer than p vertices which are bad upwards. Otherwise, some set $S \subseteq M_i$ of p vertices bad upwards would have fewer than $p^2 \leq q$ neighbors in $M_{i+1} \cup M_{i+2}$ and thus fewer than $n - q$ neighbors in G . A similar contradiction establishes that, for $j \geq 3$, the set M_j contains fewer than p vertices bad downwards. Let W consist of all vertices bad upwards in $\cup_{i=1}^{z-6} M_i$ or bad downwards in $\cup_{j=8}^{z+1} M_j$. Note that $|W| < (z - 6) \cdot p + (z - 6) \cdot p < 2n \cdot p/q$.

Next we consider the s, t -closure D^* of D . We claim that any two vertices $v, v' \in V - W$ with $v - v' > 7q$ are adjacent in D^* . Clearly, v is a vertex of M_j with $j \geq 8$ and hence is good downwards; similarly, v' is a vertex of M_i with $i \leq z - 6$ and hence is good upwards; furthermore, $j - i \geq 7$. Vertex v has (in G) at least p neighbors in $M_{j-1} \cup M_{j-2}$, which have (in G) more than $(j - i - 6)q$ neighbors in the set $M_{i-3} \cup M_{i+4} \cup \dots \cup M_{j-4} \cup M_{j-3}$ (of $(j - i - 5)q \geq 2q$ elements). Vertex v' has (in G) at least p neighbors in $M_{i+1} \cup M_{i+2}$, which have more than $(j - i - 6)q$ neighbors in the same set $M_{i+3} \cup \dots \cup M_{j-3}$. Consequently, there is a vertex $u \in M_{i+3} \cup \dots \cup M_{j-3}$ adjacent in G to a neighbor $x \in M_{j-1} \cup M_{j-2}$ of $v \in M_j$, and to a neighbor $y \in M_{i+1} \cup M_{i+2}$ of $v' \in M_i$. Since $v' < y < u < x < v$, v and v' are adjacent in D^* .

By taking complements we conclude that if $v, v' \in V - W$ satisfy $v - v' > 7q$, then v, v' are not adjacent in \tilde{D} . Hence $\tilde{D} - W$ is a subgraph of P_n^{7q} .

COROLLARY. For each sufficiently large n , there exists a graph G with n vertices and at most $2n^{5/3} \log n$ edges, such that for any acyclic orientation D of G , the transitive closure of D has at least $\binom{n}{2} - 10n^{5/3}$ arcs.

Proof. Let $\alpha = \frac{5}{3}$ and let G be a graph whose existence is assured by the lemma. Recall that $pqr \leq 2n^\alpha \cdot \log n$. According to the theorem, $\tilde{D} - W$ is a subgraph of P_n^{7q} for some W , $|W| < 2n \cdot p/q$. Hence \tilde{D} has fewer than

$$7qn + 2n^2 \cdot \frac{p}{q} < 10n^{5/3}$$

edges (for all large n), and the conclusion follows.

It should be clear from the above corollary that $c(2, n) = O(n^{5/3} \log n)$. That fact also follows from Theorem 2 in the next section. In fact, the corollary implies that, for large n , $c(2, n) \leq 3n^{5/3} \log n$, and it is easy to see that the constant 3 may be replaced by any $C > 1$. (One can take $r = \lfloor (1 + \varepsilon/2)n^{4\alpha-6} \log n \rfloor$ in the statement of the lemma, thus obtaining $pqr \leq (1 + \varepsilon/2) \cdot n^\alpha \cdot \log n$; moreover, $10n^{5/3} < (\varepsilon/2)n^{5/3} \log n$, for large n .)

4. An upper bound on the complexity of k -round sorting. We define $\alpha_k = (3 \cdot 2^{k-1} - 1)/(2^k - 1)$, $k \geq 1$. Note that $\alpha_1 = 2$, $\alpha_2 = \frac{5}{3}$, $\alpha_3 = \frac{11}{7}$, $\alpha_4 = \frac{23}{15}$, and $\lim \alpha_k = \frac{3}{2}$.

THEOREM 2. For any $k \geq 1$, $c(k, n) = O(n^{\alpha_k} \log n)$.

Proof. When $k = 1$, $c(1, n) = \binom{n}{2} = O(n^{\alpha_1} \log n)$ and we proceed by induction on k . Let $\alpha = \alpha_k$ and let G be a graph whose existence is implied by the lemma. We shall define a k -round algorithm for sorting the vertex-set V of G . First, define $\phi(D_0) = G$ for any D_0 . Second, to define $\phi(D_0, D_1, \dots, D_t)$ for $1 \leq t \leq k - 2$, we let m be the smallest integer such that V admits subsets W, V_1, V_2, \dots, V_m satisfying

- (i) W is disjoint from V_1, V_2, \dots, V_m ;
- (ii) $|W| \leq 4n^{\alpha-1}$;
- (iii) $|V_1| = |V_2| = \dots = |V_m| = 14q$;
- (iv) each edge of \tilde{D}_1 lies within some V_j or is incident to a vertex of W .

By the induction hypothesis, there exists a $(k - 1)$ -round algorithm ψ for sorting a set of $14q$ elements, with complexity $c(\psi) = O((14q)^{\alpha_{k-1}} \log 14q) = O(q^{\alpha_{k-1}} \log q)$. We define $\phi(D_0, D_1, \dots, D_t)$, $1 \leq t \leq k - 2$, to consist of all edges of the graphs $\psi(D_1[V_j], \dots, D_t[V_j])$, $j = 1 \dots, m$. (Here $D_i[V_j]$ denotes the subdigraph of D_i induced on V_j .) As observed earlier, $\phi(D_0, D_1, \dots, D_{k-1}) = \tilde{D}_{k-1}$; for the purposes of counting we note that the edge-set of $\phi(D_0, D_1, \dots, D_{k-1})$ is a subset of the edge-set of $\cup_{j=1}^m (\tilde{D}_{k-1}[V_j])$ together with the set $\{[w, v] : w \in W, v \in V\}$.

Consider the tower D_0, D_1, \dots, D_k determined by ϕ and some linear order $<$ on V . Since $D_0 = (V, \emptyset)$, $D_1 = G^<$ is an acyclic orientation of G , and by Theorem 1, $\tilde{D}_1 - W$ is a subgraph of P_n^{7q} for some W , $|W| < 2n(p/q) \leq 4n^{\alpha-1}$. According to the remark immediately preceding the statement of Theorem 1, $V - W$ admits $m = \lfloor n/7q \rfloor$ sets V_1, V_2, \dots, V_m satisfying (iii), (iv); furthermore, (i), (ii) are also satisfied. Hence, the complexity of ϕ is at most the sum of the number of edges of G , $n/7q$ times the complexity of ψ , and the number of edges $[w, v]$, $w \in W$, $v \in V$. Thus

$$c(\phi) \leq pqr + \frac{n}{7q} \cdot O(q^{\alpha_{k-1}} \log q) + 4n^{\alpha-1} \cdot n = O(n^\alpha \log n)$$

because $pqr = O(n^\alpha \log n)$ and

$$\frac{n}{q} \cdot q^{\alpha_{k-1}} \cdot \log q \leq n^{1+(4-2\alpha_k)(\alpha_{k-1}-1)} \cdot (4-2\alpha_k) \log n = O(n^\alpha \log n).$$

(Observe that $1 + (4 - 2\alpha_k)(\alpha_{k-1} - 1) = \alpha_k$.)

5. A lower bound on the complexity of k-round sorting.

THEOREM 3. For any $k \geq 1$, $c(k, n) = \Omega(n^{1+1/k})$.

Proof. We shall show that, for any $k \geq 1$ and all n ,

$$c(k, n) > \frac{1}{2^{k+1}} \cdot n^{1+1/k} - \frac{1}{2} \cdot n.$$

Clearly, $c(1, n) = \frac{1}{2}n^2 - \frac{1}{2}n$, and we proceed by induction on k . Assume that some k -round algorithm ϕ for sorting an n -element set V satisfies

$$c(\phi) \leq \frac{1}{2^{k+1}} \cdot n^{1+1/k} - \frac{1}{2} \cdot n, \quad \text{while } c(k-1, s) > \frac{1}{2^k} \cdot s^{1+1/k-1} - \frac{1}{2} \cdot s$$

for all s . Let $D_0 = (V, \emptyset)$ and $G = \phi(D_0)$. Since G has fewer than $(1/2^{k+1}) \cdot n^{1+1/k}$ edges, its average degree is $d \leq \lfloor (1/2^k) \cdot n^{1/k} \rfloor$. Some induced subgraph G' of G with at least $\frac{1}{2}n$ vertices has maximum degree $\Delta < 2d$. (If every such G' had $\Delta \geq 2d$, G would contain more than $\frac{1}{2}n$ vertices of degree at least $2d$, and the average degree would be greater than $\frac{1}{2}n \cdot 2d \cdot (1/n) = d$.) Thus $\Delta < \lfloor (1/2^{k-1}) \cdot n^{1/k} \rfloor = m$, and G' can be m -colored. The color-classes S_1, S_2, \dots, S_m are disjoint independent subsets of G , with $\frac{1}{2}n \leq \sum_{i=1}^m |S_i| \leq n$. There exist linear orders on V in which, for all $i < j$, any vertex of S_i precedes any vertex of S_j . Let $<$ be such an order, and let $D_1 = G^{<}$. Then each S_i is an independent set in D_1^* . Therefore, by the induction assumption, the complexity of ϕ exceeds

$$\begin{aligned} \sum_{i=1}^m \left(\frac{1}{2^k} \cdot |S_i|^{1+1/k-1} - \frac{1}{2} \cdot |S_i| \right) &= \frac{1}{2^k} \cdot \sum_{i=1}^m |S_i|^{1+1/k-1} - \frac{1}{2} \cdot \sum_{i=1}^m |S_i| \\ &\geq \frac{1}{2^k} \cdot m \cdot \left(\frac{n}{2m} \right)^{1+1/k-1} - \frac{1}{2} \cdot n \\ &> \frac{1}{2^{k+1}} \cdot n^{1+1/k-1} \cdot (2m)^{-1/k-1} - \frac{1}{2} \cdot n \\ &\geq \frac{1}{2^{k+1}} \cdot n^{1+1/k} - \frac{1}{2} \cdot n, \end{aligned}$$

contrary to our assumption on $c(\phi)$.

COROLLARY. If G is a graph with n vertices and no more than $\frac{1}{8} \cdot n^{3/2} - \frac{1}{2} \cdot n$ edges, then there is an acyclic orientation D of G whose transitive closure has fewer than $\binom{n}{2} - (\frac{1}{8} \cdot n^{3/2} - \frac{1}{2} \cdot n)$ arcs.

6. Comments. To interpret our results for parallel sorting algorithms, we consider, for example, the corollary of Theorem 3. Suppose that there is an algorithm for multiprocessor sorting of an n -element set by binary comparisons which uses $\frac{1}{8}n^{3/2} - \frac{1}{2}n$ processors and always terminates in two time intervals. The comparisons performed during the first time interval define a graph G with n vertices and at most $\frac{1}{8}n^{3/2} - \frac{1}{2}n$ edges; by the corollary, in the worst case, more than $\frac{1}{8}n^{3/2} - \frac{1}{2}n$ comparisons will remain to be made during the second time interval. We conclude that there is no algorithm to sort n items by $\frac{1}{8}n^{3/2} - \frac{1}{2}n$ parallel processors in two time intervals.

Valiant [8] investigated parallelism in comparison problems, and our model is largely drawn from [8]. Valient uses $\text{Sort}_m(n)$ to denote the maximum number of time intervals needed by the best algorithm to sort n items with m parallel processors. In the spirit of the above example of two time intervals, we may interpret our results to imply

that for

$$m \leq \frac{n^{1+1/k}}{2^{k+1} \cdot k} - \frac{n}{2k},$$

$\text{Sort}_m(n) > k$; otherwise a parallel algorithm for sorting an n -element set with m processors in k time intervals would yield a k -round sorting algorithm of complexity at most mk , contrary to Theorem 3.

On the other hand, an algorithm for sorting an n -element set with m parallel processors in two time intervals (by binary comparisons), is essentially a graph G (the comparisons made during the first interval) with n vertices and at most m edges, such that for any acyclic orientation D of G , the number of arcs in the transitive closure of D is at least $\binom{n}{2} - m$. Therefore, such an algorithm exists provided n is large enough and $m \geq 2n^{5/3} \log n$; i.e., if $m \geq 2n^{5/3} \log n$ and n is sufficiently large, $\text{Sort}_m(n) \leq 2$. The corollary to Theorem 1, which justifies the statement, also implies that the algorithm results in approximately equal order of magnitude (except for the log term) of the numbers of comparisons made in the first and second time intervals; thus we would be making a relatively efficient use of the m processors. (Similar comments apply to sorting in k time intervals; with a slight abuse of notation, $\text{Sort}_m(n) \leq k$ provided $m \geq O(n^{\alpha_k} \log n)$, and the corresponding algorithms perform about the same number of comparisons during each time interval; cf. Theorem 2 and its proof.)

Conversely, the results of [1], [6], [8] have an interpretation for $c(k, n)$ with a variable k . We state these without further discussion:

$$\begin{aligned} c(C \log n, n) &\leq n \log^2 n, \\ c(2 \log n \cdot \log \log n + O(\log n), n) &\leq n \cdot \log n \cdot \log \log n, \\ c(n, n) &\leq \log n. \\ c(k, n) &\geq n \log n. \end{aligned}$$

We have stressed earlier that we do not explicitly construct the multiprocessor algorithms guaranteed to complete the sorting in k time intervals with the stated number of processors (or the k -round sorting algorithms of the stated complexity). In all such algorithms we use directly, or recursively, the graphs described in the lemma of § 3. A constructive proof of our bounds would require us to find a graph G with n vertices and $2n^{5/3} \log n$ edges for which the transitive closure of each acyclic orientation has at least $\binom{n}{2} - 2n^{5/3} \log n$ arcs. We note that the proofs in § 3 imply that a random graph G with n vertices and $2n^{5/3} \log n$ edges (each such graph being chosen equally likely) has the property that, with probability tending to 1, the transitive closure of each acyclic orientation has at least $\binom{n}{2} - 10n^{5/3}$ arcs. We paraphrase this by saying (somewhat imprecisely) that the Random graph G will work. (In other words, assigning random $2n^{5/3} \log n$ comparisons to the processors will almost surely result in fewer than $10n^{5/3}$ comparisons left for the second time interval.)

It is instructive to try to construct G when n is small. Every graph G with fewer than 5 vertices admits a transitive orientation. The pentagon C_5 admits an orientation in which only one arc is implied by transitivity, i.e., it defines a 2-round algorithm for sorting a 5-element set of complexity 9. The Petersen graph defines one of complexity 39. Taking balanced incomplete block designs and imposing the structure of G on each block, these observations can be extended to higher n . For instance, for all sufficiently large $n \equiv 1$ or $10 \pmod{90}$, there is [9] a BIBD with $k = 10$, $\lambda = 1$, $v = n$, and making

each block a copy of the Peterson graph, we obtain a 2-round sorting algorithm of complexity $\frac{39}{45}\binom{n}{2}$. The contrast with Theorem 2 is obvious.

Addendum. Lately, we have considered other k -round problems for linearly ordered sets [3]. In two rounds, both merging and selecting the i th smallest element (i fixed) takes between $Cn^{4/3}$ and $C'n^{4/3}$ comparisons; finding the median takes between $C \cdot n^{4/3}$ and $C' \cdot n^{8/5} \log n$ comparisons. We have extended these results to k rounds, [3]. The merging and selection algorithms do not depend on the non-constructive method, and we were able to use repeated merging to *construct* a sorting algorithm which will sort a set of n items in $k \geq 3$ rounds with $O(n^{\alpha_k})$ comparisons, where $\lim \alpha'_k = 1$. (Recall that $\lim \alpha_k = \frac{3}{2}$.) As a very rough illustration, in 50 rounds, n items may be sorted with about $O(n^{1.1})$ comparisons. These results will be published at a later date.

Bollobás and Rosenfeld [2], studied the problem of almost-sorting a set in one round. That is, after the first round, all but $o(n^2)$ pairs have had their order discovered. Their results are independent of ours, and provide an interesting comparison with the two-round sorting problem considered here.

Acknowledgments. We wish to acknowledge inspiring discussions with B. Alspach, D. Kirkpatrick, I. Rabinovitch, F. Roberts and M. Rosenfeld.

REFERENCES

- [1] K. E. BATCHER, *Sorting networks and their applications*, Proc. AFIPS Spring Joint Computer Conference, 32 (1968), pp. 307–314.
- [2] B. BOLLOBÁS AND M. ROSENFELD, *Sorting in one round*, Israel J. Math., to appear.
- [3] R. HÄGGKVIST AND P. HELL, *Graphs and parallel comparison algorithms*, Proceedings of the XI Southeastern Conference on Combinatorics and Computing, Boca Raton, FL., 1980.
- [4] E. HOROWITZ AND S. SAHNI, *Fundamentals of Computer Algorithms*, Computer Science Press, 1978.
- [5] D. KNUTH, *The art of computer programming, vol. 3, Sorting and searching*, Addison-Wesley, Reading, MA, 1973.
- [6] F. P. PREPARATA, *New parallel sorting schemes*, IEEE Trans. Comput., C-27 (1978), pp. 669–673.
- [7] S. SCHEELE, *Final report to Office of Environmental Education, Department of Health, Education, and Welfare*, Social Engineering Technology, Los Angeles, California, 1977.
- [8] L. G. VALIANT, *Parallelism in comparison problems*, this Journal, 4 (1975), pp. 348–355.
- [9] R. M. WILSON, *Decompositions of complete graphs into subgraphs isomorphic to a given graph*, Proc. 5th British Combinatorial Conference, 1975, pp. 647–659.

A NEW APPROACH TO PLANAR POINT LOCATION*

FRANCO P. PREPARATA †

Abstract. Given a planar straight line graph G with n vertices and a point P_0 , locating P_0 means to find the region of the planar subdivision induced by G which contains P_0 . Recently, Lipton and Tarjan presented a brilliant but extremely complex point location algorithm which runs in time $O(\log n)$ on a data structure using $O(n)$ storage. This paper presents a *practical* algorithm which runs in less than $6\lceil\log_2 n\rceil$ comparisons on a data structure which uses $O(n \log n)$ storage, in the worst case. The method rests crucially on a simple partition of each edge of G into $O(\log n)$ segments.

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

1. Introduction. The problem of locating a point in a planar subdivision—briefly called “point location”—is quite important in computational geometry and has received considerable attention in the recent past. It is stated as follows: Given a connected planar straight-line graph G on n vertices and a point P_0 , find which region of the planar subdivision induced by G contains P_0 .

An early solution to this problem was proposed by Dobkin and Lipton [1], whose location algorithm runs in time $O(\log n)^1$ on a data structure which uses $O(n^2)$ space and can be built in $O(n^2)$ time. More recently Lee and Preparata [2] [3] developed an $O(\log^2 n)$ time location algorithm on a data structure constructed in $O(n \log n)$ time and using $O(n)$ space. Observing the trade-off between space/preprocessing on one side and search time on the other, Shamos [4] raised the question of whether $O(\log n)$ search time was achievable with less than quadratic storage. This issue was definitively settled by Lipton and Tarjan [5] who showed that the point location problem—called by them “triangle problem”—could be solved in $O(\log n)$ time on a data structure which uses $O(n)$ space and can be constructed in time $O(n \log n)$. Their brilliant method, which is based on a theoretically far-reaching planar separator theorem [6], is, however, algorithmically extremely complicated; to quote Lipton and Tarjan themselves, “. . . this algorithm [is not advocated] as a practical one, but its existence suggests that there may be a practical algorithm with $O(\log n)$ time bound and $O(n)$ space bound”.

The result presented in this paper comes very close to providing a complete substantiation of the above conjecture; specifically, we shall exhibit a *practical* point location algorithm which runs in $O(\log n)$ time on a data structure, which can be constructed in $O(n \log n)$ time, but which uses $O(n \log n)$ space rather than just $O(n)$.²

Our method could be viewed as an evolution of the original technique of Dobkin and Lipton [1], which we now briefly review. A horizontal line is drawn through each vertex of G , thereby slicing the plane into horizontal strips called “slabs”; each slab contains no vertex of G and is subdivided by the transversal edges into an ordered set of $O(n)$ regions. Point location is accomplished by *first* searching the horizontal lines to

* Received by the editors November 21, 1978, and in final form December, 1980. This work was supported in part by the National Science Foundation under grants MCS76-17321 and MCS78-13642 and the Joint Services Electronics Program under contract DAAB-07-72-C-0259.

† Coordinated Science Laboratory and Departments of Electrical Engineering and of Computer Science, University of Illinois at Urbana, Urbana, Illinois 61801.

¹ All logarithms in this paper are to the base 2.

² Subsequent to the original writing of this paper, D. Kirkpatrick [17] found an entirely new point location algorithm, with the same orders of complexity as Lipton-Tarjan's. A careful analysis as to the practicality of Kirkpatrick's method, however, has to my knowledge not yet been done.

locate a slab and *next* searching the segments crossing the slab to locate a region. Clearly this search is carried out in $O(\log n)$ comparisons, but since an edge is partitioned by $O(n)$ horizontal lines, $O(n^2)$ storage is used. In contrast, our method interleaves tests against horizontal lines and tests against edges; thus it will not be necessary to decompose the edges into $O(n)$ portions. In particular, the method rests crucially on the observation that each edge of G can be decomposed uniquely into $O(\log n)$ fragments.

2. Preliminaries. Let $G = (V, E)$ be a planar graph embedded in the plane. A vertex v of G is a point of the plane given as a pair of coordinates $x(v)$ and $y(v)$, and an edge of G is a straight line segment. Letting $V = \{v_1, \dots, v_n\}$, we assume that $y(v_1) \leq y(v_2) \leq \dots \leq y(v_n)$. (In the sequel we shall assume for simplicity that these ordinates are distinct; the details of the general case are straightforward.) For additional simplification, and without loss of generality, we may assume that $y(v_i) = i$; so, when we say that the ordinate of a point u in the (x, y) -plane is i we mean $y(u) = y(v_i)$.

The graph G is represented as a *doubly-connected edge list* (DCEL) [8], where each edge e in E is described by a 6-field node (V1, V2, F1, F2, P1, P2), and: (i) the edge e is directed from vertex V1 to vertex V2; (ii) F1 and F2 are the names of plane regions (faces) lying to the right and to the left of the directed edge e , respectively; (iii) P1 and P2 point, respectively, to the edges which follow $\overline{V1V2}$ counterclockwise around V1 and V2 (Fig. 1). In this representation, clearly the counterclockwise sequence of edges

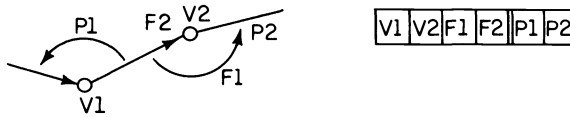


FIG. 1. Illustration of node of DCEL.

incident to a vertex and the clockwise sequence of edges bordering a face can each be obtained in time proportional to their cardinality.

Our preliminary objective is to obtain from G a partial ordering relation $<$ on E defined as follows: for $e_1, e_2 \in E$, $e_1 < e_2$ means that there is a horizontal line l , intersecting both e_1 and e_2 such that the intersection of l with e_1 is to the left of that with e_2 . A topological sorting \mathcal{S} of $<$ will be called a *consistent ordering* of the edge of set E . (For any horizontal line l , the left-to-right sequence of the edges intersected by l is a subsequence of \mathcal{S} .)

The relation $<$ can be obtained by a procedure analogous to “regularization”, as described in [2]. This procedure maintains a representation of the intersections of a horizontal line with the planar embedding of G (a left-to-right sequence of edges of G , stored as a dictionary). The vertices of G are scanned—say, in order of decreasing ordinate—and, for each scanned vertex v , the horizontal intersection of G with the line $y = y(v)$ is obtained, by updating the previous intersection in a straightforward manner (by deleting the “upwards” edges and inserting the “downwards” edges issuing from v). In addition, for each edge e inserted at this point two relation pairs, $e' < e$ and $e < e''$, are generated, where e' and e'' are respectively the (left) predecessor and (right) successor of e in the updated intersection at $y = y(v)^-$ (Fig. 2). In this fashion, all pairs defining the

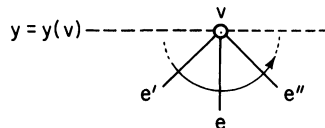


FIG. 2. Pairs $e' < e$ and $e < e''$ are contributed to the relation $<$.

relation $<$ are obtained; the running time is clearly $O(n \log n)$, since that is the time used by the initial sorting of the vertex ordinates, and for each of the $O(n)$ edges of G we have a bounded number of dictionary operations, each costing time $O(\log n)$.³ Note also that the relation $<$ can be defined (and is correctly found by the above procedure) in a more general case than that of rectilinear edge planar embeddings, that is, as long as each embedded edge is intersected at most once by an arbitrary horizontal line (for an application of this remark, see § 6).

Once $<$ is available, the consistent ordering can be obtained in time $O(n)$ by a standard topological sorting technique (see [16 p. 262]). One such consistent ordering \mathcal{F} of a graph given in Fig. 3a is shown by labeling each edge with its index in the list \mathcal{F} .

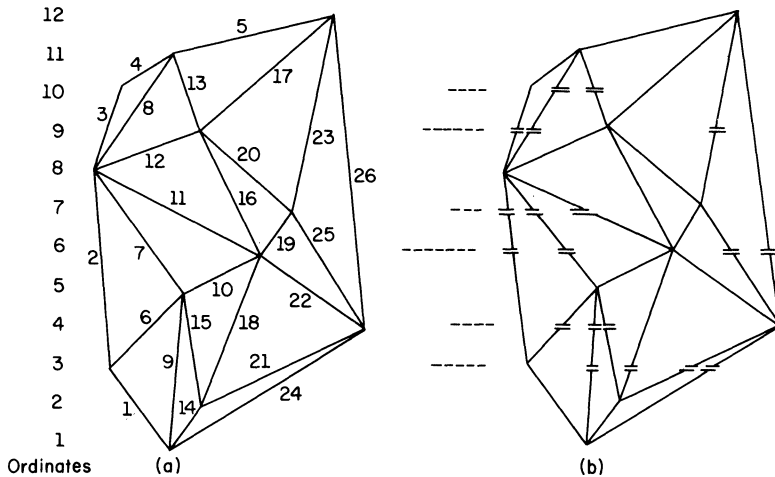


FIG. 3. (a) A graph G and a corresponding consistent ordering \mathcal{F} (indices in \mathcal{F} are shown as labels); (b) edge segmentation as induced by the segment tree $T(1, 12)$.

3. Definition and construction of the search structure. We now describe the search data structure \mathcal{H} (a tree) which can be produced for graph G . In the construction of \mathcal{H} we shall make use of the list \mathcal{F} previously obtained, and of an auxiliary structure, a “segment tree”, which we now recall. A *segment tree* $T(a, b)$ [9], [10], for an integer interval $[a, b]$ ($a < b$), consists of a root v with $B[v] = a$, $E[v] = b$, and, if $b - a > 1$, of a left subtree $T(a, \lfloor (a + b)/2 \rfloor)$ and a right subtree $T(\lfloor (a + b)/2 \rfloor, b)$, pointed to respectively by $\text{LSON}[v]$ and $\text{RSON}[v]$; if $b - a = 1$, then $\text{LSON}[v] = \text{RSON}[v] = \Lambda$. In Fig. 4 we illustrate the tree $T(1, 12)$, where each node is labeled with the pair $(B[v], E[v])$. We now explicitly recall how an interval $[i, j]$ ($i < j$) can be segmented into $O(\log n)$ fragments by means of a segment tree. Let $l(i, j)$ be the node u of $T(a, b)$ with $B[u] = i$ and such that $E[u] \cong j$ be maximum; similarly, let $r(i, j)$ be the node v of $T(a, b)$ with $E[v] = j$ and such that $B[v] \cong i$ be minimum. Let P_l and P_r be the sets of nodes on the paths from the root of T to $l(i, j)$ and $r(i, j)$, respectively (Fig. 4). It is easily seen that the segmentation of the interval $[i, j]$ is given by the intervals associated with $l(i, j)$ and $r(i, j)$, and with the right sons of $P_l - P_r$, not in P_r , and the left sons of $P_r - P_l$, not in P_l . Since $T(a, b)$ is nearly balanced and has $(n - 1)$ leaves, it follows that each edge is segmented into $O(\log n)$ pieces. If we segment each edge of G by partitioning the

³ In the special case when the graph G is a triangulation, the relation $<$ is readily obtained as follows: each triangle contributes two pairs to $<$ and $<$ is completely described by this set of pairs.

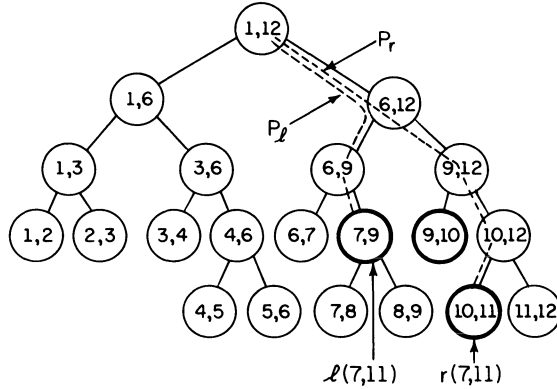


FIG 4. Illustration of the segment tree $T(1, 12)$. Shown are also $l(7, 11)$, $r(7, 11)$, P_l , P_r and the segmentation of $[7, 11]$ (boldface nodes).

y -interval defined by the ordinates of its extremes, tree $T(1, 12)$ (Fig. 4) induces on the given G the segmentation shown in the Fig. 3b.

In the sequel, we let $\mathcal{T} \triangleq T(1, n)$. For a node v of \mathcal{T} , we let $M[v] \triangleq \lfloor (B[v] + E[v])/2 \rfloor$, and call *slab* (v) the plane strip comprised between $y = B[v]$ and $y = E[v]$; a segment e , with extreme ordinate r and s ($r < s$), is said to *span* slab (v) if $r = B[v]$ and $s = E[v]$; slab (v_1) and slab (v_2) are said to be *companion* if v_1 and v_2 are siblings in \mathcal{T} . Notice that a segment spans a given slab if and only if the edge to which it belongs has one extreme in the companion slab.

In \mathcal{H} we have two types of nodes, with different graphical representations: ∇ , a ∇ -node or “horizontal node”, is associated with a horizontal line and has an ordinate $Y[\cdot]$ as discriminator; O , an O -node or “segment node”, is associated with a straight-line segment e and has as a discriminator a linear function $f[e]$ of x and y such that $f[e] = 0$ is the equation of the line containing e . A subtree of \mathcal{H} whose root is a ∇ -node is briefly called a ∇ -tree.

Each call of the algorithm which constructs \mathcal{H} processes one slab. Specifically, for some node v in \mathcal{T} , it accepts the left-to-right sequences S of the segments which either span or are contained in slab (v) and organizes them in a search tree. This is done by recursively processing all the segments in S which are inside the *trapeze* lying between two consecutive spanning segments (or the open trapezes lying respectively either to the left of the first or to the right of the last spanning segments) and proceeding until S has been completely scanned. Thus \mathcal{H} is built by $TREE(\mathcal{I}, \text{root}(\mathcal{T}))$ where \mathcal{I} is the previously defined consistent ordering of the edges of G , structured as a queue, and $TREE(S, v)$ is the following recursive procedure (where S, S_1, S_2 and U are queues):⁴

```

procedure TREE ( $S, v$ )
1  begin if ( $S = \emptyset$ ) then  $U \leftarrow \emptyset$  else  $S_1 \leftarrow S_2 \leftarrow U \leftarrow \emptyset$ 
2      while  $S \neq \emptyset$  do
3          begin  $e \in S$ 
4              if ( $(B[v] < B[e])$  or ( $E[e] < E[v]$ )) then (* $e$  does not span slab ( $v$ )*)
5                  begin if  $B[e] < M[v]$  then  $S_1 \leftarrow e$ 
6                      if  $M[v] < E[e]$  then  $S_2 \leftarrow e$ 
                      end (*queues  $S_1$  and  $S_2$  are being built*)
          end

```

⁴ In compliance with fairly standard notation, “ $S \leftarrow$ ” and “ $\leftarrow S$ ” denote respectively the operations “add to” and “remove from” a queue S [11].

```

7         if ( $(B[e] \leq B[v])$  and  $[E[v] \leq E[e])$ ) or  $(S = \emptyset)$  then
           (*e either spans slab (v) or is last term in  $S^*$ )
8         begin if  $(S_1 \cup S_2 \neq \emptyset)$  then (*the trapeze is nonempty*)
9             begin  $w \leftarrow$  new horizontal node of  $\mathcal{H}$ 
10                 $Y[w] \leftarrow M[v]$ 
11                 $LTREE[w] \leftarrow TREE(S_1, LSON[v])$ 
12                 $RTREE[w] \leftarrow TREE(S_2, RSON[v])$ 
                (*the segments in a trapeze are organized
                by joining together the structures
                corresponding to companion slabs*)
13                 $U \leftarrow w$ 
14                end
                 $U \leftarrow e$ 
           end
       end
15      $\mathcal{U} \leftarrow BALANCE(U)$ 
16     return  $\mathcal{U}$ 
end

```

The procedure BALANCE, to be discussed in the next section, takes a sequence of terms, which are either trees or segments, and arranges them in a conveniently balanced tree.

We now analyze the performance of the procedure TREE with the exception of that portion of the computational work done by BALANCE. It is convenient to charge the work to the individual edges of G .

Specifically, we interpret the previously discussed segmentation as induced by \mathcal{T} within the framework of the procedure TREE. The visit of each node of $P_l \cup P_r$ in \mathcal{T} (see discussion at the beginning of this section) corresponds to a subsequence of the sequence of steps (3, 4, 5, 6, 14) in procedure TREE, which globally use time bounded by a constant. Since there are $O(n)$ edges in G (by Euler's theorem on planar graphs) and each edge—as we have just seen—is charged $O(\log n)$ work, the generation of all “segment nodes” of \mathcal{T} uses work $O(n \log n)$, globally.

Turning now our attention to the ∇ -nodes of \mathcal{H} , each such node is produced and processed in steps 9 and 13, again using work bounded by a constant. We now exhibit a simple argument to show that the number of ∇ -nodes is $O(n \log n)$. Let $l(v) = E[v] - B[v]$ denote the *width* of slab (v), for some $v \in \mathcal{T}$; clearly $l(v) - 1$ is the number of vertices of G contained in slab (v). Notice that a ∇ -node pertaining to slab (v) (steps 9-13 of TREE) is created if and only if $S_1 \cup S_2 \neq \emptyset$, that is, there is at least one vertex in the trapeze; thus there are at most $l(v) - 1$ trapezes in slab (v); i.e., a ∇ -node with $Y = M[v]$ can occur at most $l(v) - 1$ times. It is therefore immediate that for each level of \mathcal{T} —starting from the root—we have less than n ∇ -nodes, and since \mathcal{T} has $O(\log n)$ levels, the claim follows.

Thus we conclude that the total work used by the procedure TREE to produce the tree \mathcal{H} , except for the work attributable to subroutine BALANCE, is $O(n \log n)$.

4. The BALANCE procedure: description and performance analysis. We have just seen that \mathcal{H} has $O(n \log n)$ nodes. If \mathcal{H} were balanced, it would have depth $O(\log n)$. However, there is no explicit provision in algorithm TREE to achieve such property; as a matter of fact, the depth of \mathcal{H} critically depends on the subroutine BALANCE. Indeed, suppose that in step 15 of TREE, the set U contains $O(n)$ ∇ -trees. The increase in depth produced by BALANCE (U) could be $O(\log n)$, thereby

resulting in an $O(\log^2 n)$ depth for \mathcal{H} . However, we shall now describe a procedure BALANCE which produces a global $O(\log n)$ depth \mathcal{H} . The procedure is based on the following lemmas (the first of which is a variant of another lemma presented in [12]).

LEMMA 1. Let $\mathcal{A} = a_1, a_2 \cdots a_p$ be a string with $p > 1$ and let the positive integer $|a_j|$ denote the weight of a_j ; also, let $|\mathcal{A}| = \sum_{j=1}^p |a_j|$ and $M = \max_{j=1}^p |a_j|$. Then for any number $M \leq m < |\mathcal{A}|$, the string \mathcal{A} can be algorithmically partitioned in time $O(p)$ as $\mathcal{A} = \mathcal{A}_1 \mathcal{A}_2 \mathcal{A}_3 \mathcal{A}_4$, so that $|\mathcal{A}_2| \leq m$, $|\mathcal{A}_3| \leq m$, and $|\mathcal{A}_2| + |\mathcal{A}_3| > m$.

Proof. Arrange the terms of \mathcal{A} as the leaves of a balanced binary tree $t(\mathcal{A})$, and for each node V of this tree $t(\mathcal{A})$ compute the weight $|V|$ as $|\text{LEFTSON}(V)| + |\text{RIGHTSON}(V)|$; obviously $|\text{ROOT}(t(\mathcal{A}))| = |\mathcal{A}|$. If we trace a path from the root of $t(\mathcal{A})$ following at each node the branch of larger weight, the weights of the traversed nodes form a decreasing sequence whose minimum is guaranteed to be no larger than M . Thus there is a unique node V^* on this path such that $|V^*| > m$, $|\text{LEFTSON}(V^*)| \leq m$, $|\text{RIGHTSON}(V^*)| \leq m$. We then let $\mathcal{A}_2 :=$ string of leaves of $\text{LEFTSON}(V^*)$, $\mathcal{A}_3 :=$ string of leaves of $\text{RIGHTSON}(V^*)$, while \mathcal{A}_1 and \mathcal{A}_4 are the (possibly empty) prefix and suffix of \mathcal{A} . The time bound $O(p)$ is immediate. \square

In terms of our discussion in the preceding section, in any given slab a trapeze between two consecutive spanning segments is nonempty, i.e., it gives rise to a ∇ -tree, if and only if there is at least one vertex in its interior (an appropriately modified statement holds for each of the two terminal open trapezes in the slab). Thus, for any ∇ -tree H we define its weight $|H|$ as the number of vertices of G properly contained in the plane trapeze associated with H ; obviously, $|H| \geq 1$.

We make at this point the simplifying assumption that $(n - 1)$ is a power of 2; the details for the general case can be easily supplied, while this assumption greatly simplifies the following discussion, since the width $l(v)$ of slab (v) , for any v in the segment tree \mathcal{T} , is itself a power of 2. A ∇ -tree H pertains to slab (v) if and only if $M[v]$ is the discriminator of the root of H ; we define the *level* of H , $\text{level}(H)$, as $\log(l(v))$.

As we noted earlier, the argument U of the procedure BALANCE is a string of segments and ∇ -trees; specifically, U has the general form $\tau_0 H_1 \tau_1 \cdots \tau_{r-1} H_r \tau_r$, where the H 's are all nonempty ∇ -trees of identical level, and the τ_i 's are (possibly empty) strings of segments. We define the weight $|U|$ of U as $|U| = \sum_{j=1}^r |H_j|$, and $\text{level}(U) = \text{level}(H_j)$. If $\text{level}(U) = l$, U is called an l -string. Below, the notation $H = \mathcal{U}' \nabla \mathcal{U}''$ means that the ∇ -tree H is obtained by joining via a ∇ -node a left subtree \mathcal{U}' and a right subtree \mathcal{U}'' .

LEMMA 2. Let U be an l -string and let \mathcal{U} be the tree produced by BALANCE(U). We claim

$$\text{depth}(\mathcal{U}) < \log(n - 1) + 2 \log |U| + 3 \text{level}(U) + 1.$$

Proof. For simplicity, let $\delta(\cdot) \triangleq \text{depth}(\cdot) - \log(n - 1)$. We make the following inductive hypothesis:

P. If U is a j -string with $|U| < K$ and $j \leq l$, then $\delta(\mathcal{U}) < 2 \log |U| + 3j + 1$.

The induction can be started with $j = 1$. In this case U pertains to a slab of width at most 2, whence $|U| = 1$. So U is of the form $\tau_0 H \tau_1$, where $H = \mathcal{U}_1 \nabla \mathcal{U}_2$ and \mathcal{U}_1 and \mathcal{U}_2 each consists of $O(n)$ O -nodes. It follows that $\delta(\mathcal{U}_i) \leq 0$ ($i = 1, 2$); whence $\delta(H) \leq 1$ and $\delta(\mathcal{U}) \leq 3 < 4$ (see Fig. 5a).

To prove P, let $U = \tau_0 H_1 \tau_1 \cdots \tau_{r-1} H_r \tau_r$ with $|U| = K$ and $j = l$. Notice that $\text{depth}(\tau_i) \leq \log(n - 1)$ (i.e., $\delta(\tau_i) \leq 0$, for $1 \leq i \leq r$). The proof explicitly exhibits the procedure BALANCE.

Step 1. If U consists of segments, then arrange them in a balanced tree of O -nodes, else find in U tree H_s such that $|H_s| = \max_{j=1}^r |H_j|$.

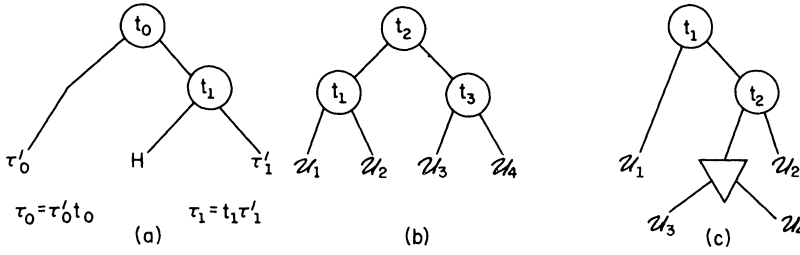


FIG. 5. Illustration for the proof of Lemma 2.

Step 2. If $|H_s| < K/2$, apply Lemma 1 to U with $m = K/2$ and obtain $U_1 t_1 U_2 t_2 U_3 t_3 U_4$, where t_1, t_2, t_3 are segments and the U_i 's are l -strings. Apply the algorithm recursively to U_i to obtain a tree $\mathcal{U}_i (i = 1, \dots, 4)$ and structure the \mathcal{U}_i 's as in Fig. 5b.

Comment. By Lemma 1, $|U_2|, |U_3| \leq K/2$ and $|U_2| + |U_3| > K/2$, which implies $|U_1| + |U_4| < K/2$, i.e., $|U_1|, |U_4| < K/2$. By the inductive hypothesis, $\delta(\mathcal{U}_1), \delta(\mathcal{U}_2), \delta(\mathcal{U}_3), \delta(\mathcal{U}_4) < 2 \log(K/2) + 3l + 1 = 2 \log K + 3l - 1$, whence clearly (Fig. 5b) $\delta(\mathcal{U}) < 2 \log K + 3l + 1$.

Step 3. ($|H_s| \geq K/2$). In this case \mathcal{U} has the form $U_1 t_1 H_s t_2 U_2$, where t_1, t_2 are segments and U_1, U_2 are l -strings. Apply the algorithm recursively to U_1 and U_2 to obtain \mathcal{U}_1 and \mathcal{U}_2 , and structure \mathcal{U}_1, H_s , and \mathcal{U}_2 as in Fig. 5c (where $H_s = \mathcal{U}_3 \nabla \mathcal{U}_4$).

Comment. We distinguish two cases.

(1) $|H_s| < K$. Since $|H_s| \geq K/2$, then $|U_1|, |U_2| \leq K/2$, whence, by the inductive hypothesis, $\delta(\mathcal{U}_i) < 2 \log |U_i| + 3l + 1 \leq 2 \log K + 3l - 1 (i = 1, 2)$. As regards H_s , we have $H_s = \mathcal{U}_3 \nabla \mathcal{U}_4$, with $|U_3|, |U_4| \leq |H_s| < K$; since \mathcal{U}_3 and \mathcal{U}_4 are $(l - 1)$ -strings, we have $\delta(\mathcal{U}_3), \delta(\mathcal{U}_4) < 2 \log K + 3(l - 1) + 1 = 2 \log K + 3l - 2$ and obviously $\delta(H_s) < 2 \log K + 3l - 1$ (Fig. 5c). \square

(2) $|H_s| = K$. Again, $H_s = \mathcal{U}_3 \nabla \mathcal{U}_4$. If $|U_3|, |U_4| < K$, then we argue as in case (2). Suppose instead that, say, $|U_3| = K (|U_4| = 0)$. We then set $H^{(l)} \triangleq H_s$ and $U_3 = U^{(l-1)}$; denoting by $H^{(j)}$ the heaviest term of $U^{(j)}$, if $|H^{(j)}| = K$ and the heavier of its subtrees has also weight K , the corresponding $(j - 1)$ -string is defined as $U^{(j-1)}$. Thus we obtain a sequence $H^{(l)}, H^{(l-1)}, \dots, H^{(p)}$ such that $|H^{(l)}| = |H^{(l-1)}| = \dots = |H^{(p+1)}| = K$ and $|H^{(p)}| < K$. It follows that $U^{(p)}$ is a p -string ($p < l$), which falls in the cases discussed either in step 2 or in step 3 case (1) above, whence $\delta(\mathcal{U}^{(p)}) < 2 \log K + 3p + 1$. Now notice that $\delta(\mathcal{U}^{(j)}) - \delta(\mathcal{U}^{(j-1)}) \leq 3$; thus, $\delta(\mathcal{U}_3) = \delta(\mathcal{U}^{(l-1)}) < 2 \log K + 3p + 1 + 3(l - 1 - p) = 2 \log K + 3l - 2$, thus extending the inductive hypothesis (Fig. 5c). \square

In conclusion we have:

THEOREM. *The depth of the tree \mathcal{H} is less than $6 \log(n - 1) - 2$.*

Proof. If the root of \mathcal{H} is a ∇ -node, then $\mathcal{H} = \mathcal{U}_1 \nabla \mathcal{U}_2$ where U_1 and U_2 are l -strings, with $l = \log(n - 1) - 1$ and $|U_1|, |U_2| \leq (n - 1)/2$. Thus by Lemma 2, $\text{depth}(\mathcal{H}) < \log(n - 1) + 2 \log(n - 1) + 3 \log(n - 1) - 4 + 1 = 6 \log(n - 1) - 3$. If the root of \mathcal{H} is an O -node, then there is an edge t in G spanning the slab $[1, n]$. In this case G appears as $G_1 t G_2$, where both G_1 and G_2 are graphs with no more than n vertices; G_1 and G_2 can be structured into ∇ -trees \mathcal{H}_1 and \mathcal{H}_2 , respectively, of depth less than $6 \log(n - 1) - 3$, as we have just seen. Thus $\text{depth}(\mathcal{H}) < 6 \log(n - 1) - 2$.

⁵ Indeed, it can be shown that $\delta(\mathcal{U}^{(j)}) - \delta(\mathcal{U}^{(j-1)}) = 1$.

When $(n - 1)$ is not a power of 2, it can be easily shown that $6 \lceil \log n \rceil$ is a (generous) bound to depth (\mathcal{H}). The search tree \mathcal{H} for the graph of Fig. 3 is shown in Fig. 6.

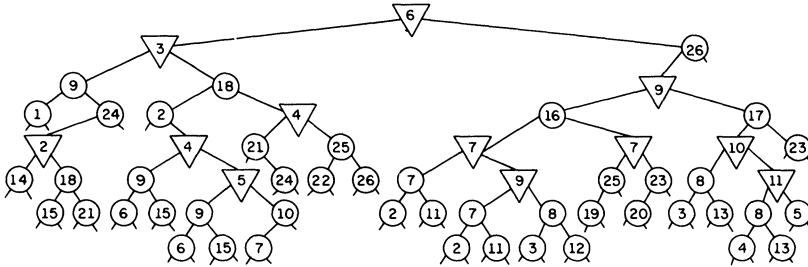


FIG. 6. The search data structure \mathcal{H} for the graph of Fig. 3. Leaves with region identifiers are not explicitly shown.

We now estimate the running time of BALANCE. If U contains r ∇ -trees, the call BALANCE (U) runs in time $O(r \log r)$, by a result of [13]. To estimate the number of ∇ -trees involved in balancing operations, we consider a tree $\mathcal{H}^{(\nabla)}$ obtained from \mathcal{H} by deleting and bypassing all O -nodes of \mathcal{H} ; i.e., (1) replace each O -node path between two ∇ -nodes by a single edge, and (2) delete each O -node path terminating in a O -node leaf. The nodes of $\mathcal{H}^{(\nabla)}$ are (∇ -nodes) of three types: *regular*, with two or more “children”; *singular*, with just one child; *leaves*, with no child. Clearly $\mathcal{H}^{(\nabla)}$ has at most $(n - 2)$ leaves, since a leaf corresponds to a vertex of G ; also, it is easy to realize that only the children of regular nodes participate in the balancing process, and their number is less than twice the number of leaves; i.e., it is $O(n)$. Thus the global running time of BALANCE is $O(n \log n)$.

5. Point location. To locate a point $P_0 = (x_0, y_0)$ in the planar subdivision induced by G , we use \mathcal{H} as a binary search tree. With each O -node of \mathcal{H} which has one or no descendant we append one or two leaves (Fig. 6), respectively, and with each such leaf we associate the identifier of a plane region (bordering with the edge associated with the parent O -node). The point location proceeds as follows: at each node V of \mathcal{H} we choose a branch: if V is a ∇ -node, by comparing y_0 with $Y[V]$; if V is an O -node, by testing the sign of $f(x_0, y_0)$, where $f(x, y)$ is the discriminant function of V . Thus we trace a unique path from the root to a leaf, at which stage the point location is completed. By the preceding discussion this process uses a number of comparisons bounded by the depth of \mathcal{H} , i.e., $6 \lceil \log n \rceil$.

6. Comments and applications. As the previous analysis indicates, planar point location is simply done in time $O(\log n)$ using a search structure which can be stored in $O(n \log n)$ space. Specifically, less than $6 \lceil \log n \rceil$ comparisons are ever needed, although the analysis which establishes the upper-bound on the depth of \mathcal{H} is overly generous and multiplicative constants for $\lceil \log n \rceil$ substantially lower than 6 can be expected.

As to the storage requirement, the analysis refers to the case in which each of $O(n)$ edges is partitioned into $O(\log n)$ segments; this intuitively corresponds to a large fraction of long edges, which presumably is not the average case; however, graphs can be constructed for which this situation occurs. It is conceivable that the simple approach presented in this paper could be further refined to achieve $O(n)$ storage while maintaining $O(\log n)$ search time.

Notice that the described point location method is not restricted to triangulations, nor to planar subdivisions induced by straight-line graphs. Indeed the straight-line segments may be replaced by other curves if the following two properties hold: (i) the curves are single-valued in one selected coordinate (say, y), and (ii) the discrimination of a point with respect to any of the curves can be done in constant time. For example, these conditions are clearly met by arcs of circles or of other conics, if they have no horizontal tangent except possibly at their extremes. We can now mention two applications of the given method. Both problems have recently received consideration in the literature [14], [15].

1. *Fixed-radius near neighbor searching.* This problem involves finding all points of a set F in the plane which are within some fixed radius r of a "query point" [14]. Bentley and Maurer have recently proposed—among other methods—a locus approach, which consists in subdividing the plane into regions each of which is the locus of the points within distance r from a given subset F' of F (this region is clearly the intersection of all the circles with radius r centered at the points in F'). Let $F = \{p_1, \dots, p_n\}$, and let C_i be the circle of radius r with center in $p_i \in F$. For each C_i , let u_i and l_i be the two points on the circle C_i with largest and smallest ordinates, respectively, and let I denote the set of intersections of pairs of circles in $\{C_i | i = 1, \dots, n\}$. If we define $V \triangleq I \cup \{u_i | i = 1, \dots, n\} \cup \{l_i | i = 1, \dots, n\}$, the circumference of each C_i is partitioned into a set of arcs which have properties (i) and (ii) given above. Therefore V is the vertex set of a planar graph G whose edges are the arcs just described. To this planar graph the method of this paper is applicable. Since $|V| = |I| + |\{u_i | i = 1, \dots, n\}| + |\{l_i | i = 1, \dots, n\}| \leq 2\binom{n}{2} + n + n = n(n + 1)$, graph G is planar with $O(n^2)$ vertices. Thus fixed-radius near-neighbor searching can be solved in $O(\log n)$ time with a data structure using $O(n^2 \log n)$ space and constructible in $O(n^2 \log n)$ time; in [14] the latter two quantities are both $O(n^3)$.

2. *Maxima testing in three dimensions.* For points u and v in three-dimensional Euclidean space \mathbb{R}^3 , u is said to *dominate* v if $x_i[u] \geq x_i[v]$ ($i = 1, 2, 3$). Given a finite set F of points \mathbb{R}^3 , $u \in F$ is a *maximum* of F if it is not dominated by any other point in F . Suppose now that F is a set of maxima of F ; testing a target point p for maximum in F means to determine if there is at least a point $u \in F$ which dominates p .

Letting $|F| = n$, Bentley [15] solves this problem in $O(\log^2 n)$ time on a search data structure that is stored in $O(n \log n)$ space and is constructed in $O(n \log n)$ time. We now show that the same storage and preprocessing time can be maintained while reducing the test time to $O(\log n)$.

Let $F = \{u_1, \dots, u_n\}$. Let v be the point such that $x_j[v] = \min_{i=1}^n x_j[u_i]$ ($j = 1, 2, 3$); for convenience we may assume that v be the origin of \mathbb{R}^3 , so that all points of F lie in the positive orthant \mathbb{R}_+^3 . Let M_i be the domain of points of \mathbb{R}_+^3 dominated by $u_i \in F$, and let $M = \bigcup_{i=1}^n M_i$. Consider now the surface of M and suppose it projected on one of the coordinate planes, say (x_1, x_2) . This projection appears as a planar straight-line graph G , each finite region of which is the projection of a portion of the surface of M_i , for some i (Fig. 7); it follows that if the (x_1, x_2) -projection of the target point p falls in the region of G associated with $u_i \in F$, then the maxima testing reduces to comparing $x_3[p]$ with $x_3[u_i]$. Thus maxima testing is done via point-location in G . Notice now that G has two edges—respectively parallel to the x_1 and x_2 axes—issuing from the (x_1, x_2) -projection of each $u_i \in F$. It is easy to realize that the point-location procedure can be applied to the graph consisting of the n edges parallel to, say, the x_2 -axis, and the positive x_2 -axis itself (see Fig. 7). Obviously the search data structure can be stored in $O(n \log n)$ space and is constructible in $O(n \log n)$ time. Referring to the arguments of Bentley [15] shows that

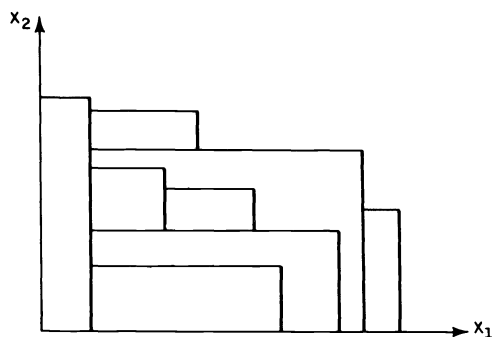


FIG. 7. Typical projection of the surface of M on the plane (x_1, x_2) . The vertical edges are shown as thick lines.

the time for worst-case maxima testing in k dimensions can be reduced from $O(\log^{k-1} n)$ to $O(\log^{k-2} n)$ for $k \geq 3$.

REFERENCES

- [1] D. P. DOBKIN AND R. J. LIPTON, *Multidimensional searching problems*, this Journal, 5 (1976), pp. 181–186.
- [2] D. T. LEE AND F. P. PREPARATA, *Location of a point in a planar subdivision and its applications*, Proc. 8th ACM Symposium on Theory of Computing, Hershey, PA, 1976, pp. 231–235.
- [3] ———, *Location of a point in a planar subdivision and its applications*, this Journal, 6 (1977), pp. 594–606.
- [4] M. I. SHAMOS, *Computational Geometry*, Dept. of Computer Science, Yale University, 1977, Springer-Verlag, to appear.
- [5] R. J. LIPTON AND R. E. TARJAN, *Applications of a planar separator theorem*, Proc. 18th IEEE Symposium on Foundation of Computer Science, Providence, RI, 1977, pp. 162–170.
- [6] ———, *A separator theorem for planar graphs*, Conference on Theoretical Computer Science, Waterloo, Ont., 1977, pp. 1–10.
- [7] M. R. GAREY, D. S. JOHNSON, F. P. PREPARATA AND R. E. TARJAN, *Triangulating a simple polygon*, Inform. Proc. Letters, 7 (1978) pp. 175–179.
- [8] D. E. MULLER AND F. P. PREPARATA, *Finding the intersection of two convex polyhedra*, Theoret. Comput. Sci., 7 (1978), pp. 217–236.
- [9] J. L. BENTLEY, *Decomposable search problems*, Inform. Proc. Letters, 8 (1977), pp. 244–251.
- [10] J. L. BENTLEY, *Solution to Klee's rectangle problems*, unpublished manuscript, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, 1977.
- [11] E. M. REINGOLD, J. NIEVERGELT AND N. DEO, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.
- [12] R. P. BRENT, D. J. KUCK, AND K. MARUYAMA, *The parallel evaluation of arithmetic expressions without division*, IEEE Trans. Comput., C-22 (1973), pp. 532–534.
- [13] D. E. MULLER AND F. P. PREPARATA, *Restructuring of arithmetic expressions for parallel evaluation*, J. Assoc. Comput. Mach., 23 (1976), pp. 534–543.
- [14] 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.
- [15] J. L. BENTLEY, *Multidimensional divide and conquer*, Research Review, Dept. Computer Science, Carnegie-Mellon University, 1977.
- [16] D. E. KNUTH, *The Art of Computer Programming, vol. 1, Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1969.
- [17] D. G. KIRKPATRICK, *Optimal search in planar subdivisions*, manuscript, Dept. Comput. Science, University of British Columbia, Vancouver, British Columbia, 1979.

UNIFORM INTERPRETATIONS OF GRAMMAR FORMS*

H. A. MAURER†, A. SALOMAA‡ AND D. WOOD§

Abstract. Encouraged by positive experiences with so-called uniform-interpretations of L-forms, we investigate in this paper a suitable analogous definition of uniform interpretations of grammar forms. Concerning CF grammar forms it is shown that a rich variety of language families can be obtained. CF grammar forms with a single variable are extensively examined both with regard to generative capacity (including a characterization of subregular, sublinear and subfinite index families) and with regard to the notion of goodness and badness. Concerning non-CF grammar forms it is shown that each of the families of EOL-, ETOL-, matrix-, scattered context-, context sensitive, type 0-languages (and many others) can be obtained by using interpretations of one form specific to this family.

Key words. grammar form, EOL form, Chomsky hierarchy, uniform interpretation, generative capacity

1. Introduction. In the (albeit brief) history of grammar forms one important problem raised in the initial paper of Cremers and Ginsburg [CG] has remained unanswered. This problem can be formulated as:

Are there grammar form characterizations of the various well-known non-context-free language families?

For example, are there grammar form characterizations of the ETOL or matrix language families?

In [CG] and in [W] this question, under the g -interpretations of [CG], can be stated in its original manner as:

Are there any g -grammatical families lying strictly between the context-free and recursively enumerable language families?

It is conjectured that there are no such families, and some preliminary evidence to support this conjecture is given in [MPSW]. Because of this conjecture many other interpretation mechanisms were studied in [MPSW], and of these the s -interpretation was singled out for particular study. It was demonstrated that this interpretation mechanism, together with a further restriction upon it, indeed enabled many of the sought-for language families to be generated.

However, these results, while pleasing, did not completely satisfy us. The major reason for this fact was that the additional restriction, while simple, destroyed the transitivity of interpretation. Hence when in [MSW3] the notion of a "constant context" interpretation was discovered, it was felt this would also serve in the general grammar form situation. While, unfortunately, this turned out not to be the case, our investigations led to the interpretation mechanism discussed in this paper.

Under uniform interpretation, given a production

$$(1.1) \quad A \rightarrow ABcB,$$

then

$$D \rightarrow DCaE$$

* Received by the editors March 13, 1979, and in revised form August 14, 1980. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada under grant A-7700, and by the Austrian Federal Ministry for Science and Research.

† Institut für Informationsverarbeitung, TU Graz, Steyrergasse 17, A-8010 Graz, Austria.

‡ Department of Mathematics, University of Turku, SF-20500 Turku 50, Finland.

§ Unit for Computer Science, McMaster University, Hamilton, Ontario, L8S 4K1, Canada.

is a uniform interpretation of (1.1), while

$$F \rightarrow GHbN$$

is not. A *uniform interpretation* is an s -interpretation with the further constraint that the substitution be uniform upon all symbols appearing on *both* sides of a production.

Up until now uniform interpretations have only been studied for EOL forms (see [MSW1] and [MSW4], where only terminals have been uniformly interpreted). We feel that uniformly interpreting grammars is a natural mechanism: for example, in van Wijngaarten two-level grammars a similar technique is used. Also in logic the notion of a uniform substitution is widely applied.

The results given in the remainder of the paper serve to establish the viability of uniform interpretations of general grammar forms. We show that there are grammar forms whose families are the EOL, ETOL, matrix, scattered context and context-sensitive language families, this is seen in § 4. However, we obtain, for example, not only the ETOL languages and the matrix languages but also many others of this ilk. Hence, in particular, many of the controlled-context-free language families are obtained.

In § 3 we exhibit many nonreduction results and also exhibit a grammatical family which is an anti-AFL. It is also shown that every s -interpretation family is a uniform interpretation family and that the converse is not true.

In § 5 forms with one variable are examined with regard to generative capacity, and a characterization is given of when forms with a single variable generate families which are subregular, sublinear and sub-finite-index. Two-variable sequential forms are also studied, in which case it is shown how to obtain the context-free languages and the recursively enumerable languages. Finally in § 6 the notions of goodness and badness for single variable forms are investigated.

We now turn in § 2 to the basic terminology and notation used throughout the paper.

2. Definitions. We first need the definition of general grammars used in this paper.

We say $G = (V, \Sigma, P, S)$ is a *general grammar* if V is a finite alphabet, $\Sigma \subseteq V$ is a finite terminal alphabet, $V - \Sigma$ is the nonterminal alphabet, $P \subseteq V^*(V - \Sigma)V^* \times V^*$ is a finite set of productions and S in $V - \Sigma$ is the sentence symbol. Productions are usually written $\alpha \rightarrow \beta$. As usual we write $\alpha_1\alpha\alpha_2 \Rightarrow \alpha_1\beta\alpha_2$ in G to mean, “ $\alpha_1\alpha\alpha_2$ is rewritten by G using the production $\alpha \rightarrow \beta$ in P ”, and this usage is extended to \Rightarrow^+ and \Rightarrow^* in the standard manner. The language generated by G , denoted $L(G)$, is defined as

$$L(G) = \{x : x \text{ in } \Sigma^* \text{ and } S \Rightarrow^+ x\}.$$

We say $G = (V, \Sigma, P, S)$ is *context-free* if $\alpha \rightarrow \beta$ in P implies α is in $V - \Sigma$.

We say $L \subseteq \Sigma^*$ is a *context-free language* if $L = L(G)$ for some context-free grammar G .

Let $G = (V, \Sigma, P, S)$ be a context-free grammar. We say a production $A \rightarrow \alpha$ is *recurrent* (*nonrecurrent*) if α contains an A (α does not contain an A). Similarly we say A in $V - \Sigma$ is *recurrent* (*nonrecurrent*) if A has a recurrent production (has no recurrent production). A derivation in G

$$A \Rightarrow^+ \alpha$$

is *nonrecurrent* if no recurrent production is used in the derivation. We write this as

$$A \underset{\text{nr}}{\Rightarrow^+} \alpha.$$

A production $A \rightarrow \alpha$ in P is *separated* if α is in $\Sigma^* \cup (V - \Sigma)^*$. A context-free grammar is separated if all its productions are separated.

A nonterminal A is said to be *finite* if the set $\{x: A \Rightarrow^+ x \text{ in } \Sigma^*\}$ is finite; otherwise A is said to be *infinite*. A context-free grammar is infinite if it is reduced and has an infinite nonterminal; otherwise it is finite.

We now turn to the definition of a grammar form and its interpretations.

Let $M, N \subseteq V^*$, where V is an alphabet; $M \rightarrow N$ denotes the set $\{\alpha \rightarrow \beta: \alpha \text{ in } M \text{ and } \beta \text{ in } N\}$.

A *disjoint finite letter substitution* (dfi-substitution) μ defined on an alphabet V satisfies the following properties:

- (i) $\mu(X)$ is finite for each X in V .
- (ii) $\mu(X)$ consists only of symbols.
- (iii) $\mu(X) \cap \mu(Y) = \emptyset$, for all X, Y in $V, X \neq Y$.

A *general grammar form* or *context-free grammar form* is a general grammar or a context-free grammar, respectively.

DEFINITION. Let $G_i = (V_i, \Sigma_i, P_i, S_i), i = 1, 2$, be two general grammar forms. We say G_1 is an interpretation of G_2 modulo μ , denoted $G_1 \triangleleft G_2(\mu)$, where μ is a dfi-substitution on V_2 , if the following conditions hold:

- (i) $\mu(A) \subseteq V_1 - \Sigma_1$, for all A in $V_2 - \Sigma_2$.
- (ii) $\mu(a) \subseteq \Sigma_1$, for all a in Σ_2 .
- (iii) $P_1 \subseteq \mu(P_2)$, where $\mu(P_2) = \bigcup_{\alpha \rightarrow \beta \text{ in } P_2} \mu(\alpha) \rightarrow \mu(\beta)$.
- (iv) S_1 is in $\mu(S_2)$.

Note that μ has a well-defined inverse μ^{-1} . If we replace (iii) by

- (iii)' $P_1 \subseteq \mu_u(P_2)$, where $\mu_u(P_2) \subseteq \mu(P_2)$ and $\alpha' \rightarrow \beta'$ is in $\mu_u(P_2)$ iff $\alpha' = X_1 \cdots X_m$, $\beta' = Y_1 \cdots Y_n$ and for all $X_i, Y_j, 1 \leq i \leq m, 1 \leq j \leq n, \mu^{-1}(X_i) = \mu^{-1}(Y_j)$ implies $X_i = Y_j$,

then we say that G_1 is a *uniform interpretation* of G_2 modulo μ , denoted $G_1 \triangleleft_u G_2(\mu)$.

We often write simply $G_1 \triangleleft G_2$ or $G_1 \triangleleft_u G_2$, when μ is understood.

In the literature, [GLMW], [MPSW] and [MSW5] for example, \triangleleft is said to be a *strict interpretation*. The notion of uniform interpretation for terminals was first introduced in [MSW1]. Our uniform interpretation could in the terminology of [MSW1] be called a weak uniform interpretation.

The family of general grammar forms obtained from a general grammar form G is defined by

$$\mathcal{G}(G) = \{G': G' \triangleleft G\},$$

and the uniform family is defined by

$$\mathcal{G}_u(G) = \{G': G' \triangleleft_u G\}.$$

Clearly $\mathcal{G}_u(G) \subseteq \mathcal{G}(G)$ for all G .

The family of languages defined by a general grammar form G is

$$\mathcal{L}(G) = \{L(G'): G' \triangleleft G\},$$

and the uniform family is

$$\mathcal{L}_u(G) = \{L(G'): G' \triangleleft_u G\}.$$

Clearly $\mathcal{L}_u(G) \subseteq \mathcal{L}(G)$.

Throughout this paper we use the following λ -convention.

Convention. Two languages L_1 and L_2 are said to be *equal modulo λ* (the empty word) if $L_1 - \{\lambda\} = L_2 - \{\lambda\}$. Two language families \mathcal{L}_1 and \mathcal{L}_2 are said to be equal if for

every nonempty language in \mathcal{L}_1 (modulo λ) there is an equal language (modulo λ) in \mathcal{L}_2 and conversely.

Two general grammar forms G_1 and G_2 are termed *form equivalent* if

$$\mathcal{L}(G_1) = \mathcal{L}(G_2),$$

and *uniform form equivalent* (u-form equivalent) if

$$\mathcal{L}_u(G_1) = \mathcal{L}_u(G_2).$$

If $G = (V, \Sigma, P, S)$ is a general grammar form, we say G is a $(V - \Sigma, \Sigma)$ -form. If either $V - \Sigma$ or Σ is a singleton set we represent it by its single element. Thus we speak of (S, a) -forms in the sequel.

For a word α and a symbol X , $|\alpha|_X$ denotes the number of occurrences of X in α . By $|\alpha|$ we mean $\sum_{X \text{ in } \alpha} |\alpha|_X$.

In the following section there are a number of examples illustrating the differences between strict and uniform interpretations; hence we conclude the present section with a single example.

Let G be defined by the productions

$$Z \rightarrow AS; \quad AS \rightarrow AB; \quad AS \rightarrow aA; \quad AB \rightarrow ASS; \quad A \rightarrow \lambda.$$

Then $\mathcal{L}_u(G) = \mathcal{L}(\text{CF})$, the family of context-free languages. Let F be an arbitrary context-free grammar in Chomsky normal form; then it has productions only of the types

$$C \rightarrow DE \quad \text{and} \quad C \rightarrow a.$$

Letting $F = (V, \Sigma, P, S)$, we construct a uniform interpretation G' of G as follows: $G' = (V', \Sigma, P', Z)$, where $V' = V \cup \{Z, A\} \cup \{\bar{B} : B \text{ in } V - \Sigma\}$ and P' contains:

- (i) $Z \rightarrow AS; A \rightarrow \lambda$,
- (ii) $AC \rightarrow A\bar{C}$ and $A\bar{C} \rightarrow ADE$ if $C \rightarrow DE$ is in P ,
- (iii) $AC \rightarrow aA$ if $C \rightarrow a$ is in P .

Then G' simulates each left derivation of F ; hence $L(G') = L(F)$.

Conversely, consider an arbitrary $G' \triangleleft_u G(\mu)$. Because of uniformity, once a particular A' in $\mu(A)$ is introduced via $Z \Rightarrow A'S'$, then A' is preserved in the continuing derivation until it is erased. Since $\mathcal{L}(\text{CF})$ is closed under union we need only show that a particular choice of A' and S' gives a context-free language. Hence, assume $Z \rightarrow AS$ is the only Z -production in G' . If $A \rightarrow \lambda$ is not in G' then $L(G') = \emptyset$. Now construct a corresponding context-free grammar F which contains a production $C \rightarrow a$ only if $AC \rightarrow aA$ is in G' and a production $C \rightarrow DE$ only if $AC \rightarrow AB'$ and $AB' \rightarrow ADE$ are in G' . It should now be clear that $L(F) = L(G')$. Thus we have demonstrated that $\mathcal{L}(\text{CF})$ is obtained as $\mathcal{L}_u(G)$ where G and its interpretations *left derivation simulate* the corresponding context-free grammar.

3. Preliminary results. We discuss some implications of the choice of the uniform interpretation mechanism as compared with the use of the usual interpretation.

First, observe that every family $\mathcal{L}(F)$ is also a uniform family; hence under uniform interpretations we can do at least as much as under the usual interpretation. Let $F = (V, \Sigma, P, S)$ be a general grammar form and construct a new grammar form $G = (V \cup \{[\alpha \rightarrow \beta] : \alpha \rightarrow \beta \text{ in } P\}, \Sigma, \{\alpha \rightarrow [\alpha \rightarrow \beta], [\alpha \rightarrow \beta] \rightarrow \beta : \alpha \rightarrow \beta \text{ in } P\}, S)$. We say G is the *stretched* version of F . It should be clear that $\mathcal{L}_u(G) = \mathcal{L}(F)$, since the stretching has prevented uniformity from having an effect. Furthermore, the stretched derivation in G corresponding to a production in F is unique.

We will see that there are many more uniform families than strict families. Before demonstrating this fact it is appropriate to compare the closure properties of uniform and strict families.

It is well known [MPSW] and [MSW5] that each strict family is closed under intersection with regular sets. Moreover if such a family is generated by a *unary* form, that is, a form with one terminal letter, it is also closed under union. However, for uniform families neither of these closure results holds true in general.

Consider

$$(3.1) \quad H_1 : S \rightarrow aS; \quad S \rightarrow \lambda;$$

then each uniform interpretation of H_1 is of the type

$$H'_1 : S \rightarrow a_i S, \quad 1 \leq i \leq m; \quad S \rightarrow \lambda$$

for $m \geq 0$. Hence $L(H'_1) = \{a_1, \dots, a_m\}^*$. Therefore $\mathcal{L}_u(H_1) = \{\Sigma^* : \Sigma \text{ is an alphabet}\} \cup \{\{\lambda\}\}$. Obviously $\mathcal{L}_u(H_1)$ is closed neither under union nor under intersection with regular sets, and is therefore *not* a strict family.

However, $\mathcal{L}_u(H_1)$ is closed under star (since $(\Sigma^*)^* = \Sigma^*$, under reversal and also under inverse homomorphism.

Secondly, consider

$$(3.2) \quad H_2 : S \rightarrow SS; \quad S \rightarrow a;$$

then each uniform interpretation of H_2 is either of the type

$$H'_2 : S \rightarrow SS; \quad S \rightarrow a_i, \quad 1 \leq i \leq m,$$

or of the type

$$H''_2 : S \rightarrow a_i, \quad 1 \leq i \leq m,$$

giving either $\{a_1, \dots, a_m\}^*$ or $\{a_1, \dots, a_m\}$. Thus $\mathcal{L}_u(H_2) = \{\Sigma, \Sigma^* : \Sigma \text{ is an alphabet}\}$. In this case $\mathcal{L}_u(H_2)$ is closed under star and reversal. It is not closed under union, product, intersection with regular sets, inverse homomorphism or homomorphism.

In both cases $\mathcal{L}(H_1)$ and $\mathcal{L}(H_2)$ are AFLs, $\mathcal{L}(H_1)$ is the family of regular languages and $\mathcal{L}(H_2)$ the context-free languages. By the stretching technique given above we can obtain

$$(3.3) \quad H_3 : S \rightarrow A; \quad S \rightarrow B; \quad A \rightarrow aS; \quad B \rightarrow \lambda$$

from H_1 and $\mathcal{L}_u(H_3) = \mathcal{L}(\text{REG})$. That we can obtain AFL's as uniform families is not surprising; however, we can also obtain anti-AFL's.

Consider

$$(3.4) \quad H_4 : S \rightarrow SSS; \quad S \rightarrow aa;$$

then $\mathcal{L}_u(H_4)$ is not closed under star, since $L(H_4)^*$ is not in $\mathcal{L}_u(H_4)$. In particular, no interpretation of H_4 can generate a word of length four. That $\mathcal{L}_u(H_4)$ is not closed under the other AFL operations is easy to see.

We now demonstrate that there are many more uniform families than strict families by way of the following family of grammar forms:

$$(3.5) \quad G_i : S \rightarrow SS; \quad S \rightarrow a^i, \quad 1 \leq j \leq i$$

for all $i \geq 1$.

Clearly $G_k \triangleleft_u G_l$ for all $k, l, 1 \leq k \leq l$, hence $\mathcal{L}_u(G_k) \subseteq \mathcal{L}_u(G_l)$. Further, this inclusion is proper for all $l > k$, since $\mathcal{L}_u(G_l)$ contains $\{a^l\}^*$, which is not in $\mathcal{L}_u(G_k)$. Now

since $\mathcal{L}_u(G_i)$ is not closed under intersection with regular sets, $\mathcal{L}_u(G_i)$ is not a strict family. This gives the result.

Continuing to contrast uniform and strict interpretations we next consider reduction results, or rather, as we shall see, nonreduction results. However, we will present first the positive results.

LEMMA 3.1. *Let $F = (V, \Sigma, P, S)$ be an infinite context-free grammar form. Then there exists a u-form equivalent context-free grammar form G , every nonterminal of which is infinite.*

Proof. Assume F has at least one nonterminal H which is finite. If there is no such nonterminal take G to be F . Without loss of generality we may assume that F is reduced.

Observe that we can also assume that F has no productions of the type $A \rightarrow A$, since under uniform interpretation such productions yield productions of exactly the same type, and hence can be omitted from F without affecting its generative power.

Now two finite nonterminals A and B are said to be equivalent, $A \equiv B$, iff $A \Rightarrow^* \alpha_1 B \beta_1$ and $B \Rightarrow^* \alpha_2 A \beta_2$ in F for some $\alpha_1, \alpha_2, \beta_1, \beta_2$. Let $[A]$ denote the equivalence class of A under \equiv . Clearly the equivalence classes of finite nonterminals are partially ordered under \leq defined as follows:

$[A] \leq [B]$ iff there is A' in $[A]$ and B' in $[B]$ such that

$$A' \Rightarrow^* \alpha B' \beta \quad \text{for some } \alpha \text{ and } \beta.$$

Consider a maximal equivalence class $[A]$; that is, if there is an equivalence class $[B]$ with $[A] \leq [B]$, then $[A] = [B]$. Such an equivalence class must exist. From $F = (V, \Sigma, P, S)$ we will construct a $G = (U, \Sigma, Q, S)$ such that $\mathcal{L}_u(F) = \mathcal{L}_u(G)$ and G has fewer finite nonterminals than F . Let $[A] = \{A_1, \dots, A_m\}$, $m > 0$ and $L([A]) = \{x : x \text{ in } \Sigma^* \text{ and } A_i \Rightarrow x \text{ for some } A_i \text{ in } [A]\}$; this is well defined since if A_i, A_j are in $[A]$, then $A_i \Rightarrow^* \alpha A_j \beta$ implies either $\alpha\beta = \lambda$, or $\alpha\beta$ contains only nonterminals equivalent to A , in which case $L([A]) = \{\lambda\}$. Let $U = V - [A]$ and define a substitution τ on V as follows:

$$\tau(X) = X \quad \text{for all } X \text{ in } U, \quad \tau(A_i) = L([A]) \quad \text{for all } A_i \text{ in } [A].$$

Now let $Q = \{B \rightarrow \tau(\alpha) : B \text{ is in } U, B \rightarrow \alpha \text{ is in } P\}$. The equality of $\mathcal{L}_u(F)$ and $\mathcal{L}_u(G)$ should now be clear and furthermore the lemma follows by induction on the number of equivalence classes. \square

Our second result concerns "isolation". Under strict interpretation of a form F any derivation according to F can be isolated. By this we mean the chosen derivation can be so interpreted as to be the only possible derivation in an interpretation grammar. This is not true, in general, under uniform interpretations. Consider

$$F : S \rightarrow SS; \quad S \rightarrow a;$$

then $S \Rightarrow SS \Rightarrow aa$ in F , but there is no uniform interpretation $F' \triangleleft_u F$ such that this is the only derivation in F' . For, if this were so, the language $\{aa\}$ would be in $\mathcal{L}_u(F)$, but as we have seen above this is not the case. We do have a weaker "isolation" result, namely:

LEMMA 3.2. *Let $F = (V, \Sigma, P, S)$ be a context-free grammar form with $L(F) \neq \emptyset$. Then there exist a word x in $L(F)$ and an interpretation $F' \triangleleft_u F$ such that $L(F') = \{x\}$; that is, the derivation of x in F can be isolated.*

Proof. Since $L(F)$ is nonempty, there is a word in $L(F)$ and moreover there is a smallest such word; let one of these be x . Now there is a shortest derivation $S \Rightarrow_{nr}^+ x$ in F . For if this were not so then we would have:

$$S \Rightarrow^* \alpha_1 A \alpha_2 = \alpha_1 \beta_1 A \beta_2 \alpha_2 \Rightarrow^* x \quad \text{in } F.$$

In this case we obtain $S \Rightarrow^* \alpha_1 A \alpha_2 \Rightarrow^* x_1$ by omitting the production $A \rightarrow \beta_1 A \beta_2$. Now either $|x_1| < |x|$, giving a contradiction, or $x_1 = x$, in which case the derivation can be shortened, again a contradiction.

Since $S \Rightarrow_{nr}^+ x$ in F then there is trivially an $F' \triangleleft_u F$ such that $L(F') = \{x\}$. \square

The remarks previous to this lemma also indicate that the following ‘‘contraction’’ transformation does not preserve u-form equivalence. Let $F = (V, \Sigma, P, S)$ be a context-free grammar form and $A \Rightarrow^* \alpha$ be some derivation in F for some nonterminal A in $V - \Sigma$.

Define $G = (V, \Sigma, P \cup \{A \rightarrow \alpha\}, S)$. Under strict interpretation $\mathcal{L}(F) = \mathcal{L}(G)$. However, consider F to be $S \rightarrow SS; S \rightarrow a$ and G to be

$$S \rightarrow SS; \quad S \rightarrow a; \quad S \rightarrow aaa,$$

since $S \Rightarrow^* aaa$ is in F . It is easily seen that $\mathcal{L}_u(F) \neq \mathcal{L}_u(G)$.

This transformation is usually used in the proof of the following ‘‘simulation’’ result. Let $F_i = (V_i, \Sigma_i, P_i, S_i)$, $i = 1, 2$ be two context-free grammar forms such that for each production $A \rightarrow \alpha$ in P_1 there is a derivation $A \Rightarrow^* \alpha$ in P_2 ; then $\mathcal{L}(F_1) \subseteq \mathcal{L}(F_2)$. Now for all productions in G there is a derivation in F . However $\mathcal{L}_u(G) \not\subseteq \mathcal{L}_u(F)$, since this would imply $\mathcal{L}_u(F) = \mathcal{L}_u(G)$, which does not hold. Thus the ‘‘simulation’’ result does not hold for uniform interpretation.

Our final positive result is similar in nature to the spanning normal form theorem of [MSW4]. We say a context-free grammar form $F = (V, \Sigma, P, S)$ is in *recurrent normal form* if the following conditions hold:

(i) Each nonrecurrent nonterminal A has only productions of the types

$$A \rightarrow BC; \quad A \rightarrow a.$$

(ii) Each recurrent nonterminal A has only productions of the types:

(a) $A \rightarrow BC, B \neq A \neq C$,

(b) $A \rightarrow a$,

(c) $A \rightarrow \lambda$,

(d) $A \rightarrow \alpha_0 A \alpha_1 \cdots A \alpha_m, \alpha_i$ in $(V - \{A\}) \cup \{\lambda\}, 0 \leq i \leq m$.

A context-free grammar $F = (V, \Sigma, P, S)$ is said to be *n-recurrent*, $n \geq 0$, if for all recurrent productions $A \rightarrow \alpha, |\alpha|_A \leq n$. We say a context-free grammar form $F = (V, \Sigma, P, S)$ is in *n-recurrent normal form*, $n \geq 0$, if it is in recurrent normal form and it is *n-recurrent*.

We now have:

THEOREM 3.3. *For every n-recurrent context-free grammar form F there exists a u-form equivalent context-free grammar form G in n-recurrent normal form.*

Proof. This is clear. \square

Theorem 3.3 is the best we can do in two senses. First, if there are no recurrent nonterminals in F then we have the Chomsky normal form result as in [MSW5]. However, in general we cannot reduce an *n-recurrent* grammar form to an $(n - 1)$ -recurrent grammar form. For example, consider:

$$F: S \rightarrow SSS; \quad S \rightarrow a;$$

then $\mathcal{L}_u(F) = \{\Sigma, \Sigma(\Sigma\Sigma)^*: \Sigma \text{ an alphabet}\}$.

It is easy to see that if G is u-form equivalent to F and 2-recurrent then $G = (V, \{a\}, P, S)$ must at least have productions

$$S \rightarrow \alpha_1 S \alpha_2 S \alpha_3; \quad S \rightarrow a,$$

where $\alpha_1 \alpha_2 \alpha_3$ are in $(V - \{S\})^*$. Now if $\alpha_1 \alpha_2 \alpha_3 = \lambda$ then the language $\{a\}^*$ can be

obtained which is not in $\mathcal{L}_u(F)$. If $\alpha_1\alpha_2\alpha_3 \neq \lambda$, then we can ensure via interpretation that

$$\alpha_1\alpha_2\alpha_3 \Rightarrow^* x \text{ implies } x \text{ is in } \{b\}^* \text{ for some } b.$$

In this case we can obtain via interpretation an infinite language which contains the word a but not the word b . Again, this language cannot be found in $\mathcal{L}_u(F)$.

We have almost proved our first nonreduction result, namely:

THEOREM 3.4. *There are n -recurrent grammar forms for $n > 0$ which have no u -form equivalent $(n - 1)$ -recurrent grammar form.*

Proof. For $n > 1$ consider the forms

$$F_n : S \rightarrow S^n; \quad S \rightarrow a,$$

proceeding as above. For the case $n = 1$, consider

$$F_1 : S \rightarrow aS; \quad S \rightarrow \lambda.$$

Now $\mathcal{L}_u(F_1) = \{\Sigma^* : \Sigma \text{ an alphabet}\}$. It should be clear that no 0-recurrent grammar form is u -form equivalent to F_1 , since in such a grammar form any length word can be isolated. \square

COROLLARY 3.5. *Reduction to Chomsky normal form is not in general possible under uniform interpretation.*

Similarly, removal of left recursion and reduction to Greibach normal form is also not possible. We leave the details to the interested reader, suggesting F_2 of the proof of Theorem 3.4 as a candidate grammar form for this task.

Theorem 3.3 is also the best we can do in a second sense; namely, we cannot remove λ -productions for recurrent nonterminals.

THEOREM 3.6. *Let F be given by $S \rightarrow aS; S \rightarrow \lambda$. Then there is no λ -free context-free grammar form G such that $\mathcal{L}_u(G) = \mathcal{L}_u(F)$.*

Proof. Assume there is such a grammar form G . Then by Lemma 3.2 there is a word x in $L(G)$ such that $\{x\}$ is in $\mathcal{L}_u(G)$. Now since G is λ -free, $x \neq \lambda$. Thus because $\mathcal{L}_u(F) = \{\Sigma^* : \Sigma \text{ an alphabet}\} \cup \{\{\lambda\}\}$, we obtain a contradiction. \square

Our next result shows that even chain-productions cannot be removed in a λ -free grammar form. A *chain-production* is a production $A \rightarrow B$, A, B both nonterminal.

THEOREM 3.7. *Let F be given by*

$$S \rightarrow A; \quad S \rightarrow bbb; \quad A \rightarrow aA; \quad A \rightarrow a.$$

Then there is no chain-free λ -free context-free grammar form G with $\mathcal{L}_u(G) = \mathcal{L}_u(F)$.

Proof. Assume such a $G = (V, \Sigma, P, S)$ exists. Now $S \Rightarrow^* aa$ is not a nonrecurrent derivation for any a in Σ , since this would imply $\{aa\}$ is in $\mathcal{L}_u(F)$, which is a contradiction. Therefore consider a derivation $S \Rightarrow^* aa$ for some a in Σ . Since G is both chain-free and λ -free we must have either

$$S \Rightarrow aB \Rightarrow aa, \quad B \text{ in } V - \Sigma (S \Rightarrow Aa \Rightarrow aa, A \text{ in } V - \Sigma)$$

or

$$S \Rightarrow AB \Rightarrow aB \Rightarrow aa, \quad A, B \text{ in } V - \Sigma.$$

Since $A \rightarrow a$ and $B \rightarrow a$ are both nonrecurrent, either A or B is equal to S . Assume $B = S$ without any loss of generality. Then we have $S \Rightarrow^* aS$ in both cases. Thus $S \Rightarrow^* a^n S$ for all $n \geq 1$. Secondly, observe that there must be some b in Σ such that $S \Rightarrow_{nr}^+ bbb$. Otherwise $\{bbb\}$ is not in $\mathcal{L}_u(G)$. Moreover $a \neq b$, otherwise $\{a, aaa\}$ is in $\mathcal{L}_u(G)$, a

contradiction. Hence there is a derivation

$$S \Rightarrow^* a^n S \Rightarrow a^n bbb \quad \text{in } G.$$

But such words are not in $L(F')$ for any $F' \triangleleft_u F$. \square

Note that if we remove the λ -free condition, a chain-free grammar form can easily be obtained by replacing $A \rightarrow B$ with $A \rightarrow BC$ and $C \rightarrow \lambda$.

4. Generative power of general grammar forms. In [MPSW] it was shown how the introduction of a specific interpretation mechanism, the production-restricted interpretation, enabled numerous non-context-free families to be generated by general grammar forms. In the present section we demonstrate how the EOL, ETOL, matrix and scattered context language families can be generated by general grammar forms under uniform interpretation in a simpler manner than that of [MPSW]. While the generation of EOL and ETOL is immediate, the generation of matrix and scattered context depends upon the introduction of the notion of a complete set of productions for general grammars. This notion, which is usually associated with EOL and EIL grammars (see [MSW3] or [RS], for example), we now define.

DEFINITION. Let $G = (V, \Sigma, P, S)$ be a general grammar (form) and $m > 0$ an integer. We say G is an m -smooth grammar if

- (i) for all $\alpha \rightarrow \beta$ in P , $|\alpha| \leq m$, and
- (ii) for all α in $V^*(V - \Sigma)V^*$ with $1 \leq |\alpha| \leq m$ there is a production $\alpha \rightarrow \beta$ in P , for some β in V^* .

Convention. When specifying a particular general grammar form, we assume that all unspecified left-hand sides have identity productions $\alpha \rightarrow \alpha$ so as to ensure that the form is m -smooth for some $m > 0$.

We now modify the notion of interpretation by taking into account the notion of smoothness. If F is an m -smooth general grammar form and $F' \triangleleft_u F$ (or $F' \triangleleft F$), then F' must also be an m -smooth general grammar form.

That this is a subtle restriction can be seen by considering the 2-smooth general grammar form

$$F: S \rightarrow AA; \quad AA \rightarrow a; \quad A \rightarrow a.$$

Under smoothness $\mathcal{L}_u(F)$ only contains nonempty languages of at least two words. For consider an $F' \triangleleft_u F$; then there must be at least one production

$$S' \rightarrow A'A''$$

in F' , at least one production

$$A'A'' \rightarrow a'$$

in F' and finally at least one of each production

$$A' \rightarrow a'' \quad \text{and} \quad A'' \rightarrow a'''.$$

Otherwise, since F' must be 2-smooth by our convention, the absence of a specific production for an α implies that there is a production $\alpha \rightarrow \alpha$. However, if F' has $S' \rightarrow S'$, $A'A'' \rightarrow A'A''$, $A' \rightarrow A'$ or even $A'' \rightarrow A''$, then F' is not an interpretation of F . Thus $\#L(F') \geq 2$.

It is of course clear that there are singleton languages in $\mathcal{L}(F)$, or even in $\mathcal{L}_u(F)$ when F is not assumed to be smooth.

This example also illustrates that under the smoothness convention isolation is not necessarily possible. This is in contradistinction to all other interpretation mechanisms including the uniform interpretation considered in the present paper.

Finally, add the productions

$$S \rightarrow S; \quad A \rightarrow A; \quad AA \rightarrow AA$$

to F , in which case $\mathcal{L}_u(F)$ is the same whether or not F is assumed to be 2-smooth.

This is, in fact, generally true. For an arbitrary general grammar form H add the productions $\alpha \rightarrow \alpha$ to H for all $\alpha \rightarrow \beta$ in H . Let this new form be G , then $\mathcal{L}_u(G)$ gives the same language family whether or not G is assumed to be smooth. This together with the particular form F above demonstrates that the smoothness convention enables all uniform families to be generated and more. However, it is not clear when $\mathcal{L}_u(G) = \mathcal{L}_u(H)$ under the smoothness convention. Later we make intrinsic use of the smoothness convention so that we can obtain a G such that $\mathcal{L}_u(G)$ is the family of matrix languages. We claim that the star closure of the matrix languages is included in $\mathcal{L}_u(G)$ when the smoothness convention is ignored. Thus, in this case the question of the equality of $\mathcal{L}_u(G)$ with and without smoothness is reduced to that of whether the matrix languages are star closed. This is, in fact, a well-known open problem.

In the case of context-free grammar forms 1-smoothness does not make any essential difference to the generative power. This means that up to the empty set the same language families can be obtained.

We now turn to our first results:

THEOREM 4.1. *Let G be a 2-smooth general grammar form specified by the productions:*

$$\begin{aligned} S &\rightarrow ALZR; \quad AL \rightarrow AAL; \quad AL \rightarrow A; \\ AZ &\rightarrow AZ\bar{Z}; \quad A\bar{Z} \rightarrow ZA; \quad A\bar{Z} \rightarrow ZZA; \quad A\bar{Z} \rightarrow aA; \\ AR &\rightarrow R; \quad AR \rightarrow \lambda. \end{aligned}$$

Then $\mathcal{L}_u(G) = \mathcal{L}(\text{EOL})$.

Proof. Intuitively the A acts as an activation messenger, each A sweeping from left to right, causing a rewriting of each symbol until the R is reached. At this point either the A is erased or both the A and R are erased. This latter case corresponds to the termination of the simulated derivation. Finally, a number of A 's are first generated before the simulation is begun.

(i) $\mathcal{L}(\text{EOL}) \subseteq \mathcal{L}_u(G)$. Consider an arbitrary EOL language L . Then L can be generated by an EOL grammar $F = (V, \Sigma, P, Z)$, all of whose productions are of the following types:

$$\begin{aligned} B &\rightarrow CD; \quad B \rightarrow C; \quad B \rightarrow a; \\ a &\rightarrow N; \quad N \rightarrow N, \quad \text{where } N \text{ is a blocking symbol.} \end{aligned}$$

Construct $G' \triangleleft_u G$ as follows. Let $G' = (V', \Sigma, P', S)$, where $V' = \{S, A, L, R\} \cup V \cup \{\bar{B} : B \text{ in } V - \Sigma\}$, and P' contains the productions

- (i) $S \rightarrow ALZR; AL \rightarrow AAL; AL \rightarrow A; AR \rightarrow R; AR \rightarrow \lambda,$
- (ii) $AB \rightarrow A\bar{B};$ for all B in $V - \Sigma,$
- (iii) $A\bar{B} \rightarrow CA$ if $B \rightarrow C$ is in $P,$
- (iv) $A\bar{B} \rightarrow CDA$ if $B \rightarrow CD$ is in $P,$
- (v) $A\bar{B} \rightarrow aA$ if $B \rightarrow a$ is in $P.$

A derivation in G' proceeds as follows:

$$S \Rightarrow^+ A^{n-1}LZR \Rightarrow A^nZR \Rightarrow A^n\bar{Z}R \Rightarrow A^{n-1}\alpha AR \Rightarrow A^{n-1}\alpha R \Rightarrow \dots$$

Clearly, since activation symbols cannot affect each other, any n -step derivation in F which does not produce any terminal words before the n th step can be simulated by G' .

Conversely, each A introduced must sweep completely from left to right up to R . Hence the only way a terminal word is produced is when an n -step terminating derivation of F is simulated by G' . Thus $L(G') = L(F)$.

(ii) $\mathcal{L}_u(G) \subseteq \mathcal{L}(\text{EOL})$. Consider an arbitrary $G' \triangleleft_u G$. First observe that there may be many interpretations of A, L, Z and R . However there must be a finite number of them, so it is sufficient to consider one A, L, Z and R such that $S \rightarrow ALZR$, since $\mathcal{L}(\text{EOL})$ is closed under union. By the choice of G , when $S \rightarrow ALZR$ is fixed as the starting production in G' , then A, L and R cannot be changed via productions since we are dealing with uniform interpretations. Further, for termination purposes $AR \rightarrow \lambda$ must be present as must $AL \rightarrow A$. Since G and G' are 2-smooth, then G' must have nonblocking productions for interpretations of AZ and $A\bar{Z}$. Let $G' = (V', \Sigma, P', S)$, where $\{Z, S, A, L, R\} \subseteq V'$. Construct an EOL grammar $F = (V, \Sigma, P, Z)$ from G' as follows: $V = (V' - \{S, A, L, R\}) \cup \{N\}$ and P contains the productions

- (i) $N \rightarrow N; a \rightarrow N$ for all a in Σ ,
- (ii) $B \rightarrow \bar{B}$ if $AB \rightarrow A\bar{B}, B$ in $V' - \Sigma$,
- (iii) $\bar{B} \rightarrow C$ if $A\bar{B} \rightarrow CA, B$ in $V' - \Sigma$
- (iv) $\bar{B} \rightarrow CD$ if $A\bar{B} \rightarrow CDA, B$ in $V' - \Sigma$
- (v) $\bar{B} \rightarrow a$ if $A\bar{B} \rightarrow aA, a$ in Σ, B in $V' - \Sigma$

By the remarks in part (i) it should be clear that $L(F) = L(G')$, when $AL \rightarrow AAL; AL \rightarrow A; AR \rightarrow R; AR \rightarrow \lambda$ are all in P' . In the case that neither $AL \rightarrow A$ nor $AR \rightarrow \lambda$ are in P' , then $L(G') = \emptyset$. If $AL \rightarrow AAL$ is not in P' , then take $Z \rightarrow a$ into P for all $A\bar{Z} \rightarrow aA$ in P' , since this is the only terminating production. The situation is similar if $AR \rightarrow R$ is not in P' . \square

We may generalize this result in two ways. First, note that interpretations of G in Theorem 4.1 give rise to simulations of interpretations of the synchronized EOL grammar $F: Z \rightarrow ZZ; Z \rightarrow Z; Z \rightarrow a$ where productions for a are blocking. Let $\text{EOL}(G)$ denote the simulated EOL grammar F . Then it follows, by arguments similar to those used in Theorem 4.1 that for each $G' \triangleleft_u G, \mathcal{L}_u(G') = \mathcal{L}_u(\text{EOL}(G'))$. Since $\text{EOL}(G)$ is synchro-very-complete (see [MSW7]), every u-synchro-EOL family can be obtained as an $\mathcal{L}_u(G')$ for some $G' \triangleleft_u G$.

THEOREM 4.2. *Let \mathcal{L} be a uniform synchronized EOL form family of languages. Then there exists a smooth general grammar form G such that $\mathcal{L} = \mathcal{L}_u(G)$.*

In fact, this result can be extended to include every uniform EOL family, by adding the productions

$$S \rightarrow ALaR; Aa \rightarrow A\bar{a}; A\bar{a} \rightarrow ZA; A\bar{Z} \rightarrow A$$

to G , where \bar{a} is a new nonterminal.

Second, we can also generate the ETOL languages.

THEOREM 4.3. *Let H be a 2-smooth general grammar form specified by*

$$\begin{aligned} S &\rightarrow LZR; L \rightarrow AL; AL \rightarrow A; \\ AZ &\rightarrow A\bar{Z}; A\bar{Z} \rightarrow ZA; A\bar{Z} \rightarrow ZZA; A\bar{Z} \rightarrow aA; \\ AR &\rightarrow R; AR \rightarrow \lambda. \end{aligned}$$

Then $\mathcal{L}_u(H) = \mathcal{L}(\text{ETOL})$.

Proof. The essential change is that L may introduce many (but a finite number) different A 's. Each different A corresponds to a different table. The proof is analogous to that of Theorem 4.1 and is omitted. However the reader should be aware of one subtle use of smoothness. Let H' be an interpretation of H , and $\{A_1, \dots, A_m\}$ be the interpretation of A . If $L \rightarrow A_i L$ is not in H' for any i , then $A_i L \rightarrow A_i$ can never be applied;

hence the productions $L \rightarrow A_i L$ give $\Sigma^* L$ for some $\Sigma \subseteq \{A_1, \dots, A_m\}$. Now note that by smoothness $A_i L \rightarrow A_i$ must be in H' for each i , $1 \leq i \leq m$; hence we obtain $\Sigma^+ Z'R$ as initial configurations of the simulation. If we do not rely on smoothness then we have $KZ'R$ as initial configurations, where $K \subseteq \Sigma^+$ is regular. However regular-controlled ETOL grammars give ETOL languages; therefore the result again follows. \square

The final remarks in this proof of Theorem 4.3 lead to the following generalization. Let \mathcal{C} be a family of control languages for ETOL grammars such that \mathcal{C} is generated by some general grammar form. Then the class of \mathcal{C} -controlled ETOL languages can also be defined by a general grammar form.

We now turn to sequential rewriting and demonstrate a similar result in this case, namely that classes of controlled context-free languages can be defined by general grammar forms.

THEOREM 4.4. *Let G be a 2-smooth general grammar form defined by the productions:*

$$\begin{aligned} S &\rightarrow \bar{A}AS; \quad AS \rightarrow AZ, \\ AZ &\rightarrow ZA; \quad \bar{A}\bar{Z} \rightarrow Z\bar{A}; \quad Aa \rightarrow aA; \quad \bar{A}a \rightarrow a\bar{A}, \\ A &\rightarrow \hat{A}; \quad \bar{A} \rightarrow \hat{\bar{A}}, \\ \hat{A}\bar{Z} &\rightarrow \hat{A}\bar{Z}; \quad \hat{A}\bar{Z} \rightarrow Z; \quad \hat{A}\bar{Z} \rightarrow ZZ; \quad \hat{A}\bar{Z} \rightarrow a, \\ \hat{\bar{A}}\bar{Z} &\rightarrow \hat{\bar{A}}\bar{Z}; \quad \hat{\bar{A}}\bar{Z} \rightarrow Z; \quad \hat{\bar{A}}\bar{Z} \rightarrow ZZ; \quad \hat{\bar{A}}\bar{Z} \rightarrow a. \end{aligned}$$

Then $\mathcal{L}_u(G) = \mathcal{L}(\text{MAT})$, the family of matrix languages.

Proof. That $\mathcal{L}(\text{MAT}) \subseteq \mathcal{L}_u(G)$ can be seen by simulating with interpretations of G matrix grammars whose matrices contain exactly two productions. The A and \bar{A} are activation messengers for each of the two productions in a matrix.

The productions in a matrix grammar can be assumed to be of types

$$B \rightarrow C, \quad B \rightarrow CD, \quad B \rightarrow a$$

without any loss of generality. Hence these can be obtained as interpretations of the Z and \bar{Z} productions.

Conversely, consider an interpretation $G' = (V, \Sigma, P, S') \triangleleft_u G(\mu)$. We construct a matrix grammar $M = (V', \Sigma, P', Z)$ such that $L(M) = L(G')$. First note that since $\mathcal{L}(\text{MAT})$ is closed under union we need only consider the case that G' has one production $AS \rightarrow AZ$.

Second, note that since G and G' are smooth then for all A' in $\mu(A)$ and Z' in $\mu(Z)$ there is a production $A'Z' \rightarrow Z'A'$ in P . Otherwise G' would not be an interpretation of G . This is once more an implication of smoothness. This fact is important since it means each A' will indeed give rise to an \hat{A}' (again because of smoothness), $\hat{A}'Z' \rightarrow \hat{A}'\bar{Z}'$ is in G' and also there is a production $\hat{A}'\bar{Z}' \rightarrow \alpha$ in G' . Of course the same holds true for the \bar{A} symbols as well. Let

$$R(\bar{A}'A') = \{\hat{A}'\hat{A}': \bar{A}' \rightarrow \hat{A}' \text{ and } \hat{A}' \rightarrow \hat{A}' \text{ are in } G'\},$$

denote the pairs of activation symbols derived from a pair $\bar{A}'A'$ where $S \rightarrow \bar{A}'A'S$ is in G' . Let

$$\begin{aligned} T(\hat{A}'\hat{A}') &= \{[Z_1 \rightarrow \alpha_1, Z_2 \rightarrow \alpha_2]: \hat{A}'Z_1 \rightarrow \hat{A}'\bar{Z}_1; \\ &\quad \hat{A}'\bar{Z}_1 \rightarrow \alpha_1; \hat{A}'Z_2 \rightarrow \hat{A}'\bar{Z}_2; \hat{A}'\bar{Z}_2 \rightarrow \alpha_2 \text{ are in } G'\}. \end{aligned}$$

Finally, let

$$P' = \bigcup_{S \rightarrow \bar{A}'A'S \text{ in } G'} \bigcup_{\hat{A}'\hat{A}' \text{ in } R(\hat{A}'A')} T(\hat{A}'\hat{A}')$$

Since the \bar{A}' and A' symbols can migrate freely to the right, letting $V' = \Sigma \cup \mu(Z)$, we have $L(M) = L(G')$. \square

That G is 2-smooth is crucial to the validity of the proof of Theorem 4.4. For if G were not a 2-smooth grammar, then under interpretation the productions $Aa \rightarrow aA$ and $\bar{A}a \rightarrow a\bar{A}$ could be omitted. Let G_1 be G with $Aa \rightarrow aA$ and $\bar{A}a \rightarrow a\bar{A}$ omitted; then even $\mathcal{L}_u(G_1) \supseteq \mathcal{L}_u(G)$. (Note, however, that G_1 is not an interpretation of G .) This can be seen since by preventing the messengers from passing over terminals we can obtain the star of an arbitrary matrix language. Hence proper containment of $\mathcal{L}_u(G)$ in $\mathcal{L}_u(G_1)$ depends upon whether the star closure of the matrix languages properly contains the matrix languages. This is a well-known open problem.

To give the reader some intuition for this claim, consider the generation of $\{a^n b^n c^n : n > 0\}^*$.

Let H be the matrix grammar given by the productions:

$$\begin{aligned} m_1: & [Z \rightarrow Z; Z \rightarrow \bar{B}_1 B_2], \\ m_2: & [\bar{B}_1 \rightarrow \bar{a} B_1 \bar{b}; B_2 \rightarrow \bar{c} B_2], \\ m_3: & [B_1 \rightarrow \bar{a} B_1 \bar{b}; B_2 \rightarrow \bar{c} B_2], \\ m_4: & [B_1 \rightarrow \lambda; B_2 \rightarrow \lambda], \\ m_5: & [\bar{a} \rightarrow \bar{a}; \bar{a} \rightarrow a], \\ m_6: & [\bar{a} \rightarrow \bar{a}; \bar{a} \rightarrow Za], \\ m_7: & [\bar{a} \rightarrow \bar{a}; \bar{a} \rightarrow a], \\ m_8: & [\bar{b} \rightarrow \bar{b}; \bar{b} \rightarrow b], \\ m_9: & [\bar{c} \rightarrow \bar{c}; \bar{c} \rightarrow c]. \end{aligned}$$

Now $L(H) \supseteq \{a^n b^n c^n : n > 0\}^*$ since in a matrix grammar words of the form $a^{n_1} b^{n_1} c^{m_1} a^{n_2} b^{n_2} c^{m_2}$ are obtained, where $n_1 + n_2 = m_1 + m_2$. In other words the matrix grammar knows no boundaries. However, in the simulation of H by an interpretation G'_1 of G in which the messenger symbols cannot skip over terminals, the application of m_6 implies everything to the right of \bar{a} must be terminal. Hence $L(G'_1) = \{a^n b^n c^n : n > 0\}^*$.

Hence considering G not to be a smooth grammar means that G'_1 is also an interpretation of G . Thus smoothness is crucial.

By a slight modification of the grammar G of Theorem 4.4 we are also able to obtain the family of scattered context languages, $\mathcal{L}(\text{SCAT})$.

THEOREM 4.5. *Let F be a 2-smooth general grammar form defined by the productions:*

- (1) $S \rightarrow AS; AS \rightarrow AZ,$
- (2) $AZ \rightarrow ZA; Aa \rightarrow aA; A \rightarrow \hat{A},$
- (3) $\hat{A}Z \rightarrow \hat{A}\bar{Z}; \hat{A}\bar{Z} \rightarrow Z\bar{A}; \hat{A}\bar{Z} \rightarrow ZZ\bar{A}; \hat{A}\bar{Z} \rightarrow a\bar{A},$
- (4) $\bar{A}Z \rightarrow Z\bar{A}; \bar{A}a \rightarrow a\bar{A}; \bar{A} \rightarrow \hat{A},$
- (5) $\hat{A}Z \rightarrow \hat{A}\bar{Z}; \hat{A}\bar{Z} \rightarrow Z; \hat{A}\bar{Z} \rightarrow ZZ; \hat{A}\bar{Z} \rightarrow a.$

Then $\mathcal{L}_u(F) = \mathcal{L}(\text{SCAT})$.

Proof. Again because of smoothness, any interpretation F' of F has for each A' productions of types (2) and (3), and hence productions of types (4) and (5) for some \bar{A}' , for all α in $\mu(Z) \cup \mu(a)$. This enables a two-production scattered-context-production to be simulated. The details of the proof are similar to those of Theorem 4.4. \square

Theorem 4.4 is a special case of a more general theorem. Let \mathcal{C} be a family of control languages, where each control language represents a subset of P^* , P being some finite set of context-free productions. Then if \mathcal{C} is generable by some general grammar form we can also define the \mathcal{C} -controlled context-free language family by a general grammar form.

Finally, note that smoothness is again crucial to the equality $\mathcal{L}_u(F) = \mathcal{L}(\text{SCAT})$ in Theorem 4.5. Without it, type (4) productions $\bar{A}Z \rightarrow Z\bar{A}$; $\bar{A}a \rightarrow a\bar{A}$ could be omitted in interpretations, and hence context-sensitive rewriting could be simulated. This follows from the observation that the \bar{A} messenger no longer moves right but immediately causes the rewriting of the symbol to its right, if there is a production for this symbol. Thus context-sensitive productions of the type

$$BC \rightarrow \alpha \quad \text{with } |\alpha| \geq 2$$

can be simulated. Hence without the smoothness convention $\mathcal{L}_u(F) = \mathcal{L}(\text{CS})$. If we include erasing productions

$$\hat{A}\bar{Z} \rightarrow \bar{A} \quad \text{and} \quad \hat{A}\bar{Z} \rightarrow \lambda$$

in F , then without smoothness $\mathcal{L}_u(F) = \mathcal{L}(\text{RE})$.

5. Generative power of some simple types of grammar forms under uniform interpretation. In this section we study the generative power of (Φ, Σ) -forms under uniform interpretation, where Φ consists of just one or two variables. Everything up to Theorem 5.5 deals with *context-free* forms. We start by presenting a characterization of the language families generated by separated (S, Σ) -forms. Forms considered will be assumed to generate infinite languages throughout.

We need a few preliminary notions. A finite collection \mathcal{K} of languages over $\{S\}$, $\mathcal{K} = \{\{S\}, L_1, L_2, \dots, L_n\}$, $L_i \subseteq S^*$, is called *closed* if there is a finite set of productions $R \subseteq S \times S^*$ such that

$$\mathcal{K} = \{L(Q) : Q \subseteq R, L(Q) \text{ is the language of sentential forms generated from } S \text{ by } Q\}.$$

Let Σ be an alphabet, M a finite subset of Σ^* . A substitution τ defined on a single symbol S is called *M-restricted*, if for some dfl-substitution μ we have $\tau(S) \subseteq \mu(M)$.

Using the above notions the language families generated by (S, Σ) -forms can be characterized as follows:

THEOREM 5.1. *A family of languages \mathcal{L} is equal to $\mathcal{L}_u(F)$ for some separated (S, Σ) -form F iff there exists a closed class of languages \mathcal{K} and a finite subset M of Σ^* such that*

$$\mathcal{L} = \{\tau(L) : L \text{ is in } \mathcal{K}, \tau \text{ is } M\text{-restricted}\}.$$

Proof. Left to the reader. \square

THEOREM 5.2. *Let F be an (S, Δ) -form with productions P such that $S \rightarrow z$ is in P for some z in Δ^+ . Then $\mathcal{L}_u(F) \subseteq \mathcal{L}(\text{REG})$ iff P is a finite subset of $S \rightarrow \Delta^* \cup S^+ \cup \Delta^+ S \cup S \Delta^+ \cup S \Delta^+ S$.*

Proof.

Part 1. Suppose F is an (S, Δ) -form with productions $P \subseteq S \rightarrow \Delta^* \cup S^+ \cup \Delta^+ S \cup S \Delta^+ \cup S \Delta^+ S$. Consider an arbitrary $G = (\{S\} \cup \Sigma, \Sigma, Q, S)$, $G \triangleleft_u F$. Then we have $Q = Q_1 \cup Q_2 \cup Q_3$, where $Q_1 = S \rightarrow A \cup B S \cup S C$, A, B, C finite subsets of Σ^* , $Q_2 = S \rightarrow SDS$, D a finite subset of Σ^+ , and $Q_3 = S \rightarrow E$, E a finite subset of S^+ .

Observe that in any derivation of a word x in $L(G)$ productions can be rearranged (if necessary) such that

$$S \xrightarrow[Q_3]{*} y \xrightarrow[Q_2]{*} z \xrightarrow[Q_1]{*} x \text{ holds.}$$

Starting with S , the productions of Q_1 yield the language $L_{Q_1} = B^*AC^*$, the productions of Q_2 the (sentential form) language $L_{Q_2} = (SD)^+S$; hence Q_2 and Q_1 together yield (when starting from S) the (regular) language $L_{Q_2, Q_1} = (B^*AC^*D)^+B^*AC^*$. The (sentential form) language L_{Q_3} generated by productions Q_3 from S is well known to be regular. Since $L(G)$ is obtained from L_{Q_3} by the substitution $L(G) = \tau(L_{Q_3})$ with $\tau(S) = L_{Q_2, Q_1}$ and since regular languages are closed under substitution, $L(G)$ is in $\mathcal{L}(\text{REG})$ as stated.

Part 2. Suppose F is an (S, Δ) -form containing a production not in $S \rightarrow \Delta^* \cup S^+ \cup \Delta^+S \cup S\Delta^+ \cup S\Delta^+S$. Then F contains a production of type (i), (ii) or (iii):

- (i) $S \rightarrow uSv$ with u, v in Δ^+ ;
 - (ii) $S \rightarrow \alpha\beta S u S v$
 - (iii) $S \rightarrow v S u S \beta a \alpha$
- } with α, β in $(\{S\} \cup \Delta)^*$, u, v in Δ^* , a in Δ .

Since cases (ii) and (iii) are symmetric, it suffices to consider cases (i) and (ii). We first discuss case (i). Consider an interpretation $G \triangleleft_u F$ such that G contains exactly two productions $S \rightarrow \bar{u}S\bar{v}$ and $S \rightarrow \bar{w}$, where \bar{u} is in Σ_1^+ , \bar{v} is in Σ_2^+ and \bar{w} is in Σ_3^* , $\Sigma_1, \Sigma_2, \Sigma_3$ mutually disjoint alphabets. (Such a G clearly exists.)

Since $L(G) = \{\bar{u}^n \bar{w} \bar{v}^n : n \geq 0\}$, $L(G)$ is nonregular. Thus $\mathcal{L}_u(F) \subseteq \mathcal{L}(\text{REG})$ does not hold.

Consider now a production $p: S \rightarrow \alpha\beta S u S v$ as listed under case (ii). Let $\Delta^{(1)}, \Delta^{(2)}, \dots, \Delta^{(10)}$, and $\{a, b\}$ be mutually disjoint alphabets, and let h_i be a homomorphism defined on $\Delta \cup \{S\}$ by

$$h_i(a) = \begin{cases} a^{(i)} & \text{for } a \text{ in } \Delta, \\ s & \text{for } a = S. \end{cases}$$

Let $q: S \rightarrow z$ be the shortest non-empty terminal production of F , $|z| = m > 0$. Observe that such a production exists by the assumption in the theorem statement.

Let G be an interpretation of F , $G \triangleleft_u F$ with the following productions:

$$p_1: S \rightarrow h_1(\alpha) a h_2(\beta) S h_3(u) S h_4(v),$$

$$p_2: S \rightarrow h_5(\alpha) b h_6(\beta) S h_7(u) S h_8(v),$$

$$p_3: S \rightarrow h_9(z),$$

$$p_4: S \rightarrow h_{10}(z).$$

Let x_1, x_2, y_1, y_2 be the unique terminal words defined by

$$h_1(\alpha) \xrightarrow[p_3]{*} x_1, \quad h_2(\beta) \xrightarrow[p_3]{*} x_2, \quad h_5(\alpha) \xrightarrow[p_4]{*} y_1, \quad h_6(\beta) S h_7(u) S h_8(v) \xrightarrow[p_4]{*} y_2.$$

Hence we have

$$S \xrightarrow[\{p_1, p_3\}]{*} x_1 a x_2 S h_3(u) S h_4(v) \quad \text{and} \quad S \xrightarrow[\{p_2, p_4\}]{*} y_1 b y_2.$$

Consider now the CF grammar $H = (\{S, a, b\}, \{a, b\}, \{S \rightarrow aSS|b\}, S)$ and the

homomorphism h defined by

$$h(\alpha) = \begin{cases} a & \text{for } \alpha = a, \\ b & \text{for } \alpha = b, \\ \lambda & \text{otherwise.} \end{cases}$$

We have

$$L(H) = h(L(G) \cap \{x_1ax_2, h_3(u), h_4(v), y_1by_2\}^*).$$

However, $L(H)$ is well known to be equal to $D_2\{b\}$, where D_2 is the Dyck-language over the alphabet $\{a, b\}$. D_2 is not linear. (It is indeed known to be of infinite index.) Since $\mathcal{L}(\text{LIN})$, the family of linear languages is a semi-AFL, hence closed under intersection by regular sets and homomorphism, $L(G)$ is not linear. This establishes for case (ii) that $\mathcal{L}_u(F) \subseteq \mathcal{L}(\text{REG})$ does not hold. \square

A few remarks concerning the above proof are in order. Case (ii) of Part 2 is much easier to carry out if F can be assumed to be λ -free. In this case, G can be chosen to contain only two productions. The only point in having productions p_2 and p_4 in the above proof is to obtain a “discernable” derivation $S \Rightarrow^* y_1by_2$ of a nonempty terminal word.

The proof of Part 1 not only establishes that for an (S, Δ) -form F with productions $P \subseteq S \rightarrow \Delta^* \cup S^+ \cup \Delta^+S \cup S\Delta^+ \cup S\Delta^+S$, $\mathcal{L}_u(F) \subseteq \mathcal{L}(\text{REG})$ holds. It also shows that we have $\mathcal{L}_u(F) \subseteq \mathcal{L}(\text{REG}(3))$, the family of regular languages of star height $h \leq 3$. We thus have as an immediate corollary.

COROLLARY 5.3. *Let F be an (S, Δ) -form with at least one terminating production $S \rightarrow z \neq \lambda$. If $\mathcal{L}_u(F)$ contains a regular language of star height $h \geq 4$, then $\mathcal{L}_u(F)$ also contains nonregular languages.*

Before presenting a characterization of linear and finite index (S, Δ) -forms, we need a technical lemma concerning the invariance of the infinite index property under certain operations.

LEMMA 5.4. *(Infinite index lemma). Suppose L is a context-free language, R a regular set, h a homomorphism, $L' = h(L \cap R)$ and L' is of infinite index. Then L is of infinite index.*

Proof. Obvious by the closure properties of finite index languages. \square

THEOREM 5.5. *Let F be an (S, Δ) -form with productions P such that $S \rightarrow z$ is in P for some z in Δ^+ . Then $\mathcal{L}_u(F) \subseteq \mathcal{L}(\text{LIN})$ iff P is a finite subset of either*

$$M_1 = S \rightarrow \Delta^* \cup S^+ \cup \Delta^+S \cup S\Delta^+ \cup S\Delta^+S \quad \text{or} \quad M_2 = S \rightarrow \Delta^* \cup \Delta^*S\Delta^*.$$

Proof.

Part 1. Suppose P is a subset of M_1 . Then $\mathcal{L}_u(F) \subseteq \mathcal{L}(\text{REG})$ by Theorem 5.2. Suppose P is a subset of M_2 . Then, clearly, $\mathcal{L}_u(F) \subseteq \mathcal{L}(\text{LIN})$, holds.

Part 2. Suppose P is not a subset of M_1 or M_2 . If P is not a subset of $M_1 \cup M_2$, then P contains a production of type (ii) or (iii) as discussed in Part 2 of Theorem 5.2. By that proof, we have $L = h(L' \cap R)$ for some language L' of infinite index, for some regular set R and homomorphism h . By the infinite-index lemma $\mathcal{L}_u(F)$ contains a language of infinite index and hence a nonlinear language.

It remains to consider the case that $P \subseteq M_1 \cup M_2$ but neither $P \subseteq M_1$ nor $P \subseteq M_2$ holds. This clearly means that P contains a production of $S \rightarrow \Delta^+S\Delta^+$ and a production of either $S \rightarrow S^k$ ($k \geq 2$) or $S \rightarrow SuS$ with u in Δ^+ . In both cases, nonlinear products of linear languages can easily be obtained as interpretations of F . \square

THEOREM 5.6. *Let F be an (S, Δ) -form with productions P containing a nonempty terminal production. Then $\mathcal{L}_u(F) \subseteq \mathcal{L}(\text{FINDEX})$ (the class of finite index CF languages) iff P is a finite subset of*

$$Z = S \rightarrow \Delta^* \cup \Delta^* S \Delta^* \cup S \Delta^+ S \cup S^+.$$

Proof. If P is a finite subset of Z , then $\mathcal{L}_u(F)$ contains (following the idea of the proof of Part 1 of Theorem 5.2) only substitutions of linear languages into regular languages. All such languages are known to be of finite index.

Conversely, suppose P is not a finite subset of Z . Then P contains productions as listed under case (ii) and (iii) in Part 2 of the proof of Theorem 5.2. As was argued there, this implies that $\mathcal{L}_u(F)$ contains languages of infinite index. \square

In the next theorem we establish that generative power increases dramatically when going from the (S, Σ) case to only the sequential $(\{S, A\}, \Sigma)$ case, and that even in the (S, Σ) case general forms are much more powerful than context-free forms.

THEOREM 5.7.

- (i) *There exists a regular language L such that for no general (S, Σ) -form F is L in $\mathcal{L}_u(F)$.*
- (ii) *For every type 0 language L there exists a sequential general $(\{S, A\}, \{a, b\})$ -form F with L in $\mathcal{L}_u(F)$.*
- (iii) *There are general (S, Σ) -forms F such that $\mathcal{L}_u(F)$ contains non-context-free languages.*

Proof of (i). Consider $L = ca^+c$ and suppose L is in $\mathcal{L}_u(F)$ for some general (S, a) -form F ; i.e., $L = L(G)$ for some general (S, Σ) -grammar G . We must have $S \Rightarrow_G^* cac$, and there must be a production $S \rightarrow \alpha S \beta$ in G . Because $S \Rightarrow_G^* cac$, we must have $\alpha\beta = \lambda$, a contradiction.

Proof of (ii) and (iii). Let L be an arbitrary type 0 language. We may assume that $L = L(G)$, $G = (V, \Sigma, P, A_1)$, where $V - \Sigma = \{A_1, A_2, \dots, A_n\}$ and $P \subseteq (V - \Sigma)^+ \times (V - \Sigma)^* \cup (V - \Sigma) \times \Sigma$. Define a homomorphism h on V by

$$h(\alpha) = \begin{cases} \alpha & \text{if } \alpha \text{ is in } \Sigma, \\ Sa^iS & \text{if } \alpha = A_i. \end{cases}$$

Consider $F_1 = (\{S, A, a, b\}, \{a, b\}, P_1, A)$ and $F_2 = (\{S, a, b\}, \{a, b\}, P_2, S)$ as follows:

$$P_1 = \{A \rightarrow SaS\} \cup Q, \quad P_2 = \{S \rightarrow aSaS\} \cup Q,$$

$$Q = \{h(\alpha) \rightarrow h(\beta) : \alpha \rightarrow \beta \text{ is in } P \cap (V - \Sigma)^+ \times (V - \Sigma)^*\}$$

$$\cup \{Sa^iS \rightarrow b : A_i \rightarrow c \text{ is in } P \cap (V - \Sigma) \times \Sigma\}.$$

By interpreting $Sa^iS \rightarrow b$ as $Sa^iS \rightarrow c$ for every $A_i \rightarrow c$ in $P \cap (V - \Sigma) \times \Sigma$ and by interpreting $S \rightarrow aSaS$ (in the case of F_2) as $S \rightarrow \$SaS$ (where $\$$ is a terminal symbol not used otherwise) we obtain interpretations F'_1 and F'_2 of F_1 and F_2 , respectively, such that $L(F'_1) = L$ and $L(F'_2) \cap \Sigma^* = \L , establishing our theorem. \square

We do not know whether the terminal alphabet in F_1 and F_2 can be reduced to one letter.

We conclude this section by showing that (despite the fact that there are context-free complete $(\{S, A\}, a)$ -forms by § 3) sequential $(\{S, A\}, a)$ -forms do not have great generative power.

THEOREM 5.8. *There exists no sequential $(\{S, A\}, a)$ -form F such that $\mathcal{L}_u(F)$ contains L , where $L = \{cb^n a^+ b^n c : n \geq 1\}$.*

Proof. Suppose that we have $L = L(G)$, $G = (V, \Sigma, P, S)$, $V = \{S\} \cup \{A_1, \dots, A_n\} \cup \Sigma$, $G \triangleleft_u F$ and without loss of generality that $S \rightarrow S$ and $A_i \rightarrow A_j$ do not occur in G . We may further assume that every A_i is infinite.

Observe that

$$P \subseteq \{S \rightarrow V^*\} \cup \bigcup_{i=1}^n \{A_i \rightarrow (\Sigma \cup \{A_i\})^*\}.$$

Now $S \Rightarrow_G^* cbabc$ must hold. Hence $S \Rightarrow^* \alpha S \beta$ and $\alpha \beta \Rightarrow^* x$ in Σ^+ implies $\alpha \beta = \lambda$.

Thus, S does not occur on the right side of any production. For every sufficiently large m we thus have $S = x \Rightarrow^* cb^m a^m b^m c$, where x contains some A in $\{A_1, \dots, A_m\}$. Since A generates an infinite language, $A \Rightarrow^* uAv$ holds for some uv in Σ^+ . Note that $\{u, v\} \subseteq a^* \cup b^*$. Since $|u|_b = |v|_b$ must hold, A is of either type (1) or (2):

- (1) $A \Rightarrow^* b^i A b^i \quad (i \geq 1)$,
- (2) $A \Rightarrow^* a^k A a^l \quad (k + l \geq 1)$.

Observe that A cannot be of both type (1) and (2) simultaneously, and that for every A of type (1) a terminal production containing an a must exist. However, to generate $cb^m a^m b^m c$, an A of type (1) must occur and can only be surrounded by terminals.

Thus we have

$$S \Rightarrow y_1 A y_2 \Rightarrow^* cb^m a^m b^m c,$$

where y_1, y_2 are in Σ^* and A is of type (1). Since A must produce a^m , A must also be of type (2) for sufficiently large m , a contradiction. \square

6. Goodness and badness. In this section we consider goodness and badness of context-free grammar forms under uniform interpretation. In analogy to [MSW2] or [HMO] we call a context-free grammar form F *good* if $\mathcal{L}_u(G) \subseteq \mathcal{L}_u(F)$ implies that for some $F' \triangleleft_u F$, $\mathcal{L}_u(G) = \mathcal{L}_u(F')$ holds. We call F *p-good* if $\mathcal{L}_u(G) \subseteq \mathcal{L}_u(F)$ and G λ -free (“propagating”) imply that for some $F' \triangleleft_u F$, $\mathcal{L}_u(G) = \mathcal{L}_u(F')$ holds. Accordingly, F is called *bad*, (*p-bad*), iff F is not good (not *p-good*).

Despite the fact that results on goodness and badness are often difficult to obtain, cf. the papers quoted above, we present a number of both positive (goodness) and negative (badness) results. In particular, we show that every separated (S, a) -form is *p-good* but *bad* (once more stressing the significance of the empty word in form theory; cf. [AiM]). We then show that many nonseparated (S, a) -forms are *bad*, and give a result allowing us to establish badness of certain forms based on an “isolation property”.

We begin with a technical lemma.

LEMMA 6.1. *Let F be a separated (S, a) -form and Σ an alphabet. If an infinite language L contains a word x in Σ^+ and all but a finite number of words of L contain symbols not in Σ then L is not in $\mathcal{L}_u(F)$.*

Proof. Assume $F' \triangleleft_u F$ and $L(F') = L$, $F' = (\{S\} \cup \Sigma', \Sigma', P, S)$. Since x is in L , $S \rightarrow z$ is in P for some z in Σ^+ . Hence L contains arbitrarily long words over Σ . \square

THEOREM 6.2. *Let $F = (\{S, a\}, \{a\}, P, S)$ be a separated form with $L(F)$ infinite. Then F is *p-good*.*

Proof. We first establish that F is *p-good*. Consider some G with $\mathcal{L}_u(G) \subseteq \mathcal{L}_u(F)$ and G λ -free. We may assume that G is reduced, that $L(G)$ is infinite and that, by Lemma 3.1, every nonterminal is infinite.

By analyzing $G = (V, \Sigma, Q, S)$ carefully we will be able to show that $G \triangleleft_u F$ holds, implying the *p-goodness* of F .

First observe that $Q = Q_1 \cup Q_2 \cup Q_3 \cup Q_4 \cup Q_5$ with $Q_1 \subseteq S \rightarrow \Sigma^+$, $Q_2 \subseteq S \rightarrow \Sigma^+$, $Q_3 \subseteq S \rightarrow \Sigma^+ S (\Sigma \cup \{S\})^* U (\Sigma \cup \{S\})^* S \Sigma^+$, $Q_4 \subseteq S \rightarrow V^* (V - \Sigma - \{S\}) V^*$, and $Q_5 \subseteq (V - \{S\}) \rightarrow V^*$. It is our first aim to establish $Q_3 = Q_4 = \emptyset$. Since G is reduced, this implies $Q_5 = \emptyset$. Once $Q = Q_1 \cup Q_2$ is established, our claim follows readily.

Suppose first that $Q_4 \neq \emptyset$. Then Q contains a production $S \rightarrow \alpha A \beta$ with $A \neq S$. Let Σ and Δ be disjoint terminal alphabets. By Lemma 3.2 there exists a $G' \triangleleft_u G$ such that $L(G') = \{x\}$, x in Σ^+ , $G' = (\{S\} \cup \Phi_1 \cup \Sigma, \Sigma, P', S)$. Consider a $G'' \triangleleft_u G$ with $G'' = (\{S\} \cup \Phi_2 \cup \Delta, \Delta, P'', S)$ such that (i) no production contains a symbol of Σ , (ii) the only production involving S is $S \rightarrow \bar{\alpha} A \bar{\beta}$ obtained from $S \rightarrow \alpha A \beta$ by replacing any terminal symbol in $\alpha\beta$ by some symbol of Δ and (iii) adding the production $S \rightarrow x$ would give a grammar generating an infinite language. We may assume that $\Phi_1 \cap \Phi_2 = \emptyset$. By Lemma 3.1 the grammar

$$G''' = (\{S\} \cup \Phi_1 \cup \Phi_2 \cup \Sigma \cup \Delta, \Sigma \cup \Delta, P' \cup P'', S)$$

can be assumed to be a uniform interpretation of G . $L(G''')$ contains the word x in Σ^+ but all other words contain symbols of Δ . Hence, by Lemma 6.1, $L(G''')$ is not in $\mathcal{L}_u(F)$, i.e. $\mathcal{L}_u(G) \not\subseteq \mathcal{L}_u(F)$. Thus $Q_4 \neq \emptyset$ is impossible.

Assume next that $Q_3 \neq \emptyset$. Then Q contains a production $S \rightarrow y S \alpha$ with y in Σ^+ and α in $(\Sigma \cup \{S\})^*$. Construct an interpretation G' of G , $G' \triangleleft_u G$ by taking into G' the production $S \rightarrow y S \alpha$ and productions necessary for an isolated derivation $S \Rightarrow^* x$, x in Δ^+ , $\Delta \cap \Sigma = \emptyset$. (Possible by Lemma 3.2). Then $L(G')$ contains a word in Δ^+ , but all other words contain symbols of Σ . By Lemma 6.1, $\mathcal{L}_u(G) \not\subseteq \mathcal{L}_u(F)$ and hence $Q_3 \neq \emptyset$ is also impossible.

We have now established $Q = Q_1 \cup Q_2$. For any pair of productions $S \rightarrow S^k$ in Q_1 and $S \rightarrow w$ in Q_2 consider $G' \triangleleft_u G$ where G' consists of only the productions $S \rightarrow S^k$ and $S \rightarrow w$; i.e., $L(G') = \{w^{(k-1)n+1} : n \geq 0\}$. Since $L(G') = L(F')$ must hold for some $F' \triangleleft_u F$, $S \rightarrow S^k$ must occur in F' and hence in F . Since $S \rightarrow w$ must occur in F' , F must contain a production $S \rightarrow a^l$, where $l = |w|$.

Thus, $Q_1 \subseteq P$ holds. Further, for every production $S \rightarrow x$ in Q_2 , P contains $S \rightarrow a^{|x|}$. Thus $G \triangleleft_u F$, establishing the p -goodness of F . \square

We now establish that a large class of (S, a) -forms is bad:

THEOREM 6.3. *Let F be a λ -free (S, a) -form containing the production $S \rightarrow a$, a production $S \rightarrow \alpha S \beta$ with $\alpha\beta \neq \lambda$ but not containing $S \rightarrow S^k$, $k = |\alpha S \beta|$. Then F is bad.*

Proof. Consider the grammar form G with productions $S \rightarrow S^k$ and $S \rightarrow a$. Clearly, $\mathcal{L}_u(G) \subseteq \mathcal{L}_u(F)$. But $F' \triangleleft_u F$ implies $\mathcal{L}_u(F') \neq \mathcal{L}_u(G)$. For suppose $\mathcal{L}_u(F') = \mathcal{L}_u(G)$, i.e., $\mathcal{L}_u(F') \subseteq \mathcal{L}_u(G)$. By the proof of Theorem 6.2, F' contains the production $S \rightarrow S^k$, a contradiction. \square

We believe that Theorem 6.3 can be strengthened considerably. Indeed, we conjecture that any (S, a) -form F generating an infinite language is bad.

To formulate the final theorem in this section, one further notion proves useful.

A grammar form F has the *isolation property* if for every integer k there exists an $F' \triangleleft_u F$ such that $L(F') = \{x\}$, where $|x| \geq k$.

THEOREM 6.4. *Suppose F is a grammar form, \mathcal{H} a closed class and M a finite subset of Σ^* , Σ an alphabet. If $\mathcal{L}_u(F) \supseteq \{\tau(L) : L \text{ is in } \mathcal{H}, \tau \text{ is } M\text{-restricted}\}$ and every infinite interpretation of F has the isolation property, then F is bad.*

Proof. By Theorem 5.1, there exists a separated (S, a) -form G such that $\mathcal{L}_u(G) = \{\tau(L) : L \text{ is in } \mathcal{H}, \tau \text{ is } M\text{-restricted}\}$ and hence $\mathcal{L}_u(G) \subseteq \mathcal{L}_u(F)$. Suppose for some $F' \triangleleft_u F$ we have $\mathcal{L}_u(G) = \mathcal{L}_u(F')$. Since, by assumption F' has the isolation property, G would have the isolation property, a contradiction. \square

We conclude this section with a sample application of Theorem 6.4.

Example. $F = (\{S, A, a\}, \{a\}, P, S)$ with $P = \{S \rightarrow A, A \rightarrow SS, A \rightarrow a\}$ is bad. Consider $\mathcal{H} = \{\{S\}, \{S^+\}\}$, $M = \{a\}$,

$$\mathcal{L}_u(F) \supseteq \{\tau(L) : L = \{S\} \text{ or } L = S^+, \tau(S) \subseteq \mu(a)\}.$$

F has the isolation property. Hence F is bad.

REFERENCES

- [AiM] W. AINHORN AND H. A. MAURER, *On ϵ -productions for terminals in EOL forms*, Discrete Appl. Math. 1 (1979), pp. 155–156.
- [CG] A. CREMERS AND S. GINSBURG, *Context-free grammar forms*, J. Comput. System Sci., 11 (1975), pp. 86–116.
- [GLMW] S. GINSBURG, B. L. LEONG, O. MAYER AND D. WOTSCHKE, *On strict interpretations of grammar forms*, Math. Systems Theory, 12 (1979), pp. 233–252.
- [HMO] H. HULE, H. A. MAURER AND TH. OTTMANN, *Good OL forms*, Acta Informatica, 9 (1978), pp. 345–353.
- [MPSW] H. A. MAURER, M. PENTTONEN, A. SALOMAA AND D. WOOD, *On non context-free grammar forms*, Math. Systems Theory, 12 (1979), pp. 297–324.
- [MSW1] H. A. MAURER, A. SALOMAA AND D. WOOD, *Uniform interpretations of L forms*, Inform. Control, 36 (1978), pp. 157–173.
- [MSW2] ———, *On good EOL forms*, this Journal, (1978), pp. 158–166.
- [MSW3] ———, *Context-dependent L forms*, Inform. Control, 42 (1979), pp. 97–118.
- [MSW4] ———, *Synchronized EOL forms under uniform interpretations*, Rev. Francaise d'Automat. Inform. Recher. Operationelle (RAIRO), to appear.
- [MSW5] ———, *Context-free grammar forms with strict interpretations*, J. Comput. System Sci., 21 (1980), pp. 110–135.
- [MSW6] ———, *Pure grammars*, Inform. Control, 44 (1980), pp. 47–72.
- [MSW7] ———, *Synchronized EOL forms*, Theoret. Comput. Sci., 12 (1980), pp. 135–159.
- [RS] G. ROZENBERG AND A. SALOMAA, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.
- [S] A. SALOMAA, *Formal Languages*, Academic Press, New York, 1973.
- [W] D. WOOD, *A survey of grammar and L form theory—1978*, in Proceedings of the Mathematical Foundations of Computer Science 1979, J. Becvar, ed., Lecture Notes in Computer Science 74, Springer-Verlag, New York, 1979, pp. 191–200.

ATTRIBUTE GRAMMARS AND MATHEMATICAL SEMANTICS*

BRIAN H. MAYOH†

Abstract. Attribute grammars and mathematical semantics are rival language definition methods. We show that any attribute grammar G has a reformulation $MS(G)$ within mathematical semantics. Most attribute grammars have properties that discipline the sets of equations the grammar gives to derivation trees. We list six such properties, and show that for a grammar G with one of these properties both $MS(G)$ and the compiler for G can be simplified. Because these compiler-friendly properties are of independent interest, the paper is written in such a way that the first and last sections do not depend on the other sections.

Keywords. mathematical semantics, attribute grammars, compiler properties

1. Introduction. Attribute grammars [9] give a systematic way of expressing such restrictions on a programming language as that variables must be declared before use or that the types of the two sides of an assignment statement must agree. In this paper we show that any attribute grammar can be given an equivalent and elegant formulation within the mathematical semantics of D. Scott and C. Strachey [16], [17]. This reformulation is of interest because of the widespread acceptance of the advantages of mathematical semantics for the description of real programming languages [1], [2], [5], [12], [13], [18], [19].

Before looking at the details of the reformulation, let us look at Knuth's simple example of an attribute grammar BIN for binary notation:

$$\begin{array}{ll}
 B \rightarrow 0 & v[B] = 0, \\
 B \rightarrow 1 & v[B] = 2^{c[B]}, \\
 L \rightarrow B & v[L] = v[B], c[B] = c[L], l[L] = 1, \\
 L_0 \rightarrow L_1 B_2 & v[L_0] = v[L_1] + v[B_2], c[B_2] = c[L_0], \\
 & c[L_1] = c[L_0] + 1, l[L_0] = l[L_1] + 1, \\
 N \rightarrow L & v[N] = v[L], c[L] = 0, \\
 N \rightarrow L_1 \cdot L_2 & v[N] = v[L_1] + v[L_2], c[L_1] = 0, \\
 & c[L_2] = -l[L_2],
 \end{array}$$

As explained in [9, p. 131] one can deduce from the grammar that the number 13.25 is the meaning of the expression 1101.01, because the equations given by the grammar can be ordered suitably. This difficulty with the ordering of equations does not arise when the grammar is reformulated within mathematical semantics:

$$\begin{array}{l}
 bv[0](c) = 0, \\
 bv[1](c) = 2^c, \\
 lv[B](c) = bv[B](c), \quad ll[B] = 1, \\
 lv[LB](c) = lv[L](c+1) + bv[B]c, \quad ll[LB] = ll[L] + 1, \\
 nv[L] = lv[L](0), \\
 nv[L_1 \cdot L_2] = lv[L_1](0) + lv[L_2](c_2) \text{ where } c_2 = -ll[L_2].
 \end{array}$$

* Received by the editors August 25, 1978, and in final revised form October 10, 1980.

† Department of Computer Science, University of Aarhus, Aarhus, Denmark.

Here we have the definition of four functions: bv for the synthesized attribute v of the symbol B , lv for the synthesized attribute v of the symbol L , ll for the synthesized attribute l of the symbol L , and nv for the synthesized attribute v of the symbol N . The first two functions have an argument in round brackets for an inherited attribute. All four functions have an argument in square brackets for derivation trees. In all our examples we will have an unambiguous grammar so we can use strings instead of trees for square bracket arguments. The deduction that the number 13.25 is the meaning of the string 1101.01 now becomes

$$\begin{aligned}
 nv[1101.01] &= lv[1101]0 + lv[01]c_2, \text{ where } c_2 = -ll[01] \\
 &= lv[1101]0 + lv[01](-2) \\
 &= (lv[110]1 + 1) + (lv[0](-1) + 0.25) \\
 &= (12 + 1) + (0 + 0.25) = 13.25.
 \end{aligned}$$

This example is too small to justify the claim that the reformulation of an attribute grammar within mathematical semantics is easier to understand because it only uses functions, whereas an attribute grammar uses attributes, functions and equations. The example in § 4 better illustrates the advantages of a reformulation $MS(G)$ within mathematical semantics of an attribute grammar G . There is always an $MS(G)$, equivalent to G (Theorem 1); if G is well defined, then $MS(G)$ does not use recursion (Theorem 2). Some attribute grammars have properties that discipline the set of equations the grammar gives to derivation trees. If a grammar G has one of these properties then Table 1 shows that both $MS(G)$ and the compiler for G can be simplified.

TABLE 1

Property	Compiler Simplification	$MS(G)$ Simplification
unordered	subtrees in arbitrary order	as compiler
ordered	subtrees from left to right	as compiler
reordered	subtrees in fixed order	as compiler
tangled	one pass	no splitting
benign	attributes in fixed order	determinate
well defined	—	no recursion

Because these compiler friendly properties are of independent interest this paper has been written in such a way that those unconcerned with mathematical semantics can omit all but the last section, and scan the earlier sections when they meet undefined notation.

2. Reformulation of an arbitrary attribute grammar. An *attribute structure* consists of:

- (1) disjoint sets $\mathcal{A}, \bar{A}, \bar{A}$;
- (2) for each X in \mathcal{A} , subsets $\underline{X} \subset \bar{A}$, and $\bar{X} \subset \bar{A}$;
- (3) for each a in $\bar{A} \cup \bar{A}$, a set V_a^0 .

The elements of \mathcal{A} are called *symbols*, the elements of \bar{A} are called *synthesized attributes*, and the elements of \bar{A} are called *inherited attributes*. For each X in \mathcal{A} we define

$$\begin{aligned}
 \text{SYN}^0[X], & \text{ the Cartesian product of } V_a^0 \text{ for } a \text{ in } X, \\
 \text{INH}^0[X], & \text{ the Cartesian product of } V_a^0 \text{ for } a \text{ in } \bar{X}.
 \end{aligned}$$

By convention $\text{SYN}^0[X]$ ($\text{INH}^0[X]$) has precisely one element if \bar{X} (\bar{X}) is empty. An attribute grammar consists of:

- (1) a context free grammar $(\mathcal{T}, \mathcal{H}, S, \mathcal{P})$; where the start symbol S does not occur on the right side of a production;
- (2) an attribute structure such that $\mathcal{A} = \mathcal{T} \cup \mathcal{H}$, S is empty, and \bar{X} is empty for X in \mathcal{T} ;
- (3) for every production $X_{p,0} \rightarrow X_{p,1} \cdots X_{p,-1}$ in \mathcal{P} we have a partial function $f_p^0: L_p^0 \rightarrow (R_p^0 \rightarrow R_p^0)$,

$$L_p^0 = \text{INH}^0(X_{p,0}) \times \text{SYN}^0(X_{p,1}) \times \text{SYN}^0(X_{p,2}) \times \cdots \times \text{SYN}^0(X_{p,-1}),$$

$$R_p^0 = \text{SYN}^0(X_{p,0}) \times \text{INH}^0(X_{p,1}) \times \text{INH}^0(X_{p,2}) \times \cdots \times \text{INH}^0(X_{p,-1}).$$

Here and later we avoid a sea of subscripts by using a convention due to B. Rosen in which $X_{p,-1}$, rather than X_{p,n_p-1} , is the last symbol of production p . In practice we usually have a function $q_p^0: L_p^0 \rightarrow R_p^0$ such that $f_p^0(l)(r) = q_p^0(l)$ for all l in L_p^0 and r in R_p^0 , and we say that an attribute grammar is in normal form if we have such a function for each production.

Example. For the attribute grammar BIN we have Table 2.

TABLE 2

syntactic rule	semantic rule
$B \rightarrow 0$	$B = \{v\}, \quad \bar{B} = \{c\}, \quad L = \{v, l\}, \quad \bar{L} = \{c\}, \quad N = \{v\}, \quad \bar{N} = \{\cdot\}$ $\text{SYN}^0(B) = V_v^0, \quad \text{SYN}^0(L) = V_v^0 \times v_l^0, \quad \text{SYN}^0(N) = V_v^0,$ $\text{INH}^0(B) = V_c^0, \quad \text{INH}^0(L) = V_c^0, \quad \text{INH}^0(N) = \{\cdot\}.$
$B \rightarrow 1$	$f_a^0: V_c^0 \rightarrow (V_v^0 \rightarrow V_v^0)$ $f_a^0(c[B])(v[B]) = 0$
$L \rightarrow B$	$f_b^0: V_c^0 \rightarrow (V_v^0 \rightarrow V_v^0)$ $f_b^0(c[B])(v[B]) = 2^{c[B]}$
$L_0 \rightarrow L_1 B_2$	$f_c^0: V_c^0 \times V_v^0 \rightarrow (V_v^0 \times V_l^0 \times V_c^0 \rightarrow V_v^0 \times V_l^0 \times V_c^0)$ $f_c^0(c[L], v[B])(v[L], l[L], c[B]) = (v[B], 1, c[L])$
$N \rightarrow L$	$f_d^0: V_v^0 \times V_l^0 \rightarrow (V_v^0 \times V_c^0 \rightarrow V_v^0 \times V_c^0)$ $f_d^0(v[L], l[L])(v[N], c[L]) = (v[L], 0)$
$N \rightarrow L_1 \cdot L_2$	$f_e^0: V_v^0 \times V_l^0 \times V_v^0 \times V_l^0 \rightarrow (V_v^0 \times V_c^0 \times V_c^0 \rightarrow V_v^0 \times V_c^0 \times V_c^0)$ $f_e^0(v[L_1], l[L_1], v[L_2], l[L_2])(v[N], c[L_1], c[L_2])$ $= (v[L_1] + v[L_2], 0, -l[L_2])$

Note that our reformulation of BIN borrows notation like $v[B]$ for attribute values from the original formulation, and it shows that the attribute grammar is in normal form. The differences between our definition of attribute grammar and that in [9] are minor and inessential, but they pave the way to the lattices and functions of mathematical semantics. For each symbol X in $\mathcal{H} \cup \mathcal{T}$ the productions of the grammar give

$\text{DOM}^0(X)$, the set of derivation trees that can be generated from X . As described in [16], [17] one can convert the sets $V_a^0, \text{SYN}^0(X), \text{INH}^0(X), \text{DOM}^0(X)$ by adding a bottom element \perp and a top element \top , to lattices $V_a, \text{SYN}(X), \text{INH}(X), \text{DOM}(X)$, and one can form a lattice of continuous functions

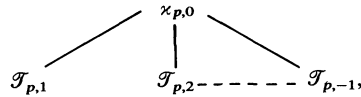
$$\text{CONT}(X) = \text{DOM}(X) \rightarrow (\text{INH}(X) \rightarrow \text{SYN}(X)).$$

A reformulation of a grammar in mathematical semantics will define precisely one element of $\text{CONT}(S)$.

Convention. When specifying a function x over $\text{DOM}(X)$, we may do so by a set of equations

$$x[X_{p,1} \cdots X_{p,-1}] = \cdots,$$

with one equation for each production p with $X_{p,0} = X$. We use $x[X_{p,1} \cdots X_{p,-1}]$ as a convenient way of writing the value of X on a derivation tree of the form



where $\mathcal{T}_{p,1} \cdots \mathcal{T}_{p,-1}$ are derivation trees with $X_{p,1} \cdots X_{p,-1}$ at their roots. Because $\text{DOM}(X)$ is the lattice sum of $\text{DOM}(X_{p,1}) \times \cdots \times \text{DOM}(X_{p,-1})$ for p such that $X_{p,0} = X$, our sets of equations do determine functions over $\text{DOM}(X)$. The equation pairs for bv, ll, lv, nv in § 1 determine functions

$$\begin{aligned} bv &: \text{DOM}(B) \rightarrow V_c \rightarrow V_v, \\ ll &: \text{DOM}(L) \rightarrow V_b, \\ lv &: \text{DOM}(L) \rightarrow V_c \rightarrow V_c, \\ nv &: \text{DOM}(N) \rightarrow V_v. \end{aligned}$$

When specifying these functions we used the convention that parentheses can be omitted if this does not lead to confusion. This convention usually allows us to omit parentheses around empty sets of arguments.

DEFINITION 1. Let $G = (\mathcal{T}, \mathcal{H}, \mathcal{S}, \mathcal{P})$ be an attribute grammar.

An *assignment* to a derivation tree π of G is a pair of functions (sy, in) from nodes of π to attribute values such that:

$$\text{sy}(u) \in \text{SYN}(X_u) \quad \text{and} \quad \text{in}(u) \in \text{INH}(X_u),$$

where X_u is the symbol at node u . The assignment is said to be *complete* if for all nodes u we have

$$\text{sy}(u) \in \text{SYN}^0(X_u) \quad \text{and} \quad \text{in}(u) \in \text{INH}^0(X_u).$$

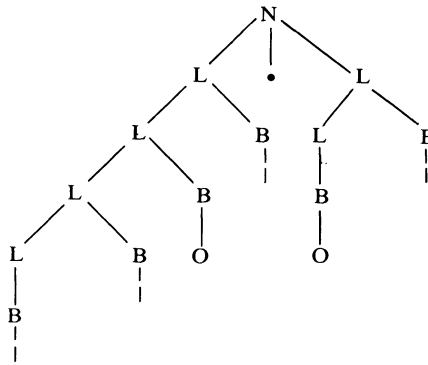
For every complete assignment (sy, in) we can define $\text{Next}(\text{sy}, \text{in})$ as the assignment (sy', in') given by

$$\begin{aligned} (*) \quad & (\text{sy}'(u_0), \text{in}'(u_1) \cdots \text{in}'(u_{-1})) \\ & = f_p^0(\text{in}(u_0), \text{sy}(u_1) \cdots \text{sy}(u_{-1}))(\text{sy}(u_0), \text{in}(u_1) \cdots \text{in}(u_{-1})) \end{aligned}$$

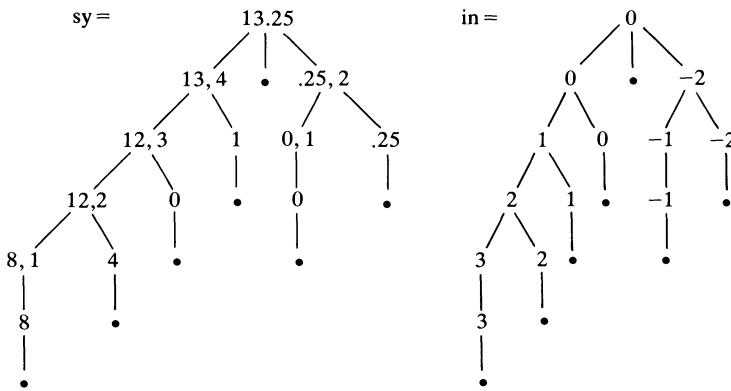
for each application $u_0 \rightarrow u_1 \cdots u_{-1}$ of the production $X_{p,0} \rightarrow X_{p,1} \cdots X_{p,-1}$ in the tree π .

The assignment (sy, in) *fits* π if $(\text{sy}, \text{in}) = \text{Next}(\text{sy}, \text{in})$. The grammar G *assigns* w to π if $w = \text{sy}(\text{root of } \pi)$ for every complete assignment (sy, in) that fits π .

Example. The derivation tree



for the grammar BIN fits the assignment



An assignment τ to a derivation tree π gives a value $\tau(u, \alpha)$ to each attribute α of each node u . We say π has a computation sequence if there is a sequence $(u_1, \alpha_1) \cdots (u_n, \alpha_n)$ such that

- (1) each u_j is a node of π ;
- (2) each α_j is an attribute of the symbol at the node u_j ;
- (3) the pair (u, α) occurs in the sequence for each attribute a of each node u ;
- (4) if τ and τ' are complete assignments such that

$$\tau(u_1, \alpha_1) = \tau'(u_1, \alpha_1) \cdots \tau(u_{j-1}, \alpha_{j-1}) = \tau'(u_{j-1}, \alpha_{j-1}),$$

then $\text{Next}(\tau)(u_j, \alpha_j) = \text{Next}(\tau')(u_j, \alpha_j) \neq \perp$.

LEMMA. If the derivation tree π has a computation sequence, then there is precisely one complete assignment that fits π .

Proof. We define an assignment τ_w by:

if τ is any complete assignment such that

$$\tau(u_1, \alpha_1) = \tau_w(u_1, \alpha_1) \cdots \tau(u_{j-1}, \alpha_{j-1}) = \tau_w(u_{j-1}, \alpha_{j-1}),$$

then $\tau_w(u_j, \alpha_j) = \text{Next}(\tau)(u_j, \alpha_j)$.

A simple induction argument using requirement (4) in the definition of computation

sequences gives

$$\tau_w(u_j, \alpha_j) \neq \perp \text{ does not depend on the choice of } \tau.$$

This implies that τ_w is a complete assignment.

If we take τ_w as τ in the definition of $\tau_w(u_j, \alpha_j)$ we get $\tau_w(u_j, \alpha_j) = \text{Next}(\tau_w)(u_j, \alpha_j)$, so the complete assignment τ_0 fits the tree π . Suppose τ_1 is a complete assignment that fits π . If we have $\tau_w(u_1, \alpha_1) = \tau_1(u_1, \alpha_1) \cdots \tau_w(u_{j-1}, \alpha_{j-1}) = \tau_1(u_{j-1}, \alpha_{j-1})$ requirement (4) gives $\text{Next}(\tau_w)(u_j, \alpha_j) = \text{Next}(\tau_1)(u_j, \alpha_j)$ and $\tau_w(u_j, \alpha_j) = \tau_1(u_j, \alpha_j)$ follows from $\tau_w = \text{Next}(\tau_w) \& \tau_1 = \text{Next}(\tau_1)$. We infer that $\tau_w = \tau_1$.

THEOREM 1. *For any attribute grammar G with start symbol S we can define a function s in $\text{CONT}(S)$ such that for any derivation tree π we have*

- (a) *if $s[\pi] = w \in \text{SYN}^0(s)$, then G assigns w to π ;*
- (b) *if π has a computation sequence and G assigns w to π , then $s[\pi] = w$.*

Proof.

(a) If we extend the functions f_p^0 to continuous functions $f_p: L_p \rightarrow (R_p \rightarrow R_p)$, and we use f_p instead of f_p^0 in the equations (*) in the definition, our function Next becomes a continuous function from assignments to assignments. For any derivation tree π there is a least assignment τ satisfying $\tau = \text{Next}(\tau)$. If (sy, in) is this least assignment and we take $\text{sy}(\text{root})$ as the value of $s[\pi]$, then “ (sy, in) is less than every assignment that fits π ” gives part (a) of our theorem.

- (b) Let us agree on the following continuous extension of f_p^0 :

$$f_p(l)(r) = \text{greatest lower bound of } f_p^0(l')(r') \text{ for } l < l', r < r'$$

and look at the definition of τ_w in the proof of the lemma. Because of the way we have extended f_p^0 we have

$$\tau_w(u_j, \alpha_j) = \text{Next}(\tau)(u_j, \alpha_j)$$

for every assignment satisfying

$$\tau_w(u_1, \alpha_1) = \tau(u_1, \alpha_1) \cdots \tau_w(u_{j-1}, \alpha_{j-1}) = \tau(u_{j-1}, \alpha_{j-1}).$$

Suppose we define $\tau_j = \text{Next}(\tau_{j-1})$ and take τ_0 as the assignment that gives \perp to all attributes of all nodes in a derivation tree. If we have

$$\tau_w(u_1, \alpha_1) = \tau_{j-1}(u_j, \alpha_j) \cdots \tau_w(u_{j-1}, \alpha_{j-1}) = \tau_{j-1}(u_{j-1}, \alpha_{j-1}),$$

we also have $\tau_w(u_j, \alpha_j) = \text{Next}(\tau_{j-1})(u_j, \alpha_j) = \tau_j(u_j, \alpha_j)$.

Since Next is continuous we also have

$$\tau_w(u_1, \alpha_1) = \tau_j(u_1, \alpha_j) \cdots \tau_w(u_{j-1}, \alpha_{j-1}) = \tau_j(u_{j-1}, \alpha_{j-1}),$$

and induction gives

$$\tau_w(u_1, \alpha_1) = \tau_1(u_1, \alpha_1) \cdots \tau_w(u_r, \alpha_r) = \tau_r(u_r, \alpha_r),$$

so τ_w agrees with the least assignment $\tau_0 \cup \tau_1 \cup \cdots$.

If the grammar G assigns w to π , then the lemma ensures that w is the value of the synthesized attributes in the complete assignment τ_w . By definition $s[\pi]$ is the value of these attributes in the least assignment. These two values must be the same.

Comment. So far we have only considered assignments to derivation trees with the start symbol of an attribute grammar at their roots. For any derivation tree π with root symbol $X \in \mathcal{H} \cup \mathcal{T}$ we can extend definition 1 and the proof of Theorem 1 to give a

function $x[\tau]$ in $\text{INH}(X) \rightarrow \text{SYN}(X)$. These functions are defined by the equations

$$\begin{aligned} \text{sy}_0 &= x_{p,0}[X_{p,1} \cdots X_{p,-1}]\text{in}_0, \\ \text{sy}_1 &= x_{p,1}[X_{p,1}]\text{in}_1 \cdots \text{sy}_{-1} = x_{p,-1}[X_{p,-1}]\text{in}_{-1} \\ (\text{sy}_0, \text{in}_0 \cdots \text{in}_{-1}) &= f_p(\text{in}_0, \text{sy}_1 \cdots \text{sy}_{-1})(\text{sy}_0, \text{in}_1 \cdots \text{in}_{-1}). \end{aligned}$$

This was proved in an earlier version of this paper but the details are so similar to those for the independent result in [4] that they are omitted here. In a suitable specification language, the unique function $x_{p,0}$ given by equations (*) is

$$(**) \quad x_{p,0}[X_{p,1} \cdots X_{p,-1}]\text{in}_0 = YH \downarrow 1,$$

where $H(\text{sy}_0, \text{in}_1 \cdots \text{in}_{-1}) = f_p(\text{in}_0, x_{p,1}[X_{p,1}]\text{in}_1, \cdots, x_{p,-1}[X_{p,-1}]\text{in}_{-1})(\text{sy}_0, \text{in}_1 \cdots \text{in}_{-1})$.

Here Y is the fixed point operator, $\downarrow 1$ selects the first component of a list, and we include the trivial functions $x_{p,i}$ for terminal symbols $X_{p,i}$. In practice such trivial functions can be omitted.

Note that $x_{p,0}$ is a member of $\text{CONT}(X_{p,0})$, and s in Theorem 1 is the least upper bound in $\text{CONT}(S)$ of the functions $x_{p,0}$ for the productions with $S = X_{p,0}$.

Applying our construction to the grammar BIN gives the somewhat obscure

$$\begin{aligned} b[0]c &= YH \downarrow 1 && \text{where } H(v) = 0, \\ b[1]c &= YH \downarrow 1 && \text{where } H(v) = 2^c, \text{ where } \Gamma_0 = c v, \quad \Gamma_1 = c \rightarrow v, \\ l[B]c &= YH \downarrow 1 && \text{where } H((v, l), \text{in}_1) = ((b[B]\text{in}_1, l), c), \\ l[LB]c &= YH \downarrow 1 && \text{where } H((v, l), \text{in}_1, \text{in}_2) \\ &&& = ((l[L]\text{in}_1 \downarrow 1 + b[B]\text{in}_2, l[L]\text{in}_1 \downarrow 2 + 1), c + 1, c), \\ n[L] &= YH \downarrow 1 && \text{where } H(v, \text{in}_1) = (l[L]\text{in}_1 \downarrow 1, 0), \\ n[L_1 \cdot L_2] &= YH \downarrow 1 && \text{where } H(v, \text{in}_1, \text{in}_2) \\ &&& = (l[L_1]\text{in}_1 \downarrow 1 + l[L_2] \downarrow 1, 0, -l[L_2]\text{in}_2 \downarrow 2). \end{aligned}$$

Straightforward fixed point elimination gives

$$\begin{aligned} b[0]c &= 0, \\ b[1]c &= 2^c, \\ l[B]c &= (b[B]c, 1), \\ l[LB]c &= (v_1 + b[B]c, l_1 + 1) \text{ where } (v_1, l_1) = l[L](c + 1), \\ n[L] &= l[L]0 \downarrow 1, \\ n[L_1 \cdot L_2] &= v_1 + v_2 \text{ where } (v_1, l_1) = l[L_1]0 \text{ and } (v_2, l_2) = l[L_2](-l_2). \end{aligned}$$

Replacing the last line by

$$\text{and } (v_2, l_2) = YH \text{ where } H(v, l) = l[L_2](-l_1)$$

makes the recursion explicit.

3. Reformulation of a well-defined attribute grammar. In this section we show that recursion is not needed when a well-defined attribute grammar is reformulated within mathematical semantics.

DEFINITION 2. An attribute grammar $G = (\mathcal{T}, \mathcal{H}, \mathcal{S}, \mathcal{P})$ is *well defined*, if the test in [9] shows G is not circular.

THEOREM 2. For any well-defined attribute grammar G with start symbol S we can define a function s in $\text{CONT}(S)$ such that for any derivation tree π we have

- (a) the specification of s does not use recursion or the fixed point operator Y ;
- (b) G assigns w to $\pi \Leftrightarrow s[\pi] = w$;
- (c) $s[\pi] \neq \perp$.

Proof. The algorithm for testing whether an attribute grammar is well defined [9, correction] generates a finite set of directed graphs. These graphs are of three kinds. For each element X in $\mathcal{H} \cup \mathcal{T}$ we have a set of *symbol graphs* $\text{SYM}(X)$ showing how the synthesized attributes may depend on the inherited attributes of X . For each production $X_{p,0} \rightarrow X_{p,1} \cdots X_{p,-1}$ we have:

(1) a *production graph* D_p with arrows to the synthesized attributes of $X_{p,0}$ and the inherited attributes of $X_{p,1} \cdots X_{p,-1}$ from the zero, one or more attributes they depend upon;

(2) a set $\text{COMP}(p)$ of *composite graphs* of the form $D_p[Q(1) \cdots Q(-1)]$ where $Q(1) \in \text{SYM}(X_{p,1}) \cdots Q(-1) \in \text{SYM}(X_{p,-1})$. Knuth's test for circularity generates the composite graphs and $\text{SYM}(X)$ for X in \mathcal{H} from the production graphs and $\text{SYM}(X)$ for X in \mathcal{T} . If any composite graph contains a cycle, then our attribute grammar G is not well defined; otherwise these graphs tell us how to replace (** in the proof of Theorem 1 by a specification with no implicit or explicit recursion. As we shall see in the next section, this reformulation is particularly simple if for every p the union of the graphs in $\text{COMP}(p)$ contains no cycle. Even although this simplification is advocated in [4] and almost always possible in practice, we have to treat the general case if we are to prove the theorem. The nondeterminism that plagues very general attribute grammars then enters in the form of joins in function lattices. For each symbol X in $\mathcal{H} \cup \mathcal{T}$, for each graph Γ in $\text{SYM}(X)$, and each synthesized attribute α in X , we introduce a function

$$\text{symbol}(\Gamma, \alpha) : \text{DOM}(X) \rightarrow W(\Gamma, \alpha) \rightarrow V_\alpha,$$

where $W(\Gamma, \alpha)$ is the subset of $\text{INH}(X)$ given by the arrows going to the node for α in Γ . If $\alpha_1 \cdots \alpha_n$ are all the attributes of the start symbol S , this gives functions

$$\text{symbol}(\Gamma, \alpha_1) : \text{DOM}(S) \rightarrow V_{\alpha_1} \cdots \text{symbol}(\Gamma, \alpha_n) : \text{DOM}(S) \rightarrow V_{\alpha_n}$$

for each Γ in $\text{SYM}(S)$. The product of these functions is a member of $\text{CONT}(S)$, and the least upper bound (= join) of these products will be the function s of Theorem 1. For each production $X_{p,0} \rightarrow X_{p,1}, \cdots, X_{p,-1}$ there are a finite number of ways of choosing graphs $Q(0)Q(1) \cdots Q(-1)$ such that $Q(j)$ is in $\text{SYM}(X_{p,j})$ for $j = 0, 1, \cdots, -1$ and

- (***) there is an arc from α to α' in $Q(0)$ if and only if there is a directed path from $(X_{p,0}, \alpha)$ to $(X_{p,0}, \alpha')$ in $D_p[Q(1), \cdots, Q(-1)]$.

For each synthesized attribute α in $X_{p,0}$ and each such choice of $Q = (Q(0), Q(1), \cdots, Q(-1))$, we introduce a function

$$\text{rule}(Q, \alpha) : \text{DOM}(X_{p,0}) \rightarrow W(Q(0), \alpha) \rightarrow V_\alpha.$$

Because the graph $D_p[Q(1), \cdots, Q(-1)]$ contains no cycle, it gives a function rule (Q, α) that does not require recursion or the fixed point operator Y . Using these functions we complete the definition of s by

$$\text{symbol}(\Gamma, \alpha) = \text{the join of rule}(Q, \alpha) \text{ such that } Q(0) = \Gamma.$$

We have now proved (a); if we can show that every derivation tree in a well-defined grammar has a computation sequence, Theorem 1 and the lemma will give (b) and (c).

There is one and only one way of assigning symbol graphs and composite graphs to nodes of a derivation tree π so that (***) is satisfied: there is a unique choice of symbol graph for each terminal, and, working up the tree, one and only one choice of Q for each application of a production. The composite graphs partially order the attributes at the nodes of the tree because no composite graph contains a cycle. let

$$(u_1, \alpha_1) \cdots (u_n, \alpha_n)$$

be an embedding of this partially ordered set in a linear order. If τ is a complete assignment to the derivation tree, the value of $\text{Next}(\tau)(u_j, \alpha_j)$ is given by rule (Q, α_j) for the Q at node u_j and this only depends on

$$\tau(u_1, \alpha_1) \cdots \tau(u_{j-1}, \alpha_{j-1}).$$

Our linearly ordered set $(u_1, \alpha_1) \cdots (u_n, \alpha_n)$ is a computation sequence.

Comment. In a well-defined grammar we have:

- (a) every derivation tree has a computation sequence;
- (b) for each derivation tree π there is precisely one complete assignment that fits π .

Our lemma shows (a) implies (b); the grammar

$$S \rightarrow 9 \quad f(\text{sy}_0, c, d) = (c, \text{if } d \text{ even then } 1 \text{ else } d, c - 1)$$

shows (b) does not imply (a) because it is circular but there is precisely one complete assignment that fits its only derivation tree, $\text{sy}_0 = 1, c = 1, d = 0$. There are grammars satisfying (a) that are not well defined, but they must have useless productions [9].

Examples (continued). The circularity test for our grammar BIN generates

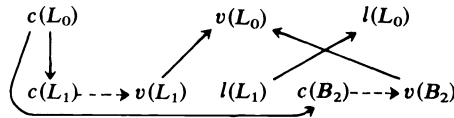
$$\text{SYM}(B) = (\Gamma_0, \Gamma_1)$$

$$\text{SYM}(L) = (\Gamma_2, \Gamma_3) \quad \text{where } \Gamma_2 = c v l, \quad \Gamma_3 = c \rightarrow v l.$$

We see that we must introduce functions

$$\begin{aligned} \text{symbol } (\Gamma_0, v) : \text{DOM}(B) \rightarrow V_v, & \quad \text{symbol } (\Gamma_1, v) : \text{DOM}(B) \rightarrow V_c \rightarrow V_v, \\ \text{symbol } (\Gamma_2, v) : \text{DOM}(L) \rightarrow V_v, & \quad \text{symbol } (\Gamma_3, v) : \text{DOM}(L) \rightarrow V_c \rightarrow V_v, \\ \text{symbol } (\Gamma_2, l) : \text{DOM}(L) \rightarrow V_b, & \quad \text{symbol } (\Gamma_3, l) : \text{DOM}(L) \rightarrow V_l. \end{aligned}$$

The test also generates four graphs in $\text{COMP}(L_0 \rightarrow L_1 B_2)$, the composite graphs given by including or excluding broken arrows in the graph



For the production $L_0 \rightarrow L_1 B_2$ there are four ways of choosing graphs $Q(0), Q(1), Q(2)$ that satisfy (***):

	Q_1	Q_2	Q_3	Q_4
$Q(0)$	Γ_2	Γ_3	Γ_3	Γ_3
$Q(1)$	Γ_2	Γ_2	Γ_3	Γ_3
$Q(2)$	Γ_0	Γ_1	Γ_0	Γ_1

Since L has two synthesized attributes, we have eight rule-functions and four symbol-functions:

$$\begin{aligned}
\text{rule } (Q_1, v)[L_1B_2] &= \text{symbol } (\Gamma_2, v)[L_1] + \text{symbol } (\Gamma_0, v)[B_2], \\
\text{rule } (Q_2, v)[L_1B_2]c &= \text{symbol } (\Gamma_3, v)[L_1] + \text{symbol } (\Gamma_1, v)[B_2]c, \\
\text{rule } (Q_3, v)[L_1B_2]c &= \text{symbol } (\Gamma_3, v)[L_1](c+1) + \text{symbol } (\Gamma_0, v)[B_2], \\
\text{rule } (Q_4, v)[L_1B_2]c &= \text{symbol } (\Gamma_3, v)[L_1](c+1) + \text{symbol } (\Gamma_1, v)[B_2]c, \\
\text{symbol } (\Gamma_2, v)[L_1B_2] &= \text{rule } (Q_1, v)[L_1B_2], \\
\text{symbol } (\Gamma_3, v)[L_1B_2]c &= \text{rule } (Q_2, v)[L_1B_2]c \cup \text{rule } (Q_3, v)[L_1B_2]c \\
&\quad \cup \text{rule } (Q_4, v)[L_1B_2]c, \\
\text{rule } (Q_1, l)[L_1B_2] &= \text{symbol } (\Gamma_2, l)[L_1] + 1, \\
\text{rule } (Q_2, l)[L_1B_2] &= \text{symbol } (\Gamma_2, l)[L_1] + 1, \\
\text{rule } (Q_3, l)[L_1B_2] &= \text{symbol } (\Gamma_3, l)[L_1] + 1, \\
\text{rule } (Q_4, l)[L_1B_2] &= \text{symbol } (\Gamma_3, l)[L_1] + 1, \\
\text{symbol } (\Gamma_2, l)[L_1B_2] &= \text{rule } (Q_1, l)[L_1B_2], \\
\text{symbol } (\Gamma_3, l)[L_1B_2] &= \text{rule } (Q_2, l)[L_1B_2] \cup \text{rule } (Q_3, l)[L_1B_2] \\
&\quad \cup \text{rule } (Q_4, l)[L_1B_2],
\end{aligned}$$

In the next section we show that the above twelve equations can be replaced by

$$\begin{aligned}
\text{symbol } (\Gamma_3, v)[L_1B_2]c &= \text{symbol } (\Gamma_3, v)[L_1](c+1) \\
&\quad + \text{symbol } (\Gamma_1, v)[B_2](c), \\
\text{symbol } (\Gamma_3, l)[L_1B_2] &= \text{symbol } (\Gamma_3, l)[L_1] + 1;
\end{aligned}$$

clearly these are unsugared versions of our original equations

$$\begin{aligned}
lv[LB]c &= lv[L](c+1) + lv[B]c, \\
ll[LB] &= ll[L] + 1.
\end{aligned}$$

4. Other desirable properties. Well-defined attribute grammars can have other desirable properties that simplify the task of making a compiler for the language they generate. In this section we introduce six such properties and show how the reformulation within mathematical semantics of an attribute grammar G becomes simpler when G has one of these properties.

DEFINITION 3. Let D_p be the graph introduced in [9] for a production $p: X_{p0} \rightarrow X_{p1} \cdots X_{p,-1}$ in an attribute grammar. Let W_p be the subgraph of D_p formed by deleting every arrow from an inherited attribute of X_{p0} and every arrow to a synthesized attribute from an attribute of $X_{p,1} \cdots X_{p,-1}$. We say that the production p is

<i>unordered</i>	if W_p is empty;
<i>ordered</i>	if each arrow in W_p from an attribute of $X_{p,j}$ to an attribute of $X_{p,k}$ satisfies $0 < j < k$;
<i>reordered</i>	if there is a permutation f of $1, 2, \dots, n(p) - 1$ such that each arrow in W_p from an attribute of $X_{p,j}$ to an attribute of $X_{p,k}$ satisfies $j \neq 0 \wedge f(j) < f(k)$;

- tangled* if there are no cycles in the graph $D_p(\text{ALL}(X_{p,1}) \cdots \text{ALL}(X_{p,-1}))$ where $\text{ALL}(X)$ is the graph with an arrow from every inherited attribute of X to every synthesized attribute of X ;
- benign* if there are no cycles in the graph $D_p(\text{SOME}(X_{p,1}) \cdots \text{SOME}(X_{p,-1}))$ where $\text{SOME}(X)$ is the union of the graphs in $\text{SYM}(X)$;
- well defined* if there are no cycles in any of the graphs $D_p(Q(1), \cdots Q(-1))$ for $Q(1) \in \text{SYM}(X_{p,1}) \cdots Q(-1) \in \text{SYM}(X_{p,-1})$.

Remark. A grammar is not circular if all its productions are well defined. If a production has one of the other properties we have defined, then the order of evaluating the attributes of the symbols on the right side of the production (right symbols) is simplified. For a benign production this order does not depend on the productions used to expand the right symbols. For a tangled production this order can be such that all the inherited attributes of a right symbol occur before any of its synthesized attributes. For an ordered (reordered, unordered) production this order can be such that one can evaluate all attributes of a right symbol $X_{p,i}$ before evaluating any attribute of the next right symbol (the symbol following $X_{p,i}$ in some permutation of the right side of the production, any other right symbol). Clearly these distinctions are significant when designing a compiler for the language given by an attribute grammar.

Example. Consider the attribute grammar CONTRIVED, presented in Table 3. The broken arrows in a production graph D_p are those that are not in W_p . We see that the productions $U \rightarrow 5$, $U \rightarrow 7$, $S \rightarrow O$, $O \rightarrow X$, $X \rightarrow 9$ are unordered, the production $O \rightarrow RT$ is ordered, and the production $R \rightarrow TO$ is reordered.

THEOREM 3.

- (a) *unordered* \rightarrow *ordered* \rightarrow *reordered* \rightarrow *tangled* \rightarrow *benign* \rightarrow *well defined*.
- (b) *The chain of implications in (a) is proper.*
- (c) *A production $X_{p,0} \rightarrow X_{p,1} \cdots X_{p,-1}$ is tangled if and only if the attributes of $X_{p,0}, X_{p,1} \cdots X_{p,-1}$ can be ordered in such a way that every inherited attribute of $X_{p,i}$ can be evaluated before a synthesized attribute of $X_{p,i}$ is evaluated.*

Proof.

(a) The first, second, fourth and fifth implications follow directly from the definitions. For the third implication assume that production $X_{p,0} \rightarrow X_{p,1} \cdots X_{p,-1}$ is reordered and $D_p(\text{ALL}(X_{p,1}) \cdots \text{ALL}(X_{p,-1}))$ has a cycle. This cycle cannot pass through an inherited attribute of $X_{p,0}$ because there are no arrows *to* these attributes; it cannot pass through a synthesized attribute of $X_{p,0}$ because there are no arrows *from* these attributes in a reordered production; it cannot use an arrow of W_p because f increases along such an arrow and f is constant on the arrows of $\text{ALL}(X)$. Since $\text{ALL}(X)$ has no cycle, our assumption is absurd.

(b) Consider the grammar CONTRIVED. The production $O \rightarrow RT$ is ordered, but not unordered; the production $R \rightarrow TO$ is reordered, but not ordered; the production $T \rightarrow BX$ is tangled, but not reordered; the production $B \rightarrow X$ is not tangled.

As the production graph D_i is the only symbol graph in $\text{SYM}(X)$ it is also the graph $\text{SOME}(X)$. Since $D_k[D_i]$ has no cycles, the production $B \rightarrow X$ is benign. Now consider the attribute grammar given by the first three productions of CONTRIVED. The production $S \rightarrow U$ is well defined because there are no cycles in $D_a[D_b]$ and $D_a[D_c]$, but it is not benign because there is a cycle in $D_a[D_b \cup D_c]$.

(c \rightarrow) If the composite graph $D_p(\text{ALL}(X_{p,1}) \cdots \text{ALL}(X_{p,-1}))$ has no cycles, it is the graph of a partial order on the attributes of $X_{p,0}, X_{p,1} \cdots X_{p,-1}$. Because any partial order can be embedded in a linear order we can evaluate attributes in an order satisfying:

TABLE 3

$\bar{S} = \{\bar{s}\}$, $\bar{U} = \{\bar{u}1, \bar{u}2\}$, $\bar{O} = \{\bar{o}\}$, $\bar{R} = \{\bar{r}\}$, $\bar{T} = \{\bar{t}\}$, $\bar{B} = \{\bar{b}1, \bar{b}2\}$, $\bar{X} = \{\bar{x}1, \bar{x}2\}$,
 $\underline{S} = \{\underline{s}\}$, $\underline{U} = \{\underline{u}1, \underline{u}2\}$, $\underline{O} = \{\underline{o}\}$, $\underline{R} = \{\underline{r}\}$, $\underline{T} = \{\underline{t}\}$, $\underline{B} = \{\underline{b}\}$, $\underline{X} = \{\underline{x}1, \underline{x}2\}$

name	production	semantic rule	production graph
a	$S \rightarrow U$	$f_a(\underline{u}1, \underline{u}2)(\bar{s}, \bar{u}1, \bar{u}2)$ $= (\underline{u}1 + \underline{u}2, \underline{u}2, \underline{u}1)$	
b	$U \rightarrow 5$	$f_b(\bar{u}1, \bar{u}2)(\underline{u}1, \underline{u}2)$ $= (23 \times \bar{u}1, 5)$	
c	$U \rightarrow 7$	$f_c(\bar{u}1, \bar{u}2)(\underline{u}1, \underline{u}2)$ $= (7, 29 \times \bar{u}2)$	
d	$S \rightarrow O$	$f_d(\underline{o})(\bar{s}, \bar{o})$ $= (3 \times \underline{o}, 2)$	
e	$O \rightarrow X$	$f_e(\bar{o}, \underline{x}1, \underline{x}2)(\underline{o}, \bar{x}1, \bar{x}2)$ $= (\underline{x}1 / \underline{x}2, \bar{o}, \bar{o})$	
f	$O \rightarrow RT$	$f_f(\bar{o}, \underline{r}, \underline{t})(\underline{o}, \bar{r}, \bar{t})$ $= (19 \times \underline{t}, 31 \times \bar{o}, 37 \times \underline{r})$	
g	$R \rightarrow TO$	$f_g(\bar{r}, \underline{t}, \underline{o})(\underline{r}, \bar{t}, \bar{o})$ $= (11 \times \underline{t}, 13 \times \underline{o}, 17 \times \bar{r})$	
h	$T \rightarrow BX$	$f_h(\bar{t}, \underline{b}, \underline{x}1, \underline{x}2)(\underline{t}, \bar{b}1, \bar{b}2, \bar{x}1, \bar{x}2)$ $= (\underline{b} + \underline{x}2, \bar{t}, \bar{x}1, \bar{t}, \bar{b}2)$	
k	$B \rightarrow X$	$f_k(\bar{b}1, \bar{b}2, \underline{x}1, \underline{x}2)(\underline{b}, \bar{x}1, \bar{x}2)$ $= (\bar{b}2 - \underline{x}2, \bar{b}1, \underline{x}1)$	
l	$X \rightarrow 9$	$f_l(\bar{x}1, \bar{x}2)(\underline{x}1, \underline{x}2)$ $= (\bar{x}1, \bar{x}2)$	

- (1) if attribute β depends on attribute α , then α is evaluated before β ;
 (2) if α is an inherited attribute of $X_{p,j}$ and β is a synthesized attribute of $X_{p,j}$, then α is evaluated before β .

(c \leftarrow) Assume the attributes of $X_{p,0}, X_{p,1}, \dots, X_{p,-1}$ can be evaluated in some order satisfying (1) and (2). Consider an edge from attribute α to attribute β in D_p ($\text{ALL}(X_{p,1}) \dots \text{ALL}(X_{p,-1})$). If the edge is in D_p , α is evaluated before β by (1); if the edge is in $\text{ALL}(X_{p,j})$, α is evaluated before β by (2). Since “is evaluated before” is a linear order, D_p ($\text{ALL}(X_{p,1}) \dots \text{ALL}(X_{p,-1})$) has no cycles.

Comment. An attribute grammar with only ordered productions allows “evaluation in one pass from left to right” [3], [8]. One applies the following recursive algorithm for each application of a production $X_{p,0} \rightarrow X_{p,1} \dots X_{p,-1}$ in a derivation tree:

```

lr evaluate: begin fetch inherited attributes of  $X_{p,0}$ ;
               for  $X : X_{p,1}$  to  $X_{p,-1}$ 
                 do begin Use  $f_p$  to evaluate inherited attributes of  $X$ ;
                       Call lr evaluate to calculate synthesized attributes of  $X$ ;
                 end
               Use  $f_p$  to calculate synthesized attributes of  $X_{p,0}$ ;
    end.
    
```

A similar argument shows that one can evaluate an attribute grammar with only tangled productions in one pass, if we allow a pre-evaluation phase in which we either rearrange the derivation tree or add a next sibling pointer to each node in the tree. One applies the following recursive algorithm for each application of a production $X_{p,0} \rightarrow X_{p,1} \dots X_{p,-1}$ in the tree

```

t-evaluate: begin fetch inherited attributes of  $X_{p,0}$ ;
               for  $\alpha$  : attribute of  $X_{p,0} X_{p,1} \dots X_{p,-1}$  in order given by
                 Theorem 3c
                 do if  $\alpha$  is inherited attribute of  $X_{p,j}$  for  $j \neq 0$ 
                   or  $\alpha$  is synthesized attribute of  $X_{p,0}$ 
                   then Use  $f_p$  to calculate  $\alpha$ 
                   else if  $\alpha$  not already calculated
                   then Call t-evaluate to calculate
                       all synthesized attributes of the  $X_{p,j}$  to which
                        $\alpha$  belongs;
    end.
    
```

An algorithm for finding the finite number of passes required to evaluate a well-defined attribute grammar is given in [15].

DEFINITION 4. An attribute grammar is in *normal form* if for every production the function

$$f_p : L_p^0 \rightarrow R_p^0 \rightarrow R_p^0$$

satisfies

$$f_p(l)(r) = f_p(l)(r')$$

for any l in L_p^0 and any r, r' in R_p^0 .

Comment. For well-defined attribute grammars in normal form, many tiresome distinctions disappear.

THEOREM 4. (Hanne Riis). *If an attribute grammar is in normal form then the production p is reordered \Leftrightarrow production p is tangled.*

Proof. Suppose $X_{p,0} \rightarrow X_{p,1} \dots X_{p,-1}$ is a tangled production of an attribute grammar G . If G is in normal form, we can evaluate all attributes of a right symbol $X_{p,i}$ “at

the same time'', because we can wait until a synthesized attribute is required before evaluating the inherited attributes. To make this argument precise we introduce the relation R by

$$jRk \Leftrightarrow \text{there is an arrow in } D_p \text{ from a synthesized attribute} \\ \text{of } X_{p,j} \text{ to an inherited attribute of } X_{p,k}$$

The reflexive transitive closure R^* of this relation is a partial order because jR^*k , kR^*j , $j \neq k$ implies a chain of arrows in D_p that becomes a cycle in D_p [ALL ($X_{p,1}$) \cdots ALL ($X_{p,-1}$)]; and this cannot happen when p is a tangled production.

Embed the partial order R^* in a total order and define $f(j)$ as the position of $X_{p,j}$ in this total order. Clearly f is a permutation and jRk implies $f(j) < f(k)$. Because G is in normal form, there are no arrows in W_p from either synthesized attributes of $X_{p,0}$ or inherited attributes of $X_{p,1} \cdots X_{p,-1}$. If there is an arrow in W_p from an attribute of $X_{p,j}$ to an attribute of $X_{p,k}$ we must have $j \neq 0$ and jRk . The tangled production p must be reordered; the converse implication is given by Theorem 3.

Example (continued). As an illustration of the simplifications possible when productions have our "compiler-friendly" properties we reformulate our grammar CONTRIVED within mathematical semantics in Table 4.

TABLE 4

syntactic rule	semantic function
$S \rightarrow U$	$s[U] = \underline{u1} + \underline{u2}$ where $(\underline{u1}, \underline{u2}) = u[U](\underline{u2}, \underline{u1})$
$U \rightarrow 5$	$u[5](\underline{u1}, \underline{u2}) = (23 \times \underline{u1}, 5)$
$U \rightarrow 7$	$u[7](\underline{u1}, \underline{u2}) = (7, 29 \times \underline{u2})$
$S \rightarrow O$	$s[O] = 3 \times o[O]2$
$O \rightarrow X$	$o[X](\bar{o}) = \underline{x1}/\underline{x2}$ where $(\underline{x1}, \underline{x2}) = x[X](\bar{o}, \bar{o})$
$O \rightarrow RT$	$o[RT](\bar{o}) = 19 \times \underline{t}$ where $\underline{t} = r[R](31 \times \bar{o})$ and $\underline{t} = t[T](37 \times \underline{r})$
$R \rightarrow TO$	$r[TO](\bar{r}) = 11 \times \underline{q}$ where $\underline{q} = o[O](17 \times \bar{r})$ and $\underline{t} = t[T](13 \times \underline{q})$
$T \rightarrow BX$	$t[BX](\bar{t}) = \underline{b} + \underline{x2}$ where $\underline{b} = b[B](\bar{t}, \bar{t})$ and $(\underline{x1}, \underline{x2}) = x[X](\bar{t}, \bar{t})$
$B \rightarrow X$	$b[X](\bar{b1}, \bar{b2}) = \bar{b2} - \underline{x2}$ where $(\underline{x1}, \underline{x2}) = x[X](\bar{b1}, \underline{x1})$
$X \rightarrow 9$	$x[X](\underline{x1}, \underline{x2}) = (\underline{x1}, \underline{x2})$

The only productions which are not tangled are $S \rightarrow U$ and $B \rightarrow X$. For these two productions and no others we have recursion in the corresponding semantic function. Since our grammar is well defined, this recursion can be eliminated by Theorem 2. For the nonbenign production $S \rightarrow U$, the proof of the theorem suggests replacing the semantic functions for the first three productions by

$$s[U] = sb[U] \cup sc[U], \\ sb[U] = \underline{u1} + \underline{u2} \quad \text{where } \underline{u2} = u2b[U] \text{ and } \underline{u1} = u1b[U](\underline{u2}), \\ sc[U] = \underline{u1} + \underline{u2} \quad \text{where } \underline{u1} = u1c[U] \text{ and } \underline{u2} = u2c[U](\underline{u1}), \\ u1b[5](\underline{u1}) = 23 \times \underline{u1}, \quad u2b[5] = 5, \\ u1c[7] = 7, \quad u2c[7](\underline{u2}) = 29 \times \underline{u2}.$$

The proof of our next theorem shows why the join operator \cup is not needed when removing the recursion in the semantic function for the benign production $B \rightarrow X$:

$$b[X](\bar{b1}, \bar{b2}) = \bar{b2} - \underline{x2} \quad \text{where } \underline{x1} = \bar{b1} \text{ and } \underline{x2} = \underline{x1}.$$

Convention. We use $MS[G]$ as an abbreviation for a specification in mathematical semantics of the function s in the proof of Theorem 1 for an attribute grammar G .

Determinacy theorem. *If all productions in an attribute grammar G are benign, then the join operator U need not appear in any of the functions specified by $MS[G]$.*

Proof. For each X the set of symbol graphs $SYM(X)$ can be replaced by their union $SOME(X)$. If we make this replacement in the proof of Theorem 2, there is one and only one Q satisfying requirement (***) for a production $X_{p,0} \rightarrow X_{p,1} \cdots X_{p,-1}$. The function rules (Q, α) that are joined in the definition of symbol (Γ, α) come from different productions with the same left side. Such joins do not appear in an $MS[G]$ specification because of the convention in § 2.

Comment. Because our grammar $MS(G)$ works on derivation trees, the implicit joins in the § 2 convention do not destroy determinacy. The convention that $MS(G)$ semantic functions may be specified in terms of one another seems just as harmless. In our statement of Theorem 2 we avoided the fixed point operator Y used to unravel this mutual recursion.

Splitting theorem. *If all productions in an attributed grammar G are tangled, then we can construct an $MS(G)$ such that*

- (a) *no function specified in $MS(G)$ uses the operators Y and U*
- (b) *every function specified in $MS(G)$ is in $CONT(X)$ for some X in $N \cup T$.*

Proof.

- (a) Combine Theorem 2 and the determinacy theorem.

(b) Consider the $MS(G)$ formulation given by part (a). It consists of specifications of the functions $Symbol(SOME(X), \alpha)$ for each X in $N \cup T$ and each synthesized attribute α in \underline{X} . For a tangled production $X_{p,0} \rightarrow X_{p,1} \cdots X_{p,-1}$, all inherited attributes of $X_{p,i}$ can be evaluated before any synthesized attributes of $X_{p,i}$. Thus each function $Symbol(SOME(X), \alpha)$ can be extended from $DOM(X) \rightarrow W(SOME(X), \alpha) \rightarrow V_\alpha$ to $DOM(X) \rightarrow INH(X) \rightarrow V_\alpha$. Our theorem now follows from the fact that the lattice product of $DOM(X) \rightarrow INH(X) \rightarrow V_\alpha$ for α in \underline{X} is isomorphic to the lattice $CONT(X) = DOM(X) \rightarrow INH(X) \rightarrow SYN(X)$.

Comment. When we removed recursion from the semantic function for the benign production $B \rightarrow X$ in our grammar, the required splitting of $SYN(X)$ was implicit. The general construction would give

$$b[X](\overline{b1}, \overline{b2}) = \overline{b2} - \underline{x2} \quad \text{where } \underline{x1} = x1[X]\overline{b1} \text{ and } \underline{x2} = x2[X]\underline{x1},$$

$$x1[9](\overline{x1}) = \overline{x1},$$

$$x2[9](\overline{x2}) = \overline{x2},$$

and minor changes in the specifications for productions $O \rightarrow X$ and $T \rightarrow BX$.

Concluding remarks. The converse of the problem in this paper—forming an attribute grammar from a specification in mathematical semantics—is the subject of [8], [11]. Is there any good reason for basing a compiler generator on attribute grammars, rather than mathematical semantics [14]? If there is, should one allow for attribute grammars that are well defined but not benign? Any algorithm for checking that an attribute grammar is well defined is computationally intractable [6], [7]. Chirica and Martin [4] give a pragmatic reason for preferring benign grammars for particular languages; our determinacy theorem gives a theoretical reason for this preference.

Acknowledgment. The author would like to thank Ole Lehmann Madsen, Hanne Riis, and Erik Meineche Schmidt for many fruitful discussions on this and the other topics discussed in this paper.

REFERENCES

- [1] L. AIELLO, M. AIELLO AND R. W. WEYRAUCH, *The semantics of PASCAL in LCF*, STAN-74-447, Stanford University, Stanford CA, 1974.
- [2] D. BJØRNER AND C. B. JONES, *Vienna Development Method: The Meta Language*, Lecture Notes in Computer Science, Springer-Verlag, New York, 1978.
- [3] L. M. CHIRICA AND D. F. MARTIN, *An order-algebraic definition of Knuthian semantics*, Math. Sys. Theory, 13 (1979), pp. 1–27.
- [4] ———, *An algebraic formulation of Knuthian semantics*, Proc. of the 17th IEEE Symposium on the Foundations of Computer Science, 1976, pp. 127–136.
- [5] J. B. DENNIS, *On storage management for advanced programming languages*, Project MAC Computation Structures Group, memo 109-1, Massachusetts Institute of Technology, Cambridge, MA, 1974.
- [6] M. JAZAYERI, W. F. OGDEN AND W. C. ROUNDS, *The intrinsically exponential complexity of the circularity problem for attribute grammars*, Comm. ACM, 12 (1975), pp. 697–721.
- [7] N. JONES, *Circularity testing of attribute grammars requires exponential time: a simpler proof*, DAIMI PB-107, Aarhus, 1980.
- [8] H. GANZINGER, *Some principles for the development of compiler descriptions from denotational language definitions*, Tech. Univ. München, preprint, 1980.
- [9] D. E. KNUTH, *Semantics of context free languages*, Math. Sys. Theory, 2 (1968), pp. 127–145; correction, *ibid*, 5 (1971), p. 95.
- [10] P. M. LEWIS, P. J. ROSENKRANTZ AND R. E. STEARNS, *Attributed translations*, J. Comput. Sys. Sci., 9 (1974), pp. 279–307.
- [11] O. L. MADSEN, *On defining semantics by means of extended attribute grammars*, DAIMI PB-90, Aarhus, 1978.
- [12] R. E. MILNE, *The formal semantics of computer language and their implementations*, Ph.D. thesis, Cambridge University, Cambridge, 1974.
- [13] P. D. MOSSES, *The mathematical semantics of Algol 60*, Program Research Group PRG-12, Oxford, 1974.
- [14] ———, *Mathematical semantics and compiler generation*, Ph.D. thesis, Oxford University, 1975.
- [15] H. RIIS AND S. SKYUM, *K-visit attribute grammars*, DAIMI PB-121, Aarhus, 1980.
- [16] D. SCOTT, *Mathematical concepts in programming language semantics*, AFIPS Proc., 40 (STCC 1972), pp. 225–242.
- [17] D. SCOTT AND C. STRACHEY, *Towards a Mathematical Semantics for Computer Languages*, Proc. Symposium on Computers and Automata, Brooklyn Polytechnic, New York, 1971.
- [18] R. D. TENNENT, *Mathematical semantics and the design of programming languages*, Ph.D. thesis, Toronto University, 1973.
- [19] ———, *The denotational semantics of programming languages*, Comm. ACM, 8 (1976), pp. 437–453.

AUTOMATIC PROGRAMMING OF FINITE STATE LINEAR PROGRAMS*

AMIR PNUELI[†] AND GIORA SLUTZKI[‡]

Abstract. Finite State Linear Programs (FSLP) are introduced to model simple data processing applications. Essentially these are finite automata with the added capability of performing linear operations on a set of registers and the input. Algorithmic constructions are given to test equivalence of FSLP programs, and to minimize the number of states and registers. Linear algebraic methods are used for the register minimization procedure (and its correctness proof).

Key words. automatic programming, finite state programs, analysis, synthesis, optimization, register optimization.

1. Introduction. The prominent problems in the area of analysis and synthesis of programs can all be described under the uniform framework of translation or alternative descriptions of a common object. We may consider two languages: \mathcal{S} , a specification language, being a high level, nonprocedural description language (such as the predicate calculus) and \mathcal{L} , a programming language. With respect to this pair we may investigate the following questions.

- (1) *Analysis.* Given a program in \mathcal{L} , produce its specification in \mathcal{S} .
- (2) *Synthesis.* Given a specification in \mathcal{S} , construct a program in \mathcal{L} satisfying the specification.
- (3) *Verification.* Given a specification in \mathcal{S} , and a program in \mathcal{L} , check them for consistency.
- (4) *Optimization.* Find the best program in \mathcal{L} equivalent to a given program in \mathcal{L} or satisfying a given specification in \mathcal{S} .

The present approaches to these very important problems can be divided into two types.

(a) Considering the problems in their full generality, devise heuristics which will cover as many cases as possible. All the efforts in [1], [3], [5] can be grouped under this approach.

(b) Construct submodels of the general case for which there exist algorithms solving some of the problems (1)–(4). A successful example of this approach is [8].

The main difference between (a) and (b) (which may be argued to have a large overlap) is that (a) is essentially procedure oriented while (b) is model oriented. In this paper we adopt the approach in (b), calling upon extensions of models developed in automata theory. More specifically, we suggest an extension to finite state automata which will approximate a simple class of data processing programs which satisfy the following properties.

- (a) The program reads an input tape and outputs a single (numeric) result.
- (b) The tape contains both numeric and alphabetic fields. The numeric values are used for computation, using a finite set of numeric registers (program variables). The operations allowed are linear in the input values (i.e., additions and multiplications by constants).
- (c) The symbolic (alphabetic) fields are used for control decisions, i.e., selection of the next operation, or next part in the program to be executed.

* Received by the editors July 13, 1978, and in final form October 23, 1980.

[†] Department of Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel.

[‡] Department of Computer Science, University of Kansas, Lawrence, Kansas 66045.

The two critical restrictions stipulated in the model are (1) the operations are linear and (2) control decisions depend only on the symbolic fields of the input, and *not* on the results of the computation.

Thus, the finite state linear program (FSLP) model imposes a separation between arithmetic (numeric) manipulation and control. For the control component we assume a finite state control. This separation ensures, as shown in the paper itself, decidability for the questions (1)–(4) above. Simple as this class is, it does cover some nontrivial examples and with additional extensions (discussed in § 5) can cover an interesting subclass of data processing applications.

While this work reports only results about relations and transformations between programs, and thus is restricted to investigations within \mathcal{L} (the programming language), there has been developed an appropriate specification language \mathcal{S} . Just as the class of FSLP programs is an extension of finite state automata, the specification language is an extension of the regular expression language obtained by allowing numerical coefficients in the expressions. Relative to these two languages the questions of analysis and synthesis have been investigated and proved decidable, and algorithms have been developed for their general solution. These results are reported in [7].

The paper is organized as follows. The second section introduces the FSLP model. The two results proved in this section are (1) nondeterminism does not increase the computational power of FSLP programs and (2) allowing nondeterminism, one register is enough to carry out the computation. Section 3 treats the *zero computation problem* (i.e., whether the program computes zero on its domain) and the (related) *equivalence problem*. Both problems are shown to be decidable. In § 4 we deal with *optimization problems*. It is shown that state optimization can be carried out by identifying equivalent states, much in the spirit of the traditional automata theory. The main result of this section is a procedure to minimize the number of registers subject to a given control structure. This enables one first to minimize the number of states in the program and then to minimize the number of registers without compromising the state minimality. The last section concludes the paper, summarizing the results and pointing out several potentially useful extensions. All constructions presented in this paper are effective.

2. The class of FSLP programs. The class of programs we study is a class of programs which process two data types. The first is a finite set of letters Σ (finite alphabet), while the second is some fixed field of characteristic zero, R (the rationals will be used in our examples). The program can manipulate a finite set of registers of type R , denoted by Z_1, \dots, Z_n , and it can read a file which contains a sequence of pairs $(\sigma_1, y_1) \cdots (\sigma_i, y_i) \cdots$, with $\sigma_i \in \Sigma$ and $y_i \in R$. Each pair (σ_i, y_i) is called a record. Thus, an input to an FSLP program is a “word” over the (infinite) alphabet of records $\Sigma \times R$. The empty word will be denoted by Λ . The program uses the σ_i ’s to determine its next move, and it can add or subtract the y_i ’s from some of its registers. The program cannot test the current value of any of its registers in order to determine its next move. Thus the control capabilities of programs (in our case finite state control) and their computational capabilities are separated in the FSLP model. The operations allowed on the R -data type are addition, subtraction and multiplication by a constant.

We will have to deal (occasionally) with undefined values. Denote by ω , $\omega \notin R$, the undefined value, and let $R^+ = R \cup \{\omega\}$. We set the following rules for arithmetical operations for any $r \in R$ and ω : (1) $r \pm \omega \equiv r$, (2) $r \cdot \omega \equiv \omega$, where the “ \equiv ” relation means that either both sides are undefined (i.e., ω) or both are defined and equal to each other. As it turns out, the seemingly bizarre rule $r \pm \omega = r$ is very convenient technically (see Definitions 2.1(5) and 2.3). Apparently, the reason is that ω behaves both like a zero

and like the usual undefined symbol precisely where the respective properties are needed.

DEFINITION 2.1. Let R be any field of characteristic 0. An R -FSLP program is a pair $P = (\mathcal{A}, \mathcal{F})$, where $\mathcal{A} = (K, \Sigma, \delta, B, F)$ is the *underlying* (nondeterministic) *finite state automaton* with state set, input set, move function, initial states and final states respectively, and $\mathcal{F} = (\bar{Z}, T, U, A, V)$ is the *computation structure*, where:

(1) $\bar{Z} = (Z_1, \dots, Z_n)$ is a (row) vector of registers capable of storing values from R . The register vector is modified during transitions.

(2) $T: K \times \Sigma \times K \rightarrow R^{n \times n}$, where $R^{n \times n}$ is the set of $n \times n$ matrices over R ; for $q' \notin \delta(q, \sigma)$, $T(q, \sigma, q') = 0$ (all-zero arrays will be denoted by 0).

(3) $U: K \times \Sigma \times K \rightarrow R^n$, where R^n is the set of (row) n -vectors over R ; for $q' \notin \delta(q, \sigma)$, $U(q, \sigma, q') = 0$.

T and U assign to each transition (q, σ, q') a matrix $T(q, \sigma, q')$ and a vector $U(q, \sigma, q')$ such that the transformation associated with this transition is $\bar{Z} \leftarrow \bar{Z} \cdot T(q, \sigma, q') + U(q, \sigma, q') \cdot y$, where y is the numerical input read during the transition.

(4) $A: B \rightarrow R^n$ assigns a (row) n -vector $A(q)$ over R to each initial state $q \in B$. $A(q)$ is the initial value of \bar{Z} for computations starting at q .

(5) $V: F \rightarrow R^n$ assigns a (column) n -vector $V(q)$ over R to each accepting state $q \in F$. If the value of \bar{Z} at state $q \in F$ is \bar{u} , then the output value associated with q is $\bar{u} \cdot V(q)$.

A and V may be extended to all of K , in which case we set $A(q) = 0$ for $q \notin B$ and $V(q) = \omega$ (ω will denote arrays all of whose components are ω) for $q \notin F$.

To save repetitions we agree that, unless stated otherwise, a program P is denoted by the pair $(\mathcal{A}, \mathcal{F})$ and that \mathcal{A} and \mathcal{F} always have the (same) components $(K, \Sigma, \delta, B, F)$ and (\bar{Z}, T, U, A, V) respectively.

An FSLP program P is *deterministic* if $|B| = 1$ (i.e., B contains a single state) and for every $q \in K$ and $\sigma \in \Sigma$, $|\delta(q, \sigma)| \leq 1$ (i.e., there is at most one transition from q by σ). In this case we prefer to specify a single initial state, denoted usually by q_1 , instead of the set B . Since all the programs discussed in this paper are FSLP programs, we shall sometimes omit the adjective "FSLP". Hopefully, no confusion will be caused.

In our examples we shall usually describe FSLP programs as transition graphs whose nodes correspond to states and edges to transitions. Each edge is labeled by a letter $\sigma \in \Sigma$ which selects the transition and a set of assignments of the form $Z_i \leftarrow a_{i1}Z_1 + \dots + a_{in}Z_n + b_i y$, $i = 1, 2, \dots, n$, with $a_{ij}, b_i \in R$. y stands for the currently input R -value. The assignments are assumed to be carried out simultaneously; i.e., all right-hand side expressions are calculated first and then assigned to the left-hand side variables. Of course, all these assignments can be represented as a single vector operation $\bar{Z} \leftarrow \bar{Z} \cdot M + \bar{\eta} \cdot y$, where M and $\bar{\eta}$ are the T and U for the specific transition according to Definition 2.1. Also, if no operation is performed during a transition (i.e., M is the identity matrix and $\bar{\eta}$ is the null vector) we prefer not to write any assignments on the edge corresponding to that transition. This omission will make the figures more readable and stress the fact that such transitions are considered as purely control-transitions.

For any word (file) $w = (\sigma_1, y_1)(\sigma_2, y_2) \dots (\sigma_s, y_s) \in (\Sigma \times R)^*$ we define $\lambda(w)$ as $\sigma_1 \dots \sigma_s$ (the Σ -component of w). For any such word and for any program P we define the following computation functions.

2.2. DEFINITION. The function $\text{Val}_P: (\Sigma \times R)^* \times K \rightarrow R^n$ is a vector-valued function such that $\text{Val}_P(w, q)$ expresses the current value of the register vector \bar{Z} at the state q , after reading the input word w . Since P is nondeterministic, there may be many paths reading w (w -paths) and leading from B to q . Roughly, $\text{Val}_P(w, q)$ is the combined

value obtained by summing computations over the different paths, but discarding the multiplicity introduced by shared subpaths. We explicitly stipulate that if $q \notin \delta(B, \lambda(w))$ then $\text{Val}_P(w, q) = 0$. In all other cases Val_P is given by the following recursive definition:

$$\text{Val}_P(\Lambda, q_j) = A(q_j),$$

$$\text{Val}_P(w \cdot (\sigma, y), q_j) = \sum_{q_k \in \delta(B, \lambda(w))} [\text{Val}_P(w, q_k) \cdot T(q_k, \sigma, q_j) + U(q_k, \sigma, q_j) \cdot y].$$

Note that in the deterministic case the definition of Val_P coincides with the intuitive notion of computation sequence corresponding to an input w , that is, proceeding along the uniquely determined path and performing the computations associated with the edges in a stepwise manner.

DEFINITION 2.3. We define a function $\text{Out}_P : (\Sigma \times R)^* \rightarrow R$ by

$$\text{Out}_P(w) \equiv \sum_{q \in \delta(B, \lambda(w))} \text{Val}_P(w, q) \cdot V(q).$$

Note that if P contains no w -path leading from B to F , then $\text{Out}_P(w)$ is undefined. Thus in contrast to the totality of Val_P we prefer Out_P to be a partial function, a feature that corresponds to the possibility of program results being undefined. Note also that in the case that $\text{Out}_P(w)$ is defined we may extend the summation over all states of P .

Example 2.4. Consider the following “real-life” situation. A wholesale store has two types of customers denoted by u and v . It is selling two commodities a and b , and the prices vary among the customers according to Table 1.

TABLE 1

Product	Price/Unit Customer u	Price/Unit Customer v
a	3	4
b	7	6

An input file is presented which contains several groups of records, each listing transactions for a certain customer. Each such group is headed by a record $(u, 0)$ or $(v, 0)$ specifying the customer’s type. The records in the group are of the form (a, y) or (b, y) where y is the quantity purchased at the transaction. The file is terminated by an $(s, 0)$ record. Thus the structure of the file can be described symbolically as

$$[[(u, 0) + (v, 0)] [(a, y) + (b, y)]^*]^* (s, 0),$$

where $+$ stands for choice (disjunction).

The (deterministic) program in Fig. 1 will read such a file and compute the balance of all the transactions.

Consider now an alternative situation in which the record which identifies the customer’s type succeeds rather than precedes the relevant transaction group. In this case we will construct a nondeterministic program as in Fig. 2. Program P_2 when presented with the input $(a, 1)(a, 2)(b, 3)(u, 0)(b, 2)(b, 2)(v, 0)(s, 0)$ outputs the value 54.

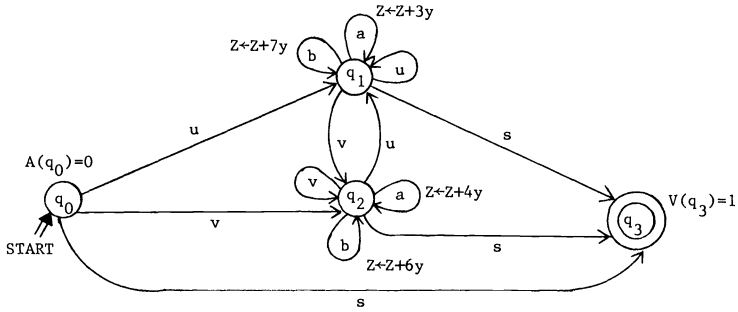


FIG. 1. Program P_1 .

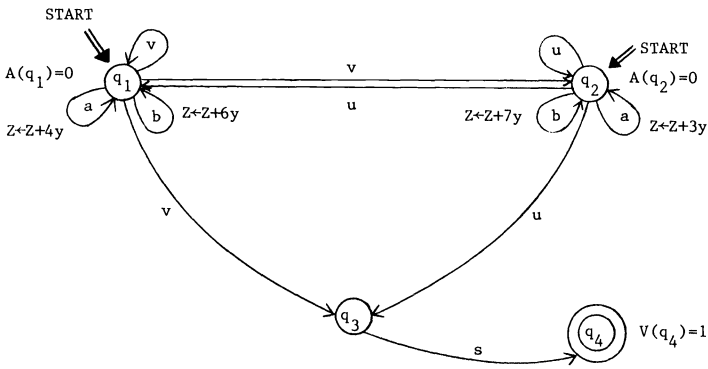


FIG. 2. Program P_2 .

There are special cases of the general model of FSLP programs that are worth noting. Let P be a program. We already mentioned the deterministic programs. If for each $(q, \sigma, q') \in K \times \Sigma \times K$, $U(q, \sigma, q') = 0$ then the R values on the input tape can be ignored. In this case we obtain a *multiplicative automaton* (MA). If, in addition, $\dim \bar{Z} = 1$ (i.e., there is only one register) then we get a scalar MA. A model similar to our scalar MA is discussed extensively in [4, Chapt. 6].

Let P_1 and P_2 be FSLP programs. We say that P_1 and P_2 are *equivalent*, $P_1 \approx P_2$, if their $\text{Out}(w)$ functions are equivalent; i.e., for every $w \in (\Sigma \times R)^*$, $\text{Out}_{P_1}(w) \equiv \text{Out}_{P_2}(w)$.

We will show now that the well-known tradeoff between determinism and number of states in the classical automata theory is extended here to a trade-off between determinism and the number of registers and states. This is expressed in the following two theorems.

THEOREM 2.5. *For every FSLP program P there is an equivalent deterministic FSLP program \hat{P} .*

THEOREM 2.6. *For every FSLP program P there is an equivalent (possibly nondeterministic) FSLP program \hat{P} utilizing a single register.*

Proof of Theorem 2.5. Let $P = (\mathcal{A}, \mathcal{F})$ be an FSLP program. The idea of the construction is the following. The underlying control structure of \hat{P} is exactly the one derived when constructing the deterministic version of the finite automaton \mathcal{A} . The new register vector \hat{Z} will be a multiple copy of the original \bar{Z} having a unique copy for each

state $q \in K$. When entering a state $\hat{q} \in \hat{K}$ (which is a subset of K) the copy belonging to $q_i \in \hat{q}$ will undergo the transformation applied to \bar{Z} when entering q_i in P . In addition we shall ensure that copies belonging to $q_i \notin \hat{q}$ will be deleted (reset to 0) on entrance to \hat{q} . Assume that $|K| = m$, $\dim \bar{Z} = n$. Let $\hat{n} = m \cdot n$. The required deterministic \hat{P} is $\hat{P} = (\hat{\mathcal{A}}, \hat{\mathcal{F}})$, where $\hat{\mathcal{A}} = (\hat{K}, \Sigma, \hat{\delta}, \hat{q}_1, \hat{F})$ is the deterministic version of \mathcal{A} and $\hat{\mathcal{F}} = (\hat{Z}, \hat{T}, \hat{U}, \hat{A}, \hat{V})$ is defined below. $\hat{Z} = (\hat{Z}^1, \dots, \hat{Z}^m)$ is a “supervector” assuming values from $R^{\hat{n}}$, where \hat{Z}^i is a copy of the original \bar{Z} associated with the state $q_i \in K$. $\hat{T} : \hat{K} \times \Sigma \times \hat{K} \rightarrow R^{\hat{n} \times \hat{n}}$ is an assignment of an $\hat{n} \times \hat{n}$ matrix to each edge $(\hat{q}, \sigma, \hat{q}')$. For each $(\hat{q}, \sigma, \hat{q}')$ such that $\hat{q}' = \hat{\delta}(\hat{q}, \sigma)$ it is given by the block form $\hat{T}(\hat{q}, \sigma, \hat{q}') = (T_{i,j})$, where $T_{i,j} = T(q_i, \sigma, q_j)$, $1 \leq i, j \leq m$. Note that \hat{T} depends only on $\sigma \in \Sigma$ and not on \hat{q} or \hat{q}' . We extend the definition of \hat{T} by setting $\hat{T}(\hat{q}, \sigma, \hat{q}') = 0$ if $\hat{q}' \neq \hat{\delta}(\hat{q}, \sigma)$. $\hat{U} : \hat{K} \times \Sigma \times \hat{K} \rightarrow R^{\hat{n}}$ is an assignment of an \hat{n} -vector to each edge $(\hat{q}, \sigma, \hat{q}')$. For each $(\hat{q}, \sigma, \hat{q}')$ satisfying $\hat{q}' = \hat{\delta}(\hat{q}, \sigma)$, \hat{U} is given by the block form $\hat{U}(\hat{q}, \sigma, \hat{q}') = (\hat{U}^1, \dots, \hat{U}^m)$ where $\hat{U}^j = \sum_{q_i \in \hat{q}} U(q_i, \sigma, q_j)$. Note that $\hat{U}(\hat{q}, \sigma, \hat{q}')$ is independent of \hat{q}' . We extend the definition of \hat{U} by setting $\hat{U}(\hat{q}, \sigma, \hat{q}') = 0$ if $\hat{q}' \neq \hat{\delta}(\hat{q}, \sigma)$. $\hat{A}(\hat{q}_1) = (A(q_1), A(q_2), \dots, A(q_m))$; i.e., $\hat{A}(\hat{q}_1)$ is the supervector composed of all the initial vectors, each in the position corresponding to its state. Note that zero vectors will correspond to noninitial states. $\hat{V} : \hat{F} \rightarrow R^{\hat{n}}$ assigns a (column) supervector in $R^{\hat{n}}$ to states $\hat{q} \in \hat{F}$ by the block form $\hat{V}(\hat{q}) = (V'(q_1)^T, \dots, V'(q_m)^T)^T$ where $V'(q_i) = V(q_i)$ if $V(q_i)$ is defined and 0 otherwise. The superscript T denotes the transposition operation. Note that for \hat{q} and \hat{p} in \hat{F} we have $\hat{V}(\hat{q}) = \hat{V}(\hat{p})$. Again we extend the definition by setting $\hat{V}(\hat{q}) \equiv \omega$ for $\hat{q} \notin \hat{F}$. This ends the construction of \hat{P} . The proof that $P \approx \hat{P}$ rests on the assertion $[\text{Val}_{\hat{P}}(w, \hat{q}_w)]_j = \text{Val}_P(w, q_j)$, where $\hat{q}_w = \hat{\delta}(\hat{q}_1, \lambda(w))$ and the subscript j on the left-hand side picks the j th block (which is an n -vector) of $\text{Val}_{\hat{P}}(w, \hat{q}_w)$. The proof of this assertion is a straightforward induction on $|w|$ (the length of w). \square

Proof of Theorem 2.6. Let $P = (\mathcal{A}, \mathcal{F})$ be given with $K = \{q_1, \dots, q_m\}$. We construct a 1-register FSLP program $\hat{P} = (\hat{\mathcal{A}}, \hat{\mathcal{F}})$ where $\hat{\mathcal{A}} = (\hat{K}, \Sigma, \hat{\delta}, \hat{B}, \hat{F})$ and $\hat{\mathcal{F}} = (\hat{Z}, \hat{T}, \hat{U}, \hat{A}, \hat{V})$. The idea of the construction is to have a new state corresponding to each register in each state of P . Thus when reaching state $\hat{q}_i \in \hat{K}$ we intend \hat{Z} (the single register of \hat{P}) to hold the value that Z_i originally held on reaching q_i in P . The formal definition follows.

$\hat{K} = \{q_i^j \mid i = 1, \dots, m; j = 1, \dots, n\}$ the set of states of \hat{P} . \hat{Z} is the single register of \hat{P} . $\hat{\delta} : \hat{K} \times \Sigma \rightarrow 2^{\hat{K}}$ is defined by $\hat{\delta}(q_i^j, \sigma) = \{q_k^l \mid k = 1, \dots, m; q_l \in \delta(q_i, \sigma)\}$. $\hat{T} : \hat{K} \times \Sigma \times \hat{K} \rightarrow R$ is a scalar mapping defined by $\hat{T}(q_i^j, \sigma, q_k^l) = [T(q_i, \sigma, q_l)]_{i,k}$; i.e., the (i, k) entry of the matrix $T(q_i, \sigma, q_l)$. $\hat{U} : \hat{K} \times \Sigma \times \hat{K} \rightarrow R$ is a scalar mapping defined by $\hat{U}(q_i^j, \sigma, q_k^l) = (1/n)[\hat{U}(q_i, \sigma, q_l)]_k$, where $1/n$ is the inverse of n , which is assumed to exist in R . Note that the right-hand side is independent of i . $\hat{B} = \{q_j^i \mid i = 1, \dots, n; q_j \in B\}$. $\hat{A} : \hat{B} \rightarrow R$ is given by $\hat{A}(q_j^i) = [A(q_j)]_i$. \hat{A} can be extended in the usual manner. $\hat{F} = \{q_j^i \mid i = 1, 2, \dots, n; q_j \in F\}$. $\hat{V} : \hat{F} \rightarrow R$ is defined by $\hat{V}(q_j^i) = [V(q_j)]_i$. Again \hat{V} is extended to be ω for states not in \hat{F} . The inductive claim which is proved in this case (by induction on $|w|$) is

$$[\text{Val}_P(w, q_j)]_i = \text{Val}_{\hat{P}}(w, q_j^i),$$

and it serves to establish the equivalence of P and \hat{P} . \square

Example 2.7. The deterministic version of the program P_2 of Fig. 2 is P_3 shown in Fig. 3. We have slightly deviated from the formal construction given in the proof of Theorem 2.5 in that one (redundant) register (corresponding to the state q_4) was deleted and the register corresponding to q_3 assumed its role. In Fig. 3 the actions C_i associated

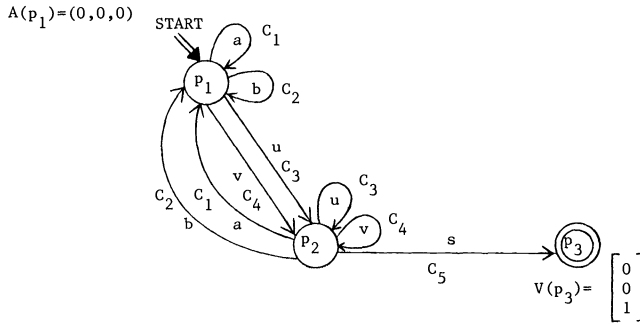


FIG. 3. Program P₃.

with the edges are the following:

$$\begin{aligned}
 C_1: \bar{Z} &\leftarrow \bar{Z} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + (4, 3, 0) \cdot y, & C_2: \bar{Z} &\leftarrow \bar{Z} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + (6, 7, 0) \cdot y \\
 C_3: \bar{Z} &\leftarrow \bar{Z} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, & C_4: \bar{Z} &\leftarrow \bar{Z} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, & C_5: \bar{Z} &\leftarrow \bar{Z} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.
 \end{aligned}$$

3. Zero computation and equivalence testing. In this section we assume that the FSLP programs are deterministic unless stated otherwise. By Theorem 2.5 no generality is lost.

The intuitive notion of the FSLP model being fully analyzable, which we wish to stress throughout the paper, should clearly imply algorithmic solutions to questions like the *zero computation problem* and *equivalence*. An FSLP program is *zero computing* (ZC) if it outputs zero for each word w in $(\Sigma \times R)^*$ for which it is defined. The *zero computation problem* (ZCP) is to determine for a given program P whether or not it is ZC. This problem is algorithmically solvable. One approach to the proof of this fact is to extend the result in [4] (given there for scalar MA's). The basic idea would be to show that for each program P there is a bound N such that P is zero computing for all inputs if and only if it is zero computing for all words not longer than N . Of these it is sufficient to consider inputs whose R -components are all zero except one which is 1. Thus by a finite amount of checking one can verify that P is zero computing for all inputs.

We adopt here a more constructive approach. Let $P = (\mathcal{A}, \mathcal{F})$ be a deterministic program with $\dim \bar{Z} = n$. For each $q \in K$, each row vector $\bar{v} \in R^n$ and $x \in \Sigma^*$ we define $\text{Amp}(q, \bar{v}, x) = 0$ if $\delta(w, x) \notin F$, and otherwise to be the value output by P if started at q with initial vector \bar{v} and fed with $w = (\sigma_1, 0) \cdots (\sigma_b, 0)$, where $\lambda(w) = x$.

For each $q \in K$ we will construct a *zero amplification subspace* (ZAS) S_q (of column vectors), which is a subspace of R^n such that

$$\bar{v} \perp S_q \Leftrightarrow \text{Amp}(q, \bar{v}, x) = 0 \quad \text{for all } x \in \Sigma^*.$$

$\bar{v} \perp S_q$ denotes here that \bar{v} is orthogonal to S_q . Thus the subspace orthogonal to S_q is the subspace of all initial values, at q , that will cause a zero amplification for any zero input word w (i.e., word whose R -components are all zero). The bases of the spaces S_q are constructed backwards by successive accumulation. At any stage they will be represen-

ted, at each q , by a set N_q of independent n -vectors whose number is obviously bounded by n .

3.1. The S_q -algorithm.

Initialization. Set $N_q = \emptyset$ for each nonaccepting state q and $N_q = \{V(q)\}$ for each $q \in F$. Obviously, for each q in F , the vector $V(q)$ should be in S_q since any initial \bar{v} which causes zero amplification must yield 0 for the empty word Λ , and hence must satisfy $\bar{v} \cdot V(q) = 0$.

Iteration step. This step consists of consideration of all edges (q, σ, q') labeled by the operation $\bar{Z} \leftarrow \bar{Z} \cdot M + \bar{\eta} \cdot y$. Here, for each $\bar{u}' \in S_{q'}$ we should have $M \cdot \bar{u}' \in S_q$. This is so because any \bar{v} which satisfies $\bar{v} \perp S_{q'}$ can, by reading σ , go into q' with value $\bar{v} \cdot M$, and hence must satisfy $\bar{v} \cdot M \perp S_{q'}$. Hence $\bar{v} \cdot M \cdot \bar{u}' = 0$, i.e., \bar{v} is orthogonal to $M \cdot \bar{u}'$, which gives $M \cdot \bar{u}' \in S_q$. The iteration step consists therefore of taking each $\bar{u}' \in N_{q'}$ and calculating $\bar{u} = M \cdot \bar{u}'$. \bar{u} is then added to N_q if it is not already in $S_q = L(N_q)$ (i.e., cannot be expressed as a linear combination of the current basis N_q). $L(N_q)$ is the linear subspace spanned by N_q). The iteration step is repeated until after a complete cycle, including all edges, no N_q has increased.

It is obvious that the algorithm converges. After the iteration step of the S_q -algorithm has been completed we can conclude that each state $q \in K$ has an associated subspace $S_q = L(N_q)$ of R^n .

The fact that this algorithm actually computes the desired S_q 's is proved in the following lemma.

LEMMA 3.2.

$$\bar{v} \perp S_q \Leftrightarrow \forall x \in \Sigma^* [\text{Amp}(q, \bar{v}, x) = 0].$$

Proof. (\Leftarrow) Assume that for all $x \in \Sigma^* [\text{Amp}(q, \bar{v}, x) = 0]$ and that $\bar{v} \cdot \bar{u}' \neq 0$ for some $\bar{u}' \in S_{q'}$. Since $N_{q'}$ spans $S_{q'}$, there must exist a vector \bar{u} in $N_{q'}$ such that $\bar{v} \cdot \bar{u} \neq 0$. Then by the inductive definition of the $N_{q'}$'s we can choose a path $\pi = (q, \sigma_1, q_1)(q_1, \sigma_2, q_2) \cdots (q_{k-1}, \sigma_k, q_k)$ with $q_k \in F$ such that (1) $\bar{v} \cdot \bar{u} \neq 0$ and (2) for each i , $1 \leq i \leq k$, we can choose \bar{u}_i in N_{q_i} such that $\bar{u} = T(q, \sigma_1, q_1)T(q_1, \sigma_2, q_2) \cdots T(q_{i-1}, \sigma_i, q_i) \cdot \bar{u}_i$ with $\bar{u}_k = V(q_k) \in N_{q_k}$. Since P is deterministic, $\text{Amp}(q, \bar{v}, \sigma_1 \cdots \sigma_k) = \bar{v} \cdot T(q, \sigma_1, q_1) \cdots T(q_{k-1}, \sigma_k, q_k) = \bar{v} \cdot \bar{u} \neq 0$, contrary to our assumption.

(\Rightarrow) This direction is proved by induction on $|x|$, assuming $\bar{v} \perp S_q$. If $x = \Lambda$ then q is an accepting state and hence $\bar{v} \perp S_q$ implies $\bar{v} \cdot V(q) = 0$. Since in this case $\text{Amp}(q, \bar{v}, \Lambda) = \bar{v} \cdot V(q)$, we are done.

Assume the conclusion to be true for all states q and for all $x \in \Sigma^*$ such that $|x| < k$. Consider the word $x' = \sigma x$, and an edge (q, σ, q') labeled by $\bar{Z} \leftarrow \bar{Z} \cdot M + \bar{\eta} \cdot y$. Obviously, since P is deterministic, we have $\text{Amp}(q, \bar{v}, \sigma x) = \text{Amp}(q', \bar{v} \cdot M, x)$. Thus

$$\begin{aligned} \bar{v} \perp S_q &\Leftrightarrow \forall \bar{u} \in S_{q'} [\bar{v} \cdot \bar{u} = 0] \\ &\Rightarrow \forall \bar{u}' \in S_{q'} [\bar{v} \cdot (M \cdot \bar{u}') = 0] \\ &\Leftrightarrow \forall \bar{u}' \in S_{q'} [(\bar{v} \cdot M) \cdot \bar{u}' = 0] \\ &\Leftrightarrow \bar{v} \cdot M \perp S_{q'}, \end{aligned}$$

where the implication is justified by the construction which ensured that $M \cdot S_{q'} \subseteq S_q$. Hence, by the induction hypothesis, $\text{Amp}(q', \bar{v} \cdot M, x) = 0$, which implies the required result. \square

Now let P be any deterministic program, q_0 its initial state, $A(q_0)$ the corresponding initial vector. For each edge e of P let $\bar{Z} \leftarrow \bar{Z} \cdot T(e) + U(e) \cdot y$ be the action labeling it, and assume that the subspaces S_q have been calculated, for each state q of P , as described above. The following lemma follows easily from Lemma 3.2.

LEMMA 3.3. P is ZC if and only if

- (1) $A(q_0) \perp S_{q_0}$ and
- (2) for each edge e entering state q , $U(e) \perp S_q$.

Once the zero amplification spaces S_q have been computed for each q it is possible to bring any FSLP program into a form which is easier to analyze.

DEFINITION 3.4. An FSLP program P is said to be *proper* if (1) P is deterministic, (2) $A(q_0) \in S_{q_0}$ (q_0 is the initial state of P), and (3) for each edge (q', σ, q) of P , $U(q', \sigma, q) \in S_q$.

There is an obvious advantage to a proper program in that it does not operate on input values y if they do not participate in the output. Thus the following is an immediate consequence of the above definition.

COROLLARY 3.5. A proper FSLP program is ZC if and only if $A(q_0) = 0$ and for each edge (q, σ, q') , $U(q, \sigma, q') = 0$.

In order to show that every program can be brought to a proper form we first bring a linearity result which can be proved easily by induction, using the definitions of Val and Out functions.

LEMMA 3.6. Let $P_i = (\mathcal{A}, \mathcal{F}_i)$, $\mathcal{F}_i = (\bar{Z}, T, U^i, A^i, V)$, $i = 1, 2$, be two FSLP programs differing only in their initial (the A -) and nonhomogeneous (the U -) components. Then $P = (\mathcal{A}, \mathcal{F})$, where $\mathcal{F} = (\bar{Z}, T, U^1 + U^2, A^1 + A^2, V)$, satisfies $\text{Out}_P(w) \equiv \text{Out}_{P_1}(w) + \text{Out}_{P_2}(w)$ for every $w \in (\Sigma \times R)^*$.

LEMMA 3.7. Every FSLP program can be brought to proper form.

Proof. Let $P = (\mathcal{A}, \mathcal{F})$. Define two programs $P_i = (\mathcal{A}, \mathcal{F}_i)$, $\mathcal{F}_i = (\bar{Z}, T, U^i, A^i, V)$, $i = 1, 2$, as follows. $U^1(e) = \text{pr}(U(e), S_q)$ for every edge $e = (q', \sigma, q)$; i.e., $U^1(e)$ is the orthogonal projection of $U(e)$ on S_q (\bar{u} is an orthogonal projection of \bar{v} on X if $\bar{v} = \bar{u} + \bar{w}$ where \bar{w} is orthogonal to X). Also define $A^1(q_0) = \text{pr}(A(q_0), S_{q_0})$. Similarly, $U^2(e) = U(e) - U^1(e)$ for every edge e of P , and $A^2(q_0) = A(q_0) - A^1(q_0)$. Note that $U^2(q_i, \sigma, q_j) \perp S_{q_j}$ and $A^2(q_0) \perp S_{q_0}$. Thus P_2 is ZC. By Lemma 3.6, $\text{Out}_P(w) \equiv \text{Out}_{P_1}(w) + \text{Out}_{P_2}(w) \equiv \text{Out}_{P_1}(w)$ since $\text{Out}_{P_2}(w) \equiv 0$ (or ω). Hence P_1 is a proper FSLP program equivalent to P . \square

In order to test the equivalence of two programs P_1 and P_2 we first check that their domains are identical; i.e., they are defined for exactly the same set of words. This is done by testing the underlying finite automata \mathcal{A}_1 and \mathcal{A}_2 for equivalence. Once the domains are found to be equal we construct the program $P = P_1 - P_2$ as follows.

Let $P_i = (\mathcal{A}_i, \mathcal{F}_i)$ with $\mathcal{A}_i = (K_i, \Sigma, \delta_i, B_i, F_i)$, $\mathcal{F}_i = (\bar{Z}^i, T_i, U_i, A_i, V_i)$, $i = 1, 2$, be the two given programs. We may assume that the dimensions of \bar{Z}^1 and \bar{Z}^2 are equal. If they are not, the shorter can be extended to match the longer. We define $P = (\mathcal{A}, \mathcal{F})$ with $K = K_1 \cup K_2$, $\bar{Z} = \bar{Z}^1 = \bar{Z}^2$, $\delta = \delta_1 \cup \delta_2$, $T = T_1 + T_2$, $B = B_1 \cup B_2$, $A = A_1 + A_2$, $F = F_1 \cup F_2$, $V = V_1 - V_2$ and $U = U_1 + U_2$.

In the definition of V we used our convention that $a \pm \omega \equiv \omega \pm a \equiv a$ (for $a \in R$ and ω the undefined value). Thus if $q \in F_1$ then $q \notin F_2$, implying $V(q) \equiv V_1(q) - V_2(q) \equiv V_1(q) - \omega \equiv V_1(q)$, and similarly in the case of $q \in F_2$.

P is actually the nondeterministic union of the two programs with $V_2(q)$, for $q \in F_2$, negated. Since the computation in disjoint nondeterministic programs is the sum of the computations in the two components, the result will be zero if and only if the computations in P_1 and P_2 produce the same results. Thus $P_1 \approx P_2$ if and only if both

have the same domain and $P = P_1 - P_2$ is a ZC program. To determine whether P is ZC we first obtain a deterministic program \hat{P} which is equivalent to P and then test it for the ZC property. We summarize the above discussion in the following theorem.

THEOREM 3.8. *The properties of ZC and equivalence are decidable for FSLP programs.*

Example 3.9. The bases of the ZAS for the program P_3 (see Example 2.7) are the following:

$$N_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad N_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad N_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Note that P_3 is already in proper form.

4. Optimization. Most of the constructions illustrated above, as well as initial designs produced by programmers, are usually less than optimal in the sense that the FSLP programs are not necessarily the most economical. Therefore, it would be most desirable if we could offer an optimization procedure which for a given program P derives its most economical counterpart P' . When measuring the structural complexity of FSLP programs two criteria should be considered: (a) number of states, and (b) number of registers. Unfortunately there is a tradeoff between the two, and one of them can be optimized only at the expense of the other. Since for nondeterministic programs the state optimization problem is not tractable even for the finite automaton case, and the register optimization problem is trivially solvable by Theorem 2.5, we restrict our discussion to deterministic programs.

This section is divided into two parts. The first part treats state optimization, and the results are similar to those in the usual automata theory. The second and the central part discusses a register minimization procedure which keeps the state structure fixed. We obtain an algorithm that minimizes the number of registers subject to a given control configuration, i.e., finds the program with the minimal number of registers which is equivalent to a given program and has the same state structure.

4.1. State optimization. As in the finite automaton case, the state reduced (optimal) program P' equivalent to a given program P can always be obtained by identifying equivalent states. For a program P and a state q of P , we define the program P_q as the program P modified by taking q to be the initial state with initial value $A(q) = 0$. The appropriate equivalence relation (on the state set) turns out to be the following.

DEFINITION 4.1.1. Two states q_1 and q_2 of P are said to be *equivalent*, $q_1 \cong q_2$, if $P_{q_1} \approx P_{q_2}$.

Given a program P we can partition its states into equivalence classes (using, for example, the equivalence test of § 3). Let $[q]$ denote the equivalence class containing all states equivalent to q . Let N be the number of different equivalence classes of states in P .

LEMMA 4.1.2. *In a proper FSLP program P , let $q_1 \cong q_2$ be such that $\delta(q_1, \sigma) = \delta(q_2, \sigma) = q$. Then $U(q_1, \sigma, q) = U(q_2, \sigma, q)$.*

Proof. Obviously, if $q_1 \cong q_2$, then all computations starting from q_1 and q_2 respectively (with initial value 0) should yield identical results. Thus in particular if the first read letter is σ , leading both q_1 and q_2 into q , this implies $\text{Amp}(q, U(q_1, \sigma, q), x) = \text{Amp}(q, U(q_2, \sigma, q), x)$ for all $x \in \Sigma^*$. By linearity $\text{Amp}(q, U(q_1, \sigma, q) - U(q_2, \sigma, q), x) = 0$ for all $x \in \Sigma^*$. By Lemma 3.2, $U(q_1, \sigma, q) - U(q_2, \sigma, q) \perp S_q$. P being proper implies $U(q_1, \sigma, q) - U(q_2, \sigma, q) \in S_q$, which is possible only if $U(q_1, \sigma, q) = U(q_2, \sigma, q)$. \square

THEOREM 4.1.3. *Let N be the number of different equivalence classes of states in P . Then there exists an FSLP program P' such that $P \approx P'$ and P' has N states.*

Proof. Let P be a proper FSLP program given by $P = (\mathcal{A}, \mathcal{F})$ with $K = \{q_1, \dots, q_m\}$, q_1 the initial state and $\dim \bar{Z} = n$. Let $n' = m \cdot n$. We define $P' = (\mathcal{A}', \mathcal{F}')$, where $K' = \{[q] \mid q \in K\}$; i.e., each state of P' is an equivalence class of states of P . The initial state of P' is $[q_1]$. $\bar{Z}' = (\bar{Z}^1, \dots, \bar{Z}^m)$ where $\dim \bar{Z}^i = n$. Thus \bar{Z}' is an n' -dimensional supervector having a copy of the original register vector \bar{Z} , for each state $q_i \in K$. $\delta' : K' \times \Sigma \rightarrow K'$ is defined by $\delta'([q], \sigma) = [\delta(q, \sigma)]$. It is easy to show that this definition is consistent and that it makes P' deterministic. T' yields $n' \times n'$ matrix for each edge of P' . Actually, T' depends only on σ and is given by block form

$$T'(\sigma) = (T'_{ij}(\sigma)), \quad 1 \leq i, j \leq m, \quad \text{where } T'_{ij}(\sigma) = T(q_i, \sigma, q_j).$$

U' is given by the block form $U'([q], \sigma, [q']) = (U^1, \dots, U^m)$ where $U^j = U(q, \sigma, q_i)$ for $j = 1, 2, \dots, m$. By Lemma 4.1.2, this definition is independent of the state chosen to represent $[q]$, but the same representative should be used to define all the U^j 's. Note also that $U'([q], \sigma, [q'])$ is independent of $[q']$. $A'([q_1])$ is n' -dimensional supervector whose block form is $A'([q_1]) = (A(q_1), A(q_2), \dots, A(q_m))$. Note that since P is deterministic, the n -components corresponding to $q_i \neq q_1$ will be explicitly zero. $F' = \{[q] \mid q \in F\}$. Again the definition of equivalence implies identical acceptance behavior; i.e., $q_1 \equiv q_2$ implies $[q_1] \in F$ iff $[q_2] \in F$. V' yields for each $[q]$ in F' an n' -dimensional (column) supervector defined by $V'([q]) = ((V^1)^T, (V^2)^T, \dots, (V^m)^T)^T$, where $V^i = V(q_i)$ if $q_i \in F$ and $V^i = 0$ otherwise. We extend V' as usual by setting $V'([q]) \equiv \omega$ for $[q] \notin F'$. This completes the definition of P' . We omit the proof that $P \approx P'$. \square

Actually, at some expense of simplicity (of construction and proof) we could have reduced the number n' (in the above proof) to be $n' = k \cdot n$, where k is the maximal number of states (of P) in any equivalence class $[q]$. Further improvements, not using register reduction (of the next section) are not obvious.

In order to prove that the program P' of Theorem 4.1.3 has in fact a minimal number of states, we shall show that there cannot exist an equivalent program with less than N states. For our FSLP programs we assume that every state is accessible (i.e., the programs do not contain useless states).

LEMMA 4.1.4. *Let P and P' be FSLP programs, with state sets K and K' respectively, and let $P \approx P'$. Then for each $q \in K$ there exists $q' \in K'$ such that $P_q \approx P'_{q'}$.*

Proof. It is easy to see that $P_{q_1} \approx P'_{q'_1}$ (q_1 and q'_1 being the initial states of P and P' respectively). Were this not the case, then, since the contribution of the initial vector for a given $w \in (\Sigma \times R)^*$ is constant, we could choose the R -values in w , so as to make $\text{Out}_P(w) \neq \text{Out}_{P'}(w)$, which contradicts $P \approx P'$. Now it is easy to see that $P_q \approx P'_{q'}$ implies $P_{\delta(q, \sigma)} \approx P'_{\delta'(q', \sigma)}$ for every $\sigma \in \Sigma$. Thus, since every q in P is accessible from q_1 , say $q = \delta(q_1, \lambda(w))$, we can take $q' = \delta'(q'_1, \lambda(w))$ (being sure of its existence) and get $P_q \approx P'_{q'}$. \square

Now let P be an FSLP program which is reduced in the sense of Theorem 4.1.3, i.e., for any q_i and q_j in K , $P_{q_i} \not\approx P_{q_j}$. Let P' be an FSLP program equivalent to P , but assume that P' has fewer states. By Lemma 4.1.4 each state in P is equivalent to some state in P' , and there are fewer states in P' than in P . Hence there must exist two states q_i and q_j in K and a state q' in K' such that $P_{q_i} \approx P'_{q'}$ and $P_{q_j} \approx P'_{q'}$, implying $P_{q_i} \approx P_{q_j}$ in contradiction to P being reduced. Thus we have the following corollary.

COROLLARY 4.1.5. *The FSLP program P' defined in Theorem 4.1.3 is a statewise optimal program for P , i.e., for any program P'' equivalent to P , the number of states of P'' is equal to or greater than the number of states of P' .*

For example, it is easy to check that the FSLP program of Fig. 3 is already statewise optimal, i.e., no two states of that program are equivalent.

4.2. Reduction of the number of registers. We now present a method for reducing the number of registers of an FSLP program while retaining its state structure together with all its states. For the scope of the subsequent discussion we introduce a slightly extended version of the FSLP model. In this (still deterministic) model we do not insist on the program having a common set of registers, but instead, each state $q_i \in K$ may have its private register vector $Z(q_i)$. As a result, different states may have register vectors of different dimensions and correspondingly the transformation matrices need not be square. The only motivation for introducing this complication is to make the description of the register reduction procedure easier.

DEFINITION 4.2.1. An *extended FSLP program* (EFSLP) is a triple $P = (\mathcal{A}, \mathcal{F}, d)$, where \mathcal{A} is defined as usual and $\mathcal{F} = (Z, T, U, A, V)$ where Z is a mapping associating with each $q \in K$ its private register vector $Z(q)$. d is the *dimension mapping* such that for each q in K , $d(q) = \dim Z(q)$; $T(q, \sigma, q')$ is now a rectangular $d(q) \times d(q')$ matrix; $U(q, \sigma, q')$ is a $d(q')$ -dimensional vector; $A(q_1)$ is the initial $d(q_1)$ -dimensional (row) vector and for $q \in F$, $V(q)$ is a $d(q)$ -dimensional (column) vector. All mappings are extended as in Definition 2.1.

The next lemma follows immediately from the definitions.

LEMMA 4.2.2. *The models of FSLP and EFSLP are intertranslatable.*

Let P be an FSLP program. Following the S_q -algorithm we can construct for each state q in K its zero amplification subspace (ZAS) S_q . These subspaces are represented by rectangular matrices. Thus for q_i in K we have a $d(q_i) \times f(q_i)$ matrix with $f(q_i)$ ($\cong d(q_i)$) being the rank of N_i . Hence we can construct a (not necessarily unique) pseudo-inverse N_i^\dagger such that $N_i^\dagger \cdot N_i = I_{f(q_i)}$, $I_{f(q_i)}$ being the identity matrix of order $f(q_i)$ (see [2] for a detailed study of properties of such matrices). The fact that N_i represents the ZAS at state q_i is reflected by the property that for each row $d(q_i)$ -vector \bar{u} , $\bar{u} \cdot N_i = 0$ if and only if for all $x \in \Sigma^*$ $[\text{Amp}(q_i, \bar{u}, x) = 0]$. $f(q_i)$ is called the *forward rank* of state q_i . Going back to the construction of the N_i we recall that for each (column) vector $\bar{v} \in N_i$ there exists a path from q_i to some accepting state such that if the register value at q_i is $\bar{Z}(q_i) = \bar{u}$ the output value produced at the end of this path is $\bar{u} \cdot \bar{v}$. Thus there exist $f(q_i)$ input words $x_1, \dots, x_{f(q_i)}$ in $(\Sigma \times \{0\})^*$ (i.e., the R -components being zero) such that for any initial register value at q_i , \bar{u} say, $\bar{u} \cdot N_i$ is the set of output values produced by the $f(q_i)$ computations induced by these words, leading from q_i to the respective accepting states.

Similarly to the “forward matrices” N_i we can construct “backward matrices” X_i for each $q_i \in K$. The subspaces represented by these matrices are the subspaces spanned by all possible values of $Z(q_i)$ for all possible computations starting from the initial state q_1 . Following is the procedure for constructing the X_i ’s.

- (1) Initially set $X_1 = \{A(q_1)\}$ if $A(q_1) \neq 0$; otherwise set $X_1 = \emptyset$; for $i \neq 1$ set $X_i = \emptyset$.
- (2) If $q_j = \delta(q_i, \sigma)$ and $U(q_i, \sigma, q_j) \notin L(X_j)$ adjoin $U(q_i, \sigma, q_j)$ to X_j . Repeat step (2) for all edges (q_i, σ, q_j) .
- (3) If $q_j = \delta(q_i, \sigma)$ and $\bar{u} \in X_i$ and $\bar{v} = \bar{u} \cdot T(q_i, \sigma, q_j) \notin L(X_j)$ adjoin \bar{v} to X_j . Repeat step (3) until a complete cycle including all edges has not increased any of the X_i .

In the above, X_j is a set of row vectors comprising a rectangular matrix, and $\bar{v} \in L(X_j)$ if \bar{v} is a linear combination of the rows of X_j . The final X_i ’s are called *attainability matrices* at state q_i ; they are $b(q_i) \times d(q_i)$ matrices of rank $b(q_i)$. We may

assume that (each) X_i has the property that each row in X_i is $\text{Val}_P(w, q_i)$ for some input word $w \in (\Sigma \times R)^*$. Were this not the case we could easily modify X_i while keeping the same row space (i.e., we may change the basis of the row space). As in the case of the N_i 's we can construct the pseudo-inverse matrices X_i^\dagger such that for each state q_i in K , $X_i^\dagger \cdot X_i = I_{b(q_i)}$.

DEFINITION 4.2.3. An EFSLP program P is called *economical* if for each state q_i in K , $b(q_i) = d(q_i) = f(q_i)$.

THEOREM 4.2.4. Each EFSLP program P is equivalent to an economical EFSLP program P'' with identical (state) structure.

Proof. Let P be given by $P = (\mathcal{A}, \mathcal{F}, d)$. The procedure will consist of two steps. First we will obtain an EFSLP program P' such that $b'(q_i) \subseteq d'(q_i) = f'(q_i) = f(q_i)$ and then construct an EFSLP program P'' such that $b'(q_i) = b''(q_i) = d''(q_i) = f''(q_i)$.

Assume that the matrices N_i have been computed for each state q_i of P . Define $P' = (\mathcal{A}', \mathcal{F}', d')$, where $Z'(q_i) = (Z_1 \cdots Z_{d'(q_i)})$; $d'(q_i) = f(q_i)$; $T'(q_i, \sigma, q_j) = N_i^\dagger \cdot T(q_i, \sigma, q_j) \cdot N_j$; $U'(q_i, \sigma, q_j) = U(q_i, \sigma, q_j) \cdot N_j$; $A'(q_1) = A(q_1) \cdot N_1$; $V'(q_i) = N_i^\dagger \cdot V(q_i)$.

It is easy to check that the dimensions match and all assignments to $Z'(q_i)$ in fact have dimension $f(q_i)$. In order to show that $P \approx P'$, we first prove the following claim:

(*) If $q_i = \delta(q_1, \lambda(w))$ then $\text{Val}_{P'}(w, q_i) = \text{Val}_P(w, q_i) \cdot N_i$.

The proof is by induction on $|w|$. Let $w = \Lambda$; then $\text{Val}_{P'}(\Lambda, q_1) = A'(q_1) = A(q_1) \cdot N_1 = \text{Val}_P(\Lambda, q_1) \cdot N_1$. Assume that (*) is true for w , and let $w' = w \cdot (\sigma, y)$, with $q_j = \delta(q_i, \sigma)$. Then

$$\begin{aligned} \text{Val}_{P'}(w', q_j) &= \text{Val}_{P'}(w, q_i) \cdot T'(q_i, \sigma, q_j) + U'(q_i, \sigma, q_j) \cdot y \\ &= \text{Val}_P(w, q_i) \cdot N_i \cdot N_i^\dagger \cdot T(q_i, \sigma, q_j) \cdot N_j + U(q_i, \sigma, q_j) \cdot N_j \cdot y. \end{aligned}$$

By the construction of the N_i 's it was ensured that $T(q_i, \sigma, q_j) \cdot N_j \subseteq L(N_i)$ (i.e., $T(q_i, \sigma, q_j) \cdot \bar{v} \in L(N_i)$ for each column \bar{v} of N_j). It is also obvious that for each $\bar{v} \in L(N_i)$, $N_i \cdot N_i^\dagger \cdot \bar{v} = \bar{v}$.

Thus

$$\begin{aligned} \text{Val}_{P'}(w', q_j) &= [\text{Val}_P(w, q_i) \cdot T(q_i, \sigma, q_j) + U(q_i, \sigma, q_j) \cdot y] \cdot N_j \\ &= \text{Val}_P(w', q_j) \cdot N_j \end{aligned}$$

which proves (*). Now, if $\delta(q_1, \lambda(w)) = q_k$ is in F then, since $V(q_k)$ is in $L(N_k)$ and hence $N_k \cdot N_k^\dagger \cdot V(q_k) = V(q_k)$, we obtain

$$\begin{aligned} \text{Out}_{P'}(w) &= \text{Val}_{P'}(w, q_k) \cdot V'(q_k) \\ &= \text{Val}_P(w, q_k) \cdot N_k \cdot N_k^\dagger \cdot V(q_k) \\ &= \text{Val}_P(w, q_k) \cdot V(q_k) \\ &= \text{Out}_P(w). \end{aligned}$$

This proves that $P \approx P'$. For the derived P' it would have been possible to recalculate the ZAS S'_{q_i} for each q_i . However, this is not necessary. By following the construction of the original S_{q_i} in P , we can observe a constant relation between vectors in S_{q_i} and vectors in S'_{q_i} . The relation is that for each vector \bar{v} , $\dim \bar{v} = d(q_i)$, $\bar{v} \in S_{q_i}$ implies $N_i^\dagger \cdot \bar{v} \in S'_{q_i}$. Hence $N_i^\dagger \cdot S_{q_i} \subseteq S'_{q_i}$. However, $N_i^\dagger \cdot S_{q_i} = N_i^\dagger \cdot L(N_i) = L(N_i^\dagger \cdot N_i) = L(I_{f(q_i)}) \subseteq S'_{q_i}$. Thus the rank of S'_{q_i} is at least $f(q_i)$, but the rank being also at most

$d'(q_i) = f(q_i)$, it follows that N'_i must be of full rank, and is actually the identity matrix $I_{f(q_i)}$. Consequently $d'(q_i) = f'(q_i)$ holds for P' .

We now calculate the attainability matrices X'_i for each state q_i of P' as outlined before. X'_i is a $b'(q_i) \times d'(q_i)$ matrix. Define an EFSLP program P'' by $P'' = (\mathcal{A}, \mathcal{F}'', d'')$, where $Z''(q_i) = (Z_1, \dots, Z_{d'(q_i)})$; $d''(q_i) = b'(q_i)$; $T''(q_i, \sigma, q_j) = X'_i \cdot T'(q_i, \sigma, q_j) \cdot X_j'^{\dagger}$; $U''(q_i, \sigma, q_j) = U'(q_i, \sigma, q_j) \cdot X_j'^{\dagger}$; $A''(q_1) = A'(q_1) \cdot X_1'^{\dagger}$; $V''(q_i) = X'_i \cdot V'(q_i)$. Again it is routine to check that the dimensions are compatible. To show that $P' \approx P''$ we first prove the following assertion.

$$(**) \quad \text{If } q_i = \delta(q_1, \lambda(w)) \text{ then } \text{Val}_{P''}(w, q_i) = \text{Val}_{P'}(w, q_i) \cdot X_i'^{\dagger}.$$

The proof is by induction on $|w|$. For $w = \Lambda$ we have $\text{Val}_{P''}(\Lambda, q_1) = A''(q_1) = A'(q_1) \cdot X_1'^{\dagger} = \text{Val}_{P'}(\Lambda, q_1) \cdot X_1'$. Assume that $(**)$ holds for w , and consider $w' = w \cdot (\sigma, y)$, where $\delta(q_i, \sigma) = q_j$.

$$\begin{aligned} \text{Val}_{P''}(w', q_j) &= \text{Val}_{P''}(w, q_i) \cdot T''(q_i, \sigma, q_j) + U''(q_i, \sigma, q_j) \cdot y \\ &= \text{Val}_{P'}(w, q_i) \cdot X_i'^{\dagger} \cdot X'_i \cdot T'(q_i, \sigma, q_j) \cdot X_j'^{\dagger} + U'(q_i, \sigma, q_j) \cdot X_j'^{\dagger} \cdot y. \end{aligned}$$

By the construction of the X'_i 's $\text{Val}_{P'}(w, q_i) \in L(X'_i)$ and therefore $\text{Val}_{P'}(w, q_i) \cdot X_i'^{\dagger} \cdot X'_i = \text{Val}_{P'}(w, q_i)$. Thus

$$\begin{aligned} \text{Val}_{P''}(w', q_j) &= [\text{Val}_{P'}(w, q_i) \cdot T'(q_i, \sigma, q_j) + U'(q_i, \sigma, q_j) \cdot y] \cdot X_j'^{\dagger} \\ &= \text{Val}_{P'}(w', q_j) \cdot X_j'^{\dagger}, \end{aligned}$$

which proves $(**)$. Assuming now that $\delta(q_1, \lambda(w)) = q_k$ is in F we have, arguing as before, that

$$\begin{aligned} \text{Out}_{P''}(w) &\equiv \text{Val}_{P''}(w, q_k) \cdot V''(q_k) \equiv \text{Val}_{P'}(w, q_k) \cdot X_k'^{\dagger} \cdot X'_k \cdot V'(q_k) \\ &\equiv \text{Val}_{P'}(w, q_k) \cdot V'(q_k) \equiv \text{Out}_{P'}(w). \end{aligned}$$

For P'' we have $d''(q_i) = b'(q_i)$ by definition. In order to prove that P'' is economical we have to show that $f''(q_i) = b''(q_i) = d''(q_i)$. In fact, it is easy to see from $(**)$ that the new attainability matrix at state q_i is $X''_i = X'_i \cdot X_i'^{\dagger} = I_{b'(q_i)}$. Thus $b''(q_i) = b'(q_i)$. Now, in view of $N'_i = I_{d'(q_i)}$, the rank of $X'_i \cdot N'_i$ is $b'(q_i)$ and so is the rank of N''_i (which consists of just the independent columns of $X'_i \cdot N'_i$). We conclude that $f''(q_i) = b'(q_i) = b''(q_i) = d''(q_i)$. \square

We wish to show now that an economical EFSLP program utilizes a minimal number of registers at each of its states. We shall make this statement precise. Let $P = (\mathcal{A}, \mathcal{F}, d)$ be an EFSLP program and denote by $[P]$ the set of all EFSLP programs which satisfy (1)–(2) below.

(1) The control structure of each member of $[P]$ is identical to that of P , i.e., for each R in $[P]$, $R = (\mathcal{A}_R, \mathcal{F}_R, d^R)$ and $\mathcal{A}_R = \mathcal{A}$.

(2) For each $R \in [P]$, $R \approx P$.

Thus it makes sense to talk about some particular state q_i in each program in $[P]$. For $R \in [P]$ denote by $d^R(q_i)$ the dimension of the register vector in R at state q_i . We shall fix our attention on an arbitrary state q_i .

Consider any r ($r > 0$) different input words w_j , $j = 1, \dots, r$, which lead from q_1 to q_i . For R in $[P]$ each such word yields, on reaching q_i , a $d^R(q_i)$ -dimensional row vector,

\bar{u}_j which is the current value of the register vector as computed on w_j . Let us construct an $r \times d^R(q_i)$ matrix C_R^r whose rows are $\bar{u}_j, j = 1 \dots, r$.

Now consider s ($s > 0$) different words $x_k, k = 1, \dots, s$, which lead from q_i to any accepting state in F and whose numerical components are all zero. For R in $[P]$ any such word induces a linear homogeneous transformation from the initial value of $Z^R(q_i)$ (the register vector of R at q_i) to an output value. For each x_k this transformation can be represented by a column vector \bar{v}_k such that for any initial register vector \bar{u} , $\text{Amp}(q_i, \bar{u}, \lambda(x_k)) = \bar{u} \cdot \bar{v}_k$. Again we construct a $d^R(q_i) \times s$ matrix, D_R^s , with columns $\bar{v}_k, k = 1, \dots, s$.

Now consider the matrix $E^{q_i} = (E_{jk}), j = 1, \dots, r, k = 1, \dots, s$, where $E_{jk} = \text{Out}_R(w_j \cdot x_k)$, i.e., E^{q_i} comprises the results of computations for all input words formed by concatenating the w_j 's and the x_k 's. It is clear that $E^{q_i} = C_R^r \cdot D_R^s$. Since all programs in $[P]$ are equivalent, it follows that for any given $r \cdot s$ computations satisfying the specifications indicated above, E^{q_i} is independent of the particular program taken to carry out these computations. It is obvious also that for all R in $[P]$, $\text{rank}(E^{q_i}) \leq d^R(q_i)$. If we take now the maximum of $\text{rank}(E^{q_i})$ over all possible $r > 0$ and $s > 0$, and all possible r - s -computations we obtain a number $\rho(q_i)$ which still satisfies $\rho(q_i) \leq d^R(q_i)$ for all $R \in [P]$.

Thus it follows that $\rho(q_i)$ is a lower bound on the number of registers necessary at q_i for each program in $[P]$. Now let $Q \in [P]$ be an economical program. By the definition of $b^Q(q_i)$ there exist $b^Q(q_i)$ distinct input words $w'_j, j = 1, \dots, r = b^Q(q_i)$, such that the constructed matrix $C'_O (= X_i)$ is of rank $b^Q(q_i)$. Similarly there exist $f^Q(q_i)$ words $x'_k, k = 1, \dots, s = f^Q(q_i)$, such that $D'_O (= N_i)$ is of rank $f^Q(q_i)$. Since Q being economical implies $b^Q(q_i) = d^Q(q_i) = f^Q(q_i)$, the particular matrix $E' = C'_O \cdot D'_O$ is also of rank $d^Q(q_i)$. Q in $[P]$ implies therefore that $\rho(q_i) \leq d^Q(q_i)$ while maximality of $\rho(q_i)$ implies $\rho(q_i) \geq d^Q(q_i)$; thus $\rho(q_i) = d^Q(q_i)$. We have proved:

THEOREM 4.2.5. *Let P and R be FSLP programs such that R is in $[P]$. If P is economical then for each state q_i, P has no more registers than R has at q_i .*

The above discussion shows that after transforming a given FSLP program into an equivalent minimal state version, we may then make it economical without compromising its state minimality. In an obvious sense, minimal-state economical EFSLP programs are the best programs we can hope for, if state minimality is our chief concern.

EXAMPLE 4.2.6. To minimize the number of registers of the program P_3 (see Fig. 3) while keeping its state structure fixed, it is necessary to "go backwards and forwards" as in the proof of Theorem 4.2.4. Using the matrices N_i of Example 3.9, computing their generalized inverses and applying the "backward process" we obtain the program P'_3 of Fig. 4.

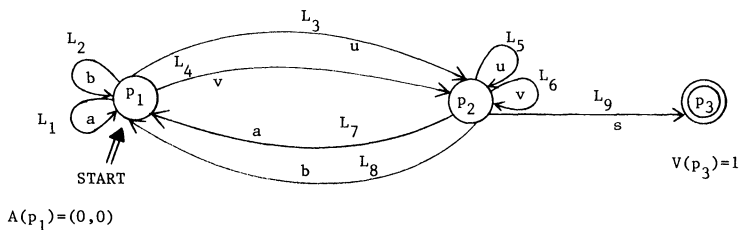


FIG. 4

In Fig. 4 the actions L_i are the following:

$$\begin{aligned}
 L_1: \bar{Z}_1 &\leftarrow \bar{Z}_1 \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + (4, 3) \cdot y, & L_2: \bar{Z}_1 &\leftarrow \bar{Z}_1 \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + (6, 7) \cdot y, \\
 L_3: \bar{Z}_2 &\leftarrow \bar{Z}_1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, & L_4: \bar{Z}_2 &\leftarrow \bar{Z}_1 \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \\
 L_5: \bar{Z}_2 &\leftarrow \bar{Z}_2 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, & L_6: \bar{Z}_2 &\leftarrow \bar{Z}_2 \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
 L_7: \bar{Z}_1 &\leftarrow \bar{Z}_2 \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} + (4, 3) \cdot y, & L_8: \bar{Z}_1 &\leftarrow \bar{Z}_2 \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} + (6, 7) \cdot y, \\
 L_9: \bar{Z}_3 &\leftarrow \bar{Z}_2 \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
 \end{aligned}$$

For P'_3 we compute the matrices X_i :

$$X_1 = \begin{bmatrix} 4 & 3 \\ 6 & 7 \end{bmatrix}, \quad X_2 = [3 \quad 3 \quad 3], \quad X_3 = [3],$$

and their pseudo-inverses:

$$X_1^\dagger = \begin{bmatrix} \frac{7}{10} & -\frac{3}{10} \\ -\frac{3}{5} & \frac{2}{5} \end{bmatrix}, \quad X_2^\dagger = \begin{bmatrix} \frac{1}{3} \\ 0 \\ 0 \end{bmatrix}, \quad X_3^\dagger = [\frac{1}{3}].$$

Applying the ‘‘forward process’’ we construct a program P''_3 , having the same control structure as P'_3 , with the following differences, $A''(p_1) = (0, 0)$ as in P'_3 ; $V''(p_3) = 3$ and the actions L_i are now

$$\begin{aligned}
 L_1: \bar{Z}_1 &\leftarrow \bar{Z}_1 \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + (1, 0) \cdot y, & L_2: \bar{Z}_1 &\leftarrow \bar{Z}_1 \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + (0, 1) \cdot y, \\
 L_3: \bar{Z}_2 &\leftarrow \bar{Z}_1 \cdot \begin{bmatrix} 1 \\ \frac{7}{3} \end{bmatrix}, & L_4: \bar{Z}_2 &\leftarrow \bar{Z}_1 \cdot \begin{bmatrix} \frac{4}{3} \\ 2 \end{bmatrix}, \\
 L_7: \bar{Z}_1 &\leftarrow \bar{Z}_2 \cdot [\frac{3}{10} \quad \frac{3}{10}] + (1, 0) \cdot y, & L_8: \bar{Z}_1 &\leftarrow \bar{Z}_2 \cdot [\frac{3}{10} \quad \frac{3}{10}] + (0, 1) \cdot y
 \end{aligned}$$

L_5, L_6 and L_9 are identity transformation. Observe that the dimensions of the final register vectors are $\dim \bar{Z}_1 = 2$, $\dim \bar{Z}_2 = \dim \bar{Z}_3 = 1$, and hence two is the minimal (and thus optimal) number of registers needed to carry out the computation of P_3 (keeping the control structure fixed). \square

5. Conclusion. It is clear that the model studied in this paper is but a first step in the general direction of identifying decidable and analyzable programming models. It is our belief that a very large part of the actual programming done today (in particular in data processing applications) naturally falls into one of the analyzable categories, and we would like therefore to encourage further research along these lines.

Several possible extensions to our model have been contemplated. The first is to remove the restriction on computational data testing. This has to be done carefully so as

not to admit to full power of a Turing machine. An example of such a careful relaxation is to allow a test on the current numeric values (input as well as registers) provided we can always modify the most recent input so that the test will take any of its possible branches. This corresponds to the notion of a "free program" in analogy to the concept of free schemes. For some results along these lines see [6].

Another useful type of extension is to allow output of an output file rather than a single output value. This will lead to generalizations of finite state transducers, and may enable modeling of a much larger class of data processing applications.

REFERENCES

- [1] R. M. BALZER, *Automatic Programming*, Tech. Memo, Information Sciences Institute, Univ. of Southern Calif., 1973.
- [2] A. BEN-ISRAEL AND T. N. E. GREVILLE, *Generalized Inverses: Theory and Applications*, Wiley, New York, 1974.
- [3] J. DARLINGTON AND R. M. BURSTALL, *A system which automatically improves programs*, Adv. Papers 3rd Intl. Conf. of AI, Stanford University, 1973.
- [4] S. EILENBERG, *Automata, Languages and Machines, vol. A*, Academic Press, New York, 1974.
- [5] Z. MANNA AND R. WALDINGER, *Toward automatic program synthesis*, Comm. ACM, 14 (1971), pp. 151-165.
- [6] A. PNUELI AND G. SLUTZKI, *Simple programs and their decision problems*, Automata, Languages and Programming, Fourth Colloquium, Turku, Finland, LNCS 52, Springer-Verlag, New York, 1977.
- [7] G. SLUTZKI, *Logical Analysis of Simple Programs*, Doctoral dissertation, Tel-Aviv University, Tel-Aviv, 1976.
- [8] N. SUZUKI AND D. JEFFERSON, *Verification decidability of Presburger array programs*, Conf. on Theoretical Computer Science, University of Waterloo, Waterloo, Ontario, 1977.

PUMPING LEMMAS FOR REGULAR SETS*

A. EHRENFUCHT†, R. PARIKH‡ AND G. ROZENBERG§

Abstract. It is well known that regularity of a language implies certain properties known as pumping lemmas or iteration theorems. However, the question of a converse result has been open. We show that the usual form of pumping is very far from implying regularity but that a strengthened pumping property, the *block* pumping property, is *equivalent* to regularity. The proof involves use of the finite version of Ramsey's theorem. We compare our results with recent results of Jaffe and Beauquier and state some open questions.

Key words. regular languages, finite automata, pumping

1. Introduction. Work on regular sets, sets recognizable by finite automata, goes back to the middle and late fifties, to Kleene, Myhill, Nerode, Rabin, Scott and Shepherdson (see, e.g., [5], [6], [8], [9], [10]). Most preliminary questions were solved at this early stage, and most of the questions that remain appear to be quite hard.

Among the properties that regular sets have are so called pumping lemmas or iteration theorems. These theorems follow from the fact that a finite automaton that accepts long strings must *repeat* internal states, i.e., loop. The existence of such a loop implies that the corresponding portion of the input string may be eliminated or iterated without affecting acceptance or rejection by the automaton.

The question that we intend to consider in this paper is that of a converse, i.e., the question whether a given pumping property implies regularity. We present both positive and negative results and compare them with recent results by Jaffe [3] and Beauquier [1].

We close the paper with some open questions and suggestions for further work.

Notational remarks. Throughout the paper, Σ will be some fixed unspecified, finite alphabet, except in Theorem 1, where Σ is given explicitly. a, b, σ are symbols, x, y, z are strings, and Λ is the null string, $|y|$ is the *length* of the string y . Letters i, j, k, l, m, n, p denote natural numbers. $\aleph = 2^{\aleph_0}$ will denote the cardinality of the set of all sets of natural numbers. \aleph is, of course, uncountable. Since there are only countably many recursive languages, it follows that if there are \aleph languages belonging to a particular class, then they cannot all be recursive.

2. Negative results. We begin this section by stating the pumping lemma.

DEFINITION. Let $L \subseteq \Sigma^*$, $x \in \Sigma^*$ and $x = uvw$. Then v is a *pump* for x relative to L iff, for all $i \geq 0$,

$$u(v)^i w \in L \quad \text{iff} \quad x \in L.$$

Note that being a pump is really a property of the *particular occurrence* of v . x may contain two occurrences of v of which one is a pump, the other not.

PUMPING LEMMA. *Let $L \subseteq \Sigma^*$ be regular. Then L satisfies the pumping condition; i.e. there exists a $k > 0$ such that for all $x, y, z \in \Sigma^*$, if $|y| \geq k$ then there are $u, v, w \in \Sigma^*$,*

* Received by the editors June 26, 1979, and in final revised form November 4, 1980.

† Mathematics Department, University of Colorado, Boulder, Colorado 80309.

‡ Mathematics Department, Boston University, and Laboratory for Computer Science, MIT, Cambridge, Massachusetts 02139. This research was partially supported by the National Science Foundation under grant MCS79 10261.

§ Mathematics Department, University of Leiden, Leiden, The Netherlands.

$v \neq \Lambda$ such that $uvw = y$ and for all $i \geq 0$

$$xu(v)^i wz \in L \quad \text{iff} \quad xyz \in L.$$

This is a well-known result [2, p. 47]. It can be proved by letting $k \geq n$, where n is the number of states of M where M is a finite state automaton that recognizes L .

Question (Rao Kosaraju [4]). Does the pumping condition imply regularity?

THEOREM. (Beauquier [1]). *There is a context-free (CF) language L which satisfies the pumping condition but is not regular.*¹

We prove below a somewhat stronger theorem.

THEOREM 1. *There are \aleph languages which satisfy the pumping condition. Thus the pumping condition does not even imply recursiveness. Some of these languages are CF but not regular.*

Proof. We prove this result by the following device. Let $\Sigma_1 = \{a, b\}$ and $X \subseteq \Sigma_1^*$. We take a 16 letter² alphabet Σ and code X as a subset $L(X)$ of Σ^* . $L(X)$ satisfies the pumping condition and the map $X \rightarrow L(X)$ is 1-1. Since X is an arbitrary subset of Σ_1^* , there are \aleph possibilities for X and hence the same number for $L(X)$. This proves the first part of the lemma. We show moreover that if X is the language $\{a^n b^n \mid n \geq 0\}$ then $L(X)$ is CF but not regular.

Let $\Sigma = \{a_{i,j} \mid 0 \leq i, j \leq 3\}$. We define two maps f_a, f_b from Σ :

$$f_a(a_{i,j}) = a_{i+1,j} \pmod{4}, \quad f_b(a_{i,j}) = a_{i,j+1} \pmod{4}.$$

The functions f_a, f_b are permutations of Σ and have moreover the property that applying two functions can *never* have the same effect as applying one or applying none; e.g., for all $\sigma \in \Sigma$,

$$f_b(f_a(\sigma)) \neq f_a(\sigma) \neq f_a(f_a(\sigma)).$$

This is because applying two functions increments both subscripts i, j by one (mod 4) or one subscript by two (mod 4), and a single application of a function can never achieve this.

We define a *legal string* as any string $x = (\sigma_1)^{n_1}(\sigma_2)^{n_2} \cdots (\sigma_m)^{n_m}$ where $m \geq 1$, σ_1 is $a_{0,0}$ and for all $i < m$, σ_{i+1} is either $f_a(\sigma_i)$ or $f_b(\sigma_i)$. The powers n_i are all required to be positive. If we think of the transition from σ_i to σ_{i+1} as being *caused* by an a or b , depending on whether $\sigma_{i+1} = f_a(\sigma_i)$ or $\sigma_{i+1} = f_b(\sigma_i)$, respectively, then there are $m - 1$ transitions in the x above, and the corresponding $m - 1$ symbols form a string y in Σ^* . Since the n_i are all positive, y is unambiguously determined by x . We shall say that x *codes* y . Thus the string $x = a_{0,0}a_{1,0}a_{1,0}a_{1,1}$ is legal, $n_1 = 1$, $n_2 = 2$, and $n_3 = 1$. The coded string y is ab . Note that y does not determine the precise values of the n_i , and hence has many codes x . Note also that deleting any $(\sigma_i)^{n_i}$ from the above legal string results in a transition that does not correspond to an a or to a b , and hence the new string must be *illegal*.

¹ Actually Beauquier's counterexample is somewhat stronger. There is a *marked* pumping lemma where k distinct symbols in y are marked and the pump is required to contain one of the marked symbols. Beauquier shows that there is a CF language that satisfies the marked pumping condition but is not regular. Subsequent to our Theorem 1, Vaughan Pratt showed (private communication) that there are \aleph languages having the marked pumping property.

² We can carry out essentially the same construction with fewer letters if we notice that the f_a, f_b defined below are elements of the group of permutations on Σ which satisfy certain conditions. However, the construction is most transparent with the Σ that we use here.

The *parity* of a string is the sum of all subscripts $i, j \pmod 2$. Thus the parity of the string x above is 0.

Now we let

$$L(X) = \{x \mid x \text{ is legal and } x \text{ codes a } y \in X\} \cup \{x \mid x \text{ is illegal and has parity } 0\}.$$

We claim that the map L is 1-1. That is, we claim that if $X \neq X'$, then $L(X) \neq L(X')$. For say $x \in X - X'$. Then any legal string y which codes x lies in $L(X) - L(X')$, and thus $L(X) \neq L(X')$.

We now show that $L(X)$ always satisfies the pumping condition. Let $k = 5$. Let $xyz \in \Sigma^*$ and $|y| \geq 5$. We consider two cases.

(1) (a) xyz is legal and y contains a doublet $\sigma\sigma$. Let $y = u\sigma w$, where the last symbol of u is also σ and let $v = \sigma$. Then for all i , $xu(\sigma)^i wz$ is legal and codes the *same* string that xyz does. Hence $xu(\sigma)^i wz \in L(X)$ iff $xyz \in L(X)$.

(b) xyz is legal but y contains no doublet. We now have to consider parities. Say for example that $xyz \in L(X)$ and has parity 1. Now one of the last two symbols of y has parity 1, since parities of consecutive symbols alternate. Let v be that symbol and express y as $y = uvw$. Then for all $i \geq 1$, $xu(v)^i wz$ codes the *same* string as xyz and is legal, so

$$xu(v)^i wz \in L(X).$$

For $i = 0$, $xu(v)^i wz = xuwz$ has parity 0 and is *illegal*, so again

$$xu(v)^i wz \in L(X).$$

The cases where xyz has parity 0 or $xyz \notin L(X)$ are similar.

(2) xyz is illegal. The illegality may be caused by the initial symbol being other than $a_{0,0}$ or by a bad transition. In any case xyz contains a substring y' of length ≤ 2 such that preserving that string will preserve illegality. Hence since $|y| \geq 5$, we can find a substring v' of y , of length 2, such that v' is *disjoint* from y' . (This would be automatic if y' were in x or z and can also be achieved if y' overlaps y .)

Now let v be a nonempty substring of v' of parity 0. There must be such a v with one or two symbols. For if there is a symbol of parity 0, then v can be that symbol. Otherwise any v of length 2 will have 0 parity. Let $y = uvw$. Then for all $i \geq 0$, $xu(v)^i wz$ has the same parity as xyz and is illegal. Hence $xu(v)^i wz$ is in $L(X)$ iff xyz is.

This proves the first part of the lemma. Intuitively, the second part depends on the fact that some PDA (or FSA) accepts X iff some other PDA (FSA) accepts $L(X)$. Hence if X is CF but not regular, then $L(X)$ is also CF but not regular.

For our specific example consider the strings $x = a^n b^n$ where n is divisible by 4, and the strings y which represent these strings x . Consider the CF rules

$$S \rightarrow A_{0,0}A_{1,0}A_{2,0}A_{3,0}SA_{0,1}A_{0,2}A_{0,3}A_{0,0},$$

$$S \rightarrow A_{0,0},$$

$$A_{i,j} \rightarrow a_{i,j}A_{i,j},$$

$$A_{i,j} \rightarrow a_{i,j}.$$

The set generated is CF and is the set of y which represent some $a^n b^n$ where $n = 0 \pmod 4$. (The cases $n = i \pmod 4$ for $i = 1, 2, 3$ are similar.) Hence $L(X) \cap \text{Legal}$ is CF, being the union of four CF sets. But then $L(X) = (L(X) \cap \text{Legal}) \cup (\text{Illegal} \cap 0\text{-parity})$, and *Illegal*, *0-parity* are regular. Thus $L(X)$ is CF, being the union of two CF sets.

But $L(X)$ cannot be regular, for consider strings y_i such that $y_i = (a_{0,0}a_{1,0}a_{2,0}a_{3,0})^{i/4} a_{0,0}$ and z_i such that $z_i = (a_{0,1}a_{0,2}a_{0,3}a_{0,0})^{i/4}$ and i is divisible by 4. (This is for convenience, since $a_{0,0}$ is the starting symbol of all legal strings, and is also the last symbol of y_i if 4 divides i .) Now for all i, j , if $i \neq j$ then

$$y_i z_i \in L(X) \quad \text{and} \quad y_j z_i \notin L(X).$$

Hence, by Myhill's theorem [8] or Nerode's theorem [6], $L(X)$ is not regular. \square

3. Positive result. We saw in the last section that the pumping lemma does not imply regularity. This is also true (cf. footnote 2) of a somewhat stronger "marked pumping lemma". Thus the question arises whether there is any form of pumping that is a necessary and sufficient condition for regularity. We begin the discussion by quoting a recent result of Jaffe [3].

THEOREM. *L is regular iff there is a k such that for all $x \in \Sigma^*$, if $|x| \geq k$ then there exist $u, v, w, x = uvw, v \neq \Lambda$ and for all z, v is a pump for xz relative to L ; i.e., for all $i \geq 0$, all $z \in \Sigma^*$,*

$$u(v)^i w z \in L \quad \text{iff} \quad xz \in L.$$

However, Jaffe's pumping condition is not *local*. Given an x it requires a pump that works not just for x but a *uniform* pump which works for all $xz, z \in \Sigma^*$. So the question that arises is whether we can find a "local" pumping condition that is equivalent to regularity. Our Theorem 2 below gives a positive solution to this question.

DEFINITION. $L \subseteq \Sigma^*$ has the *block pumping property* if there is a k such that for all $x, w, y_1, \dots, y_k, w'$ in Σ^* , if $x = wy_1 \dots y_k w'$ then there exist $m, j, 1 \leq m < j \leq k$ such that $y_{m+1} \dots y_j$ is a pump for x relative to L . (Note that because the x, w , etc. are universally quantified over, we need not specify that the y_i be nonempty. The fact that *some* cases under the condition are vacuous does not imply that the condition itself is vacuous.)

DEFINITION. $L \subseteq \Sigma^*$ has the *block cancellation property* if there is a k such that for all $x, w, y_1, \dots, y_k, w'$ in Σ^* , if $x = wy_1 \dots y_k w'$ then there exist $m, j, 1 \leq m < j \leq k$ such that $wy_1 \dots y_m y_{j+1} \dots y_k w' \in L$ iff $x \in L$. Notice that the block cancellation property is a special case of the block pumping property.

Notation. If L has the block cancellation property for a particular k we shall say that $L \in \mathcal{C}_k$.

THEOREM 2. *Regularity, the block pumping property and the block cancellation property are equivalent.*

Proof. First note that regularity implies the block pumping property, for if L is regular, let M be an automaton accepting L and let k be the number of states of M . Suppose $x \in \Sigma^*$ and $x = wy_1 \dots y_k w'$. Let s^j be the state reached by M just after reading y_j then s^0, s^1, \dots, s^k are $k+1$ occurrences of states and there must be m, j such that $m < j$ but $s^m = s^j$. Then $v = y_{m+1} \dots y_j$ is the required pump for x relative to L . Also if L satisfies the block pumping property, then it trivially satisfies the block cancellation property.

Thus the theorem reduces to the lemma below:

LEMMA 1. *The cancellation property implies regularity.*

The proof of this lemma reduces to the following three lemmas.

LEMMA 2. *There are only finitely many languages in \mathcal{C}_k .*

Notation. Given a language L and a string x , let L_x be the set

$$\{z \mid xz \in L\}.$$

LEMMA 3. *If L is in \mathcal{C}_k then so is L_σ for any $\sigma \in \Sigma$.*

LEMMA 4. *Let P be a property of languages such that (i) there are only finitely many languages that have P ; (ii) for all σ in Σ , if L has P then L_σ has P . Then P implies regularity.*

Notice that Lemma 4 is essentially the lemma used in proving Nerode's theorem [6]. Here the property P will be the property of being in \mathcal{C}_k , for some fixed k .

Before giving the proofs of these lemmas we state the following version of Ramsey's theorem. Here if X is a set, $X[2]$ denotes the set of all two element subsets of X . If X has n elements then $X[2]$ has $n(n-1)/2$ elements.

THEOREM. (Ramsey) *For every k there is a number $r(k)$ such that if a set X has $r(k)$ elements or more and $X[2] = Z \cup Z'$ then there is a $Y \subseteq X$ such that Y has at least $k+1$ elements and $Y[2] \subseteq Z$ or $Y[2] \subseteq Z'$.*

For a proof of Ramsey's theorem, see [7] or [2].

The number $r(k)$ is usually denoted $N(k+1, k+1, 2)$ corresponding to a more general statement of the theorem, but we shall use the simpler notation. In Lemmas 2 and 3, k is fixed.

Proof of Lemma 2. To prove Lemma 2, it is sufficient to show that if L, L' are in \mathcal{C}_k and for all strings x with $|x| < r(k)$, $x \in L$ iff $x \in L'$, then $L = L'$. In other words, if L, L' "agree" on all strings of length $\leq r(k)$, then they coincide. For then if Σ has n elements, $n > 1$, there are at most $m = n^{r(k)}$ strings of length $< r(k)$ and hence at most 2^m languages in \mathcal{C}_k .

Claim. We will show by induction on n that if $|x| = n$, then $x \in L$ iff $x \in L'$. This is clear if $n < r(k)$. Assume the claim for all $p < n$.

Suppose $|x| = n$ and $n \geq r(k)$. To apply Ramsey's theorem we define Z, Z' as follows. Write $x = wy_1 \cdots y_{r(k)}w'$, where all the y_i are nonempty. Let $X = \{0, \dots, r(k)\}$ and for $m, j \in X$, $m < j$, let $\{m, j\} \in Z$ if $wy_1 \cdots y_m y_{j+1} \cdots y_{r(k)}w' \in L$ and otherwise let $\{m, j\} \in Z'$.

Then $X[2] = Z \cup Z'$ and by Ramsey's theorem, there is a Y with $k+1$ elements such that $Y[2] \subseteq Z$ or $Y[2] \subseteq Z'$.

In either case the elements of Y split x into $k+2$ strings, the string u before the first element of Y , the string u' after the last element, and all the k intermediate strings, z_1, \dots, z_k . Each z will be a union of one or more consecutive y 's, and $x = uz_1 \cdots z_k u'$. We have two cases.

(i) $Y[2] \subseteq Z$. Then removing any consecutive block of z 's from x corresponds to some set $\{m, j\}$ in $Y[2]$ and the shortened string x' is *always* in L . However, by the cancellation condition, there is *some* consecutive block of z 's, whose removal leads to an x' such that $x' \in L$ iff $x \in L$. Hence $x \in L$.

(ii) $Y[2] \subseteq Z'$. A similar argument tells us that $x \notin L$.

Hence, by (i) and (ii), $x \in L$ iff there is a Y with $k+1$ elements such that $Y[2] \subseteq Z$.

The same facts hold also for L' , with the *same* Z and Z' . This is because L and L' coincide for all strings of length less than n . Thus for x also we get, $x \in L$ iff $x \in L'$. This proves the claim and Lemma 2. \square

Proof of Lemma 3. Suppose that $z \in \Sigma^*$ and $z = wy_1 \cdots y_k w'$. Consider $\sigma z = w'' y_1 \cdots y_k w'$ where $w'' = \sigma w$. Then, since $L \in \mathcal{C}_k$, there exist m, j , $1 \leq m < j \leq k$ such that

$$w'' y_1 \cdots y_m y_{j+1} \cdots y_k w' \in L \quad \text{iff} \quad \sigma z \in L.$$

But then

$$w y_1 \cdots y_m y_{j+1} \cdots y_k w' \in L_\sigma \quad \text{iff} \quad z \in L_\sigma.$$

Hence L_σ is also in \mathcal{C}_k . \square

Proof of Lemma 4. Suppose L_0 has the property P . We define the automaton M as follows:

$$\begin{aligned} S &= \text{all languages } L \text{ having property } P, \\ s_0 &= \text{start state} = L_0 \\ M(L, \sigma) &= \text{next language (state) after reading } \sigma \\ &= L_{\sigma}, \\ F &= \text{set of accepting states} = \{L \mid \Lambda \in L\}. \end{aligned}$$

We can show by induction on x that $M(L, x)$ is L_x for all x in Σ^* . For $L_{\Lambda} = L$, and for all σ in Σ ,

$$L_{x\sigma} = (L_x)_{\sigma} = M(L, x)_{\sigma} = M(M(L, x), \sigma) = M(L, x\sigma).$$

Thus M accepts x iff $M(L_0, x) \in F$ iff $\Lambda \in M(L_0, x)$
 iff $\Lambda \in (L_0)_x$
 iff $x\Lambda \in L_0$
 iff $x \in L_0$. \square

We close this section by listing some open questions:

- (1) Is there an analogue of Theorem 2 for context-free languages?
- (2) The automaton that we have constructed in the proof of Theorem 2 which recognizes the languages L_0 , has a very large number of states. For a two-element Σ , the number would be $2^{(2^{r(k)})}$. By considering nondeterministic automata we see that there is a lower bound of 2^k . Can we bridge this gap? The corresponding gap in Jaffe's construction has been closed by A. Yehudai [10].
- (3) The block pumping lemma depends for its strength principally on the cancellation property, i.e., the case $i = 0$. Is there a pumping property which is *positive*, i.e., uses $i \geq 1$ only and which is equivalent to regularity?

Acknowledgments. We are indebted to J. Jaffe, Rao Kosaraju, A. Meyer, V. Pratt and the referees for various useful suggestions.

REFERENCES

- [1] J. BEAUQUIER, personal communication from Rao Kosaraju.
- [2] M. A. HARRISON, *Introduction to Formal Language Theory*, Addison Wesley, Reading MA, 1978.
- [3] J. JAFFE, *A necessary and sufficient pumping lemma for regular languages*, Sigact News (1978), pp. 48–49.
- [4] R. KOSARAJU, personal communication.
- [5] J. MYHILL, *Finite automata and the representation of events*, WADD Technical Report 57-624 (1957), Wright-Patterson Air Force base, Ohio 45433.
- [6] A. NERODE, *Linear automata transformations*, Proc. Amer. Math. Soc., 9 (1958), pp. 541–544.
- [7] F. P. RAMSEY, *On a problem of formal logic*, in *The Foundations of Mathematics*, Routledge and Kegan Paul, 1954, pp. 82–111 and reprinted from Proc. London Math. Soc., Ser 2, 30 (1928), pp. 338–384.
- [8] M. RABIN AND D. SCOTT, *Finite automata and their decision problems*, IBM J. Res. Dev. 3 (1959), pp. 114–125.
- [9] J. SHEPHERDSON, *The reduction of two way automata to one way automata*, IBM J. Res. Dev. 3 135–136.
- [10] A. YEHUDAI, *A note on the pumping lemma for regular languages*, Inform. Proc. Letters, 9 (1979), pp.

WORST-CASE AND PROBABILISTIC ANALYSIS OF A GEOMETRIC LOCATION PROBLEM*

CHRISTOS H. PAPADIMITRIOU†

Abstract. We consider the problem of choosing K “medians” among n points on the Euclidean plane such that the sum of the distances from each of the n points to its closest median is minimized. We show that this problem is NP-complete. We also present two heuristics that produce arbitrarily good solutions with probability going to 1. One is a partition heuristic, and works when K grows linearly—or almost so—with n . The other is the “honeycomb” heuristic, and is applicable to rates of growth of K of the form $K \sim n^\epsilon$, $0 < \epsilon < 1$.

Key words. location problem, K -median problem, NP-complete problem, probabilistic analysis of algorithms

1. Introduction. In this paper we study a classical location problem: Suppose that we are given n points on the plane, and an integer $K < n$. We are asked to choose K of these n points and proclaim them to be *centers* or *medians* in such a way that, if we add the distances from each point to its closest median, this sum is as small as possible. We call this optimization problem the K -median problem.

In [FH] it is conjectured that this problem is NP-complete (see [Ka1], [GJ], [PS] for definitions concerning NP-completeness). It was already known [KH] that the K -median problem, with a metric not Euclidean but induced by a graph, is indeed NP-complete. The performance of heuristics for the problem with the general metric was analyzed both deterministically and probabilistically in [CFN] and [CNW]. Furthermore, a continuous version of the problem was of concern for a long time in economic location theory [St], [Bo], [FT1].

In § 2 we show that the Euclidean metric version of the K -median problem is NP-complete, thus proving the conjecture of [FH]. The result and its proof follow in style the analogous result about the traveling salesman problem [Pa1], [GGJ].

Once an optimization problem is shown NP-complete, the interest of researchers is usually shifted towards the analysis of efficient heuristics that, one hopes, produce good—though suboptimal—solutions (see [GJ, Chap. 6]). In fact, Karp [Ka2], [Ka3] has initiated research on a *probabilistic refinement* of this approach: He gave heuristics for several hard combinatorial optimization problems that were efficient (sometimes on the average) and produced solutions which, with probability arbitrarily close to one, were arbitrarily close to the optimum. Such an approach to the K -median problem was taken in [FH].

Before explaining the results of [FH], we need to make an observation about the K -median problem. Any instance of the K -median problem with n points can be solved exhaustively in time proportional to n^{c+1} , where $c = \min(K, n-K)$. Thus, although the problem is NP-complete when K is not fixed but comes as a part of the input, it is polynomial for any fixed K . In fact, if we restrict K to grow extremely slowly with n —say, $K = \log \log n$ —then the exhaustive algorithm is not polynomial any more, but it certainly is subexponential. It therefore makes sense to subdivide the instances of the K -median problem into classes according to the rate of growth of K with n . [FH] gives

* Received by the editors March 3, 1980, and in revised form November, 4, 1980. This research was supported by the National Science Foundation under grant MCS77-01193, and by a Miller Fellowship.

† Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

an “aggregation” heuristic, which is polynomial and has favorable error analysis when K grows slower than $\log n$. (Notice that, for this growth, the problem is most probably not NP-complete since it is solvable by a subexponential algorithm.) As a lemma, they show that there are constants c_1, c_2 such that the cost of the optimum is almost certainly between $c_1 n/\sqrt{K}$ and $c_2 n/\sqrt{K}$ when n/K goes to infinity. We improve this result in two ways: We prove it for bounded n/K (Lemma 2), and we find the exact limit $c_3 n/\sqrt{K}$ for n/K going to infinity faster than $\log n$ (corollary to Theorem 5).

In §§ 3 and 4 we give probabilistic algorithms for fast growths of K . Section 3 is concerned with the case in which K grows faster than $n(\log n)^{-1/3}$. We give a partitioning algorithm for this problem, and we show that when the points are drawn from a Poisson distribution with mean N , then this algorithm has $O(N^3/\log N)$ average execution time, and has a relative error smaller than any $\varepsilon > 0$ with probability going to 1 as N goes to infinity. Our main tool for proving this is a combinatorial lemma (Lemma 1) which shows that in the optimal solution with probability going to 1, no point is “much” further from its closest median.

In § 4 we study the case in which K grows slower than $n/\log n$, but faster than $\log n$. We notice that the continuous location problem [Sta], [Bo] becomes relevant. We give a proof that the continuous location problem is asymptotically optimized when the area is divided up into hexagonal cells (this result was apparently known to L. Fejes Toth [FT1], as quoted by Bollobas [Bo]; an independent proof was found by Mordecai Haimovich [Ha]). We then use this result to analyze a very simple “honeycomb” heuristic which, in time $O(n \log n)$ constructs a solution that has relative error smaller than $\varepsilon > 0$ with probability going to 1. Our probabilistic assumptions are that the n points are n independently and uniformly distributed variables on the unit square.

Finally, in § 5 we discuss our results, a related recent development that may simplify our approach for the case in which K grows *exactly* as n , as well as several related open problems.

2. NP-completeness. In order to show that the K -median problem is NP-complete, we have to formulate it first in a more suitable manner. We assume that the points are in the *integral* lattice, and are given as pairs of integer coordinates.

A familiar problem arises—see, for example, [Pa2], [GGJ], [Pa1], [PS]: In order to be able to argue that the problem is in NP, we must round the distances down to the closest integer; i.e., if $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, then $\text{dist}(p_1, p_2) = \lfloor ((x_1 + x_2)^2 + (y_1 + y_2)^2)^{1/2} \rfloor$. This is done in order to avoid the difficulty of comparing sums of radicals, a problem of rather mysterious complexity.

We define our problem thus:

K -MEDIAN. Given a multiset $P = \{p_1, \dots, p_n\}$ of points with integer coordinates and integers K and L , is there a subset $M = \{m_1, \dots, m_K\} \subseteq P$ such that $\sum_{j=1}^n d_j \leq L$, where $d_j = \min_{m \in M} \text{dist}(p_j, m)$?

This strict definition only serves the purposes of the present section. In order to apply probabilistic techniques, we will have to make the problem continuous. Even in the constructions of the present section, we shall allow fractional—and even irrational—coordinates. The assumption is that all coordinates, as well as the limit L , will be eventually multiplied by a sufficiently large integer and rounded, so that any required precision can be accomplished.

We shall also occasionally define a point in p with a *weight* w . This will mean that there are w points in P with exactly the same coordinate. If a fractional weight is used, we are assuming that all weights (including unit weights) will be eventually multiplied by a sufficiently (yet polynomially) large integer, so that all weights become integers. In the

sequel we shall use weights and nonintegral coordinates without further explanation.

THEOREM 1. *The K-MEDIAN problem is NP-complete.*

Proof. That K-MEDIAN, as defined above, is in NP is immediate. To prove NP-completeness, we shall reduce to K-MEDIAN the following problem:

EXACT COVER. Given a set $U = \{U_1, U_2, \dots, U_{3n}\}$ and a family $F = \{S_1, \dots, S_k\}$ of subsets of U with $|S_j| = 3, j = 1, \dots, k$, is there a cover $C \subseteq F$ such that $|C| = n$ and $\cup_C S_j = U$?

This problem is known to be NP-complete [GJ].

Before proceeding to the actual reduction, we shall discuss the properties of the configuration of points R (Fig. 1a). R is called a *row of length m* . It has $6m + 4$ points, and the two extreme points b, b' have weight m^2 ; this will imply that they *have* to be medians in any optimal solution. Suppose that we must allocate $m + 2$ medians to R . Then the two best solutions are shown in Fig. 1a. They both designate b, b' as medians,

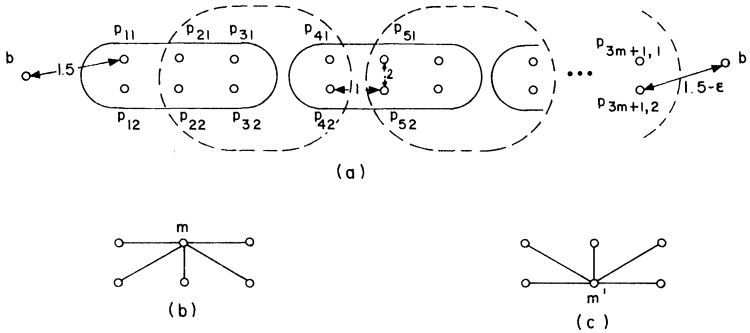


FIG. 1

plus m more points. Either $\{p_{2j_1}, p_{5j_2}, \dots, p_{3m-1,j_m}\}$ for some choices of $j_i = 1$ or $2, i = 1, \dots, m$; or $\{p_{3j_1}, p_{6j_2}, p_{9j_3}, \dots, p_{3m,j_m}\}$, again for some choices of j_i . The former is called *Solution 1* (it is really a family of solutions) and the second *Solution 2*. Solution 1 induces to the points of R the partition shown in solid lines in Fig. 1a, whereas Solution 2 the one with broken lines. Among each resulting group of 6 points the median can be chosen either as in Fig. 1b (called an *upper median*) or as in Fig. 1c (*lower median*). Notice that Solution 1 is cheaper by 2ϵ where $\epsilon = m^{-4}$. For our reduction, given any instance $U = \{u_1, \dots, u_{3n}\}$ and $F = \{S_1, \dots, S_k\}$ of the EXACT COVER problem, we shall construct a point set P (weighted) and integer K , as well as a limit L , such that P has K medians with cost L or less iff there is an exact cover $C \subseteq F$ of U . P consists of k rows R_1, \dots, R_k , each of length $3n$, arranged parallel to each other (Fig. 2, schematically). Thus, we can distinguish $3n$ columns of this formation, corresponding to the elements of U .

We shall examine in detail the “window” W of Fig. 2. It is shown in detail in Fig. 3.

The spots x, y, w, z of Fig. 3 are not points of P , but only possible positions of points. For each window, one of x, y and one of w, z positions is occupied with points of weight n^{-2} . x is occupied iff $U_i \notin S_{j-1}$; y iff $U_i \in S_{j-1}$. Similarly w is occupied iff $U_i \in S_j$; z iff $U_i \notin S_j$.

We now define $K = k(3n + 2) + 3n(k - 1)$. The first term provides enough medians for all k rows, and the second one median for the $q-q'$ pair in each window W .

L consists of 3 components $L = L_1 + L_2 + L_3$. $L_1 = k(2 * 1.5 + 3n(2.2 + 2\sqrt{1.04})) - 2n\epsilon$. This cost comes from the k rows. In order for it to be achieved, all rows

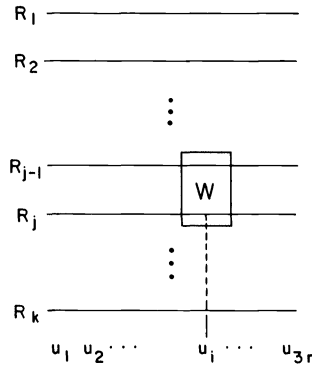


FIG. 2

must be grouped according to Solution 1 or 2, and, because of the $-2n\epsilon$ term, at least n of them must be grouped by Solution 1.

$L_2 = 3n(k - 1)$, and comes from the cost due to the q or q' points. Only one in each pair will become a median, at a cost of 1 per pair.

$L_3 = 12m(k - 1)/n^2$ is the cost of connecting each of the $6m(k - 1)$ points x, y, w, z to the closest q, q' or p point, always 2 away.

CLAIM. *There exists $M \subseteq P$ with $|M| = K$ with cost L or less iff F contains an exact cover C of U .*

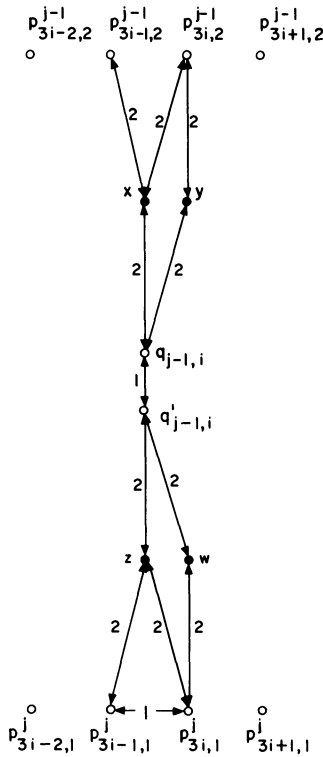


FIG. 3

Proof of claim. Suppose that such an M exists. It is not hard to see that $3n + 2$ medians must be allocated to each row and one median to each $q - q'$ pair for the cost to be L or less; to see this it suffices to consider the incremental cost or gain associated with adding or taking away one median. Each row is therefore grouped by Solution 1 or 2. Take the fact that R_j is grouped by Solution 1 to mean that $S_j \in C$, where C is the claimed exact cover. In fact, at least n rows must be grouped by Solution 1 for L to be achieved, and hence C must contain at least n sets.

Suppose that R_j is grouped by Solution 1 (i.e., $S_j \in C$). Consider the i th group, where $U_i \in S_j$. It looks like Fig. 4 (or the corresponding lower median configuration).

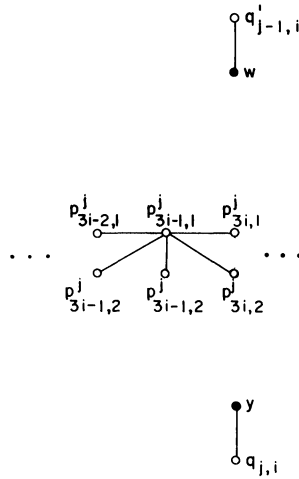


FIG. 4

Since $U_i \in S_j$, both w (above) and y (below) positions are occupied by a point. These two points *cannot* therefore be connected to their p -median with a link of length 2, as required. So they are connected to the corresponding q -medians. But this means that the x or y point of $q_{j-1,i}$ (respectively, the w or z point of $q'_{j,i}$) must be picked by their corresponding lower (respectively, upper) p -medians in R_{j+1} (respectively, R_{j-1}). By induction, therefore, the i th group of any row R_k $k < j$ (respectively, $k > j$) must have a lower (respectively, upper) median. Hence this change in this kind (upper vs. lower)—of medians can occur at most once per column. However, R_j causes this change to all three columns corresponding to the three elements $U_i \in S_j$, and thus there can be no overlaps in the sets S_j of C . So, C contains at least n sets without overlaps: it is an exact cover.

Conversely, suppose that the given instance of EXACT COVER has a solution C . Then we can infer a solution M of the K -MEDIAN problem by allocating $3n + 2$ medians to each row, and 1 to each $q - q'$ pair, having each R_j grouped by Solution 1 if $S_j \in C$, and by Solution 2 otherwise. Finally, let $j(i)$ be the index j of the unique $S_j \in C$ such that $i \in S_j$. We group the i th of R_j by an upper median if $j \geq j(i)$, and a lower median if $j < j(i)$. It follows that the solution has cost L . \square

It is more meaningful to consider the special cases of the K -MEDIAN problem, for which K is related to n in a prespecified way. Let $K: \mathbb{N} \rightarrow \mathbb{N}$ be a (polynomially computable) function. By $K(n)$ -MEDIAN we mean the set of all instances of K -MEDIAN for which $K = K(n)$. We can show, from Theorem 1, the following stronger result.

COROLLARY. Suppose that $\min(K(n), n - K(n)) = \Omega(n^\epsilon)$ for some $\epsilon > 0$. Then the $K(n)$ -MEDIAN problem is NP-complete.

Proof. The proof follows by standard “padding” arguments. \square

Corollary 1 is in a sense the strongest possible result, with the present state of our understanding of complexity theory, because if either $n - K(n)$ or $K(n)$ grows slower than n^ϵ for all ϵ , the $K(n)$ -median problem can be solved by a subexponential algorithm.

3. The linear case. In this section we consider the case in which $K(n) = \lfloor \alpha n \rfloor$ for some $\alpha < 1$. Informally, this means that a fixed fraction of the customers are proclaimed medians, and therefore each median will be, on the average, responsible for a constant number of customers. We consider point sets $P = \{p_1, \dots, p_n\}$ drawn from a Poisson process of intensity N on the unit square. As a result, the distributions of points in any two prespecified nonoverlapping subregions of the unit square are independent. n is a random variable with expected value N .

We divide the unit square into $Q = \lceil \sqrt{N/\log N} \rceil^2$ equal smaller squares S_1, \dots, S_Q , each of side $\lceil \sqrt{N/\log N} \rceil^{-1}$ and containing approximately $\log N$ points on the average. If $M \subseteq P$, we let $f_M(p_i) = m \in M$ iff $\text{dist}(p_i, m) < \text{dist}(p_i, m')$ for all $m' \in M - \{m\}$. With probability 1, f_M is well defined for all $M \subseteq P$. We let $d_i^M = \text{dist}(p_i, f_M(p_i))$ and $C(M) = \sum_{i=1}^n d_i^M$, the cost of the set M of medians. Once we have fixed S_1, \dots, S_Q , we shall define the *separable cost* of M . Let $S(p_i)$ denote the square S_i among S_1, \dots, S_Q for which $p_i \in S_i$. We define, for $p_i \in P$,

$$g_i^M = \min_{\substack{m \in M \\ S(m) = S(p_i)}} \text{dist}(p_i, m).$$

Finally, the separable cost of M is defined as $C'(M) = \sum_{i=1}^n g_i^M$. Thus, informally, $C'(M)$ is the cost of M under the additional restriction that customers must go to medians in the same square S_i .

We shall prove the following:

THEOREM 2. Let \hat{M} be the optimal solution of P . Then $C'(\hat{M}) - C(\hat{M}) = o(N^{1/2})$ with probability $1 - o(1)$.

All asymptotic statements are meant as N goes to infinity. Thus, “with probability $1 - o(1)$ ” means “with probability going to 1 as N goes to infinity”.

We first need the following lemma:

LEMMA 1. There is a constant $c_1 > 0$ such that $\max_i d_i^{\hat{M}} < c_1(\alpha^3 N)^{-1/2}$ with probability $1 - o(1)$.

Proof. It is clear that, with probability $1 - o(1)$, $n \geq N/2$ and thus $|\hat{M}| \geq \lfloor \alpha N/2 \rfloor$. If $m \in \hat{M}$, let $A(m) = \{p_i: f_{\hat{M}}(p_i) = m\}$. It follows that, with probability $1 - o(1)$, there exist at least $\lfloor \alpha N/4 \rfloor$ medians $m \in \hat{M}$ with $|A(m)| \leq 2/\alpha$. It is therefore obvious that, for some constant $c_2 > 0$, two of these medians (say, m and m') are closer to each other than $c_2(\alpha N)^{-1/2}$ with probability $1 - o(1)$. (To see this, divide the unit square into $\lfloor (\lfloor \alpha N/4 \rfloor)^{1/2} - 1 \rfloor^2$ equal squares; two of the $\lfloor \alpha N/4 \rfloor$ medians are bound, by the pigeonhole principle, to fall in the same of these squares, and hence they cannot be further apart than the diameter of the square. This argument gives $c_2 = \sqrt{8}$; $c_2 = \frac{4}{3}(12)^{1/4}$ is possible.)

Suppose now that one of the points $p_k \in P$ has $d_k^{\hat{M}} > 2c_2(N\alpha^3)^{-1/2}$. Then we claim that $C'(\hat{M} - \{m\} \cup \{p_k\}) < C(\hat{M})$ —absurd, since \hat{M} is optimum. To prove our claim, we shall construct a mapping $f: P \rightarrow \hat{M} = \{m\} \cup \{p_k\}$ such that $\sum_{j=1}^n \text{dist}(p_j, f(p_j)) <$

$C(\hat{M})$. f is defined as follows:

$$\begin{aligned} f(p_k) &= p_k, \\ f(p_j) &= m' \quad \text{if } p_j \in A(m) \text{ and } p_j \neq p_k, \\ f(p_j) &= m'' \quad \text{if } p_j \in A(m''), m'' \text{ and } p_j \neq p_k. \end{aligned}$$

Thus

$$\begin{aligned} C(\hat{M}) - \sum_{j=1}^n \text{dist}(p_j, f(p_j)) &= d_k^{\hat{M}} - \sum_{p_j \in A(m)} \text{dist}(p_j, m') - \text{dist}(p_j, m) \\ &\qquad\qquad\qquad \text{(by the triangle inequality)} \\ &\cong d_k^{\hat{M}} - |A(m)| \text{dist}(m, m') \\ &\qquad\qquad\qquad \text{(since } |A(m)| < \frac{2}{\alpha} \text{ and } \text{dist}(m, m') \leq c_2(\alpha N)^{-1/2}) \\ &\cong d_k^{\hat{M}} - 2c_2(N\alpha^3)^{-1/2} > 0. \end{aligned}$$

This proves the lemma with $c_1 = 2c_2$. \square

Proof of Theorem 2. Lemma 1 implies that with probability $1 - o(1)$ all points $p_j \in P$ such that $S(p_j) \neq S(f_{\hat{M}}(p_j))$ lie in a “corridor” of width $2c_1(\alpha^3 N)^{-1/2}$ around the perimeters of the small squares (Fig. 5). Using this, we shall show how to modify the optimal solution so as to make it separable, with a total increase in cost which is $(o(N^{1/2}))$.

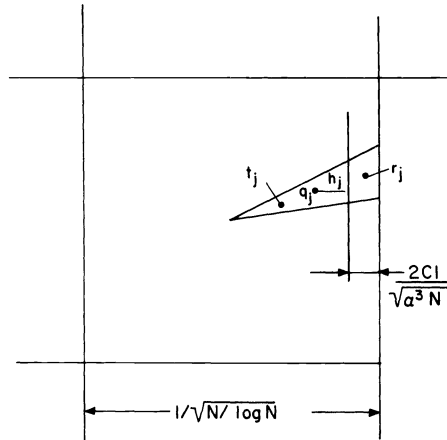


FIG. 5

The main idea is the following: It is clear that the total number of points in P that lie within these “corridors” is $o(N)$ with probability $1 - o(1)$, since each square has side asymptotically $\sim \sqrt{\log N}/N$, whereas the width of the corridor is $\sim \sqrt{1/N}$. We shall show that we can assign each of these points to a median \hat{M} in its own square which is, on the average, $O(N^{-1/2})$ away.

The details are as follows: Let us divide each square S_1 into $4\lceil\sqrt{\log N}\rceil$ triangular “slices” as shown in Fig. 5. The unit square is now divided into $R = 4\lceil\sqrt{\log N}\rceil Q$ triangles t_1, \dots, t_R and the corresponding trapezoids r_1, \dots, r_R (see Fig. 5). If $p_j \in P$, we denote by $r(p_j)$ the trapezoid it is in, or by $t(p_j)$ the triangle it is in (exactly one is well defined). Let b_j be a random variable denoting $|P \cap r_j|; j = 1, \dots, R$, and let $q_j \in P$ be the

point in t_j that is closest to the basis of t_j parallel to the side of the square, while h_j is the distance of q_j from the basis (see Fig. 5).

It is immediate that each t_j has area J/N , where $J = c_7\sqrt{\log N}$, and thus the probability that $P \cap t_j = \emptyset$ is e^{-J} . Thus we may assume that no $P \cap t_j$ is empty (i.e., q_j exists) with probability $(1 - e^{-J})^R \geq 1 - Re^{-J} = 1 - o(1)$. We now claim that, with probability $1 - o(1)$, we can assign the points of $P \cap r_j$ to the median $f_{\hat{M}}(q_j) \in \hat{M}$, and repeat this for all r_j 's, at a total incremental cost that does not exceed $c_3\sqrt{N/\log N}$ for some $c_3 > 0$. This would settle the theorem.

The cost of connecting each point p of the b_j points in $r_j \cap P$ to $f_{\hat{M}}(q_j)$, $\text{dist}(p, f_{\hat{M}}(q_j))$ can be bounded from above as follows (with probability $1 - o(1)$):

$$\begin{aligned} \text{dist}(p, f_{\hat{M}}(q_j)) &\leq \text{dist}(p, q_j) + \text{dist}(q_j, f_{\hat{M}}(q_j)) \\ &\leq \sqrt{2}(c_1(\alpha^3 N)^{-1/2} + h_j) + c_1(\alpha^3 N)^{-1/2} \end{aligned}$$

Thus the total increase in cost is bounded by

$$C'(\hat{M}) - C(\hat{M}) \leq c_4(\alpha^3 N)^{-1/2} \cdot \sum_{j=1}^R b_j + \sqrt{2} \sum_{j=1}^R b_j h_j.$$

The area of each r_j is equal to $c_5\alpha^{-3/2}/N$ for some constant c_5 ; thus the b_j 's are identically and independently distributed random variables with mean $c_5\alpha^{-3/2}$, and hence, by the central limit theorem, with probability $1 - o(1)$

$$\sum_{j=1}^R b_j \leq 2Rc_5\alpha^{-3/2} \leq c_6N/\sqrt{\log N}$$

for some $c_6 > 0$. h_j takes values from 0 to $L = \frac{1}{2}[(N/\log N)^{1/2}]^{-1}$, and the expected number of points in $P \cap t_j$ is $J = c_7\sqrt{\log N}$. To calculate $\mathcal{E}(h_j)$, we note that, $\text{prob}(h_j > xL)$ equals

$$e^{-J} \sum_{i=0}^{\infty} \frac{J^i}{i!} (1-x)^{2i} = e^{(x^2-2x)J} \leq e^{-xJ}.$$

Hence

$$\mathcal{E}(h_j) \leq -L \int_0^1 xde^{-xJ} = \frac{L}{J} - e^{-J} \frac{J+1}{J} \leq c_8N^{-1/2}.$$

Thus $\mathcal{E}(h_j b_j) \leq c_9(\alpha^3 N)^{-1/2}$, and by the central limit theorem

$$\sum_{j=1}^R h_j b_j \leq 2Rc_9(\alpha^3 N)^{-1/2} \leq c_{10}\sqrt{(N/\log N)}$$

with probability $1 - o(1)$. It follows that $C'(\hat{M}) - C(\hat{M}) \leq c_{11}(\alpha^3 \log N/N)^{-1/2}$ with probability $1 - o(1)$, and the theorem is proved. \square .

Theorem 2 implies that the optimum separable solution—the one that minimizes $C'(M)$ —would be a good approximation to the optimum solution. We shall describe below an algorithm for computing the optimum separable solution.

The algorithm is a dynamic programming scheme. Let $f(i, j)$ be the cost of the optimal way of allocating i medians to S_p , and let $F(i, j)$ be the cost of the optimal way of allocating a total of i medians to the squares S_1, S_2, \dots, S_j . It follows that

$$F(i, 1) = f(i, 1) \quad \text{for } i = 1, 2, \dots, \lfloor \alpha n \rfloor,$$

$$F(i, j+1) = \min_{j \leq k < i} [F(k, j) + f(i-k, j+1)] \quad \text{for } j = 1, \dots, Q \text{ and } i = j+1, \dots, \lfloor \alpha n \rfloor.$$

Notice that $F(\lfloor \alpha n \rfloor, Q)$ is the required optimum cost, and that the dynamic programming scheme can be implemented in such a way that the optimal allocations of medians are recovered after the determination of the optimal cost. This dynamic programming scheme requires $Q[\alpha n]$ evaluations of the function $f(i, j)$. Notice that $\mathcal{E}(Q[\alpha n]) = O(N^2/\log N)$.

For any i and fixed j , evaluating $f(i, j)$ can clearly be done in a number of operations bounded by $d2^k$, where $d > 0$ and $k = |P \cap S_j|$. Thus the expected number of operations for evaluating $f(i, j)$ is

$$\sum_{k=0}^{\infty} \frac{d \cdot 2^k (\log N)^k}{k!} e^{-\log N} = de^{\log N} = d \cdot N.$$

Therefore, this algorithm has an expected total number of steps $O(N^3/\log N)$.

To evaluate the relative error of the algorithm, we need the following lemma.

LEMMA 2. *If \hat{M} is the optimum solution, then, with probability $1 - o(1)$, $C(\hat{M}) \cong c_1 \sqrt{N}$ for some $c_1 > 0$.*

Proof. Let us fix a value for n ; the points in P_j are thus uniformly distributed in the unit square. Divide the unit square into $\lceil \sqrt{n/p} \rceil^2$ squares, where $p \ll 1$ is to be determined. How many squares contain exactly one point? This is at least as much as a binomial variable with n trials and probability $1 - p$. So, by Chebyshev's inequality, at least $(1 - 2p)n$ squares have one point, with probability $1 - o(n^{-1})$.

Consider these $(1 - 2p)n$ squares, each of side $a = \lceil \sqrt{n/p} \rceil^{-1}$. In how many cases of these squares the point is further than pa from the boundary? By Chebyshev, at least $(1 - 3p)^3 n$ with probability $1 - o(n^{-1})$. Choose $(1 - 3p)^3 = (1 - \alpha/2)$. Suppose that a_j is the distance from p_j to the point in $P - p_j$ closest to p_j , and assume that P has been ordered in decreasing a_j 's. The above argument suggests that, with probability $1 - o(1)$, $a_j \cong \sqrt{p/n}$ for $j \leq (1\alpha/2)n$. Thus

$$\sum_{j=n-\lfloor \alpha n \rfloor + 1}^n a_j \cong \sqrt{\frac{\alpha p}{2}} \sqrt{n}.$$

It is, however, clear that $\sum_{j=n-\lfloor \alpha n \rfloor + 1}^n a_j$ is a lower bound for $C(\hat{M})$. Since $n \geq N/2$ with probability $1 - o(1)$, the lemma follows. \square

Combining the preceding observations with Theorem 2 and Lemma 2 above, we obtain

THEOREM 3. *The dynamic programming algorithm above requires an expected number of operations $O(N^3/\log N)$ and produces a set M of medians with relative error $(C(M) - C(\hat{M}))/C(\hat{M}) = o(1)$ with probability $1 - o(1)$.*

We note that the same approach works for slightly sublinear growths of $K(n)$, in particular, $K(n) = \omega(n/\log^{1/3} n)$. In the next section we handle in a very different way more sublinear growths of $K(n)$, namely $K(n) = n^\epsilon$ for some $0 < \epsilon < 1$.

4. The honeycomb heuristic. In this section we consider the case in which $K(n)$ grows slower than $n/\log n$ but faster than $\log n$. For simplicity, we shall only deal explicitly with the family of growth $k(n) = \lfloor n^\epsilon \rfloor$ for some $\epsilon, 0 < \epsilon < 1$; generalization to the above mentioned range is immediate from our proofs.

A simple consequence of this growth is that both $K(n)$ and $n/K(n)$ go to infinity quite fast, as $n^\delta, \delta > 0$. That $n/K(n)$ grows quite fast means, intuitively, that the average median would, asymptotically, be responsible for a "continuum" of points. It is therefore natural to consider the following continuous, deterministic version of the

problem:

$$\text{Place } K \text{ points } M = \{m_1, \dots, m_K\} \text{ in the unit square so that } C^*(M) = \sum_{j=1}^K \int_{D_j} \text{dist}(m_j, A) dA \text{ is minimized,}$$

where $D_j = \{x \in [0, 1]^2: \text{dist}(x, m_j) \leq \text{dist}(x, m_i) \text{ for all } i \neq j\}$ are the *Dirichlet cells* (Voronoi polygons of Shamos [Sh]) of the point m_j with respect to the point set M .

D_j is thus the locus of all points that are closest to m_j ; it is easy to see that D_j is always a convex polygon (since it is the nonempty intersection of the half-planes $\text{dist}(x, m_j) \leq \text{dist}(x, m_i), i \neq j$).

Let R_n be the regular n -gon of unit area, and let c be its center. We let $\gamma(n) = \int_{R_n} \text{dist}(c, A) dA$. A simple calculation yields $\gamma(n) = \log(t + \sqrt{1+t^2})/t\sqrt{nt} + \sqrt{1+t^2}/(3\sqrt{nt})$, where $t = \tan(\pi/n)$. Some values of $\gamma(n)$ are given in Table 1. The following lemma is shown in [FT].

LEMMA 3. Let S_n be a convex n -gon with unit area, and let $p \in S_n$. Then $\int_{S_n} \text{dist}(p, A) dA \geq \gamma(n)$.

A tedious calculation yields

LEMMA 4. The function $(\gamma(x))^{-2}$ is concave for $x \geq 3$.

The following lemma says, essentially, that if we partition the unit square into many polygons, then each polygon must, on the average, have fewer than 6 sides. It is a rather surprising application of Euler's formula, and can be found, for example, in Heawood's 5-color proof [He].

LEMMA 5. Let $\{S_1, \dots, S_k\}$ be a partition of the unit square into k convex polygons, with m_1, \dots, m_k sides, respectively. Then $\sum_{j=1}^k m_j \leq 6k - 2$.

Using these lemmas, we can show the following theorem:

THEOREM 4. If $|M| = K$, then $C^*(M) \geq K^{-1/2} \gamma(6)$.

Proof. By definition, $C^*(M) = \sum_{j=1}^K \int_{D_j} \text{dist}(m_j, A) dA$, where D_j is the Dirichlet cell of m_j with respect to M . If D_j is an n_j -gon, we have, by Lemma 3,

$$C^*(M) \geq \sum_{j=1}^K |D_j|^{3/2} \gamma(n_j) = \sum_{k=3}^{\infty} \gamma(k) \sum_{j \in G_k} |D_j|^{3/2},$$

where $G_k = \{j: n_j = k\}$. Recall that $\sum_{i=1}^n x_i^{3/2}$ with $\sum_{i=1}^n x_i$ fixed is minimized when all x_i 's are equal.

Thus,

$$C^*(M) \geq \sum_{k=3}^{\infty} \gamma(k) |G_k| \left(\frac{\sum |D_j|}{|G_k|} \right)^{3/2}.$$

The latter expression can be written as

$$C^*(M) \geq \sum_{k=3}^{\infty} \frac{|G_k|}{\gamma(k)^2} \left(\sum_{j \in G_k} |D_j| \frac{\gamma^2(k)^{3/2}}{|G_k|} \right).$$

By the same argument, the right-hand side of the inequality above can be bounded from below:

$$C^*(M) \geq \left\{ \sum_{k=3}^{\infty} \left(|D_j| \frac{\gamma^2(k)}{|G_k|} \right) \left(\frac{|G_k|}{\gamma^2(k)} \right)^{3/2} \right\} / \left\{ \sum_{k=3}^{\infty} |G_k| / \gamma^2(k) \right\}^{1/2} = \left(\sum_{i=1}^K \gamma^{-2}(n_i) \right)^{-1/2}.$$

However, since $\gamma^{-2}(x)$ is concave, (Lemma 4) we have, by Jensen's inequality,

$$C^*(M) \geq K^{-1/2} \gamma \left(\frac{\sum_{j=1}^K n_j}{K} \right),$$

which, by Lemma 5 and since γ is nonincreasing, gives $C^*(M) \geq K^{-1/2} \gamma(6)$. \square

TABLE 1
Values of $\gamma(n)$

n	$\gamma(n)$	$(1/\gamma(n))^2$
3	.4036467	6.137583
4	.3825979	6.831482
5	.3784829	6.980836
6	.3771967	7.028524
7	.3766843	7.047660
8	.3764462	7.056577
9	.3763231	7.061196
10	.3762541	7.063786
20	.3761341	7.068293
100	.3761264	7.068583
∞	$2/3\sqrt{\pi}$ $\approx .3761264$	$9\pi/4$ ≈ 7.068583

Theorem 4 implies that asymptotically as K grows, the optimal partition of the square into polygons with respect to the valuation C^* is the one that consists of regular hexagons (if we ignore the effects of the boundaries of the unit square). A very simple and efficient heuristic for solving the $K(n)$ -median problem for this range of growths of $K(n)$ is immediately suggested by Theorem 4.

Given $P = \{p_1, \dots, p_n\}$:

(a) Find $K = K(n)$.

(b) Tile the plane with hexagons H_1, H_2, \dots each of area $1/K$. Choose those hexagons H_j for which $H_j \subseteq [0, 1]^2$. Let the set of their centers be $H = \{h_1, \dots, h_k\}$, $k \leq K$.

(c) Define the set of medians $M = \{m_1, \dots, m_k\} \subseteq P$ by $\text{dist}(m_i, h_j) \leq \text{dist}(p, h_j)$ for all $p \in P$.

The remaining part of this section is a probabilistic analysis of this *honeycomb heuristic*. Our probabilistic assumptions are that n is fixed and $P = \{p_1, \dots, p_n\}$ consists of points independently and uniformly distributed over the unit square. The following lemma is shown in [FH].

LEMMA 6. *If \hat{M} is the optimum solution, there is a constant c such that $C(\hat{M}) \cong cnK(n)^{1/2}$ with probability $1 - o(1)$.*

We also need the following lemma, which is a specialized result about multinomial distributions. Suppose that we have divided the unit square into 2^{2m} equal small squares, where $n2^{-2m} = n^\delta$ for some $\delta > 0$.

LEMMA 7. *With probability $1 - o(1)$ each small square will contain N_j points of P , where $|N_j - n^\delta| = o(n^\delta)$.*

To prove Lemma 7 we need a purely probabilistic fact:

LEMMA 8. *Let b be a binomially distributed random variable with probability $\frac{1}{2}$ and n trials. Then $\text{prob}(|b - n/2| > n^{5+\delta}) \leq e^{-n^\delta}$ for large enough n .*

Proof. It is well known that $\text{prob}(b = j) = B_{j,n} = \binom{n}{j} 2^{-n}$. By Stirling's formula

$$B_{j,n} = \frac{E(n, j)}{\sqrt{\pi n/2}} \left(\frac{n}{j}\right)^{j+1/2} \left(\frac{n}{n-j}\right)^{n-j-1/2} \cdot 2^{-n},$$

where $E(n, j)$ is an error term

$$E(n, j) = 1 + \frac{1}{1/12} \left(\frac{n^2 - nj + j^2}{nj(n-j)} \right) + \dots = O(1).$$

Let $j = n/2x$, $x \geq 0$ (for $x \leq 0$, similarly). Then, for some $c_1 > 0$,

$$\begin{aligned} B_{j,n} &\leq c_1 \left(n^{1/2} \left(\frac{n/2-x}{n/2} \right)^{-n/2+x} \left(\frac{n/2+x}{n/2} \right)^{-x-n/2} \right) \\ &= c_1 \left(n^{1/2} \left(1 - \frac{x^2}{(n/2)^2} \right)^{-n/2} \left(\frac{1 - \frac{x}{n/2}}{1 + \frac{x}{n/2}} \right)^x \right). \end{aligned}$$

Letting $z = x/(n/2)$, we have

$$B_{j,n} \leq c_1 (n^{1/2} (1-z^2)^{-n/2} \left(\frac{1-z}{1+z} \right)^x).$$

Let $A(j, n) = (1-z^2)^{-n/2} ((1-z/1+z))^x$; then

$$\begin{aligned} \log A(j, n) &= \frac{n}{2} \left(z^2 + \frac{z^4}{z} + \frac{z^6}{3} + \dots + x \left(-z - \frac{z^2}{z} - \frac{z^3}{3} \dots - z + \frac{z^2}{z} - \frac{z^3}{3} + \dots \right) \right) \\ &= \frac{n}{2} \sum_{i=0}^{\infty} a_i z^{2i} \quad \text{with } a_j = \frac{1}{j(2j-1)}. \end{aligned}$$

Hence $\log A(j, n) \leq -(n/2) z^2 = 2x^2/n$, and $B(j, n) \leq c_1 n^{1/2} e^{-2x^2/n}$. Hence

$$\text{prob} \left(\left| b - \frac{n}{2} \right| \geq n^{5+\delta} \right) \leq nB \left(\left| \frac{n}{2} - n^{5+\delta} \right|, n \right) \leq c_1 n^{3/2} e^{-n^{2\delta}} \leq e^{-n^\delta}$$

for large enough n . \square

Proof of Lemma 7. Fix an $\alpha > 0$, and define

$$p_j = j \exp(-n^{\alpha/2}), \quad f_j = \sum_{i=1}^j 2^{-i} (n 2^{-(j-1)})^{5+\alpha}.$$

We shall prove by induction on k that each of the 2^k subregions (squares if k even, rectangles if k odd) contain between $n 2^{-k} + f_k$ points, with probability $1 - p_k$. Notice that this settles the lemma.

To start the induction note that the assertion holds for $k = 0$. Suppose that it holds for $k = j$. Let A_{j+1} be one of the 2^{j+1} subregions, and let A_j be the unique subregion among the 2^j that contains it (if j is even, A_j is a square and A_{j+1} is its half rectangle; if j is odd, A_j is a rectangle consisting of two squares, and A_{j+1} is one of them). For N large enough, we have, by Lemma 8,

$$\text{prob} \left(A_{j+1} \text{ contains } \frac{N}{2} + N^{5+\alpha} \text{ pts} \mid A_j \text{ contains } N \right) \geq 1 - e^{-N^\alpha}.$$

Hence

$$\begin{aligned} \text{prob} (A_{j+1} \text{ has } n 2^{-(j+1)} \pm (\frac{1}{2} f_j + (n 2^{-j} + f_j)^{5+\alpha} \text{ pts} \mid A_j \text{ has } n 2^{-j} + f_j) \\ \geq 1 - \exp(-(n 2^{-j} - f_j)^\alpha). \end{aligned}$$

But this can be rewritten as

$$\begin{aligned} \text{prob}(A_{j+1} \text{ has } n2^{-(j+1)} \pm f_{j+1} \text{ pts} \mid A_j \text{ has } n2^{-j} \pm f_j) &\geq 1 - \exp(-(n2^{-j} - f_j)^\alpha), \\ \text{prob}(\text{all } A_{j+1}\text{'s have } n2^{-(j+1)} \pm f_{j+1} \text{ pts} \mid \text{all } A_j\text{'s have } n2^{-j} \pm f_j), \\ &\geq (1 - \exp(-(n2^{-j} - f_j)^\alpha))^{2^{j+1}} \geq 1 - 2^{j+1} \exp(-(n2^j - f_j)^\alpha) \\ &\geq 1 - \exp(-n^{\alpha\delta/2}). \end{aligned}$$

Thus

$$\begin{aligned} \text{prob}(\text{all } A_{j+1}\text{'s have } n2^{-(j+1)} \pm f_{j+1} \text{ pts}) \\ &\geq \text{prob}(\text{all } A_j\text{'s have } n2^{-j} \pm f_j \text{ pts})(1 - \exp(-n^{\alpha\delta/2})) \\ &\geq (1 - p_j)(1 - \exp(-n^{\alpha\delta/2})) \quad (\text{by induction hypothesis}) \\ &\geq 1 - \exp(n^{\alpha\delta/2}) - p_j \geq 1 - p_{j+1}. \end{aligned} \quad \square$$

Lemma 7 implies that with probability $1 - o(1)$ every subdivision of the unit square into not more than $n^{1-\epsilon}$ equal squares will contain only squares that have n^ϵ points, plus or minus a lower order term. It will be very useful in evaluating how well the continuous deterministic problem approximates the K -median problem.

The error analysis is rather simple, though tedious, since it involves the numbers $C(\hat{M})$, $nC^*(\hat{M})$, $C(M)$, $nC^*(H)$ and $\gamma(6)n/\sqrt{K}$; here \hat{M} is the optimal solution, M is the solution found by the heuristic and H is the set of hexagonal centers. C is our ordinary cost valuation, whereas C^* is its continuous counterpart. Our strategy is shown in Fig. 6. A solid undirected line between A and B means that we shall show—in the lemma whose number is indicated on the line—that $|A-B| = o(n/\sqrt{k(n)})$ with probability $1 - o(1)$. A broken directed line from A to B means that $A \geq B$. Once we establish all this, it is immediate that $C(M) - C(\hat{M}) = o(n/\sqrt{K(n)})$.

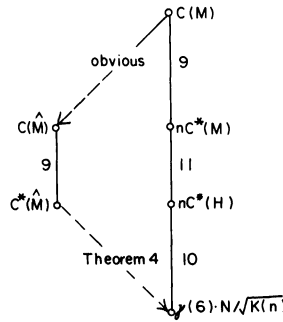


FIG. 6

LEMMA 9. $|C(\hat{M}) - nC^*(\hat{M})| = o(n/\sqrt{k(n)})$ with probability $(1 - o(1))$.

Proof. Recall that $K(n) = \lfloor n^\epsilon \rfloor$. Let us divide the unit square into 2^{2m} equal squares, where $m = \lfloor \log_2(n^\delta)/2 \rfloor$ for some $\epsilon < \delta < 1$. Let each square have a side of length Δ .

We shall show that with probability $1 - o(1)$ no point $p_j \in P$ has $d_{\hat{M}}(p_j) \geq c_1/\sqrt{K(n)}$ for some constant c_1 . Suppose that such a point p_j existed. Then, with probability $1 - o(1)$, the disk with center p_j and radius $c_1/3\sqrt{2K(n)}$ has at least $\pi c_1^2 n / 18K(n)$ points of P in it. This is established by considering the small $\Delta \times \Delta$ squares falling within this disk and applying Lemma 7. However, one could then make p_j into an additional median, at a savings of at least $(\pi n/54)(c_1/\sqrt{K(n)})^3$. Therefore, one can

argue in a manner identical to Lemma 1, that there exist two medians in \hat{M} within distance $(2K(n))^{-1/2}$, each having at most $2n/K(n)$ points. By choosing δ appropriately, we establish that $\max_j d_{\hat{M}}(p_j) \leq c_1/\sqrt{K(n)}$ with probability $1 - o(1)$.

Let us define now yet another valuation \bar{C} of any set of medians. \bar{C} is a discretized version of C^* : we assume that one point of weight $n\Delta^2$ is in the center c_i of each $\Delta \times \Delta$ square S_j and calculate

$$\bar{C}(M) = \sum_{j=1}^K n\Delta^2 \sum_{S_i \in D_j} \text{dist}(m_j, c_i).$$

Let us calculate $|\bar{C}(\hat{M}) - nC^*(\hat{M})|$. This difference is due to “lost” squares along the perimeters of the D_j ’s, and also to a “discretization” error. The perimeter does not exceed $c_2\sqrt{K(n)}$, since it consists of $O(K(n))$ sides, all bounded, by the above remark, by $c_1/\sqrt{K(n)}$; so there are at most $c_3\sqrt{K(n)}/\Delta$ lost squares, a total error $\leq n\Delta^2 c_3(\sqrt{K(n)}/\Delta) \max_j d_{\hat{M}}(p_j) = c_3 n\Delta = o(n/\sqrt{K(n)})$. The “discretization” error totals to, at most, $n\Delta\sqrt{2}/2$, also $o(n/\sqrt{K(n)})$. Thus $|\bar{C}(\hat{M}) - nC^*(\hat{M})| = o(n/\sqrt{K(n)})$.

Let us now evaluate $|\bar{C}(\hat{M}) - C(\hat{M})|$. This difference contains also a “distribution” error (squares that have more or fewer points than their share), besides the boundary and discretization errors. However, Lemma 7 says that, with probability $1 - o(1)$, this new error is $o(n) \max_j d_{\hat{M}}(p_j) = o(n/\sqrt{K(n)})$. \square

That $|n\bar{C}(\hat{M}) - C^*(\hat{M})| = o(n/\sqrt{K(n)})$ can be shown in a very similar way to Lemma 9.

We now turn to

LEMMA 10. $n|C^*(H) - \gamma(6)/\sqrt{K(n)}| = o(n/\sqrt{K(n)})$ with probability $1 - o(1)$

Proof. Each of the hexagons in the tiling $\{H_1, H_2, \dots\}$ has area $1/K(n)$, and therefore side $c_4/\sqrt{K(n)}$. Thus, there are at most $c_5\sqrt{K(n)}$ hexagons that cross the boundary of the unit square. It is contributions from these squares that increase $C^*(H)$ away from $\gamma(6)/\sqrt{K(n)}$. However, each hexagon on the boundary adds at most $c_6(K(n))^{-3/2}$ to $C^*(H)$, and thus the total deviation is $n|C^*(H) - \gamma(6)/\sqrt{K(n)}| \leq c_7 n/K(n) = o(n/\sqrt{K(n)})$. \square

LEMMA 11. $n|C^*(M) - C^*(H)| = o(n/\sqrt{K(n)})$ with probability $1 - o(1)$.

Proof. The difference between $C^*(M)$ and $C^*(H)$ is due to the “displacement” of the medians from the centers of the hexagons to the points closest to the centers. Now, each center of a hexagon falls in one of the $\Delta \times \Delta$ squares ($\Delta = n^{-1/2+\delta}$, arbitrary $\delta > 0$), and we know that, with probability $1 - o(1)$, there is at least one point from P in each square (Lemma 7). Thus this displacement is for no center greater than $\sqrt{2}\Delta$, with probability $1 - o(1)$. The total error due to displacement is therefore no greater than $n\sqrt{2}\Delta$. Taking $\delta < (1 - \epsilon)/2$, we prove the lemma. \square

We can finally show:

THEOREM 5. *The honeycomb heuristic constructs in time $O(n \log n)$ a set M of medians having relative error*

$$(C(M) - C(\hat{M})/C(\hat{M})) = o(1) \quad \text{with probability } 1 - o(1).$$

Proof. The error analysis follows from Lemmas 9–11 and Theorem 5. For the time bound, we have to show that in time $O(n \log n)$ we can find for each point in a set H , $|H| \leq N$, the closest to it from another point set P , $|P| \leq n$. This, however, is possible by the Voronoi techniques of Shamos [Sh]. \square

By Theorem 5, we can explicitly calculate the *exact limit* of the optimal cost for this range of $K(n)$:

COROLLARY. For $K(n) = \omega(\log n)$, $K(n) = o(n/\log n)$, we have

$$\text{prob} \left(\left| \frac{C(\hat{M})\sqrt{K(n)}}{n} - \gamma(6) \right| > \varepsilon \right) = 1 - o(1) \quad \text{for all } \varepsilon > 0.$$

5. Discussion. We note that no NP-completeness results are known for the following two variants of the K -median problem:

- (a) One does not restrict M to be a subset of P . That is, the K medians can be chosen to be totally new points.
- (b) The min-max version of the K -median problem.

We conjecture that both problems are NP-complete.

As far as probabilistic analysis of heuristics is concerned, our results leave open two regions of the spectrum of growth of $K(n)$:

- (a) $K(n) = c \log n$.
- (b) $\frac{c_1 n}{\log n} \leq K(n) \leq \frac{c_2 n}{(\log n)^{1/3}}$.

For the case that $K(n) = \lfloor \alpha n \rfloor$, a very interesting recent result by J. Michael Steele [Ste] may simplify our approach considerably. Steele proved the following: Let any valuation f mapping finite point sets to the reals satisfy the following properties:

- (a) f is Euclidean, i.e., linear and invariant under translations.
- (b) f is monotone; i.e., $f(P \cup \{p_{n+1}\}) \geq f(P)$.
- (c) f has bounded variance, under the uniform distribution.
- (d) f is subadditive, i.e., if $\{S_i\}_{i=1}^m$ is a partition of the unit square into squares or total perimeter L , $f(P) \leq \sum_{i=1}^m f(P \cap S_i) + O(L)$.

Then, with probability 1,

$$\lim_{n \rightarrow \infty} \left(\frac{f(p_1, \dots, p_n)}{\sqrt{n}} \right) = \beta_f \quad \text{a constant.}$$

Using this theorem, Steele gives a simple derivation of the Beardwood, Halton, and Hammersley theorem [BHH]. Notice that the valuation $f_\alpha(P) = \min \{C(M) : |M| = \lfloor \alpha |P| \rfloor\}$ for some α $0 < \alpha < 1$, does not satisfy conditions (b) and (d) above; however, Steele claims that the conclusion of the theorem still holds for F_α [Ste]. Explicit proofs of this fact have actually appeared [Ha], [Ho]. This suggests the following simple partition heuristic for the $K(n) = \lfloor \alpha n \rfloor$ case:

- 1) Partition the unit square into $\sim n/\log n$ smaller squares.
- 2) Solve the $K(n)$ -MEDIAN problem for each of the smaller squares.

As a result of Steele's theorem, the optimum such *restricted* separable solution gives a solution that is asymptotically very close to the exact optimum. We note, however, that our approach is still necessary for $K(n) = o(n)$.

Acknowledgment. Much of the motivation for this work, as well as more technical influence, came from discussions with Dorit Hochbaum and Dick Karp. The conjecture which eventually became Theorem 4 is due to the insights of Gérard Cornuejols. Some of the calculations were made using the symbolic manipulation system MACSYMA at MIT. Many thanks to Amedeo Odoni for correcting an error in the formula and table for $\gamma(n)$.

REFERENCES

- [AHU] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1978.
- [BHH] J. BEARDWOOD, J. H. HALTON AND J. M. HAMMERSLEY, *The shortest path through many points*, Proc. Cambridge Philo. Soc., 55 (1959), pp. 299–327.
- [BO] B. BOLLOBAS, *The optimal arrangement of producers*, J. London Math. Society (2), 6 (1973), pp. 417–421.
- [CFN] G. CORNUEJOLS, M. L. FISCHER AND G. L. NEMHAUSER, *Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms*, Management Sci., 23 (1978), pp. 789–810.
- [CNW] G. CORNUEJOLS, G. L. NEMHAUSER AND L. A. WOLSEY, *Worst-case and Probabilistic Analysis of Algorithms for a Location Problem*, TR375, IEOR, Cornell University, Ithaca, NY, 1978.
- [FH] M. L. FISCHER AND D. S. HOCHBAUM, *Probabilistic Analysis of the Euclidean K-median problem*, Math. Oper. Res., to appear.
- [FT1] L. FEJES TOTH, *Lagerungen in der Ebene, auf der Kugel und im Raum*, Berlin, 1953.
- [FT2] ———, *Sum of moments of convex polygons*, Acta Math. Acad. Scient. Hungaricae, 24 (3–4), (1973) pp. 417–421.
- [GGJ] M. R. GAREY, R. L. GRAHAM AND D. S. JOHNSON, *Some NP-complete geometric problems*, Proc. 8th ACM Symposium on Theory of Computing, 1976, pp. 10–27.
- [GJ] M. R. GAREY AND D. J. JOHNSON, *Computers & Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
- [Ha] M. HAIMOVICH, manuscript, Massachusetts Institute of Technology, Cambridge, MA, 1979.
- [He] P. J. HEAWOOD, *Map colour theorem*, Quart. J. Math., 24, (1980) pp. 332–338.
- [Ho] D. S. HOCHBAUM, *The probabilistic asymptotic properties of some combinatorial geometric problems*, manuscript, Carnegie-Mellon University, November, 1979.
- [Ka1] R. M. KARP, *Reducibilities among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller, J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–104.
- [Ka2] ———, *The probabilistic analysis of some combinatorial search algorithms*, in Algorithms and Complexity: New Directions and Recent Results, J. F. Traub, ed. Academic Press, New York, 1977.
- [Ka3] ———, *Probabilistic analysis of partitioning algorithms for the travelling salesman problem*, Math. Oper. Res., 2 (1977), pp. 209–224.
- [KH] O. KARIV AND S. L. HAKIMI, *An algorithmic approach to network location problems Part 2: the p-medians*, manuscript, 1976.
- [Pa1] C. H. PAPADIMITRIOU, *The Euclidean travelling salesman problem is NP-complete*, J. Theor. Comput. Sci., 4 (1977), pp. 237–244.
- [Pa2] ———, *The complexity of combinatorial optimization problems*, Ph.D. Thesis, Princeton University, Princeton, NJ, 1976.
- [PS] C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial optimization: Algorithms & Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [Sh] M. I. SHAMOS, *Computational Geometry*, Ph.D. thesis, Yale University, New Haven, CT, 1978.
- [Sta] D. A. STARRET, *Principles of optimal location in a large homogeneous area*, J. Econ. Theory, 9 (1974), pp. 418–448.
- [Ste] J. MICHAEL STEELE, *Subadditive Euclidean functionals and non-linear growth in geometric probability*, submitted to Ann. Probab.

KEY COMPARISON OPTIMAL 2-3 TREES WITH MAXIMUM UTILIZATION*

JAMES R. BITNER† AND SHOU-HSUAN HUANG†

Abstract. To study the relationship between key-comparison cost and utilization (space efficiency) of 2-3 trees, we define a class of 2-3 trees called *kcu-optimal* trees, which, out of all 2-3 trees with optimal key-comparison cost, have maximum utilization. A characterization theorem for this class is proved and the “average” utilization is found to be 64.7%, showing that these properties are not totally incompatible. Finally, a linear-time algorithm is given to create a *kcu-optimal* tree from a sorted array of keys.

Key words. B-trees, 2-3 trees, storage utilization, optimality, bushy, scrawny, compact and minimal comparison 2-3 trees

1. Introduction. Three cost measures (measures of “goodness”) can be defined for *B*-trees: node-visit cost, key-comparison cost and utilization (these terms are defined below; see § 1.1). This paper is concerned with the relationships between these different measures. We are interested in questions such as “if a *B*-tree is optimal for one given measure, is it necessarily optimal or near-optimal for another given measure?” or “can it be shown that *B*-trees which are optimal for one given measure must be very poor for another?” We study the relationship between key-comparison cost and utilization (the other two possible pairs have been previously studied; see § 1.2). Previous results (see § 1.3) suggest that these properties are incompatible, i.e., that a 2-3 tree with optimal key-comparison cost *must* have very poor utilization. Our result is to show that this is not the case. We first consider the set of 2-3 trees which have optimal key-comparison cost and find a characterization for the 2-3 trees which have maximal utilization over this set. We then compute their utilization and show it is significantly higher than the minimum. We also show the space-efficiency compares favorably with “random” 2-3 trees [5].

1.1. Definitions. Our definition of *B*-trees and 2-3 trees is taken from Knuth [2]. Note that under this definition the nodes having no sons (called *leaves*) do not carry information. The *height* of a *B*-tree is the length of the path from the root to any leaf. (Thus, the tree consisting of a single node with leaves as sons has height *one*.)

Notation. We let *K* and *N* stand for, respectively, the number of keys and *internal* (nonleaf) nodes in a given 2-3 tree.

DEFINITION. In a 2-3 tree, a *1-node* is defined as a node containing one key and a *2-node* as a node containing two keys.

We now define our cost-measures for 2-3 trees; this will be sufficient for our purposes. (All except for key-comparison cost are easily generalized to *B*-trees). In defining the *key-comparison cost* of a 2-3 tree, we assume that a “reasonable” algorithm such as Algorithm A.1 (see Appendix) is used. To calculate c_i ($i = 1, \dots, K$), the number of key comparisons made in finding key *i*, note that one comparison is made at each 1-node on the path from the root to key *i*. One comparison is also made at 2-nodes if key *i* lies in the left subtree or is the left key in the node. Otherwise, two comparisons are required. The expected number of key-comparisons is then the average, i.e., $(\sum_{i=1}^K c_i)/K$.

* Received by the editors April 2, 1979, and in final revised form September 1, 1980. This work was supported in part by the National Science Foundation under grant MCS 77-02705.

† Department of Computer Science, University of Texas, Austin, Texas 78712.

DEFINITION. Let n_i ($i = 1, \dots, K$) be the number of nodes on the path from the root to the node in which key i resides (i.e., its level plus 1). Then the *node-visit cost* is $(\sum_{i=1}^K n_i)/K$. (The node-visit cost measures the average number of distinct nodes accessed in finding a key, which gives an idea of how many pages must be brought in during such a search if the tree is stored on disk.)

DEFINITION. The *utilization* of a 2-3 tree is $K/2N$ (the ratio of the number of keys to the number of possible keys). Note the utilization is bounded between 50% and 100%.

DEFINITION. The *expansion* of a 2-3 tree is $2N/K - 1$ and is bounded between 0% and 100%. Thus, a tree with an expansion of 50% takes 50% more space than is theoretically necessary.

We study the utilization because it is a more intuitive measure and the expansion because it allows a comparison with the bounds on the average expansion of random 2-3 trees. (Note that no nontrivial bounds are known for the utilization.)

DEFINITION. A 2-3 tree is *kc-optimal* (called minimal comparison in [1]) if and only if its key-comparison cost is minimal over all 2-3 trees having the same number of keys.

DEFINITION. A 2-3 tree is respectively, *bushy* or *scrawny* if and only if its node-visit cost is, respectively, minimal or maximal over all 2-3 trees with the same number of keys.

DEFINITION. A 2-3 tree is *compact* if and only if its utilization is maximal over all 2-3 trees with the same number of keys. A new term, which this paper will discuss in detail, is:

DEFINITION. A 2-3 tree is *kcu-optimal* if and only if it has maximum utilization among all kc-optimal trees having the same number of keys.

1.2. Previous related results. Characterization theorems have been proved for bushy and scrawny trees [3], kc-optimal trees [1] and compact trees [4]. We will only make use of the characterization of kc-optimal trees:

THEOREM 1.1 (Rosenberg and Snyder [1]). *A 2-3 tree is kc-optimal if and only if no 2-node has a 2-node in either its middle or right subtrees.*

Further, the relationships between the three different cost measures have been studied. It has been shown [4] that compact trees have very good node-visit cost, at most one more than the minimum node-visit cost required of any tree with the same number of keys. However, the utilization of a compact tree is significantly higher than that of bushy trees with the same number of keys [4]. For large B -trees of order 3 (i.e., 2-3 trees), the utilization is higher by a multiplicative factor of $\frac{11}{6}$, and for order 4, by a factor of $\frac{5}{2}$. (These factors are the largest for odd and even orders, respectively; the ratios for B -trees of arbitrary order is given in [4].) Note that these ratios are extremely high. For 2-3 trees, the highest possible ratio is 2 (100% utilization versus 50%). Similarly, for B -trees of order 4 the highest possible ratio is 3.

The relationship between node-visit cost and key-comparison cost has also been studied [1]. Trying to optimize both these properties is quite difficult. The characterization theorems prescribe that the 2-nodes be near the root in bushy trees and near the leaves in kc-optimal trees. Kc-optimality and bushyness do sometimes coincide, however, but only for small trees. There exists a 2-3 tree with L leaves which is both kc-optimal and node-visit optimal iff $L \in \{2 \dots 7, 10 \dots 15, 28 \dots 31\}$. However, coincidence between kc-optimal and scrawny trees is very common. Let $\lambda(L) = \lceil \log_2 L \rceil$. For $L \geq 8$, there exists a 2-3 tree with L leaves which is kc-optimal and scrawny if and only if $2^{\lambda(L)} \leq L \leq 3 \cdot 2^{\lambda(L)-1} + 1$. Further, for all L such that $2^{\lambda(L)} \leq L \leq 3 \cdot 2^{\lambda(L)-1}$, every scrawny tree with L leaves is kc-optimal.

The relation between utilization and key-comparison cost has not yet been studied, and this will be the focus of this paper.

1.3. Our results and their relation to previous work. From previous results, it appears that low key-comparison cost and high utilization are not compatible. In [1], Rosenberg and Synder gave an algorithm to construct a kc-optimal tree from a sorted list of keys. This construction builds a tree whose 2-nodes occur only on the path from the root to the leftmost leaf. Clearly, this will asymptotically give a utilization of 50%, the minimum possible.

We show that these measures are compatible to some extent. In § 2, we prove a theorem characterizing kcu-optimal trees. Then, in § 3, we calculate the “average” utilization for large kcu-optimal trees and obtain a utilization of 64.7%, which is significantly higher than 50%. To compare their space efficiency with random 2-3 trees, we calculate the “average” expansion and obtain a result of 56.7% which compares favorably with that of random 2-3 trees which is known to be bounded [5] between 40% and 58%. Finally, in § 4, we give a linear-time algorithm for construction of a kcu-optimal tree from a sorted array of keys.

2. A characterization theorem for kcu-optimality. In this section, we prove a characterization theorem for kcu-optimal trees. Obviously, kc-optimality is a necessary condition. We will establish two additional simple and necessary conditions (Theorems 2.1 and 2.2) for kcu-optimality, then prove these three conditions to be sufficient.

THEOREM 2.1. *In a kcu-optimal tree a 2-node cannot have a 1-node as its left son.*

Proof. Suppose there is a 2-node having a 1-node as its left son. Apply the transform shown in Fig. 2.1. This preserves the number of nodes in the tree and the fact that the tree is kc-optimal (since S_3, S_4, S_5 and S_6 must be completely binary). However,

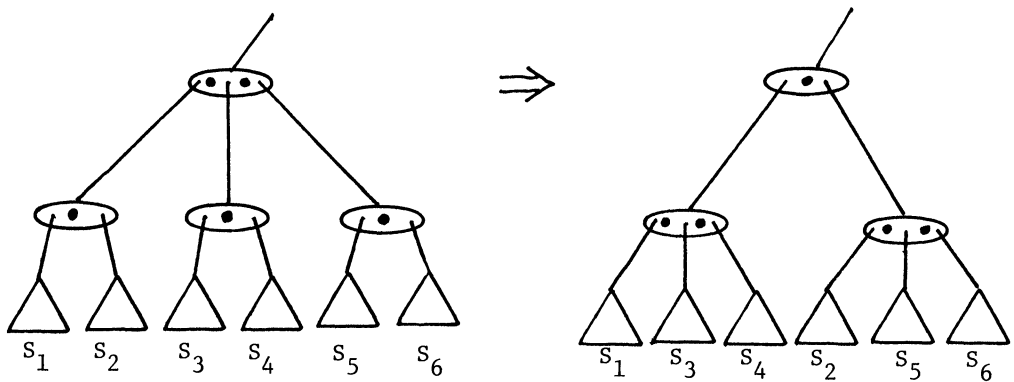


FIG. 2.1. A transformation that increases the number of 2-nodes.

the number of 2-nodes is increased by one, increasing the utilization. Therefore the original tree did not have maximum utilization, a contradiction. □

This theorem says that a kcu-optimal tree with a 2-node as its root must have a special form:

DEFINITION. A 2-3 tree is *full* if and only if it has the form shown in Fig. 2.2.

LEMMA 2.1. *A 2-node in a kcu-optimal tree must be the root of a full subtree.*

Proof. By Theorem 2.1, every internal node on the path from the root to the leftmost leaf must be a 2-node. Since the tree is kc-optimal, every other internal node must be a 1-node. □

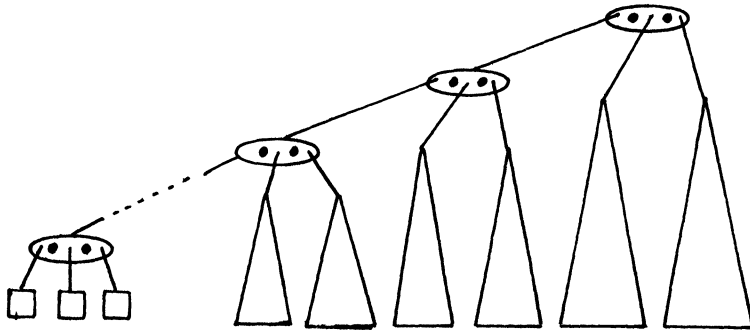


FIG. 2.2. A full tree. In this figure, the triangles are completely binary trees, and the squares are leaves.

A 2-node that has a 1-node for its father (and thus is not part of a larger full tree) is of special interest, motivating the following definition.

DEFINITION. A leader is a 2-node or a leaf which has no 2-nodes as ancestors.

THEOREM 2.2. In a kcu-optimal tree, the levels of two leaders may differ by at most one.

Proof. By Theorem 2.1 we can assume that each 2-node has a leaf or a 2-node as its left son. Suppose there are leaders (l_1 and l_2) at height i and j with $i - j > 1$. Note that l_1 cannot be an ancestor of l_2 , because l_1 's subtree would then be full and l_2 would not be a leader. Also note that l_2 's father must be a 1-node and that l_1 's and l_2 's subtrees are full. We assume without loss of generality that l_2 is the right son of its father.

We now describe the transformation shown in Fig. 2.3, which will preserve the number of keys and the kc-optimality but will increase the utilization. Applying this

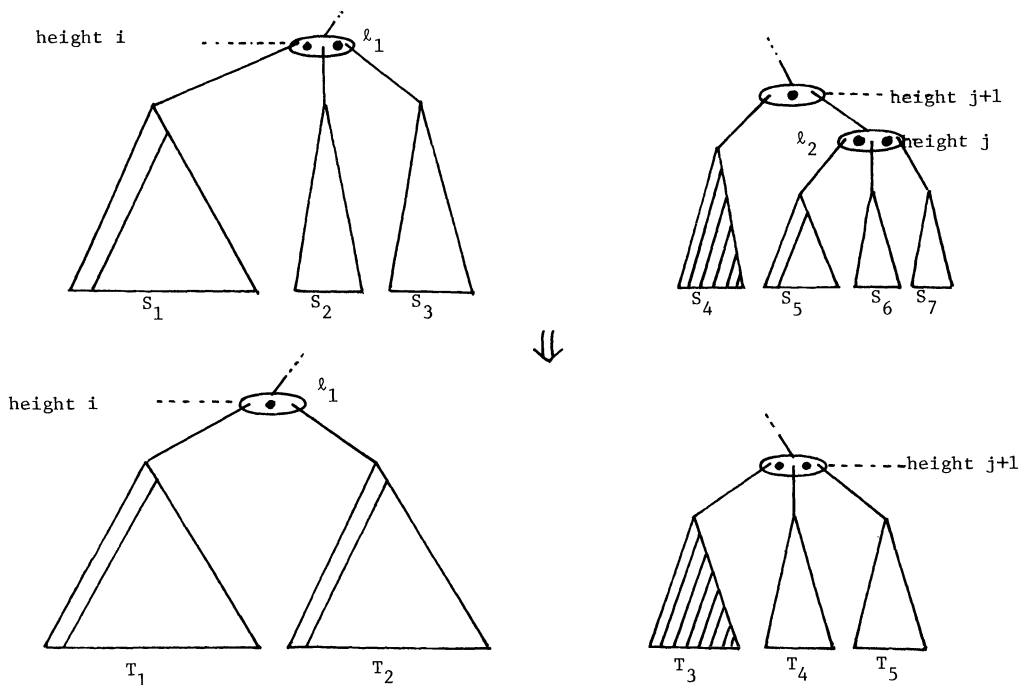


FIG. 2.3. The two subtrees affected by the transformation are shown. Triangles represent completely binary trees. Triangles with a stripe (such as S_1) are full trees. Shaded trees may have arbitrary form.

transformation to the original tree will then give a contradiction. Subtrees S_1 and S_4 are not affected and become trees T_1 and T_3 respectively. The keys in S_2 and S_3 are merged to form T_2 ; then one key is deleted from l_1 . There are $2^{j+1} - 2$ keys in l_2 and its subtrees. (It is easily shown that a full tree of height h has $2^{h+1} - 2$ keys.) These are redistributed to form T_4 and T_5 . Finally, a key is added to l_2 's father. (Note that the transformation will work even if l_2 is a leaf; then S_5, S_6 and S_7 are empty.) In total, the number of keys remained the same, and the tree is still kc-optimal. However, the number of 2-nodes has increased; originally there were $i + j + x$ 2-nodes in the affected subtrees (where x is the number of 2-nodes in S_4) and now there are $2i - 1 + x$. Hence the utilization of a kcu-optimal tree has been increased, a contradiction. \square

Trees satisfying the conditions of Theorems 2.1 and 2.2 have a very restricted form. For some l , the first $l - 1$ levels are completely binary. Then, at level l , we have some 2-nodes (which must be leaders). The remaining nodes (if any) at level l must be 1-nodes that have two 2-nodes (also leaders) as sons. (Note that if l is large enough we will sometimes have to replace "2-node" by "leaf" in the above.) Every 2-node is, by Lemma 2.1, the root of a full subtree (whose form is completely determined). Thus only three parameters are needed to determine many properties such as number of keys, 2-nodes and utilization of such a tree, and these are given in the following definition.

DEFINITION. If a 2-3 tree satisfies the conditions of Theorems 2.1 and 2.2, it is said to *have a leader profile*. Trees not satisfying either or both conditions do *not* have a leader profile. A *leader profile* is an ordered triple (h, l, x) where h is the height of the tree, l is the level of the leader with lowest (i.e., numerically smallest) level, and x is the number of leaders having lowest level. In addition, if the tree consists solely of 1-nodes (and hence all leaders are leaves) it will be more convenient to define the leader profile to be $(h, h - 1, 0)$ instead of $(h, h, 2^h)$. (Note that knowing x also determines the numbers of nodes at the higher level.)

The remainder of this section will show that kc-optimality and the conditions in Theorems 2.1 and 2.2 are sufficient for kcu-optimality. To do this, we show the leader profile uniquely determines the number of keys in a 2-3 tree (provided it has a leader profile) and its utilization. We then show two 2-3 trees have the same number of keys if and only if they have the same leader profile. These two results will allow us to prove sufficiency.

THEOREM 2.3. *Let keys (h, l, x) , nodes (h, l, x) and twos (h, l, x) be respectively the number of keys, internal nodes and 2-nodes in a 2-3 tree having leader profile (h, l, x) . These functions are well defined and have the following values:*

$$\begin{aligned} \text{keys}(h, l, x) &= 2^{h+1} - 2^{l+1} - 1 + x, \\ \text{nodes}(h, l, x) &= 2^{h+1} - (h - l)2^{l+1} + (h - l - 1)x - 1, \\ \text{twos}(h, l, x) &= (h - l - 1)2^{l+1} - (h - l - 2)x. \end{aligned}$$

Proof. The tree has the following form: For level $0, \dots, l - 1$, the tree is completely binary. At level l , there are x 2-nodes and at level $l + 1$ there are $2(2^l - x)$ 2-node leaders and x 2-nodes which are sons of 2-nodes. For level $l + 2, \dots, h - 1$, the population of nodes can be calculated by observing that a 2-node will have a 2-node and two 1-nodes as its sons and a 1-node will have two 1-nodes. Clearly, the functions are well defined, since the population of nodes at each level is completely determined by the leader profile.

These results are summarized in Table 2.1. Summing over each column proves the theorem for $l < h - 1$. For $l = h - 1$, the last two rows are zero, and it is easy to verify that the theorem is also true. \square

TABLE 2.1.
A level-by-level analysis of the structure of a 2-3 tree with leader profile (h, l, x) when $l < h - 1$. If $l = h - 1$, the last two rows should be zero.

level	number of nodes	number of 2-nodes	number of keys
$0 \leq i \leq l - 1$	2^i	0	2^i
l	2^l	x	$2^l + x$
$l + 1$	$2^{l+1} + x$	$2^{l+1} - x$	2^{l+2}
$l + 2 \leq i \leq h - 1$	$2^{i+1} - 2^{l+1} + x$	$2^{i+1} - x$	2^{i+1}

COROLLARY 2.1. If a 2-3 tree has a leader profile, then its utilization is completely determined by that leader profile.

Proof. By Theorem 2.3, the leader profile determines the number of keys, K and nodes, N , and the utilization is $K/2N$. \square

To show that two 2-3 trees have the same number of keys if and only if they have the same leader profile, i.e., the function $\text{keys}(h, l, x)$ is one-to-one, we demonstrate an order for sequencing through all the leader profiles such that $\text{keys}(h, l, x)$ is monotonically increasing. (This order is interesting in its own right.)

LEMMA 2.2. Let $\text{keys}(h, l, x)$ be the number of keys in 2-3 tree with leader profile (h, l, x) . Then

$$\begin{aligned} \text{keys}(h, h - 1, 1) &= \text{keys}(h, h - 1, 0) \quad \text{for } h > 0, \\ \text{keys}(h, l, x + 1) &= \text{keys}(h, l, x) + 1 \quad \text{for } 1 \leq x \leq 2^l - 1, \\ \text{keys}(h, l - 1, 1) &= \text{keys}(h, l, 2^l) + 1 \quad \text{for } 1 \leq l \leq h - 1, \\ \text{keys}(h + 1, h, 0) &= \text{keys}(h, 0, 1) + 1 \quad \text{for } h > 0. \end{aligned}$$

Proof. Obvious from the relations for $\text{keys}(h, l, x)$ in Theorem 2.3. \square

The order of sequencing is obviously given by:

COROLLARY 2.2. For a given h , the sequence of leader profiles for $K = 2^h - 1, \dots, 2^{h+1} - 2$ begins with $(h, h - 1, 0)$ and then consists of h subsequences. The l th subsequence from the end ($l = h - 1, \dots, 0$) consists of $(h, l, 1), (h, l, 2), \dots, (h, l, 2^l)$. (See Table 2.2 and Fig. 2.4.)

THEOREM 2.4. The function $\text{keys}(h, l, x)$ is one-to-one; that is, two trees have the same number of keys iff they have the same leader profile.

Proof. It is obvious that the order of sequencing given by Corollary 2.2 will include all leader profiles and that the value of "keys" for these leader profiles will be

TABLE 2.2
The sequence of leader profiles for $h = 3$.

K	Leader profile
7	(3, 2, 0)
8	(3, 2, 1)
9	(3, 2, 2)
10	(3, 2, 3)
11	(3, 2, 4)
12	(3, 1, 1)
13	(3, 1, 2)
14	(3, 0, 1)

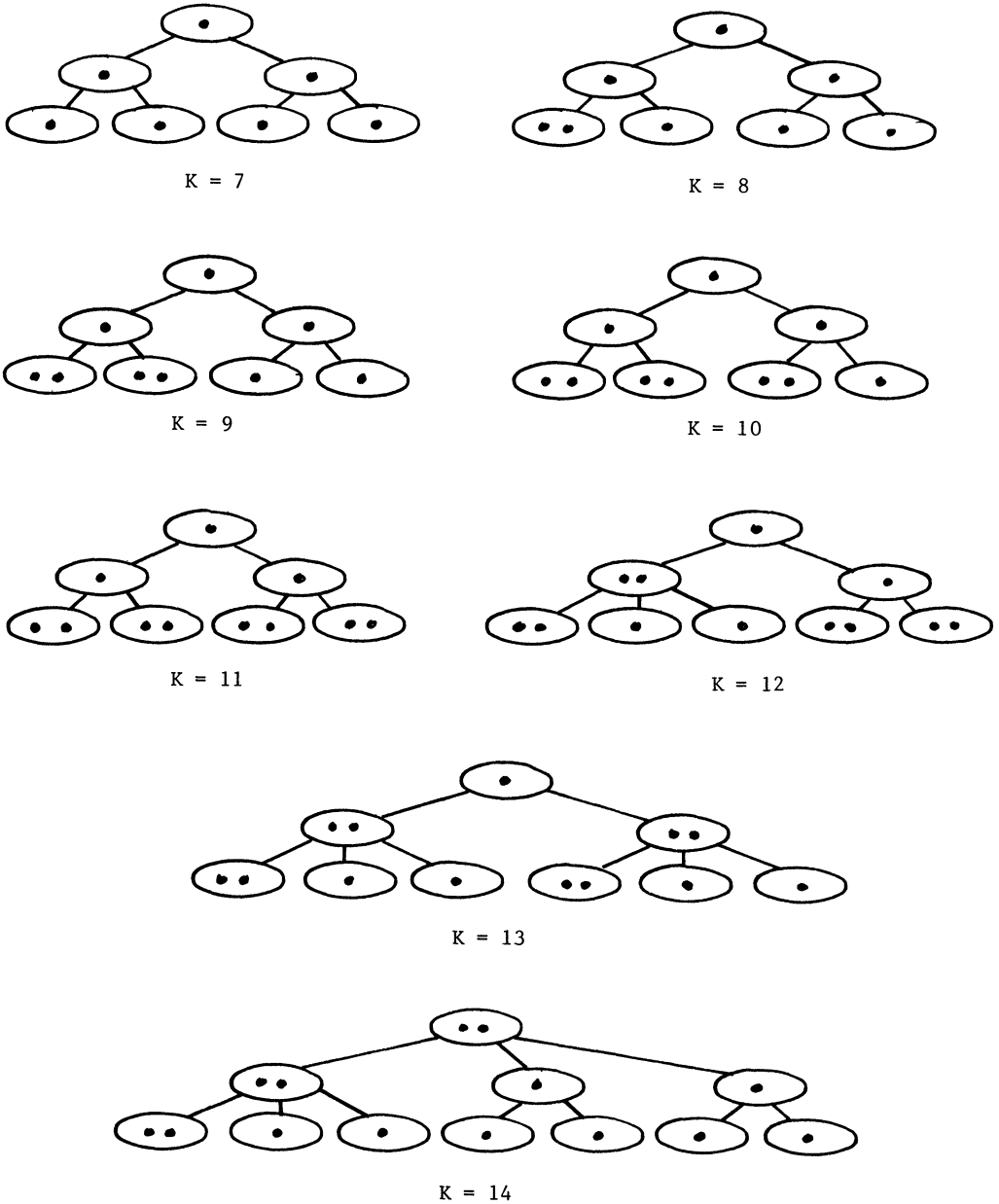


FIG. 2.4. *kcu-optimal trees of height 3. For simplicity, the leaves are not shown.*

monotonically increasing. Given two distinct leader profiles, (h_1, l_1, x_1) and (h_2, l_2, x_2) , one occurs first, say, (h_1, l_1, x_1) , and hence keys $(h_1, l_1, x_1) < \text{keys}(h_2, l_2, x_2)$. \square

THEOREM 2.5. *A 2-3 tree is kcu-optimal if and only if:*

- (1) *It is kc-optimal.*
- (2) *Every 2-node has a leaf or a 2-node as its left son.*
- (3) *The heights of any two leaders differ by at most one.*

Proof. Property (1) is necessary by the definition of kcu-optimality, and properties (2) and (3) by Theorems 2.1 and 2.2 respectively. Suppose then that these are not sufficient. Then there are two trees, T_1 and T_2 , which satisfy properties (1)–(3) and have

the same number of keys, but T_1 has higher utilization than T_2 . Since T_1 and T_2 satisfy properties (2) and (3), they have leader profiles and, in fact, by Theorem 2.4, they have the same leader profile. By Corollary 2.1, they must have the same utilization, a contradiction. \square

3. Calculating the utilization and expansion. It is clear that we can now get the utilization of a kcu-optimal tree with K keys by finding the leader profile (h, l, x) with keys $(h, l, x) = K$ (this is easily done using Corollary 2.2) and then applying Theorem 2.3. However, this formula would be complex and rather unenlightening. We would prefer a simple numerical estimate of how these trees perform on the average. Unfortunately, the utilization does not approach a limit as $K \rightarrow \infty$. The graph of utilization versus K does, however, have a very regular form (see Fig. 3.1). The K -axis can be broken into regions where region h ($h = 1, 2, \dots$) consists of $K =$

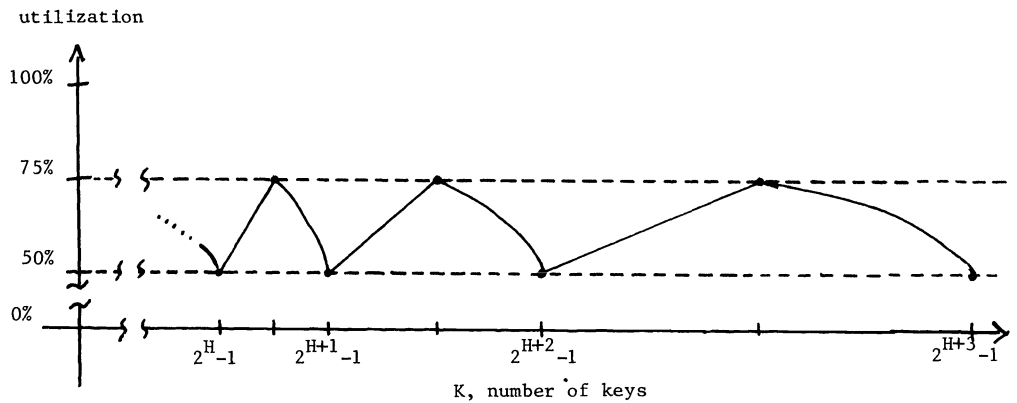


FIG. 3.1. A graph of the utilization of kcu-optimal trees versus K , the number of keys in the tree (H is some arbitrary, large integer).

$2^h - 1, \dots, 2^{h+1} - 2$ (i.e., all values of K such that a kcu-optimal tree of K keys has height h). The graph behaves nearly identically in each region: For the h th region, when $K = 2^h - 1$, the leader profile is $(h, h - 1, 0)$. The tree is completely binary, and the utilization is 50%. The utilization then increases linearly until $K = 2^h + 2^{h-1} - 1$ (i.e., the leader profile is $(h, h - 1, 2^{h-1})$). Here, the tree is completely binary except for level $h - 1$, which consists solely of 2-nodes. This tree, then, has a utilization of approximately 75%. The utilization then decreases back down to 50% which is reached when $K = 2^{h+1} - 1$, and a new region begins. Since the form of the graph is identical for each region, we can get a good idea of the average performance by calculating the average over a region for some large h (i.e., over all trees of height h). (It can, in fact, be shown that the utilization averaged over region h approaches a limit as $h \rightarrow \infty$, and we will calculate this limit.) First, two lemmas are required.

LEMMA 3.1. If $f(x)$ is monotone over $[a, b]$ then

$$\min(f(a), f(b)) + \int_a^b f(x) dx \leq \sum_{i=a}^b f(i) \leq \max(f(a), f(b)) + \int_a^b f(x) dx.$$

Hence

$$\sum_{i=a}^b f(i) = \left(\int_a^b f(x) dx \right) + \epsilon, \quad \text{where } \min(f(a), f(b)) \leq \epsilon \leq \max(f(a), f(b)).$$

LEMMA 3.2. Let t be an integer ≥ 1 , let $r \neq 0$, and suppose $-s/r \notin [1, t]$. Then

$$\sum_{i=1}^t \frac{pi+q}{ri+s} = \frac{1}{r} \left[p(t-1) + \left(q - \frac{ps}{r} \right) \ln \left(\frac{rt+s}{r+s} \right) \right] + \varepsilon,$$

where

$$\min \left(\frac{p+q}{r+s}, \frac{pt+q}{rt+s} \right) \leq \varepsilon \leq \max \left(\frac{p+q}{r+s}, \frac{pt+q}{rt+s} \right).$$

Proof. Let $f(x) = (px + q)/(rx + s)$. Then $f'(x) = (ps - qr)/(rx + s)^2$, and the sign of $f'(x)$ depends only on $ps - qr$, not x . Hence $f(x)$ is monotone over any region not containing $-s/r$. The lemma then follows directly from Lemma 3.1 and the calculation of $\int_1^t (px + q)/(rx + s) dx$ using the substitution $y = rx + s$. \square

THEOREM 3.1. Let nodes(K) be the number of nodes in a kcu-optimal tree of K keys. Then for large h , the utilization $(=K/(2 \cdot \text{nodes}(K)))$ averaged over $K = 2^h - 1, \dots, 2^{h+1} - 2$ is

$$\frac{5}{16} + \sum_{m=1}^{\infty} \frac{1}{2m} \left[1 + \left(\frac{(m-1)2^{m+2} + 2}{m} \right) \ln \left(\frac{2^{m+2} - (m+2)}{2^{m+2} - (2m-1)} \right) \right] \cdot 2^{-(m+1)},$$

which can be numerically calculated as 64.7%.

Proof. We need to calculate

$$\left(\sum_{K=2^h-1}^{2^{h+1}-2} \frac{K}{2 \cdot \text{nodes}(K)} \right) / 2^h.$$

By Corollary 2.2 this equals

$$\left(\frac{\text{keys}(h, h-1, 0)}{2 \cdot \text{nodes}(h, h-1, 0)} + \sum_{l=0}^{h-1} \sum_{x=1}^{2^l} \frac{1}{2} \cdot \frac{\text{keys}(h, l, x)}{\text{nodes}(h, l, x)} \right) / 2^h$$

We ignore the first term in the sum; when divided by 2^h , it vanishes as $h \rightarrow \infty$. Then, substituting m for $h-l-1$ and reversing the order of the first summation gives

$$\left(\sum_{m=0}^{h-1} \sum_{x=1}^{2^{h-m-1}} \frac{1}{2} \cdot \frac{\text{keys}(h, h-m-1, x)}{\text{nodes}(h, h-m-1, x)} \right) / 2^h.$$

Substituting the values from Theorem 2.3 for keys $(h, h-m-1, x)$ and nodes $(h, h-m-1, x)$ gives

$$(1) \quad \left(\sum_{m=0}^{h-1} \sum_{x=1}^{2^{h-m-1}} \frac{1}{2} \cdot \frac{2^{h+1} - 2^{h-m} - 1 + x}{2^{h+1} - (m+1)2^{h-m} + mx - 1} \right) / 2^h.$$

Define $\alpha_{h,m}$ to be the value of the inner sum for a given h and m . We want to take the limit of $\alpha_{h,m}$ as $h \rightarrow \infty$, but the number of terms increases to infinity as $h \rightarrow \infty$. Hence, we will consider $\alpha_{h,m}$ divided by the number of terms. Normalizing the sum in this manner makes the limit exist. Define $\bar{\alpha}_{h,m} = \alpha_{h,m}/2^{h-m-1}$ and $\alpha_m^* = \lim_{h \rightarrow \infty} \bar{\alpha}_{h,m}$. (Note that $\bar{\alpha}_{h,m}$ is the average of a number of utilizations, and hence $\frac{1}{2} \leq \bar{\alpha}_{h,m} \leq 1$. We use this fact in Lemma 3.3.) (1) is then equal to $\sum_{m=0}^{h-1} \bar{\alpha}_{h,m} \cdot 2^{-(m+1)}$. To calculate the limit as $h \rightarrow \infty$, we use the identity

$$(2) \quad \lim_{h \rightarrow \infty} \sum_{m=0}^{h-1} \bar{\alpha}_{h,m} \cdot 2^{-(m+1)} = \sum_{m=0}^{\infty} \alpha_m^* 2^{-(m+1)}.$$

This equality is not obvious (suppose the summand were $\delta_{h,m} \equiv 1$ if $h = m$ and 0 otherwise instead of $\bar{\alpha}_{h,m} \cdot 2^{-(m+1)}$), and is proved in Lemma 3.3. We now calculate α_m^*

for all m . If $m = 0$, $\alpha_{h,m}$ simplifies to

$$\frac{1}{2} \sum_{x=1}^{2^{h-1}} \frac{2^h - 1 + x}{2^h - 1} = \frac{5 \cdot 2^{2h-2} - 3 \cdot 2^{h-1}}{2^{h+2} - 4}.$$

Dividing by 2^{h-1} and taking the limit gives $\alpha_0^* = \frac{5}{8}$.

For $m > 0$, we use Lemma 3.2 to give

$$\begin{aligned} \alpha_{h,m} &= \frac{1}{2m} \left[2^{h-m-1} - 1 + \left(2^{h+1} - 2^{h-m} + 1 - \frac{2^{h+1} - (m+1)2^{h-m} - 1}{m} \right) \right. \\ &\quad \left. \times \ln \left(\frac{2^{h+1} - (m+1)2^{h-m} - 1 + m2^{h-m-1}}{2^{h+1} - (m+1)2^{h-m} - 1 + 2^{h-m-1}} \right) \right] + \varepsilon \\ &= \frac{1}{2m} \left[2^{h-m-1} - 1 + \left(\frac{(m-1)2^{h+1} + 2^{h-m} + (m+1)}{m} \right) \right. \\ &\quad \left. \times \ln \left(\frac{2^{h+1} - (m+2)2^{h-m-1} - 1}{2^{h+1} - (2m-1)2^{h-m-1} - 1} \right) \right] + \varepsilon, \end{aligned}$$

where $\frac{1}{2} \leq \varepsilon \leq 1$ (the minimum and maximum utilizations). Dividing by 2^{h-m-1} and taking the limit as $h \rightarrow \infty$ gives

$$\alpha_m^* = \frac{1}{2m} \left[1 + \left(\frac{(m-1)2^{m+2} + 2}{m} \right) \ln \left(\frac{2^{m+2} - (m+2)}{2^{m+2} - (2m-1)} \right) \right] \quad \text{for } m > 0.$$

Then substituting into (2) proves the theorem. \square

LEMMA 3.3. Let $\frac{1}{2} \leq \bar{\alpha}_{h,m} \leq 1$ for all $h, m \geq 0$ and let $\alpha_m^* = \lim_{h \rightarrow \infty} \bar{\alpha}_{h,m}$. Then

$$\lim_{h \rightarrow \infty} \sum_{m=0}^{h-1} \bar{\alpha}_{h,m} \cdot 2^{-(m+1)} = \sum_{m=0}^{\infty} \alpha_m^* \cdot 2^{-(m+1)}.$$

Proof. We know that for any $\varepsilon > 0$, an H_0 can be found such that

$$(3) \quad \left| \sum_{m=0}^{h-1} \bar{\alpha}_{h,m} \cdot 2^{-(m+1)} - \sum_{m=0}^{\infty} \alpha_m^* \cdot 2^{-(m+1)} \right| < \varepsilon \quad \text{for all } h \geq H_0$$

For any given ε , let $M_0 = -\log \varepsilon/2$. Note that

$$\sum_{m=M_0+1}^{h-1} \bar{\alpha}_{h,m} \cdot 2^{-(m+1)} < \sum_{m=M_0+1}^{\infty} 2^{-(m+1)} < \frac{\varepsilon}{4}.$$

Similarly, $\sum_{m=M_0+1}^{\infty} \alpha_m^* 2^{-(m+1)} < \varepsilon/4$. Now for this M_0 choose H_0 such that

$$|\alpha_m^* 2^{-(m+1)} - \bar{\alpha}_{h,m} 2^{-(m+1)}| < \frac{\varepsilon}{2M_0} \quad \text{for all } h \geq H_0, m \leq M_0.$$

Such an H_0 must exist; for every m , there exists a suitable value for H_0 , and we simply pick the maximum of these values. We now have, for any $h \geq H_0$,

$$\begin{aligned} &\left| \sum_{m=0}^{h-1} \bar{\alpha}_{h,m} 2^{-(m+1)} - \sum_{m=0}^{\infty} \alpha_m^* 2^{-(m+1)} \right| \\ &< \sum_{m=0}^{M_0} |\bar{\alpha}_{h,m} 2^{-(m+1)} - \alpha_m^* 2^{-(m+1)}| + \sum_{m=M_0+1}^{\infty} \bar{\alpha}_{h,m} 2^{-(m+1)} + \sum_{m=M_0+1}^{\infty} \alpha_m^* 2^{-(m+1)} \\ &< M_0 \cdot \frac{\varepsilon}{2M_0} + \frac{\varepsilon}{4} + \frac{\varepsilon}{4} < \varepsilon \end{aligned}$$

proving the lemma. \square

THEOREM 3.2. *Let nodes (K) be the number of nodes in a kcu-optimal tree of K keys. Then for large h, the expansion (=2 · nodes (K)/K - 1) averaged over K = 2^h - 1, . . . , 2^{h+1} - 2 is*

$$\left(2 \sum_{m=0}^{\infty} \left[m - ((m-1)2^{m+2} + 2) \ln \left(\frac{2^{m+2} - 1}{2^{m+2} - 2} \right) \right] \cdot 2^{-(m+1)} \right) - 1,$$

which can be numerically calculated as 56.7%.

Proof. In a manner analogous to Theorem 3.1, we consider

$$\left(\sum_{m=0}^{h-1} \sum_{x=1}^{2^{h-m-1}} 2 \cdot \frac{2^{h+1} - (m+1)2^{h-m} + mx - 1}{2^{h+1} - 2^{h-m} - 1 + x} \right) / 2^h - 1,$$

and define $\alpha_{h,m}$, $\bar{\alpha}_{h,m}$ and α_m^* . Using Lemma 3.2 gives

$$\alpha_{h,m} = 2 \left[m(2^{h-m-1} - 1) + (2^{h+1} - (m+1)2^{h-m} - 1 - m(2^{h+1} - 2^{h-m} - 1)) \times \ln \left(\frac{2^{h-m-1} + 2^{h+1} - 2^{h-m} - 1}{1 + 2^{h+1} - 2^{h-m} - 1} \right) \right] + \varepsilon,$$

where $1 \leq \varepsilon \leq 2$.

Dividing by 2^{h-m-1} and taking the limit as $h \rightarrow \infty$ gives

$$\alpha_m^* = 2 \left[m - ((m-1)2^{m+2} + 2) \ln \left(\frac{2^{m+2} - 1}{2^{m+2} - 2} \right) \right],$$

and substituting into $\sum_{m=0}^{\infty} \alpha_m^* \cdot 2^{-(m+1)} - 1$ proves the theorem. Note that the interchange of sum and limit is still valid; an analogue of Lemma 3.3 can easily be proved because $1 \leq \bar{\alpha}_{h,m} \leq 2$. \square

In summary, the average utilization of 64.7% shows that it is possible to have kc-optimality and reasonable utilization; these properties are not totally incompatible. In addition, the average expansion of 56.7% is comparable to the expansion of a random 2-3 tree, which is known to be bounded [5] between 40% and 58%. (Note that the expansion must be used in this comparison because no such bounds on the utilization are known.)

4. A tree construction algorithm. In this section we describe an $O(n)$ algorithm to construct a kcu-optimal 2-3 tree from a sorted array of keys (see Appendix). The algorithm builds the tree top-down. If K , the number of keys in the tree to be built, is of the form $2^{h+1} - 2$, then a full tree must be constructed. This is done in lines 5-16. If K is not of this form, (lines 17-25) the root of the tree must be a 1-node, and the other keys must be partitioned into two subtrees. Many partitioning strategies are possible. Our algorithm divides the keys as evenly as possible, with the leftover key (if there is one) going into the left subtree. We now prove that the “even splitting” tree construction algorithm actually does construct a kcu-optimal tree and that it requires time $O(n)$.

THEOREM 4.1. *The “even splitting” algorithm produces a kcu-optimal tree.*

Proof. (by induction on K). Proving the theorem for $K = 1$ is trivial, so consider $K > 1$. If K is of the form $2^{h+1} - 2$, the algorithm will construct a full tree, which is then kcu-optimal. If K is not of this form, $\lceil (K-1)/2 \rceil$ keys form the left subtree and $\lfloor (K-1)/2 \rfloor$ the right. Let $(h-1, l_1, x_1)$ and $(h-1, l_2, x_2)$ be, respectively, the leader profiles of the left and right subtrees. If K is odd, $\lceil (K-1)/2 \rceil = \lfloor (K-1)/2 \rfloor$, and the structure of the subtrees will be identical. In this case, $l_1 = l_2$. All leaders are on level l_1 or $l_1 + 1$ and the levels of two leaders can differ by at most one. Hence the tree is

kcu-optimal. If K is even, the left subtree will have one more key than the right. It may be the case that $l_1 = l_2$, in which case the proof for odd K holds. If $l_1 \neq l_2$, Corollary 3.1 states that the leader profiles must be $(h-1, l_1, 1)$ and $(h-1, l_2, 2^{l_2})$ with $l_1 = l_2 - 1$, because the number of keys in the subtrees differs by exactly one. The left subtree has leaders on level l_1 and $l_1 + 1$. The right subtree has leaders on only level $l_2 (= l_1 + 1)$, and there can be no lower leaders at level $l_2 + 1$. Again, the levels of the leaders differ by at most one, and the tree is kcu-optimal. \square

THEOREM 4.2. *The “even splitting” algorithm requires time $O(n)$.*

Proof. Since the algorithm has no loops, the time required for any invocation of Build (ignoring any calls it makes) is bounded by a constant. Since each invocation puts at least one key into the tree at most n invocations are required. \square

Appendix. Here, we give two algorithms in pseudo-PASCAL: Algorithm A.1 for searching for a value in a 2-3 tree, and Algorithm A.2 for building a kcu-optimal tree from a sorted array of keys. We assume the following global type declarations:

```

TYPE nodetype = RECORD
    onekey: BOOLEAN;
    leftkey, rightkey: keytype;
    leftson, middleson, rightson: ptnode
END;
ptnode = ↑nodetype;

```

Field “onekey” is true if and only if this node is a 1-node. For 2-nodes, the fields are used in the obvious way. For 1-nodes, the single key is stored in “leftkey” and the pointers to the sons are stored in “leftson” and “middleson” (“rightkey” and “rightson” are ignored).

ALGORITHM A.1. Searching for the key with value “keyval” in a 2-3 tree with root pointed to by “root”. A pointer to the desired node is returned in “p”. For simplicity, we assume “keyval” is in the tree. We also assume only one comparison is needed to execute each pseudo-CASE statement.

```

PROCEDURE search (root: ptnode; keyval: keytype; VAR p: ptnode);
VAR found: BOOLEAN;
BEGIN
    p := root;
    found := FALSE;
    WHILE NOT found DO BEGIN
        CASE
            keyval < p↑.leftkey: p := p↑.leftson;
            keyval = p↑.leftkey: found := TRUE;
            keyval > p↑.leftkey:
                IF p↑.onekey THEN p := p↑.middleson
                ELSE BEGIN
                    CASE
                        keyval < p↑.rightkey: p := p↑.middleson;
                        keyval = p↑.rightkey: found := TRUE;
                        keyval > p↑.rightkey: p := p↑.rightson
                    ENDCASE
                END
            ENDCASE
        END
    END
END

```

ALGORITHM A.2. The “even splitting” algorithm for constructing a k cu-optimal tree from a given set of keys. We assume the K keys are stored in sorted order in positions 1 through K of array “keys”. We assume the original invocation is $\text{build}(1, K, \text{TRUNC}(\log_2(k+1)))$, which returns a pointer to the root of the tree.

```

1. FUNCTION build (lower, upper, height: INTEGER): ptnode;
2.   VAR k1, k2: INTEGER; node: ptnode;
3.   BEGIN
4.     IF height = 0 THEN build := NIL
5.     ELSE IF upper-lower + 1 = 2**(height + 1)-2 THEN BEGIN
6.       NEW(node);
7.       k1 := lower + 2**height - 2;
8.       k2 := lower + 2**height + 2**(height-1) - 2;
9.       node↑.onekey := FALSE;
10.      node↑.leftkey := keys[k1];
11.      node↑.rightkey := keys[k2];
12.      node↑.leftson := build(lower, k1-1, height-1);
13.      node↑.middleson := build(k1 + 1, k2-1, height-1);
14.      node↑.rightson := build(k2 + 1, upper, height-1);
15.      build := node;
16.    END
17.  ELSE BEGIN
18.    NEW(node);
19.    k1 := (lower + upper + 1) DIV 2;
20.    node↑.onekey := TRUE;
21.    node↑.leftkey := keys[k1];
22.    node↑.leftson := build(lower, k1-1, height-1);
23.    node↑.middleson := build(k1 + 1, upper, height-1);
24.    build := node;
25.  END
26. END

```

REFERENCES

- [1] A. L. ROSENBERG AND L. SNYDER, *Minimal comparison 2-3 trees*, this Journal, 7 (1978), pp. 465–480.
- [2] D. E. KNUTH, *The Art of Computer Programming*, vol. 3, Addison-Wesley, Reading, MA 1973.
- [3] R. E. MILLER, N. PIPPENGER, A. L. ROSENBERG AND L. SNYDER, *Optimal 2-3 trees*, this Journal, 8 (1979), pp. 42–59.
- [4] A. L. ROSENBERG AND L. SNYDER, *Compact 2-3 trees*, IBM Tech. Rept. RC-7343.
- [5] A. C. YAO, *On random 2-3 trees*, Acta Informatica, 9 (1978), pp. 159–170.

PERFORMANCE BOUNDS FOR ORTHOGONAL ORIENTED TWO-DIMENSIONAL PACKING ALGORITHMS*

IGAL GOLAN†

Abstract. Two orthogonal oriented two-dimensional, packing algorithms are presented and their behavior is analyzed under several assumptions. For one of the algorithms we show that the ratio of the height used by the algorithm to the optimal height is asymptotically bounded by $\frac{4}{3}$. This bound is an improvement over similar bounds for previously proposed algorithms.

Key words. Two-dimensional packing

1. Introduction. Consider an “open-ended” rectangle R of width 1 and a finite collection of rectangular pieces organized into a list $L = \{P_1, P_2, \dots, P_n\}$. Each piece is defined by an ordered pair $P_i = (w(P_i), h(P_i))$, $1 \leq i \leq n$, corresponding to the width $w(P_i)$ and the height $h(P_i)$ of the piece. $w(P_i) \leq 1$ for all $1 \leq i \leq n$.

The problem is to pack the pieces into R so that no two pieces overlap and so that the height to which R is filled (the maximum height attained by any piece) is as small as possible. We shall assume that the rectangular pieces are oriented such that the sides of the piece which correspond to its width have to be parallel to the bottom edge (B) of R . The problem is a generalization of the one-dimensional bin packing problem studied in [4] and has been considered in various forms in [1], [2] and [3].

For a list L of rectangular pieces all having width less or equal to 1, let $\text{OPT}(L)$ denote the minimum possible height necessary to pack all the pieces into R . Let $A(L)$ denote the height actually used by a packing algorithm A when applied to L . We consider two types of bounds: absolute performance bounds of the form $A(L) \leq \alpha \text{OPT}(L)$ and asymptotic performance bounds of the form $A(L) \leq \alpha \text{OPT}(L) + \beta$, where α and β are constants such that $\alpha \geq 1$, $\beta \geq 0$.

In § 2 we present the split-algorithm and give absolute and asymptotic performance bounds for the general case, and for the special case where all pieces are squares. In § 3 we present the mixed-algorithm and give an asymptotic bound for its behavior.

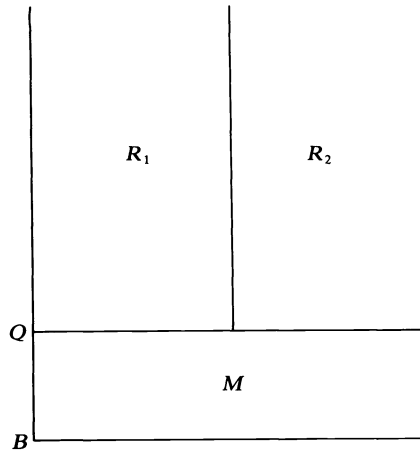
We conjecture, on the basis of the analysis of these algorithms, and other algorithms as well, (such as those described in [1], [2]) that the better performance of the mixed-algorithm as compared to the split-algorithm is not accidental. More generally it appears that algorithms treating all the components of L uniformly are outperformed by those that handle “wide” and “narrow” pieces separately.

2. Performance bounds for split packing. We can *split* an “open-ended” rectangle R into two “open-ended” rectangles R_1, R_2 and a “closed” rectangle M as in Fig. 1. If B (the bottom edge of R) and Q coincide, M will disappear.

A *level* L will be any line parallel to the bottom edge B of R ; its value is its distance from B . Let $L = \{P_1, P_2, \dots, P_n\}$ be such that $w(P_i) \geq w(P_{i+1})$ for $i < n$ (i.e., L is arranged in decreasing-width order). The *split algorithm* (SP-algorithm) will be as follows: the split algorithm packs the pieces in order of decreasing width. After packing some pieces, those left to be packed are narrower and we can split the open-ended

* Received by the editors August 16, 1979, and in final form February 21, 1980.

† Armament Development Authority, Center for Military Analyses, P.O. Box 2250, Haifa, Israel.

FIG. 1. Splitting R into R_1 , R_2 and M .

rectangle R into two open-ended rectangles each being sufficiently wide as to accommodate any of the pieces left to be packed. Packing the two rectangles simultaneously, we insert the next piece to be packed in the one with the minimum resulting overall height. As soon as possible we continue to split each of the rectangles. As the pieces get narrower we obtain more rectangles to place them in.

Begin SP-Algorithm

- (1) Define R_1 to be R . Place P_1 on B touching the left edge of R_1 . Define level $S = 0$, $k = 1$, $i = 1$.

While ($k < n$) **Repeat**

Begin We have already defined R_1, R_2, \dots, R_i . For each R_j with width W_j let P_j be the last piece packed into it. Let b_j be the distance of the bottom edge of P_j to B . Let $a_j = b_j + h(P_j)$.

- (2) Let $J = \{R_j \mid W_j \geq w(P_j) + w(P_{k+1})\}$

Comment J is the set of "open-ended" rectangles which can accommodate the next piece to be placed (P_{k+1}) side by side with the last piece placed in it (P_j).

If ($J = \Phi$) **go to** (8)

- (3) Find $R_{j'}$ in J such that $b_{j'} \leq b_j$ for all $R_j \in J$ (Resolve ties in any way you like).
 (4) Insert P_{k+1} in $R_{j'}$ with its bottom edge at height $b_{j'}$ and its left edge touching $P_{j'}$ on its right.
 (5) Split $R_{j'}$ into R_{i+1} , R_{i+2} and $M_{j'}$ with a line parallel to the bottom edge of $R_{j'}$ at height $b_{j'}$ and a vertical line touching $P_{j'}$ and P_{k+1} (see Fig. 2).
 (6) Delete $R_{j'}$ from the list of "open-ended" rectangles. Put R_{i+1} and R_{i+2} in the list. Relabel the List. $i = i + 1$, $k = k + 1$.

- (7) **Go to** (10).

- (8) Let j' be such that $a_{j'} \leq a_j$ for all $1 \leq j \leq i$ (resolve ties in any way you like). Place P_{k+1} in $R_{j'}$ with its bottom edge on the top edge of $P_{j'}$ and its left edge touching the left edge of $R_{j'}$.

- (9) Define $S = a_{j'}$, $k = k + 1$.

- (10) **End while.**

- (11) **Stop**

End SP-Algorithm.

LEMMA 1. For any “closed” rectangle M let $P_{k_1}, P_{k_2}, \dots, P_{k_f}$ be the pieces of L inside M . Let the width of M be \bar{W} . Then for all $1 \leq i \leq f$ $w(P_{k_i}) > \bar{W}/2$.

Proof. The pieces inside M are placed on top of one another. If P_{k_i} is the piece placed on top of $P_{k_{i-1}}$, $1 < i \leq f$ then $w(P_{k_i}) \leq w(P_{k_{i-1}})$. If $f = 1$ we have $w(P_{k_1}) = \bar{W}$. If $f \geq 2$ let us assume that $w(P_{k_f}) \leq \bar{W}/2$. In this case a piece would be placed side by side with P_{k_f} using step (4) of SP and the split in step (5) would separate the pieces $P_{k_{f-1}}$ and P_{k_f} . This is a contradiction and therefore, $w(P_{k_f}) > \bar{W}/2$. \square

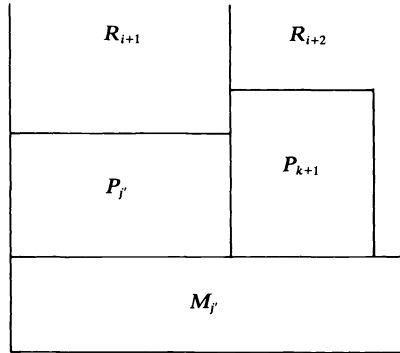


FIG. 2. Splitting R_j .

LEMMA 2. Let A be the region between B and S . Then A is at least half occupied with pieces (and parts of pieces) from L .

Proof. Initially the level S is at the bottom edge of R and obviously the lemma is true. As long as we do not perform instruction (9) in the SP-algorithm the statement will remain true. Let us assume that we have just executed instruction (9) and S is driven to a new height. Between B and S we have “closed” rectangles (and parts of “closed” rectangles) and by Lemma 1 their area is at least half occupied. In addition we have in A parts of “open-ended” rectangles $R_{i_1}, R_{i_2}, \dots, R_{i_\alpha}$. For all $1 \leq j \leq \alpha$, $S \leq a_{i_j}$ (by step (8) of the SP-Algorithm) and $b_{i_j} > W_{i_j}/2$, for otherwise, at step (2), $J \neq \Phi$ and we would not have arrived at step (9). It follows that these parts are also half occupied. \square

LEMMA 3. If $h = \max_{1 \leq i \leq n} (h(P_i))$ then for all R_j as defined by the SP-algorithm, $a_j - S \leq h$.

Proof. Assume that there is an i such that $a_i - S > h$. It follows that $b_i - S > 0$ and the P_i was inserted by using step (8). Then we would have used step (9) to move S to the bottom of P_i and $a_i - S = h(P_i)$. Thus, $a_i - S \leq h$, a contradiction and the lemma is proved. \square

THEOREM 1. For any list $L = \{P_1, \dots, P_n\}$, $SP(L) \leq 3 OPT(L)$.

Proof.

$$SP(L) \leq S + h, \quad h = \max_{1 \leq i \leq n} (h(P_i)) \quad \text{by Lemma 3,}$$

$$S \leq 2 \sum_{i=1}^n w(P_i) * h(P_i) \quad \text{by Lemma 2,}$$

$$OPT(L) \geq \max \left(h, \sum_{i=1}^n w(P_i) * h(P_i) \right).$$

Consider the two cases.

$$\begin{aligned}
 (1) \quad & h \geq \sum_{i=1}^n w(P_i) * h(P_i), \\
 \text{OPT}(L) \geq h, \quad & \frac{\text{SP}(L)}{\text{OPT}(L)} \leq \frac{S+h}{h} \leq 1 + \frac{2 \sum_{i=1}^n w(P_i) * h(P_i)}{h}, \\
 & \frac{\text{SP}(L)}{\text{OPT}(L)} \leq 3.
 \end{aligned}$$

Thus, $\text{SP}(L) \leq 3 \text{OPT}(L)$.

$$\begin{aligned}
 (2) \quad & h < \sum_{i=1}^n w(P_i) * h(P_i), \\
 \text{OPT}(L) \geq \sum_{i=1}^n w(P_i) * h(P_i), \quad & \frac{\text{SP}(L)}{\text{OPT}(L)} \leq \frac{S+h}{\sum_{i=1}^n w(P_i) * h(P_i)}, \\
 & \frac{\text{SP}(L)}{\text{OPT}(L)} \leq \frac{2 \sum_{i=1}^n w(P_i) * h(P_i)}{\sum_{i=1}^n w(P_i) * h(P_i)} + \frac{h}{\sum_{i=1}^n w(P_i) * h(P_i)} \leq 3.
 \end{aligned}$$

Therefore, $\text{SP}(L) \leq 3 \text{OPT}(L)$. \square

COROLLARY 1. For any list $L = \{P_1, P_2, \dots, P_n\}$ such that $w(P_i), h(P_i) \leq 1$ for all $1 \leq i \leq n$, $\text{SP}(L) \leq 2 \text{OPT}(L) + 1$.

Proof.

$$\text{SP}(L) \leq S + h, \quad h = \max_{1 \leq i \leq n} (h(P_i)) \leq 1,$$

$$S \leq 2 \sum_{i=1}^n w(P_i) * h(P_i) \leq 2 \text{OPT}(L).$$

Therefore, $\text{SP}(L) \leq 2 \text{OPT}(L) + 1$. \square

Example 1. For any even k let $L = \{P_i \mid i = 1, \dots, k^2/2 + k^3/2 + k + 1\}$, where

$$\begin{aligned}
 1 \leq j \leq k^2/2, \quad & P_j = (2/k^2 - \varepsilon, 2/k^2 - \varepsilon), \\
 n_1 = \frac{k^2}{2} + 1 \leq j \leq \frac{k^2}{2} + \frac{k^3}{2} + k, \quad & P_j = \left(\frac{1}{k^2}, \frac{1}{k^2}\right), \\
 n = \frac{k^2}{2} + \frac{k^3}{2} + k + 1, \quad & P_n = \left(\frac{1}{k^2} - \varepsilon, \frac{1}{2k}\right), \quad \varepsilon \ll \frac{2}{k^4}.
 \end{aligned}$$

Then an SP-packing of L will be as in Fig. 3, and the optimal packing will be as in Fig. 4.

$$\text{SP}(L) = \frac{2}{k^2} + \frac{3}{2k} - \varepsilon, \quad \text{OPT}(L) = \frac{3}{k^2} + \frac{1}{2k} - \varepsilon, \quad \frac{\text{SP}(L)}{\text{OPT}(L)} = 3 - \frac{7 - 2\varepsilon k^2}{3 + \frac{k}{2} - \varepsilon k^2}.$$

COROLLARY 2. For any $\delta > 0$ there exists a list L of rectangles arranged by decreasing width, such that $\text{SP}(L) > (3 - \delta) \text{OPT}(L)$.

Proof. Use Example 1. \square

Example 2. For any even k let $L = \{P_i \mid i = 1, \dots, (k^5/2) + k^3 + (k^2/2)\}$ where

$$1 \leq j \leq \frac{k^2}{2}, \quad P_j = \left(\frac{2}{k^2} - \varepsilon, \frac{2}{k^2} - \varepsilon \right),$$

$$n_1 = \frac{k^2}{2} + 1 \leq j \leq \frac{k^5}{2} + k^3 + \frac{k^2}{2}, \quad P_j = \left(\frac{1}{k^2}, \frac{1}{k^2} \right), \quad \text{and} \quad \varepsilon \ll \frac{2}{k^4}.$$

The SP-packing will be similar to that in Fig. 3, and the optimal packing will be similar to that in Fig. 4.

$$\begin{aligned} \text{SP}(L) &= \left(\frac{2}{k^2} - \varepsilon \right) + k^3 * \frac{1}{k^2}, & \text{SP}(L) &= k + \frac{2}{k^2} - \varepsilon, \\ \text{OPT}(L) &= \left(\frac{k^3}{2} + k \right) * \frac{1}{k^2} + \left(\frac{2}{k^2} - \varepsilon \right), & \text{OPT}(L) &= \frac{k}{2} + \frac{1}{k} + \frac{2}{k^2} - \varepsilon. \end{aligned}$$

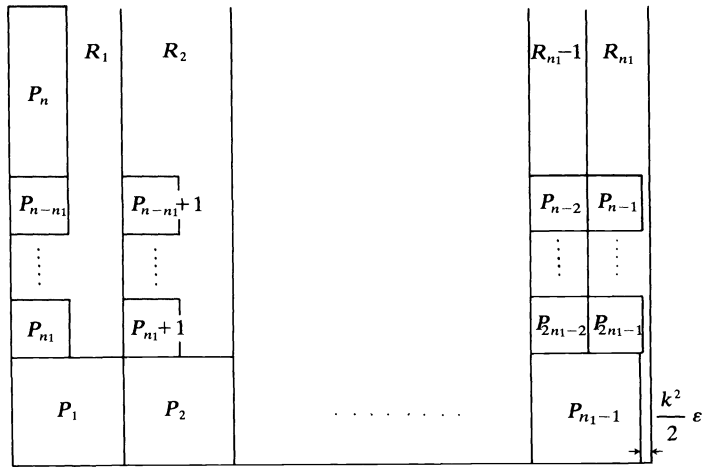


FIG. 3. $\text{SP}(L) = 2/k^2 + 3/2k - \varepsilon$.

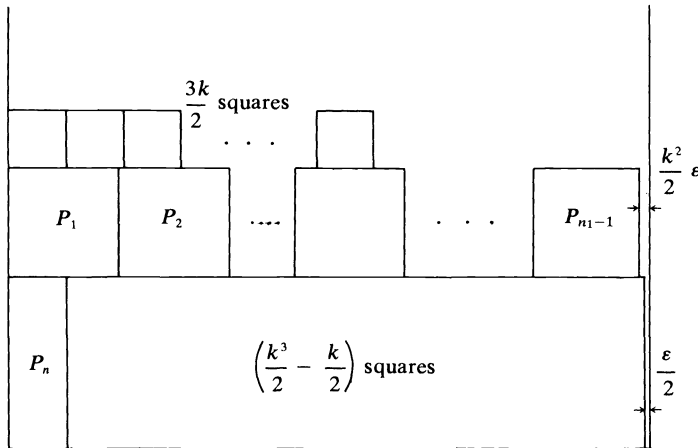


FIG. 4. $\text{OPT}(L) = 3/k^2 + 1/2k - \varepsilon$.

COROLLARY 3. For any $\delta > 0$ there exists a list L of rectangles arranged by decreasing width, such that $SP(L) > (2 - \delta)OPT(L) + C$ for any C .

Proof. Use Example 2.

$$\frac{SP(L) - C}{OPT(L)} = \frac{k + \frac{2}{k^2} - C - \epsilon}{\frac{k}{2} + \frac{1}{k} + \frac{2}{k^2} - \epsilon} = 2 - \frac{4 + 2Ck^2 + 4k - 2k^2\epsilon}{k^3 + 2k + 4 - 2k^2\epsilon}.$$

If k is large enough, $(SP(L) - C)/(OPT(L)) > 2 - \delta$. \square

When all the pieces are squares an improvement of the SP-algorithm can be obtained.

The new algorithm, called the SPS-algorithm, does not insert the squares into R in the order they appear in L . To get the SPS-algorithm replace step (8) in the SP-algorithm by the following steps:

(8') **Begin** Let j' be such that $a_{j'} \leq a_j$ for all $1 \leq j \leq i$ (resolve ties in any way you like).

Let $d = 0$.

(81) $e = W_{j'} - w(P_{j'}) - d$. If there is no m , $m > k + 1$, such that $w(P_m) \leq e$ **go to** (82). Let l , $l > k + 1$, be the smallest integer for which $w(P_l) \leq e$. Insert P_l in $R_{j'}$ with its bottom edge at distance $b_{j'}$ from B , and as far to the left as possible. Let $d = d + w(P_l)$. Take P_l out of the list L , and relabel the remaining pieces of L starting at $k + 1$. Let $n = n - 1$, **go to** (81).

(82) Insert P_{k+1} in $R_{j'}$ with its bottom edge on the top edge of $P_{j'}$ and its left edge touching the left edge of $R_{j'}$.

End

THEOREM 2. For any list $L = \{P_1, P_2, \dots, P_n\}$ of squares arranged by decreasing-width $SPS(L) \leq 2 OPT(L)$.

Proof. Let P_t be the largest square with top at height $SPS(L)$. Define level $S' = w(P_t)$ and level $S_1 = SPS(L) - w(P_t)$. By the definition of S , $S \geq S_1$. Let A be the area between levels S' and S_1 . Lemma 2 holds also for the SPS-algorithm and hence, A is at least half occupied.

Consider the area A_1 between B and S' . There are pieces of L on B arranged in decreasing width order from left to right. Let P_q be the square farthest to the right which is still greater than P_t . The distance z between the right edge of P_q and the right edge of R , is less than $w(P_t)$, otherwise, P_t would have been placed on B using step (81). The occupied area of A_1 is at least $w(P_t) * (1 - z)$. Consider the area A_2 between S_1 and $SPS(L)$. The occupied area of A_2 is at least $w(P_t) * w(P_t)$. The occupied area of both A_1 and A_2 is at least half the area, because $w(P_t) * (1 - z) + w(P_t) * w(P_t) \geq w(P_t)$. Thus, $SPS(L) \leq 2 OPT(L)$. \square

COROLLARY 4. For any $\delta > 0$, there exists a list L of squares arranged by decreasing width, such that $SPS(L) > (2 - \delta)OPT(L)$.

Proof. Use Example 1 without the last piece. \square

3. Performance bound for mixed packing. Most of the packing algorithms (see [1], [2] and [3]) packed all the pieces in the same way, an exception is [2, split-fit algorithm]. The mixed algorithm divided the pieces into several sets and packed the pieces in the sets in different ways. By not packing the pieces in decreasing-width order, it is possible to fill the gaps created by packing wide pieces, with smaller pieces and thus save space.

Given a list $L = \{P_1, P_2, \dots, P_n\}$ of rectangular pieces, such that $w(P_i), h(P_i) \leq 1$, for all $1 \leq i \leq n$, divide L into 5 sets: $A = \{P_i \mid w(P_i) > \frac{1}{2}\}$, $B = \{P_i \mid \frac{1}{3} < w(P_i) \leq \frac{1}{2}\}$, $C = \{P_i \mid \frac{1}{4} < w(P_i) \leq \frac{1}{3}\}$, $D_1 = \{P_i \mid \frac{5}{24} < w(P_i) \leq \frac{1}{4}\}$, $D_2 = \{P_i \mid w(P_i) \leq \frac{5}{24}\}$ and let $D = D_1 \cup D_2$. A piece from the A set will be called an A -piece and similarly for B, C, D_1, D_2 and D .

The mixed-algorithm (M-algorithm) is as follows:

Begin M-algorithm.

Step (1)

- (1.1) Arrange all A -pieces in decreasing width order; $A = \{a_1, a_2, \dots, a_{|A|}\}$. Set $l_1 = 0, i = 1$.
- (1.2) Place a_i in R on level l_i left justified (as far to the left edge of R as possible).
- (1.3) $l_{i+1} = l_i + h(a_i), i = i + 1$.
- (1.4) **If** $i \leq |A|$ **then go to** (1.2).

Step (2)

- (2.1) Define $L_2 = l_{|A|+1}$. Arrange all B -pieces in decreasing width order; $B = \{b_1, b_2, \dots, b_{|B|}\}$. Set level $f = 0$.
- (2.2) Let

$$k = \min_{1 \leq j \leq |B|} \{|B| + 1, j \mid b_j \in B, b_j \text{ is not marked, } b_j \text{ will fit on level } f, f + h(b_j) \leq L_2\}.$$

If $(k = |B| + 1)$ **then go to** (2.4).

- (2.3) Place b_k in R on level f right justified. Mark $b_k, f = f + h(b_k)$, **go to** (2.2).
- (2.4) Find $i = \min_{1 \leq j \leq |A|+1} (j \mid l_j > f)$.
If $(i \leq |A|)$ **then** $f = l_i$, **go to** (2.2).
- (2.5) Move up the last B -piece placed in R , until its top will be at level L_2 . Move, successively, all other B -pieces, until the top of each one will touch the bottom of the B -piece above it. Let L_1 be the height of the bottom of the lowest B -piece in R .

Step (3)

- (3.1) Find $i = \max_{1 \leq j \leq |A|+1} (0, j \mid w(a_j) > \frac{3}{4})$. Let $H_1 = l_{i+1}$, **if** $(H_1 \geq L_1)$ **then go to** (6). Find $i = \max_{1 \leq j \leq |A|+1} (0, j \mid w(a_j) > \frac{2}{3})$. Let $H_2 = \min(L_1, l_{i+1})$. Find $i = \max_{1 \leq j \leq |A|+1} (0, j \mid w(a_j) > \frac{7}{12})$. Let $H_3 = \min(L_1, l_{i+1})$.

Step (4)

- (4.1) **If** $(H_2 = H_1)$ **then go to** (5). Construct a rectangle R_1 inside R , between height H_1 and height H_2 , having width $\frac{1}{4}$, right justified (see Fig. 5). Use FFDH-algorithm [2], to pack as many of the remaining pieces as possible into R_1 . If a piece does not fit completely into R_1 , then don't pack it.
- (4.2) **If** $(H_3 = H_2)$ **then go to** (5). Construct a rectangle R_2 inside R , between height H_2 and height H_3 , having width $\frac{1}{3}$ right justified (see Fig. 5). Use FFDH-algorithm [2], to pack as many of the remaining pieces as possible into R_2 . If a piece does not fit completely into R_2 , then don't pack it.

Step (5)

- (5.1) Arrange all C -pieces not yet packed in decreasing-width order; $C = \{c_1, c_2, \dots, c_{|C|}\}$. $H_4 = H_3, i = 1$.
- (5.2) **If** $(i > |C|)$ **then go to** (5.5).
- (5.3) **If** $(H_4 + h(c_i) > L_1)$ **then** $i = i + 1$, **go to** (5.2).
- (5.4) Put c_i in R on level H_4 right justified, $H_4 = H_4 + h(c_i), i = i + 1$, **go to** (5.2).
- (5.5) **If** $(H_4 + 1 \geq L_1)$ **then go to** (6).

- (5.6) Arrange all D_1 -pieces not yet packed in decreasing-width order; $D_1 = \{d_1, d_2, \dots, d_{|D_1|}\}$. Use [1, bottom up right justified algorithm] to pack as many of the D_1 -pieces as possible, starting at height H_4 , but do not pack above L_1 . Let the height of the top of the highest D_1 -piece be H_5 .

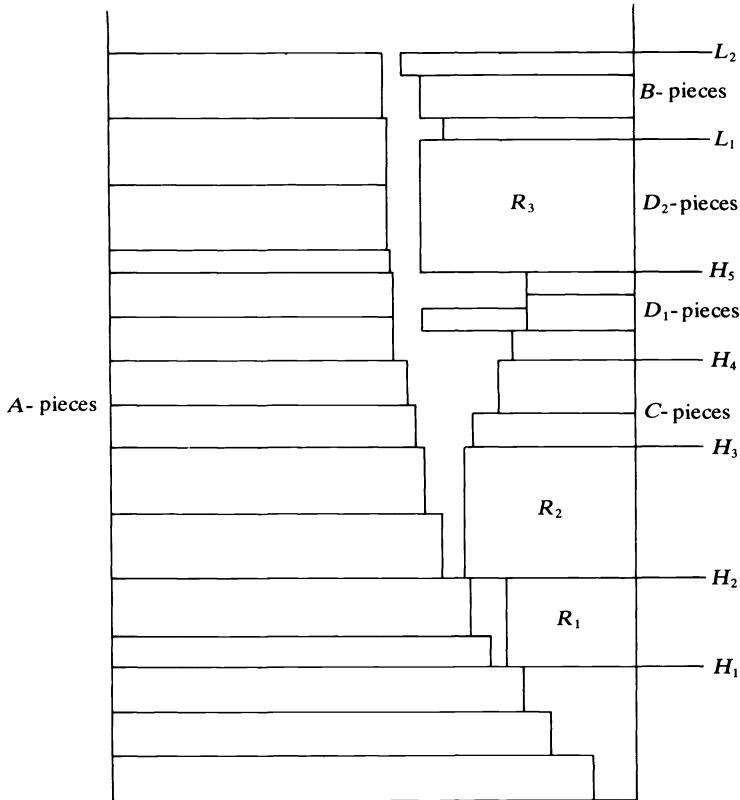


FIG. 5. M -packing steps (1)–(5).

- (5.7) **If** $(H_5 + 1 \geq L_1)$ **then go to** (6).
 (5.8) Construct a rectangle R_3 inside R , between height H_5 and height L_1 , having width $\frac{5}{12}$ right justified (see Fig. 5). Use FFDH-algorithm [2], to pack as many of the remaining D_2 -pieces as possible into R_3 . If a piece does not fit completely into R_3 , then don't pack it.

Step (6)

- (6.1) Use FFDH-algorithm [2] to pack the remaining B -pieces left justified. The pieces will be packed starting at height L_2 and ending at height L_4 .
 (6.2) Rearrange the levels of this part of the packing in such a way that all levels having total width more than $\frac{3}{4}$ will be below all other levels. Let L_3 be the highest level having total width more than $\frac{3}{4}$.

Step (7)

- (7.1) **If** $(L_3 = L_4)$ **then go to** (8). Construct a rectangle R_4 inside R , between height L_3 and height L_4 , having width $\frac{1}{4}$ right justified. Use FFDH-algorithm [2], to pack as many of the remaining pieces as possible into R_4 . If a piece does not fit completely into R_4 , then don't pack it.

Step (8)

(8.1) Use FFDH-algorithm [2], to pack all remaining pieces left justified, starting at height L_4 .

End M-algorithm.

THEOREM 3. For any list $L = \{P_1, \dots, P_n\}$, such that $w(P_i), h(P_i) \leq 1$ for all $1 \leq i \leq n$, $M(L) \leq \frac{4}{3} \text{OPT}(L) + 7\frac{1}{18}$.

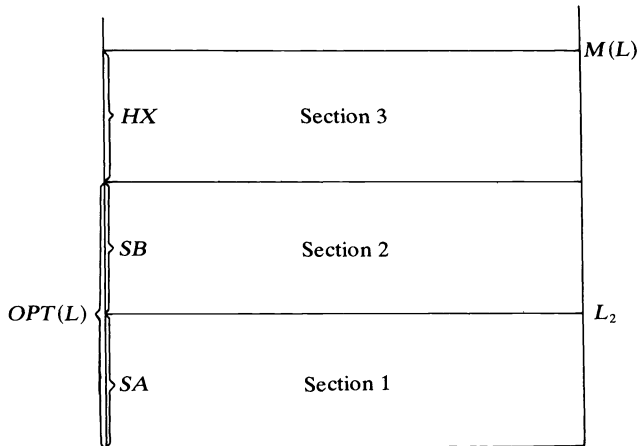


FIG. 6. *M*-packing divided into sections.

Proof.

Case 1. No D-piece left to be packed after step (7). Let $\text{OPT}(L)$ be the height of the optimal packing. Divide the optimal packing into blocks, by passing levels at the top and bottom of each A-piece. Each adjacent pair of such levels defines a block. Let a block containing an A-piece be called an A-block, and any other block a B-block. Let SA be the sum of the heights of all A-blocks and SB the sum of the heights of all B-blocks. Then

$$\text{OPT}(L) = SA + SB.$$

Let $H(Y)$ be the sum of the heights of all Y-pieces, where Y is A, B or C. Let $HH = H(A) + H(B) + H(C)$. It is clear that $SA = H(A) = L_2$.

Divide the M-packing into 3 sections as follows: section 1 between the bottom of R and L_2 , section 2 between L_2 and $\text{OPT}(L)$ and section 3 between $\text{OPT}(L)$ and $M(L)$ (Fig. 6). Let $X(B)$ be the sum of the heights of all B-pieces or parts of B-pieces, included in section 3. Let $X(C)$ be the sum of the heights of all C-pieces or parts of C-pieces, included in section 3. Let HX be the height of section 3, $M(L) = \text{OPT}(L) + HX$. Clearly the height of section 2 is SB.

In any packing, a level line can pass through at most 3 pieces from $B \cup C$. If a level line passes through an A-piece, it can pass, at most, through one more piece from $B \cup C$. Thus $HH \leq 2SA + 3SB$.

In the second section of the M-packing, FFDH-packing was used in steps (6) and (8). In each block of the FFDH-packing, in steps (6) and (8), we have at least two pieces from $B \cup C$, except maybe, the last block in each step. In a FFDH-packing of height L, if each block, except possibly the last one, contains at least K pieces, then the sum S of the heights of all pieces, satisfies $S \geq KL - (L - 1)H$, H being the height of the highest piece.

Hence, $X(B) + X(C) \leq H(B) + H(C) - (2SB - 2)$. But $H(B) + H(C) = HH - H(A)$, and $HH \leq 2SA + 3SB$. Therefore $X(B) + X(C) \leq 2SA + 3SB - SA - 2SB + 2$. Hence $X(B) + X(C) \leq SA + SB + 2$. But $\text{OPT}(L) = SA + SB$. Thus $X(B) + X(C) \leq \text{OPT}(L) + 2$.

Compare the packing of the B -pieces in the M -packing with that in the optimal packing. In step (2) of the M -algorithm, B -pieces are placed side by side with A -pieces. The B -pieces are ordered in decreasing-width order (2.1), and the k in step (2.2) is minimal, thus we always place the current widest B -piece. The cumulative height of the B -pieces put in step (2) can differ from the maximum possible only because of the additional condition $f + h(b_i) \leq L_2$ in step (2.2). This implies that there might be an additional B -piece that will fit near an A -piece, but will violate the above restriction. We conclude that the cumulative height of the B -pieces placed in the first section is not less than the cumulative height of the B -pieces, and parts of B -pieces, in the A -blocks minus 1. If $\text{OPT}(L) \geq L_4$ we have $X(B) = 0$. Assume $\text{OPT}(L) < L_4$; then the cumulative height of the B -pieces, and parts of B -pieces, placed in step (6) up to height $\text{OPT}(L)$, is at least $2SB - 1$. This is because the FFDH-algorithm will put 2-pieces in each level, except maybe the last one. The cumulative height of the B -pieces, and parts of B -pieces, packed by the optimal algorithm in the B -blocks, is not more than $2SB$. Therefore, $X(B) \leq 2$.

Divide the third section of the M packing into blocks, by passing levels at the top and bottom of each B -piece, or part of a B -piece. Each block will be between two near levels. Call a block containing a B -piece, or part of a B -piece, a BM -block and any other block a CM -block. Let SBM be the sum of the heights of all BM -blocks, and SCM the sum of the heights of all CM -blocks. Then

$$HX = SBM + SCM, \quad SBM \leq X(B) \leq 2.$$

Using step (8) we pack the C -pieces, placing them three on a level except maybe the last one. Thus

$$SCM \leq \frac{X(C)}{3} + 2, \quad HX \leq \frac{X(C)}{3} + 4.$$

But $X(B) + X(C) \leq \text{OPT}(L) + 2$. Therefore $HX \leq (\text{OPT}(L))/3 + 4\frac{2}{3}$,

$$M(L) = \text{OPT}(L) + HX \leq \frac{4}{3}\text{OPT}(L) + 4\frac{2}{3}.$$

Case 2. There are D -pieces left to be packed after step (7). We will use an area size argument in this case. In each of the 10 parts of the proof we will show that the area S , between two line levels, covered by pieces, satisfies $S \geq \alpha H + \beta$ where $\alpha \geq \frac{3}{4}$, H the distance between the levels, and β a constant.

- (i) From the bottom of R to level H_1 , the area S_1 covered by the A -pieces satisfies $S_1 \geq \frac{3}{4}H_1$ (definition of H_1 in step (3) of the M -algorithm).
- (ii) From level H_1 to level H_2 , the area S'_2 covered by the A -pieces satisfies $S'_2 \geq \frac{2}{3}(H_2 - H_1)$ (definition of H_2 in step (3) of the M -algorithm). The area S''_2 covered by pieces inside R_1 satisfies $S''_2 \geq \frac{1}{4}((H_2 - H_1)/2 - 1)$ [2, Thm. 1] and

$$S_2 = S'_2 + S''_2 \geq (\frac{2}{3} + \frac{1}{8})(H_2 - H_1) - \frac{1}{4} \geq \frac{3}{4}(H_2 - H_1) - \frac{1}{4}.$$

- (iii) From level H_2 to level H_3 , the area S'_3 covered by the A -pieces satisfies $S'_3 \geq \frac{7}{12}(H_3 - H_2)$ (definition of H_3 in step (3) of the M -algorithm). The area S''_3 covered by pieces inside R_2 satisfies $S''_3 \geq \frac{1}{3}((H_3 - H_2)/2 - 1)$ [2, Thm. 1].

Thus,

$$S_3 = S'_3 + S''_3 \geq (\frac{7}{12} + \frac{1}{6})(H_3 - H_2) - \frac{1}{3} \geq \frac{3}{4}(H_3 - H_2) - \frac{1}{3}.$$

- (iv) From level H_3 to level H_4 , the area S'_4 covered by the A -pieces satisfies $S'_4 \geq \frac{1}{2}(H_4 - H_3)$. The area S''_4 covered by the C -pieces satisfies $S''_4 \geq \frac{1}{4}(H_4 - H_3)$. Thus,

$$S_4 = S'_4 + S''_4 \geq (\frac{1}{2} + \frac{1}{4})(H_4 - H_3) \geq \frac{3}{4}(H_4 - H_3).$$

- (v) Look at any level line between levels H_4 and $H_5 - 1$. It passes through an A -piece on the left, and through a D_1 -piece on the right. In between there is a gap filled with at most one additional D_1 -piece. If the width of the gap is at least $\frac{1}{4}$, then because the D_1 -pieces have width less than $\frac{1}{4}$, the bottom up right justified algorithm used in step (5.6) will certainly place another D_1 -piece into the gap. Let S_5 be the area covered by pieces from level H_4 to level H_5 , then

$$S_5 \geq \frac{3}{4}((H_5 - 1) - H_4) + \frac{1}{2} \geq \frac{3}{4}(H_5 - H_4) - \frac{1}{4}.$$

- (vi) From level H_5 to level L_1 , the area S'_6 covered by the A -pieces, satisfies $S'_6 \geq \frac{1}{2}(L_1 - H_5)$. The area S''_6 covered by pieces inside R_3 satisfies $S''_6 \geq \frac{5}{12}(\frac{2}{3}(L_1 - H_5) - 2\frac{1}{4} \cdot \frac{2}{3})$. (The width of R_3 is $\frac{5}{12}$. The D_2 -pieces used to pack R_3 have a width of at most $\frac{5}{24}$. We use a result from the proof [2, Thm. 3] for the bound.) Thus,

$$S_6 = S'_6 + S''_6 \geq (\frac{1}{2} + \frac{10}{36})(L_1 - H_5) - \frac{5}{8} \geq \frac{3}{4}(L_1 - H_5) - \frac{5}{8}.$$

- (vii) From level L_1 to level L_2 , the area S'_7 covered by the A -pieces satisfies $S'_7 \geq \frac{1}{2}(L_2 - L_1)$. The area S''_7 covered by the B -pieces satisfies $S''_7 \geq \frac{1}{3}(L_2 - L_1)$. Thus,

$$S_7 = S'_7 + S''_7 \geq (\frac{1}{2} + \frac{1}{3})(L_2 - L_1) \geq \frac{3}{4}(L_2 - L_1).$$

- (viii) From level L_2 to level L_3 , the area S_8 covered by the B -pieces satisfies $S_8 \geq \frac{3}{4}(L_3 - L_2) - 1$ (definition of L_3 in step (6) of the M -algorithm).

- (ix) From level L_3 to level L_4 , the area S'_9 covered by the B -pieces satisfies $S'_9 \geq \frac{2}{3}(L_4 - L_3) - 1$. (There are 2 B -pieces in each level except possibly the last one.) The area S''_9 covered by pieces inside R_4 satisfies $S''_9 \geq \frac{1}{4}((L_4 - L_3)/2 - 1)$ [2, Thm. 1]. Thus,

$$S_9 = S'_9 + S''_9 \geq (\frac{2}{3} + \frac{1}{8})(L_4 - L_3) - \frac{5}{4} \geq \frac{3}{4}(L_4 - L_3) - \frac{5}{4}.$$

- (x) All pieces placed from level L_4 to the end of the packing $M(L)$ have width less than $\frac{1}{3}$. Using again the result from the proof of [2, Thm. 3] we get: $S_{10} \geq \frac{3}{4}(M(L) - L_4) - \frac{19}{9} \cdot \frac{3}{4}$. Thus,

$$S_{10} \geq \frac{3}{4}(M(L) - L_4) - \frac{19}{12}.$$

Adding the results from (i) to (x) we get $OPT(L) \geq \sum_{i=1}^{10} S_i \geq \frac{3}{4}M(L) - \frac{127}{24}$. Thus, $M(L) \leq \frac{4}{3}OPT(L) + 7\frac{1}{18}$. \square

REFERENCES

[1] B. BAKER, E. G. COFFMAN AND R. L. RIVEST, *Orthogonal packings in two dimensions*, this Journal, 9 (1980), pp. 846-855.

- [2] E. G. COFFMAN, M. R. GAREY, D. S. JOHNSON AND R. E. TARJAN, *Performance bounds for level-oriented two dimensional packing algorithms*, TR-CSL-7808 UCSB Santa Barbara, Ca. 93106; this Journal, 9 (1980), pp. 808–826.
- [3] I. GOLAN, *Two Orthogonal Oriented Algorithms for Packing in Two Dimensions*, 1979/311/MHM Computer Center M.O.D., P.O. Box 2250, Haifa, Israel.
- [4] D. S. JOHNSON, A. DEMERS, J. D. ULLMAN, M. R. GAREY AND R. L. GRAHAM, *Worst-case performance bounds for simple one-dimensional packing algorithms*, this Journal, 3 (1974), pp. 299–325.

AN AVERAGE TIME ANALYSIS OF BACKTRACKING*

CYNTHIA A. BROWN† AND PAUL WALTON PURDOM, Jr.†

Abstract. Formulas are given for the expected number of nodes in the backtrack tree that is generated while searching for all the solutions of a random predicate. The most general formulas apply to selection from any set of predicates that obeys the following conditions. Each predicate is the conjunction of t terms selected from a set of terms T . For any subset $T' \subseteq T$, the probability that the predicate contains only terms from T' depends only on the size of T' . The set T must remain unchanged if each variable x_i is replaced by $p_i(x_i)$, where p_i is a permutation function. The time needed to evaluate the general formulas is proportional to v , the number of variables in the predicate. More detailed consideration is given to predicates whose terms are random disjunctive clauses with s literals, $t = v^\alpha$ for some $1 < \alpha < s$, and the random selections are done with repetition. For this case the expected number of nodes is $\exp[O(v^{(s-\alpha)/(s-1)})]$. (Complete enumeration results in $\exp[O(v)]$ nodes.) Thus the average time for backtracking in this model is exponential with a sublinear exponent.

Key words. backtracking, analysis of algorithms, NP-completeness

1. Introduction. Many problems can be regarded as a search for all the solutions to an equation of the form $P(x_1, \dots, x_v) = \text{true}$, where P is a v -ary predicate and each x_i has a finite set of possible values. The most straightforward way to solve such a problem is by enumerating and testing each combination of possible values of the variables. If each variable has i values, there are i^v possible solutions, so the exponential time required for complete enumeration makes this method impractical for all but the smallest problems.

Suppose that, in addition to P , there are intermediate predicates $\{P_k(x_1, \dots, x_k)\}_{1 \leq k \leq v}$, where $P = P_v$, such that if $P_{k-1}(x_1, \dots, x_{k-1})$ is false, then $P_k(x_1, \dots, x_k)$ is false for all values of x_k . Then the technique of backtracking can be used to try to reduce the size of the space to be searched. The basic backtracking algorithm can be stated as follows:

1. [Initialize] Set $k \leftarrow 0$.
2. [Test] If $P_k(x_1, \dots, x_k)$ is false, go to 6.
3. [New Level] Set $k \leftarrow k + 1$.
4. [Solution?] If $k > v$, then x_1, \dots, x_v is a solution. Go to 7.
5. [First value] Set $x_k \leftarrow$ the first value of x_k and go to 2.
6. [Next value] If x_k has more values, set $x_k \leftarrow$ the next value for x_k and go to 2.
7. [Backtrack] Set $k \leftarrow k - 1$. If $k > 0$, go to 6; otherwise stop.

The values that are tested can be represented by a tree, as shown in Fig. 1. Knuth [5] gives a more complete introduction to backtracking.

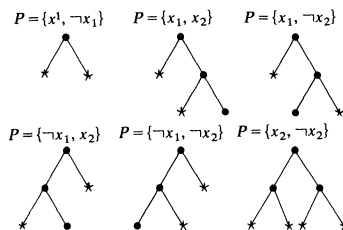


FIG. 1. The backtrack trees for six simple predicates. The false branch is to the left. Each node where a term is false is marked with an asterisk.

* Received by the editors December 19, 1979, and in revised form October 6, 1980. This research was supported in part by the National Science Foundation under grant MCS 79 06110.

† Computer Science Department, Indiana University, Bloomington, Indiana 47405.

If the intermediate predicates can be evaluated quickly and are often false for small values of k , then backtracking takes considerably less time than complete enumeration, but if the intermediate predicates are ineffective, backtracking can take considerably longer. Although there are effective intermediate predicates for many problems, no general theory has been developed for finding them, and the dependence of the running time on the intermediate predicates as well as on the original predicate makes it difficult to do realistic analyses. In particular, a naive worst-case analysis, where each intermediate predicate for $k < v$ is true, is uninteresting.

The first problem in studying backtracking is to choose a model domain of problems that are both representative and amenable to analysis. Since there is no consensus on what constitutes a typical backtracking problem, we avoid introducing arbitrary assumptions into the analysis as long as possible. Our first requirement is that the sets of predicates we consider have natural intermediate predicates. Each predicate P is the conjunction of t terms. The corresponding k th intermediate predicate is the conjunction of those terms from P that contain only variables x_1 through x_k . Some of these sets of predicates contain NP-complete problems and have $\Theta(2^v)$ worst-case solution time when using backtracking. We obtain quite general formulas for the average solution time for these predicates. The formulas can be evaluated in time $O(v)$.

As a concrete illustration for our formulas, and as a model for more detailed investigation, we use the problem of finding all solutions of conjunctive normal form formulas. These sets of formulas fit our general model, and their natural intermediate predicates have a simple form that lends itself to analysis. They contain NP-complete problems. Moreover, the trees generated by these formulas have a shape typical of those encountered by the authors in our own experience with backtracking. Thus, these sets of formulas are a good model for analysis.

It is difficult to grasp the behavior of our general formulas as t and v become large. Therefore, for one type of random conjunctive normal form predicate, we derive asymptotic results, using $t = v^\alpha$ for some α . These results show that for such problems the average solution time using backtracking is exponential in v to a power that is less than one. Comparing this with the time exponential in v required for exhaustive search, it is evident that backtracking saves considerable time for nearly all problems in the class.

We foresee two main uses for our results. To obtain an accurate estimate for the running time of a given backtrack program, the method of Knuth [5] as modified by Purdom [8] should be used. But to decide whether it will be useful to attack a problem by backtracking, before investing the effort in writing a program, the formulas in this paper provide a rough guide. The second use for these results will be in a theoretical comparison of ordinary backtracking to various modifications of backtracking [1], [9], [10].

2. Notation and description of model. In our model each predicate P is the conjunction of t terms selected randomly from a set T of possible terms. Intermediate predicate P_k is the conjunction of the terms of P that use only variables x_1, \dots, x_k . The random process for selecting terms must be such that the probability that P contains only terms from the set T_1 , where $T_1 \in T$, is proportional to $Q(|T_1|, t)$ for some function Q . (The generalization to weighted sets is straightforward.) Two important cases are

$$\begin{aligned}
 (1) \quad & Q(|T_1|, t) = \begin{cases} |T_1|^t, & \text{(selection with replacement),} \\
 (2) \quad & \begin{cases} |T_1|^t, & \\ \binom{|T_1|}{t}, & \text{(selection without replacement).} \end{cases} \end{cases}
 \end{aligned}$$

Here Q is the number of ways the terms of P can be chosen; in all cases the probability is $Q(|T_1|, t)/Q(|T|, t)$.

Let d_i be the number of possible values for variable x_i . The set of terms T must be invariant under any operation that replaces each variable x_i with $p_i(x_i)$, where p_i is a permutation. (For binary variables each p_i is either the *not* or *identity* function.) Let $F(k)$ be the number of terms in T that use only variables x_1, \dots, x_k and that are false when those variables have each been assigned some value. The invariant condition on T implies that the same number of terms are false for any set of values of the variables x_1, \dots, x_k .

Let E be the total number of terms in T . If T consists of disjunctive clauses, where each clause contains s literals randomly selected from the v variables and their negations, then

$$(3) \quad F_s(k) = \begin{cases} k^s, & k \geq 0, \\ \binom{k}{s}, & k \geq 0, \end{cases} \quad E_s = \begin{cases} 2^s v^s, & \text{(selection with replacement),} \\ 2^s \binom{v}{s}, & \text{(selection without replacement),} \end{cases}$$

with $F_s(-1) = 0$ in both cases. For (3), selection with replacement, each of the s positions may be filled with any one of the $2v$ possible literals (the v variables and their negations). In (4), s different variables are chosen for the term and each may appear either negated or not negated. We call this protocol selection without replacement, though it differs slightly from the ordinary usage of that term.

Fig. 1 shows a set of predicates over two variables with two terms, where the terms have been selected without replacement. The terms are clauses with one literal per clause. The backtrack tree for each predicate is also shown.

3. Tree size. What is the expected number of nodes in a backtrack tree? Consider the tree in which each node on level i has degree d_{i+1} (the root is level 0). The node corresponding to $x_1 = y_1, x_2 = y_2, \dots, x_k = y_k$ (x_{k+1}, \dots, x_v not set) is reached by exactly those predicates that have no terms that are false for those particular values of the variables. There are $Q(E - F(k - 1), t)$ such predicates. Altogether level k contains $\prod_{1 \leq i \leq k} d_i$ nodes. (If $d_i = d$ for all i , then level k has d^k nodes.) The uniformity conditions on the set of predicates imply that the total number of nodes in all the backtrack trees is the product of the number of predicates and the number of nodes, summed over k . Dividing this by $Q(E, t)$, the number of backtrack trees, gives the expected number of nodes in a tree:

$$(5) \quad A(v, t) = \sum_{0 \leq k \leq v} \left(\prod_{1 \leq i \leq k} d_i \right) \frac{Q(E - F(k - 1), t)}{Q(E, t)}.$$

The expected number of solutions is

$$(6) \quad S(v, t) = \left(\prod_{1 \leq i \leq v} d_i \right) \frac{Q(E - F(v), t)}{Q(E, t)}.$$

As illustrated in Fig. 1, processes consistent with the assumptions of this section may generate some predicates with less than v variables. Appendix A treats the case of formulas having exactly v variables.

Formulas (5) and (6) apply to any method of selecting predicates that obeys the restrictions of § 2. Formulas for particular cases are obtained by using appropriate

versions of Q and F . The following examples are for terms consisting of disjunctive clauses with s literals per term.

$$A_s(v, t) = \begin{cases}
 (7) & 1 + \sum_{1 \leq k \leq v} 2^k \left(1 - \left(\frac{k-1}{2v} \right)^s \right)^t, & \text{(terms and literals selected with replacement),} \\
 (8) & 1 + \sum_{1 \leq k \leq v} 2^k \binom{2^s v^s - (k-1)^s}{t} \bigg/ \binom{2^s v^s}{t}, & \text{(terms selected without replacement, literals with),} \\
 (9) & 1 + \sum_{1 \leq k \leq v} 2^k \left(1 - 2^{-s} \binom{k-1}{s} \bigg/ \binom{v}{s} \right)^t, & \text{(terms selected with replacement, literals without),} \\
 (10) & 1 + \sum_{1 \leq k \leq v} 2^k \left(\binom{2^s \binom{v}{s} - \binom{k-1}{s}}{t} \bigg/ \binom{2^s \binom{v}{s}}{t} \right), & \text{(terms and literals selected without replacement).}
 \end{cases}$$

4. Asymptotic results. For the asymptotic analysis we consider formulas in conjunctive normal form, where each clause has s literals and t , the number of terms, is v^α . We require that both literals and terms be selected randomly with replacement. The expected tree size is therefore given by (7). We compute an asymptotic expression for the number of nodes in the backtrack tree for fixed α and s as v becomes large. Cook's construction in his NP-completeness paper [2] produces predicates in conjunctive normal form, where the number of terms increases as $v^{3/2}$. The number of literals per term also increases with v , but Cook's predicates can easily be converted to a form with three literals per term, and with the number of terms proportional to the number of variables. The set of predicates we consider in this section is NP-complete. This does not, of course, necessarily imply that the average time to solve a problem in the set will be large: the average time for any NP-complete set of problems can be made arbitrarily low by adding enough easy problems to the set. The set we analyze is interesting because it is natural and because it contains hard problems.

Using $t = v^\alpha$, formula (7) can be rewritten as

$$(11) \quad A_s(v, v^\alpha) = 1 + \sum_{0 < j \leq v} \exp \left(j \ln 2 + v^\alpha \ln \left(1 - \left(\frac{j-1}{2v} \right)^s \right) \right).$$

Let k be the value of j that maximizes the summand, and let $x = (k-1)/2v$. The value of x can be found by setting the derivative of the summand to zero (if the maximum is not at an endpoint), giving

$$(12) \quad qx^s + x^{s-1} - q = 0, \quad \text{where } q = \frac{(2 \ln 2)v^{1-\alpha}}{s};$$

the left-hand side of (12) is monotonically increasing for $x > 0$. Also, when x is 0, the l.h.s. is $-q$, which is less than zero; when $x = q^{1/(s-1)}$, the l.h.s. is $q^{(2s-1)/(s-1)} > 0$. Hence a solution to (12) exists and we have the upper bound

$$(13) \quad x = O(q^{1/(s-1)}) = O(v^{(1-\alpha)/(s-1)}).$$

The value of the maximum term is

$$(14) \quad \exp [2vx \ln 2 + v^\alpha \ln (1-x^s)] = \exp [O(v^{(s-\alpha)/(s-1)})].$$

Since (11) has only v terms, all of which are positive, we also have

$$(15) \quad A_s(v, v^\alpha) = \exp [O(v^{(s-\alpha)/(s-1)})].$$

In Appendix B we show that

$$(16) \quad A_s(v, v^\alpha) = 1 + 2\pi^{1/2} \sum_{\substack{p \geq 0 \\ q \geq 0}} \left(\frac{2 \ln 2}{s}\right)^{(2q-2p-1)s+2)/(2(s-1))} w_{pq} v^{((1-2p)/2)((s-\alpha)/(s-1))+qs(1-\alpha)/(s-1)}$$

$$\exp \left[\sum_{j \geq 1} (s-1)^{2-j} e_j(s) \left(\frac{2 \ln 2}{s}\right)^{js/(s-1)} v^{(js/(s-1))(1-\alpha)+\alpha} \right],$$

where e_j and w_{pq} are given in Tables 1 and 2. The tables were calculated using REDUCE programs and also checked by hand.

TABLE 1
Coefficients for formula (16)

$e_1(s) = 1$
$e_2(s) = -\frac{1}{2}$
$e_3(s) = \frac{1}{6}(s+2)$
$e_4(s) = -\frac{1}{12}(s^2+4s+3)$
$e_5(s) = \frac{1}{120}(6s^3+37s^2+58s+24)$
$e_6(s) = -\frac{1}{60}(2s^4+17s^3+42s^2+37s+10)$
$e_7(s) = \frac{1}{5040}(120s^5+1318s^4+4553s^3+6388s^2+3708s+720)$
$e_8(s) = -\frac{1}{2520}(45s^6+612s^5+2761s^4+5456s^3+5071s^2+2124s+315)$
$e_9(s) = \frac{1}{40320}(560s^7+9148s^6+51540s^5+133769s^4+175804s^3+118236s^2+37904s+4480)$
$e_{10}(s) = -\frac{1}{45360}(504s^8+9666s^7+65881s^6+214554s^5+371550s^4+354106s^3+182529s^2+46674s+4536)$

TABLE 2
Coefficients for formula (16)

$w_{00} = 2\left(\frac{2}{s(s-1)}\right)^{1/2}$
$w_{01} = -\left(\frac{s+1}{s-1}\right)\left(\frac{2}{s(s-1)}\right)^{1/2}$
$w_{02} = \frac{3s^2+8s+3}{4(s-1)^2}\left(\frac{2}{2(s-1)}\right)^{1/2}$
$w_{10} = -\frac{4!}{2!2^3}\left(\frac{s}{4}\right)\left(\frac{2}{s(s-1)}\right)^{5/2} + \frac{6!}{3!2^6}\left(\frac{s}{3}\right)^2\left(\frac{2}{s(s-1)}\right)^{7/2}$
$w_{11} = \frac{4!}{2!2^3}\left[-\frac{1}{2}\left(\frac{2s}{4}\right) + \frac{6s+1}{s-1}\left(\frac{s}{4}\right)\right]\left(\frac{2}{s(s-1)}\right)^{5/2} + \frac{6!}{3!2^6}\left[\left(\frac{s}{3}\right)\left(\frac{2s}{3}\right) - \frac{11s-5}{2(s-1)}\left(\frac{s}{3}\right)^3\right]\left(\frac{2}{s(s-1)}\right)^{7/2}$
$w_{20} = -\frac{6!}{3!2^5}\left(\frac{s}{6}\right)\left(\frac{2}{s(s-1)}\right)^{7/2} + \frac{8!}{4!2^7}\left[\left(\frac{s}{4}\right)^2 + 2\left(\frac{s}{3}\right)\left(\frac{s}{5}\right)\right]\left(\frac{2}{s(s-1)}\right)^{9/2} + \frac{12!}{4!6!2^{11}}\left(\frac{s}{3}\right)^4\left(\frac{2}{s(s-1)}\right)^{13/2}$

When $s = 3$ and $\alpha = \frac{3}{2}$, this gives

$$(17) \quad A_3(v, v^{3/2}) = 1 + e^{0.6282482661} v^{3/4} [4.725676293 v^{3/8} - 0.9010567850 v^{-3/8} + 0.3756198156 v^{-9/8} + O(v^{-15/8})].$$

In work of this type checking is important to avoid errors. Formulas (7)–(10) were checked for small values of v , t and s by comparison with programs that generated all predicates in a class and counted the number of nodes in the backtrack trees. Special cases for formula (6) were checked in the same way. Formulas (16) and (17) were checked using numerical comparison with formula (7). In formula (16), $s = 3, 4, 5$ and 6 were checked with α varying from 1 to s in 20 steps. For $s = 3$ and 4 , the formula checked. For $s = 5$ and 6 , overflow errors prevented setting v large enough (a few thousand) for a definitive test. The successful tests check for all errors in the values given in Table 2, except in the terms in w_{20} that are multiplied by $\binom{s}{5}$ or $\binom{s}{6}$.

Mr. Khaled Bugrara is currently investigating formulas (8)–(10). Preliminary results indicate that these formulas have similar asymptotic behavior to formula (7).

5. Conclusions. Formulas (5), (6) and (21) (Appendix A) are general formulas that can be used to predict the average time required for backtracking over many classes of randomly selected predicates. Formulas (5) and (6) can be evaluated exactly in time proportional to v , the number of variables. The largest term in formula (5) determines the value to within a factor of $v^{1/2}$. For such exponential problems this accuracy is often adequate.

The asymptotic result for random conjunctive normal form predicates, given in (16), shows that backtracking can save substantial time over exhaustive search on the average. Although the average time for backtracking is exponential in v , the dependence of the exponent on v is sublinear. For random problems backtracking does best on problems with low direct interdependence (small s) and on problems with a lot of restrictions (large α).

It is important to consider whether studies of random conjunctive normal form predicates lead to valid conclusions about typical backtracking problems. Certainly in our experience the typical backtracking problem is not in conjunctive normal form. Nevertheless, the random conjunctive normal form predicates do have many properties that we regard as typical. The constraints are initially not very effective, but they become more so as one goes down the search tree, so that the number of nodes per level shows a rapid increase up to a rounded peak followed by a rapid decrease. There is some correlation between adjacent branches of the search tree, but it is not very significant. On the other hand, real problems often have solutions, while random conjunctive normal form predicates with enough terms to mimic what we consider to be a typical problem almost never have solutions. The existence of solutions, however, does not have much effect on the size of the search tree. Our model, which is qualitatively correct on the important aspects of the problem, can be expected to give qualitatively correct results.

Goldberg [4] analyzed the average time for a variant of the Putnam–Davis procedure on a class of conjunctive normal form predicates. The time he obtained was polynomial. For a further discussion of Goldberg’s results, see [11].

We have analyzed only the most straightforward backtracking algorithm. There are variations on backtracking [1], [9] that are careful about which variable to introduce at each step in the search. The investigations of Bitner and Reingold [1] as well as our own numerical studies with random conjunctive normal form predicates show that

these variations can be much more efficient than traditional backtracking. All these backtracking methods maintain the advantage of treating the predicate as a black box: the algorithms are controlled only by the results of evaluating the predicate for selected values of the variables. We are now analyzing these methods [10].

Appendix A. As illustrated in Fig. 1, processes consistent with the assumptions of § 3 may generate some predicates with less than v variables. To study predicates with exactly v variables we replace the requirement that the set T be invariant under permutations of the values of the variables with a more restrictive assumption: the number $F(k)$ of terms that are false when any k variables are set is independent of which variables are set and of the values assigned to the variables. We also require that the number of terms that use no more than k particular variables be $E(k)$, independent of which k variables are considered. Under these assumptions the number of predicates that reach a particular node on level k and that do not use variable x_j is

$$(18) \quad \begin{aligned} Q(E(v-1)-F(k-2), t) & \text{ for } j \leq k-1, \\ Q(E(v-1)-F(k-1), t) & \text{ for } j \geq k. \end{aligned}$$

The number of predicates that reach a particular node on level k and that do not use j of the variables, where i of the j variables have indices less than k , is

$$(19) \quad \binom{k-1}{i} \binom{v-k+1}{j-1} Q(E(v-j)-F(k-i-1), t),$$

where the binomials account for the number of ways the i variables less than k and the $j-i$ variables greater than or equal to k can be selected. From the principle of inclusion and exclusion, the number of predicates that use all v variables and reach a particular node on level k is (for $k \geq 1$)

$$(20) \quad \begin{aligned} & \sum_{i,j} (-1)^{v-j} \binom{k-1}{i} \binom{v-k+1}{j-i} Q(E(v-j)-F(k-i-1), t) \\ & = \sum_{i,j} (-1)^j \binom{k-1}{i} \binom{v-k+1}{j-i} Q(E(j)-F(i), t). \end{aligned}$$

Multiplying by $\prod_{1 \leq i \leq k} d_i$, summing over k and dividing by the number of predicates gives the average number of nodes for the backtrack trees for predicates that use all v variables:

$$(21) \quad A(v, t) = 1 + \frac{\sum_{1 \leq k \leq v} \left(\prod_{1 \leq i \leq k} d_i \right) \sum_{i,j} (-1)^j \binom{k-1}{i} \binom{v-k+1}{j-i} Q(E(j)-F(i), t)}{\sum_j (-1)^j \binom{v}{j} Q(E(j), t)}.$$

Appendix B. This appendix gives a derivation of (16). As we will show, the summands in (7) are approximately Gaussian. We asymptotically sum the series by finding the position of the peak, expanding the deviation from a Gaussian in a power series, and summing the power series times the Gaussian using the Euler summation formula [6, p. 110]. Using either successive approximations or power series methods [7, pp. 444–450] on (12) gives

$$(22) \quad x = \left(\frac{2 \ln 2}{s} \right)^{1/(s-1)} v^{(1-\alpha)/(s-1)} \sum_{j \geq 0} (s-1)^j f_j(s) \left(\frac{2 \ln 2}{s} \right)^{js/(s-1)} v^{js(1-\alpha)/(s-1)}.$$

The coefficients are given by the relations

$$\begin{aligned}
 f_0(s) &= 1, & g_0(s) &= 1, & g_1(s) &= -1, \\
 (23) \quad f_k(s) &= \sum_{1 \leq j \leq k} \left[\frac{s^j}{(s-1)k} - 1 \right] g_j(s) f_{k-j}(s) (s-1)^j, \\
 g_k(s) &= \sum_{1 \leq j \leq k-1} \left[\frac{(s+1)^j}{k-1} - 1 \right] f_j(s) g_{k-j}(s) (s-1)^j.
 \end{aligned}$$

The $g_j(s)$ are coefficients in a power series expansion for $1 - x^s$. Values through $f_{10}(s)$ are given in Table 3. The power series converges only for $\alpha > 1$. (For $\alpha < 1$, we have $k > v$.) Only the first few values of $f_j(s)$ are needed unless α is near one.

TABLE 3
Coefficients for formula (22)

$f_0 = 1$
$f_1 = -1$
$f_2 = \frac{1}{2}(s+2)$
$f_3 = -\frac{1}{3}(s^2+4s+3)$
$f_4 = \frac{1}{24}(6s^3+37s^2+58s+24)$
$f_5 = -\frac{1}{10}(2s^4+17s^3+42s^2+37s+10)$
$f_6 = \frac{1}{720}(120s^5+1318s^4+4553s^3+6388s^2+3708s+720)$
$f_7 = -\frac{1}{315}(45s^6+612s^5+2761s^4+5456s^3+5071s^2+2124s+315)$
$f_8 = \frac{1}{4480}(560s^7+9148s^6+51540s^5+133769s^4+175804s^3+118236s^2+37904s+4480)$
$f_9 = -\frac{1}{4536}(504s^8+9666s^7+65881s^6+214554s^5+371550s^4+354106s^3+182529s^2+46674s+4536)$
$f_{10} = \frac{1}{3628800}(362880s^9+8026416s^8+64621692s^7+255644668s^6+557061609s^5$ $+ 700870638s^4+512539012s^3+210852936s^2+44339040s+3628800)$

The value of the maximum term is

$$\begin{aligned}
 (24) \quad & \exp(2vx \ln 2 + v^\alpha \ln(1-x^s)) \\
 & = \exp\left(\sum_{j \geq 1} (s-1)^{2-j} e_j(s) \left(\frac{2 \ln 2}{s}\right)^{js/(s-1)} v^{js(1-\alpha)/(s-1)+\alpha}\right),
 \end{aligned}$$

where

$$\begin{aligned}
 (s-1)^{2-j} e_j(s) &= s f_{j-1}(s) - \sum_{1 \leq k \leq j} \frac{1}{k} h_{j-k,k}(s), \\
 h_{0k}(s) &= 1, \\
 h_{ik}(s) &= - \sum_{1 \leq j \leq i} \left(\frac{(k+1)j}{i} - 1\right) g_{j+1}(s) h_{i-j,k}(s).
 \end{aligned}$$

The h_{ik} are coefficients in the power series expansion of x^{ks} . Values of $e_j(s)$ are given in Table 2.

Replacing the j in (11) by $j+k$ and expanding the natural logarithm in a power series gives

$$\begin{aligned}
 (25) \quad A_s(v, v^\alpha) &= 1 + \sum_{-k < j \leq v-k} \exp \left[(j+k) \ln 2 + v^\alpha \ln \left(1 - \left(\frac{j+k-1}{2v} \right)^s \right) \right] \\
 &= 1 + \sum_{-k < j \leq v-k} 2 \exp \left[(j+2vx) \ln 2 - v^\alpha \sum_n \left(\sum_{i \geq 1} \frac{1}{i} \binom{is}{n} x^{is-n} \right) \left(\frac{j}{2v} \right)^n \right].
 \end{aligned}$$

The factor independent of j is

$$(26) \quad 2 \exp \left(2vx \ln 2 - v^\alpha \sum_{i \geq 1} \frac{1}{i} x^{is} \right) = 2 \exp (2vx \ln 2 + v^\alpha \ln (1 - x^s)).$$

This is the value of the maximum term; it can be moved outside the sum over j . The terms in the exponent that are proportional to j are

$$(27) \quad \ln 2 - v^\alpha \sum_{i \geq 1} \frac{sx^{is-1}}{2v} = \ln 2 - \frac{sv^\alpha}{2v} \frac{x^{s-1}}{1-x^s}.$$

This is zero by (12). Using (26) and (27) and separating the j^2 term from the rest gives

$$\begin{aligned}
 (28) \quad A_s(v, v^\alpha) &= 1 + 2 \exp (2vx \ln 2 + v^\alpha \ln (1 - x^s)) \\
 &\quad \sum_{-k < j \leq v-k} \exp (-aj^2) \exp \left(\sum_{n \geq 3} t_n j^n \right),
 \end{aligned}$$

where

$$\begin{aligned}
 a &= v^\alpha \sum_{i \geq 1} s \frac{is-1}{8v^2} x^{is-2}, \\
 t_n &= -v^\alpha \sum_{i \geq 1} \frac{1}{i} \binom{is}{n} x^{is-n} \left(\frac{1}{2v} \right)^n \quad \text{for } n \geq 3.
 \end{aligned}$$

Now

$$(29) \quad \exp \left(\sum_{n \geq 3} t_n j^n \right) = 1 + \sum_{i \geq 3} b_i j^i,$$

where

$$(30) \quad b_i = \sum_R \prod_{n \geq 3} \frac{t_n^u}{u_n!}, \quad R = \left\{ u_3 \geq 0, u_4 \geq 0, \dots \mid \sum_{n \geq 3} n u_n = i \right\}.$$

For example, $b_3 = t_3$, $b_4 = t_4$, $b_5 = t_5$ and $b_6 = t_6 + t_3^2/2$. This reduces the problem to evaluating sums of the form $\sum_{-k < j \leq v-k} b_n j^n \exp (-aj^2)$.

Using $f_n(y) = b_n y^n \exp (-ay^2)$ in the Euler summation formula [6, p.110] gives

$$\begin{aligned}
 (31) \quad &\sum_{-k < j \leq v-k} b_n j^n \exp (-aj^2) \\
 &= \int_{-k+1}^{v-k+1} f_n(y) dy + \sum_{1 \leq p \leq m} \frac{B_p}{p!} (f_n^{(p-1)}(v-k+1) - f_n^{(p-1)}(-k+1)) \\
 &\quad + \frac{(-1)^{m+1}}{m!} \int_{-k+1}^{v-k+1} B_m(\{y\}) f_n^{(m)}(y) dy,
 \end{aligned}$$

where parenthesized superscripts indicate derivatives, the B_i are Bernoulli numbers and polynomials, and $\{ \cdot \}$ is the sawtooth function. The error term is $O(vf_n^{(m)}(z))$, where z is the value of y that maximizes $f_n^{(m)}(y)$. Now $f_n^{(m)}(y)$ has the form $a^{(m-n)/2}R_{m+n}(a^{1/2}y)$, where $R_n(y)$ is e^{-y^2} times an n th degree polynomial. The coefficients of the polynomial are numbers and do not change with a , so the maximum of $R_{m+n}(a^{1/2}y)$ is independent of a . Therefore the error term is $O(a^{(m-n)/2}v)$. Since $a = O(v^{\alpha-2}x^{s-2}) = O(v^{(\alpha-s)/(s-1)})$, the error term is $O(v^{1+(m-n)(\alpha-s)/(2(s-1))})$. For $\alpha < s$ and m sufficiently large, the error term becomes small faster than any term we retain, so we will be able to neglect it if we can show that there is no problem in making m large.

To do this we show that the terms from (31) in the sum over p can be neglected. Consider $f_n^{(p-1)}(-k+1)$. Its asymptotic behavior as v becomes large depends on its exponent, which is

$$a(-k+1)^2 = O(v^{(\alpha-s)/(s-1)}v^{2+2(1-\alpha)/(s-1)}) = O(v^{(s-\alpha)/(s-1)}).$$

Therefore $f^{(p-1)}(-k+1)$ becomes exponentially small for $\alpha < s$. If $\alpha > 1$, then $f^{(p-1)}(v-k+1)$ also becomes exponentially small as v increases. Since the final answer is polynomial with fractional powers, the exponentially small terms in the sum over p can be neglected, so m can be made as large as desired.

This leaves $\int_{-k+1}^{v-k+1} f_n(y) dy$ to be evaluated. For $1 < \alpha < s$, this integral differs only by exponentially small terms from

$$(32) \quad \int_{-\infty}^{\infty} f_n(y) dy = \begin{cases} \frac{\pi^{1/2}n!}{(n/2)!2^n} a^{-(n+1)/2} b_n & \text{for } n \text{ even,} \\ 0 & \text{for } n \text{ odd.} \end{cases}$$

This gives, from (28) and (29),

$$(33) \quad A_s(v, v^\alpha) = 1 + 2\pi^{1/2} \left[a^{-1/2} + \sum_{\substack{i \geq 3 \\ i \text{ even}}} \frac{i!}{(i/2)!2^i} a^{-(i+1)/2} b_i \right] \cdot \exp(2vx \ln 2 + v^\alpha \ln(1-x^s)).$$

Expanding the factor of (33) that is in square brackets in a power series gives, using (22), (28) (for a and t_n), and (29),

$$(34) \quad \sum_{\substack{p \geq 0 \\ q \geq 0}} \left(\frac{2 \ln 2}{s} \right)^{((2q-2p-1)s+2)/(2(s-1))} w_{pq} v^{((1-2p)/2)((s-\alpha)/(s-1))+qs(1-\alpha)/(s-1)},$$

where the first few values for w_{pq} are given in Table 3. In the power series expansion w_{0q} is obtained from $a^{-1/2}$ and w_{pq} for $p \geq 1$ is obtained from all factors containing

$$(35) \quad a^{-((\sum_{r=0} r u_r + 1)/2)} \prod_n t_n^{u_n},$$

where $u_1 = 0$, $u_2 = 0$, $\sum_{r \geq 0} (r-2)u_r = 2p$, and $u_r \geq 0$ for all r . For example, w_{1q} is obtained from t_4 and t_3^2 , and w_{2q} is obtained from t_6 , $t_5 t_3$, t_4^2 and t_3^4 . Substituting (34) into (33) gives (16).

REFERENCES

[1] J. R. BITNER AND E. M. REINGOLD, *Backtrack programming techniques*, Comm. ACM, 18 (1975), pp. 651-655.

- [2] S. A. COOK, *The complexity of theorem-proving procedures*, Proc. Third ACM Symposium on Theory of Computing, 1971, pp. 151–158.
- [3] M. DAVIS AND H. PUTNAM, *A computing procedure for quantification theory*, J. Assoc. Comput. Mech., 7 (1960), pp. 201–215.
- [4] A. GOLDBERG, *Average case complexity of the satisfiability problem*, Proc. Fourth Workshop on Automated Deduction, 1979, pp. 1–6.
- [5] D. E. KNUTH, *Estimating the efficiency of backtracking programs*, Math. Comput., 29 (1975), pp. 121–136.
- [6] ———, *The Art of Computer Programming*, vol. 1, Addison–Wesley, Reading, MA, 1975.
- [7] ———, *The Art of Computer Programming*, vol. 2, Addison–Wesley, Reading, MA, 1969.
- [8] P. W. PURDOM, *Tree size by partial backtracking*, this Journal, 7 (1978), pp. 481–491.
- [9] P. W. PURDOM, E. E. ROBERTSON AND C. A. BROWN, *Backtracking with multi-level dynamic search arrangement*, Acta Informat., 15 (1981), pp. 99–114.
- [10] P. W. PURDOM AND C. A. BROWN, *An analysis of backtracking with search rearrangement*, Indiana University Computer Science Department Technical Report 89, 1980.
- [11] ———, *The Goldberg Putnam-Davis procedure requires exponential average time*, Indiana University Computer Science Department Technical Report 101, 1981.

DEADLOCK-FREE PACKET SWITCHING NETWORKS*

SAM TOUEG† AND JEFFREY D. ULLMAN‡

Abstract. Deadlock states have been observed in existing computer networks, emphasizing the need for carefully designed flow control procedures (controllers) to avoid deadlocks. Such a deadlock-free controller is readily found if we allow it global information about the overall network state. Generally, this assumption is not realistic, and we must resort to deadlock-free local controllers using only packet and node information. We present here several types of such controllers, we study their relationship and give a proof of their optimality with respect to deadlock-free controllers using the same set of local parameters.

Key words. packet switching, uniform controller, flow control, deadlock-freedom, communication network, forward-backward controller, fixed-adaptive routing

1. Introduction.

1.1. Basic definitions. A *packet switching network* is a directed graph $G = (V, E)$; the vertices V present processors and the edges E represent communication links. We assume messages, called *packets*, are to be passed between processors. Each network has an associated constant b , the number of *buffers* at each vertex; a buffer can hold exactly one packet. Associated with each packet is an acyclic *route* v_1, v_2, \dots, v_q , which is a path in G . Vertex v_1 is the *source* and v_q is the *destination* vertex for the packet. We assume a *fixed routing procedure* [KL], where a packet's route is determined at the source node. We show later how our results can be used with *adaptive routing procedures* [KL], where the packet's route is dynamically computed at each node of the path according to such factors as channel availability or channel and node congestion. We may also assume that the route of a packet is included as part of the message in the packet, although in practice the packet could hold only the source and destination, with each processor in the network responsible for deducing the next vertex to which the packet is to be passed. Also associated with a network G is the constant k , the length of the longest route taken by a packet in G . If we want to state explicitly the two constants associated with a network G we write that G is a (b, k) -network. Note that k need not be the length of the longest path in the network; we may never wish to send messages between distant nodes.

The *moves* made by the network are of three types.

1. *Generation.* A vertex v accepts, and places in an empty buffer, a packet p created by a process P residing in v .

2. *Passing.* A vertex v transfers a packet in one of its buffers to an empty buffer of vertex w , where $v \rightarrow w$ is an edge, and the route for the packet has w following v . The buffer of v holding the packet becomes empty.

3. *Consumption.* A packet in a buffer of v , such that the destination for the packet is v , is removed from that buffer and the buffer is made empty.

1.2. Controllers. A *controller* for a network is an algorithm that permits or forbids various moves in the network. One of the key problems in packet switching is

* Received by the editors July 11, 1979, and in final form August 25, 1980. This work was completed while the authors were at the Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey 08544. The original version of this paper appeared in the Proceedings of the 11th annual ACM Symposium on Theory of Computing, Atlanta, Georgia, April 1979, pp. 89-98. Copyright 1980, Association for Computing Machinery, Inc. Used by permission.

† Department of Computer Science, Cornell University, Ithaca, New York 14853. The work of this author was supported in part by the National Science Foundation under Grant GK-42048, and in part by the U.S. Army Research Office, Durham, NC under Grant DAAG29-75-0192.

‡ Department of Computer Science, Stanford University, Stanford, California 94305. The work of this author was partially supported by the National Science Foundation under grant MCS-76-15255.

preventing *deadlocks*, which are situations in which one or more packets can never reach their destination no matter what sequence of moves is subsequently performed. For example, in the network of Fig. 1, if all physically possible moves are permitted by the controller, v_1 generates b packets with destination v_2 , v_2 does the same with destination v_3 and v_3 does the same with destination v_1 ; then all buffers of all vertices will be full, no consumption moves can take place without a pass move, and no generation can take place. It is not hard to see that the network is deadlocked.

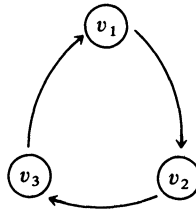


FIG.1. A network exhibiting deadlock with a trivial controller.

However, if we use a controller that simply prohibits the generation (but not passing) of a packet into the last empty buffer at a vertex, then we can show that at least one empty buffer must exist somewhere in the network. Hence it is always possible to pass or consume some packet if there are any packets in the network, and this controller is deadlock-free.

In what follows, deadlock is assumed to occur with respect to some controller. That is, a controller is *deadlock-free* (or DF) for a given network if it does not permit this network to enter a state in which one or more packets can never make a move permitted by that controller, as long as no additional packets are generated.¹

1.3. Fundamental questions. We have assumed that each packet is generated with a fixed route to travel. There are still several options left to us.

1. Is the network given, or are we designing a network of N vertices and the routes between each pair of vertices?² In each case, we could consider the *optimal controller*, the one that is deadlock-free and puts the least restriction on moves.

2. Are we looking for a (b, k) DF *uniform controller*, one that is deadlock-free for any (b, k) -network, or are we designing a controller for one particular network?

We shall consider uniform controllers first and then examine some particular networks.

1.4. Local controllers. Whereas in general a controller can examine the state of the entire network, we do not consider that this is a reasonable assumption. There has to be at least one vertex at which the controller resides, and this vertex would have to be

¹ The condition that no additional packets enter the network is essential, else we could have, for some strange controllers, a situation where a packet can move, but only if some other packet is generated; that packet can only move when a third is generated, and so on.

² Many of these design problems are difficult to solve. The following related problems are both NP-Complete [TS1]:

1. Given a network and a set of source-destination pairs, is there a corresponding set of routes such that the unrestricted flow of packets is deadlock-free?

2. Given a network and a set of source-destination routes with corresponding end-to-end window flow controls, is the network exposed to deadlock?

connected directly to every other vertex, which requires an arbitrary number of connections to one processor, or packets informing the controller of local conditions would have to be passed around the network, and this information would itself alter the state of the network (and generate too much message traffic).

Thus we shall restrict ourselves to *local controllers*, where each processor decides on the legality of accepting a packet and the decision is made according to *local information* alone. To this purpose, we shall consider the following local information defining node and packet states in a (b, k) -network G . We characterize the *local state* of a node v of G by (any subset of) the following parameters:

m , the number of free buffers in the node v ,

n , the number of packets stored in v 's buffers,

$\mathbf{i} = \langle i_0, i_1, \dots, i_k \rangle$, where i_j , $(0 \leq j \leq k)$, is the number of packets in v whose source distance to v is j , and

$\mathbf{j} = \langle j_0, j_1, \dots, j_k \rangle$, where j_i , $(0 \leq i \leq k)$, is the number of packets in v whose destination distance from v is i .

Note that we have the following relations:

$$n = \sum_{r=0}^k i_r = \sum_{r=0}^k j_r,$$

$$m = b - n.$$

Similarly, any packet p asking access to a node v has a *packet state relative to v* characterized by (any subset of) the following parameters:

i , the distance from the source of p to the node v ,

j , the distance from the node v to the destination of p .

Clearly $i + j$ is the length of p 's route; therefore $i + j \leq k$. Note that, with our assumption of fixed routing policy, all the local information listed above is readily available to any node v when it is considering whether to accept a packet p .

In our formalism, we shall define an (α, β) controller S to be a set of (α, β) -tuples where α denotes the information about a node state used and β denotes the information about a packet state used. (α_0, β_0) is in S if and only if it is permissible for an α_0 -state vertex to accept a β_0 -state packet.

We shall study the following four types of controllers, each one of them using different local information: the (m, j) or *forward-count* controllers, the (\mathbf{j}, j) or *forward-state* controllers, the (n, i) or *backward-count* controllers and the (\mathbf{i}, i) or *backward-state* controllers. For each of these four classes of controllers, and for $b > k$, we shall present a (b, k) DF uniform controller in the sense that it is deadlock-free for any (b, k) -network G . We later show that for $b \leq k$ there are no (b, k) DF uniform controllers of these types: if we want to use a (m, j) , (\mathbf{j}, j) , (n, i) or (\mathbf{i}, i) DF uniform controller in networks G where the longest packet route is of length k , then in general G must be provided with a minimal node buffer capacity of $k + 1$. This is not a serious practical limitation: in most, if not all, of the implemented packet switching networks the number of available buffers in each node far exceeds the length of the longest path taken by a packet in these networks (for example, in ARPANET the node buffer capacity is greater than 50 [KL]). We also look at the relation between the proposed DF uniform controllers,³ and we show their optimality in the sense that any other DF uniform controller of the same type (i.e., using the same local information) must be a subset of the corresponding controller proposed.

³We shall omit the parameters (b, k) whenever they are understood: "uniform controller" stands for " (b, k) uniform controller."

2. Local DF uniform controllers.

2.1. A simple DF uniform controller: the forward-count controller. We begin our study of uniform controllers by considering a forward-count controller. As we saw before, these controllers use only the following local information when deciding whether or not a node w should accept a packet p :

- 1) the length j of the path from w to the destination of p along p 's route, and
- 2) the number m of empty buffers that w currently has.

Formally, for $b > k$, let $FC(b, k)$, or just FC when (b, k) is understood, stand for

$$\{(m, j) \mid (j < m) \text{ and } (0 \leq j \leq k) \text{ and } (1 \leq m \leq b)\}.$$

That is, FC permits a packet to be generated in or passed to a vertex w , provided w has more free buffers than the packet has steps to go.

THEOREM 1. *FC is a DF uniform controller.*

Proof. Suppose FC is not a DF controller for a (b, k) -network G . After reaching a deadlock condition, make whatever moves can be made without generating any new packets. Now we have a state of G in which there is at least one packet, and no packets can move. Let p_1 be a blocked packet; p_1 is in the node v_1 , and v_1 is distance d_1 away from p_1 's destination node (note that $d_1 \geq 1$, else p_1 is consumed in v_1). Let v_2 be the next node in p_1 's route, and let m_2 be the number of free buffers in v_2 . Surely $d_1 < b$, else p_1 could not have been accepted at v_1 , and $(d_1 - 1) \geq m_2$, else p_1 could be passed to v_2 . So $m_2 < b$ and there is at least one blocked packet in v_2 . Let d_2 be the distance to the destination of the last packet p_2 to enter node v_2 . Since p_2 was accepted by v_2 then $d_2 < m_2 + 1$, thus $d_2 < d_1$. By reductio ad absurdum we can show there is a deadlocked packet with zero moves to go; but surely such a packet would be consumed. \square

2.2. A DF forward-state uniform controller. If we use more information about the packets in the receiving node, we may do better than FC . In particular, suppose we want to pass from v to w a packet p with j steps to go (from w), but w has fewer than $j + 1$ empty buffers. We may still be able to pass p if when we consider the set consisting of p and the packets already in w 's buffers, we can find some order in which they could have arrived at w according to the controller FC . To formalize the above idea, we define the following forward-state controller. For $b > k$ let $FS(b, k)$, or just FS when (b, k) is understood, be the set

$$\left\{ (\mathbf{j}, j) \mid \text{for all } i, 0 \leq i \leq j, i < b - \sum_{r=i}^k j_r, \text{ and } 0 \leq j \leq k, \text{ and } 0 \leq \sum_{r=0}^k j_r \leq b - 1 \right\}.$$

In the above, we assume $\mathbf{j} = \langle j_0, j_1, \dots, j_k \rangle$.

THEOREM 2. *FS is a DF forward-state uniform controller.*

Proof. Suppose FS is not a DF controller for a (b, k) -network G . As in Theorem 1 we can show the existence of a deadlocked packet p_1 in a node v_1 which is d_1 away from p_1 's destination ($d_1 \geq 1$). Let v_2 be the next node in p_1 's route and let $\mathbf{j} = \langle j_0, j_1, \dots, j_k \rangle$ be the state of v_2 . Since p_1 was accepted in v_1 we have $d_1 < b$. We can then deduce that $\sum_{r=0}^k j_r \neq 0$, else $(\mathbf{j}, d_1 - 1)$ is in FS and p_1 would be accepted by v_2 . So, there is at least one deadlocked packet in v_2 . Let p_2 be a packet in v_2 whose distance d_2 to its destination is minimal with respect to the other deadlocked packets in v_2 ,

$$d_2 = \min \{d \mid j_d > 0\}.$$

That is $j_0 = j_1 = \dots = j_{d_2-1} = 0$ and the state of v_2 is

$$\mathbf{j} = \langle 0, \dots, 0, j_{d_2}, j_{d_2+1}, \dots, j_k \rangle.$$

Note that $d_2 \neq 0$, else p_2 is consumed at v_2 . We now show that $d_2 < d_1$. Let p_s be the last packet accepted by v_2 . Let d_s be the distance from v_2 to the destination node of p_s . By our definition of d_2 we have $d_2 \leq d_s$. Since p_s was accepted by v_2 then

$$\langle (0, \dots, 0, j_{d_2}, \dots, j_{d_s} - 1, \dots, j_k), d_s \rangle$$

is in FS. So, for all i , $0 \leq i \leq d_s$, we have

$$i < b - (j_i + \dots + (j_{d_s} - 1) + \dots + j_k),$$

for $i = d_2$ we have

$$d_2 < b - \left(\sum_{r=d_2}^k j_r \right) + 1,$$

and therefore

$$(*) \quad d_2 \leq b - \sum_{r=d_2}^k j_r.$$

Since p_1 is not accepted by v_2 , then

$$\langle (0, \dots, j_{d_2}, j_{d_2+1}, \dots, j_{d_s}, \dots, j_k), d_1 - 1 \rangle \notin \text{FS},$$

so there exists i_0 , $0 \leq i_0 \leq d_1 - 1$ such that $i_0 \geq b - \sum_{r=i_0}^k j_r$. Suppose, for contradiction, that $d_2 > d_1 - 1$. In this case $i_0 \leq d_1 - 1 < d_2$ and since $j_0 = \dots = j_{d_2-1} = 0$, it follows that $\sum_{r=i_0}^k j_r = \sum_{r=d_2}^k j_r$. Therefore, there exists i_0 , $0 \leq i_0 \leq d_1 - 1$, such that $i_0 \geq b - \sum_{r=d_2}^k j_r$. But $d_2 > d_1 - 1 \geq i_0$, so $d_2 > b - \sum_{r=d_2}^k j_r$, contradicting (*). We conclude that $d_2 \leq d_1 - 1$, or $d_2 < d_1$. As in the previous proof, we can use reductio ad absurdum to prove the existence of a deadlocked packet with zero moves to go. \square

2.3. A backward-count controller. [G], [RH], [GHKP], [MS] discuss a controller in which the number of free buffers in the receiving vertex and the number of steps made so far (as opposed to the number of steps to go) govern the legality of a move. In our formalism, for $b > k$, a similar backward-count controller can be expressed as the following BC(b, k) set (we call it BC whenever the parameters are understood).

$$\{(n, i) \mid i \geq n - b + (k + 1) \text{ and } 0 \leq i \leq k \text{ and } 0 \leq n \leq b - 1\}.$$

With the BC controller, in the special case that $b = k + 1$, a packet that has made i moves from its source to a node w is accepted by w only if w has at most i nonempty buffers.

THEOREM 3. BC is a DF backward-count uniform controller.

Proof. Similar to the proof of Theorem 1. \square

2.4. A backward-state controller. Similarly to the case for forward-state controllers we can use more information about packets in the receiving node to define the following backward-state controller. For $b > k$, let BS(b, k) stand for

$$\left\{ (i, i) \mid \text{for all } j, i \leq j \leq k, j \geq \sum_{r=0}^j i_r - b + (k + 1) \text{ and } 0 \leq i \leq k \text{ and } 0 \leq \sum_{r=0}^k i_r \leq b - 1 \right\}.$$

THEOREM 4. BS is a DF backward-state uniform controller.

Proof. Similar to the proof of Theorem 2. \square

3. The optimality of FC, FS, BC and BS.

3.1. The relation between FC, FS, BC and BS controllers. We introduce a partial order on the family of DF uniform controllers in the following way. Let S_1 and S_2 be two DF uniform controllers. We say that S_1 is *at least as good as* S_2 if S_1 is not more

restrictive than S_2 , that is, if in any network G , any move allowed by S_2 is also allowed by S_1 . We write $S_2 \subseteq S_1$. Also, S_1 is *better than* S_2 if S_1 is strictly less restrictive than S_2 . In our notation we write $S_2 \subset S_1$ (but it is *not* necessary for two arbitrary DF uniform controllers to be comparable).

To compare FC, FS, BC and BS we must first put them in an equivalent $[(\mathbf{i}, \mathbf{j}), (i, j)]$ normal form. This is best explained with an example. The normal form of FC is given by the following set:

$$\left\{ [(\mathbf{i}, \mathbf{j}), (i, j)] \mid j < b - \sum_{r=0}^k j_r, \text{ and } 1 \leq i + j \leq k \text{ and } 0 \leq \sum_{r=0}^k i_r = \sum_{r=0}^k j_r \leq b - 1 \right\}.$$

In this form, FC has all the $\mathbf{i}, \mathbf{j}, i$ and j local information available but it uses only the j and m (i.e., $b - \sum_{r=0}^k j_r$) parameters to decide on the acceptance of a packet. It turns out that the relation between FC, FS, BC and BS can be described by the lattice shown in Fig. 2, where an edge represents the \subset relation.

THEOREM 5. $FC \subset FS$.

Proof. If $[(\mathbf{i}, \mathbf{j}), (i, j)]$ is in FC then $j < b - \sum_{r=0}^k j_r$. So, for $0 \leq i \leq j$, we have $i < b - \sum_{r=0}^k j_r$. Therefore $[(\mathbf{i}, \mathbf{j}), (i, j)]$ is in FS, and $FC \subseteq FS$. Now consider a $(b = 2, k = 1)$ -network G . Let v be a node of G with the state (\mathbf{i}, \mathbf{j}) where $\mathbf{j} = \langle 1, 0 \rangle$ (i.e., a buffer of v contains a packet whose destination is v). Let p be a packet with a state (i, j) where $j = 1$ (i.e., the destination of p is a node adjacent to v). With FC the packet p cannot be generated or passed into v since $j = b - (j_0 + j_1)$, and therefore $[(\mathbf{i}, \langle 1, 0 \rangle), (i, 1)]$ is not in FC. With the controller FS the packet p can be accepted in the node v since both inequalities $0 < b - (j_0 + j_1)$ and $1 < b - j_1$ hold, so $[(\mathbf{i}, \langle 1, 0 \rangle), (i, 1)]$ is in FS. \square

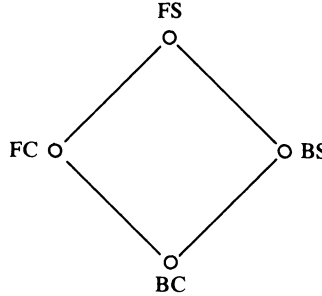


FIG. 2. Relations between controllers.

THEOREM 6. $BC \subset BS$.

Proof. If $[(\mathbf{i}, \mathbf{j}), (i, j)]$ is in BC then $i \geq \sum_{r=0}^k i_r - b + (k + 1)$. So, for $i \leq j \leq k$, we have $j \geq \sum_{r=0}^k i_r - b + (k + 1)$, and therefore $[(\mathbf{i}, \mathbf{j}), (i, j)] \in BS$. For $b = 2$ and $k = 1$, we have $[(\langle 0, 1 \rangle, \mathbf{j}), (0, j)]$ is in BS but not in BC. \square

THEOREM 7. $BC \subset FC$.

Proof. If $[(\mathbf{i}, \mathbf{j}), (i, j)] \in BC$ then $i \geq \sum_{r=0}^k i_r - b + (k + 1)$. Note that $i + j \leq k$ and therefore $j \leq k - [\sum_{r=0}^k i_r - b + (k + 1)]$. Since $\sum_{r=0}^k i_r = \sum_{r=0}^k j_r$, then $j \leq b - \sum_{r=0}^k j_r - 1$, or, equivalently, $j < b - \sum_{r=0}^k j_r$, and $[(\mathbf{i}, \mathbf{j}), (i, j)]$ is in FC. Suppose now that $b = 3, k = 2, \mathbf{i} = \langle 1, 0, 0 \rangle, \mathbf{j} = \langle 0, 1, 0 \rangle, i = 0$, and $j = 1$. In this case it is easy to check that $[(\mathbf{i}, \mathbf{j}), (i, j)]$ is in FC but not in BC. \square

The example given in the proof of Theorem 7 shows why a “forward” controller can be less restrictive than a “backward” controller. Informally, with a “backward” controller a packet may be “penalized” (i.e., its acceptance into a node may be subject to very restrictive conditions) just because the source of the packet is not far away, even

if the destination node for the packet is very near (in our example we have $i = 0$ and $j = 1$). The forward controllers follow the “intuitive” guess that the nearer a packet is to its destination (and to its eventual consumption and removal from the network), the “safer” it is for a node to accept such a packet without the danger of causing a deadlock. In the limit, it is clear that a node v can be allowed to fill *all* its buffers with packets whose destination is v without creating a deadlock, since the removal of these packets within finite time is guaranteed.⁴ Among the controllers we present here, only the forward controllers permit such node states.

THEOREM 8. $BS \subset FS$.

Proof. Suppose $[(\mathbf{i}, \mathbf{j}), (i, j)]$ is in BS. Note first that $i + j \leq k$, so $i \leq k - j$. Let $g = k - h$, then $0 \leq g \leq j$ implies $i \leq h \leq k$, and, since $[(\mathbf{i}, \mathbf{j}), (i, j)] \in BS$, this implies $h \geq \sum_{r=0}^h i_r - b + (k + 1)$. Since $g = k - h$, then $g \leq k - [\sum_{r=0}^k i_r - b + (k + 1)]$. Therefore we have

$$(*) \quad g < b - \sum_{r=0}^h i_r \quad \text{for all } g, \quad 0 \leq g \leq j.$$

We claim that $\sum_{r=0}^h i_r \geq \sum_{r=g}^k j_r$. Consider a node v with a state (\mathbf{i}, \mathbf{j}) . The left sum is the number of packets in v whose source node is at distance at most h away from v ; the right sum is the number of packets in v whose destination node is at least g away from v . We note that any packet whose destination is at least g away is at most $(k - g)$ away from its source node (no route is longer than k), and $k - g = h$. Then any packet counted in the right sum is also counted once in the left sum, and the claim is proved. With $(*)$ we now have

$$g < b - \sum_{r=g}^k j_r \quad \text{for all } g, \quad 0 \leq g \leq j,$$

and $[(\mathbf{i}, \mathbf{j}), (i, j)]$ is in FS. Then $BS \subseteq FS$, and the example given in the proof of Theorem 7 can be also used here to prove that $BS \subset FS$. \square

3.2. Node state reachability, useless elements. We now introduce the formal concept of the reachability of node states with respect to (α, β) DF uniform controllers. Let S be such a controller and let $(\alpha_0, \beta_0) \in S$. Suppose the acceptance of a β_0 -state packet into an α_0 -state node results in an α'_0 node state; we denote this state change by

$$\alpha_0 \xrightarrow[S]{\beta_0 \uparrow} \alpha'_0.$$

Similarly, if a β_0 -state packet leaves an α_0 -state node and this results in a new state α'_0 , then we denote this change by

$$\alpha_0 \xrightarrow[S]{\beta_0 \downarrow} \alpha'_0.$$

In both cases, if we are only interested in the state transition we just write $\alpha_0 \vdash_S \alpha'_0$. \vdash_S is a “state transition” relation in the set of states.

For example, if S is a (\mathbf{j}, j) DF forward-state uniform controller and $(\langle j_0, \dots, j_k \rangle, d) \in S$, then we can write

$$\langle j_0, \dots, j_d, \dots, j_k \rangle \xrightarrow[S]{d \uparrow} \langle j_0, \dots, j_d + 1, \dots, j_k \rangle;$$

⁴ In this case it is clear that the location of the source nodes for these packets is not relevant information.

provided that $j_d > 0$, we can also write

$$\langle j_0, \dots, j_d, \dots, j_k \rangle \xrightarrow[S]{d\downarrow} \langle j_0, \dots, j_d - 1, \dots, j_k \rangle$$

The transitive closure of the relation \vdash_S is denoted by \vdash_S^* . The composition of \vdash_S with itself n times is \vdash_S^n .

Let S be an (α, β) DF uniform controller and α_0 and α'_0 be node states. We say that α is *reachable from α_0 with respect to S* if and only if $\alpha_0 \vdash_S^* \alpha'_0$. If α_0 is the “empty node” state, then we simply say that α is *reachable with respect to S* and we denote this fact by $\vdash_S^* \alpha$. (The *empty node state* α_0 is the state description of a node without any packet stored in its buffers; with our parameters, α_0 is characterized by the values $\mathbf{i} = \mathbf{j} = \langle 0, 0, \dots, 0 \rangle$, $m = b$ and $n = 0$.) Note that this definition of state reachability is totally “syntactical.” In particular, the syntactical reachability of a node state α does not seem to imply the “network reachability” of this state, and the latter concept is probably of more interest to us. A node state α is *network-reachable with respect to a (b, k) DF uniform controller S* if there exist a (b, k) -network $G = (V, E)$ and a node $v \in V$ such that, the network being initially empty, there is a sequence of moves allowed by S resulting in a node state α in the node v . We denote this by $\models_S^* \alpha$.

LEMMA 1.

$$\xrightarrow[S]^* \alpha \text{ implies } \vdash_S^* \alpha.$$

Proof. Immediate. \square

We now show the converse is also true for a certain class of local DF uniform controllers. This class includes all the DF uniform controllers described in this paper.

LEMMA 2. *Let S be an (α, β) DF uniform controller where the packet state β is given by (any subset of) the distances i and j . Then*

$$\xrightarrow[S]^* \alpha \text{ is equivalent to } \models_S^* \alpha.$$

Proof. Let S be a (b, k) DF uniform controller as above, and suppose that $\vdash_S^* \alpha$. By the definition of \vdash_S^* there exists an integer n such that $\vdash_S^n \alpha$. Let $G = (V, E)$ be a $(k + 1)$ -node directed cycle $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_k \rightarrow v_0$ with b buffers in each node. The routes of G are the $k(k + 1)$ unique acyclic routes between any two distinct nodes in G (all the routes are of length less or equal to k). By induction on n , we now show $\vdash_S^n \alpha$ implies that, with G initially empty, we can reach a network state in which v_0 has the state α and all the other nodes of G are empty (implying $\models_S^* \alpha$). For $n = 0$, $\vdash_S^0 \alpha$ implies that α is the “empty node” state. Clearly, the initial empty state of G satisfies the induction statement, so $\models_S^* \alpha$. Suppose the induction hypothesis holds for n , and let $\vdash_S^{n+1} \alpha$. Then there exists a state γ such that $\vdash_S^n \gamma$ and $\gamma \vdash_S \alpha$. By the induction hypothesis, with G initially empty, we can first reach the network state in which v_0 has the state γ and all the other nodes of G are empty. Since $\gamma \vdash_S \alpha$, either $\gamma \vdash_S^{\beta\uparrow} \alpha$ or $\gamma \vdash_S^{\beta\downarrow} \alpha$, for some packet state β . If $\gamma \vdash_S^{\beta\uparrow} \alpha$, then $(\gamma, \beta) \in S$. By hypothesis, β is of the form (i, j) with $1 \leq i + j \leq k$. For any such (i, j) there is a (unique) pair v, w of source-destination nodes in G such that a packet generated in v with destination w passes through v_0 with an (i, j) local state. Since

- 1) all the nodes, except v_0 , are empty (they have b free buffers),
- 2) $(\gamma, \beta) \in S$ and
- 3) S is a (b, k) DF uniform controller,

it is permissible to generate such a packet in v and move it along its $v \dots \rightarrow w$ route up to its acceptance by v_0 , changing v_0 's state to α so, $\vdash_S^* \alpha$.

If $\gamma \vdash_S^{\beta \downarrow} \alpha$, then the state change results from the drop of a β -state packet from a γ -state node. Again, once we reach the state γ in v_0 we can drop the β -state packet by moving it along its route up to its destination where the packet is consumed (in this route, all the nodes except v_0 are empty). In both cases, a packet cannot be blocked along its route; this would contradict our assumption that S is a (b, k) DF controller. \square

Let S be a (b, k) DF uniform controller, A tuple (α, β) of S is said to be *useless* if one or more of the following conditions holds.

1. The packet state β implies a route of length d which is not in the $[0, k]$ range.
2. The node state α implies a current free buffer capacity c which is not in the $[1, b]$ range.
3. The node state α is not reachable with respect to S .

From Lemma 1, it is now clear that $S' = S - \{(\alpha, \beta)\}$ is equivalent to S in the sense that for any node state α we have $\vdash_{S'}^* \alpha$ if and only if $\vdash_S^* \alpha$. Therefore, the useless tuples can be removed from S without altering the controller. From now on, we assume that all the DF uniform controllers are free of useless elements.

3.3. Closure properties of forward and backward-count controllers. To prove the optimality of FC and of BC we can use some interesting closure properties of DF forward and backward-count controllers. The first property states that if, with such a controller, a node with m free buffers accepts a packet, then with $m + 1$ free buffers it can still safely accept it.

THEOREM 9. *Let S be a DF forward-count uniform controller and let $(m, j) \in S$. Then $S' = S \cup \{(m + 1, j)\}$ is also a DF uniform controller.*

Proof. S is free of useless elements, so $1 \leq m \leq b$ and $\vdash_S^* m$. If $m = b$ then $(m + 1, j)$ is a useless element which can be omitted from S' , and the theorem is obviously true. If $m < b$, then $\vdash_S^* m$ implies the existence of an element $(m + 1, j_0) \in S$ such that

$$b \vdash_S^* (m + 1) \vdash_S^{i_0 \uparrow} m \quad \text{with } 0 \leq j_0 \leq k.$$

Suppose S' is not deadlock-free for a (b, k) -network G . We append at each node v of G a new directed k -cycle with routes as in Lemma 2. This results in a new (b, k) -network G' with a reachable deadlock when the controller S' is used. We now show how we can reach the same deadlock state in G' when using the DF controller S . In G' , every move allowed by S' is also allowed by S with the exception of packets accepted in certain nodes because of the tuple $(m + 1, j) \in S'$. Let p be a j -state packet allowed by S' to enter a node v with $m + 1$ free buffers. Using S we can simulate this move in the following way.

1. In the k -cycle attached to v we generate and pass to v a packet q whose state is j_0 at v . Since $(m + 1, j_0) \in S$, this packet is granted access to v ; there are now m free buffers left in v .
2. Since $(m, j) \in S$, the packet p is now allowed by S to enter v ; $(m - 1)$ free buffers are left in v .
3. Cause the packet q to traverse the k -cycle and be consumed; the packet p remains in the node v , which now has m free buffers.

The generation, moves and consumption of q along the k -cycle are permissible by S , since S is a (b, k) DF uniform controller. It is now clear that S is not DF for the (b, k) -network G' , contradicting our hypothesis. \square

THEOREM 10. *Let S be a DF backward-count uniform controller and let $(n, i) \in S$. Then $S' = S \cup \{(n - 1, i)\}$ is also a DF uniform controller.*

Proof. Similar to the proof of Theorem 9. \square

The second closure property of these controllers emphasizes the fact that a packet must gather “privileges” as it travels and gets closer to its destination; a “privilege” is the right to be passed to a vertex with a small number of free buffers.

THEOREM 11. *Let S be a DF forward-count uniform controller. Then*

$$S' = \{(m, j') \mid \text{for some } j, 0 \leq j' \leq j, (m, j) \in S\}$$

is also a DF uniform controller.

Proof. Suppose S' is not deadlock-free for a (b, k) -network G . We append at each node v of G a new directed k -cycle. In this new (b, k) -network G' we can reach, with S' , the same deadlock state as in G in the following way. Let v be a node of G , let α_v be the state of v when the deadlock state is reached in G . Then $\models_S^* \alpha_v$, and we can use packets generated and passed along the k -cycle appended to v , to reach the state α_v in v . This was shown in the proof of Lemma 2. We also alter the destination of all the packets remaining in v when the state α_v is reached; these destinations are now the ones packets in v have when the deadlock state is reached in G .⁵ This simulation can be done independently with all the nodes v of G , and we reach using S' the same deadlock state as in G . We show how we can reach this deadlock in a slightly modified (b, k) -network G' , when using the DF controller S . Every move allowed by S' in G' is also allowed by S with the exception of packets accepted in certain nodes because of tuples of the form $(m, j') \in S'$, where $(m, j) \in S$ and $0 \leq j' < j$. Let p be an j' -state packet allowed by S' to enter a m -state node of G . Using S we can simulate this move in the following way. We add a $(j - j')$ -long route segment at the end of p 's route. If p 's destination is in the k -cycle, then a simple redefinition of p 's destination along this cycle is sufficient; otherwise we must append a *new* route segment to p 's destination in G . The state of p is now j , and, since $(m, j) \in S$, v can now accept p according to the controller S . Furthermore, if p is deadlocked in G' when S' is used, then p is also deadlocked in the modified G' when S is used. In fact, suppose p is deadlocked in G' when S' is used; then $(m_0, j - 1) \notin S'$, where m_0 is the state of the next node in p 's route. But this implies $(m_0, j - 1) \notin S$ and p is also deadlocked in the modified G' when S is used. It is now clear that S is not a DF uniform controller, contradicting our hypothesis. \square

We now state the corresponding theorem for backward-count controllers.

THEOREM 12. *Let S be a DF backward-count uniform controller. Then*

$$S' = \{(n, i') \mid \text{for some } i, i \leq i' \leq k, (n, i) \in S\}$$

is also a DF controller.

Proof. Similar to the proof of Theorem 11. \square

A DF forward-count uniform controller S has closure S^* defined as the set

$$\{(m', j') \mid \text{for some } (m, j) \in S, m \leq m' \leq b \text{ and } 0 \leq j' \leq j\}.$$

COROLLARY 1. *The closure S^* of a DF forward-count uniform controller S is also a DF uniform controller.*

Proof. Immediate consequence of Theorems 9 and 11. \square

Similarly, a DF backward-count controller defined by a set S has the following closure S^* :

$$\{(n', i') \mid \text{for some } (n, i) \in S, 0 \leq n' \leq n \text{ and } i \leq i' \leq k\}.$$

COROLLARY 2. *The closure S^* of a DF backward-count uniform controller S is also a DF uniform controller.*

Proof. Immediate from Theorems 10 and 12. \square

⁵ Note that while routes may appear to be determined dynamically, once we carry out the simulation to determine the correct routes, we can fix the routes initially.

3.4. Optimality of FC and BC. We first show that there are no forward or backward-count DF uniform controllers for networks G whose node buffer capacity is less than or equal to the length k of the longest route taken by a packet in G . We also prove that, for (b, k) -networks, where $b > k$, the optimal forward and backward-count controllers are respectively FC and BC.

THEOREM 13.

- (a) *There are no (b, k) DF forward-count uniform controllers for $b \leq k$.*
- (b) *For $b > k$, any (b, k) DF forward-count uniform controller is a subset of FC.*

Proof. Suppose S is a (b, k) DF forward-count uniform controller contradicting (a) or (b). We claim that for some (m, j) in S we have $m \leq j$. In fact, if S contradicts (a) then $b \leq k$, and (b, k) must be in S . If S contradicts (b) then for some (m, j) in S , (m, j) is not in FC. Either $j \notin [0, k]$, or $m \notin [1, b]$, or $m \leq j$. We can exclude the first two possibilities (S is free of useless elements) and the claim is proved. Now let S^* be the closure of S and define j_0 as

$$j_0 = \min \{j \mid \text{for some } m, (m, j) \in S^* \text{ and } m \leq j\}.$$

If $j_0 = 0$ then $m \leq 0$ and (m, j_0) cannot be in S^* (S^* is free of useless elements). Then $j_0 \geq 1$ and we can construct a directed cycle of $j_0 + 1$ vertices with b buffers in each vertex. Since $(m, j_0) \in S^*$ then

$$(m + 1, j_0), (m + 2, j_0), \dots, (b, j_0) \in S^*$$

and we can successively generate, in each vertex of the cycle, $b - m + 1$ packets with routes of length j_0 around the cycle. Then, the only permissible moves require $(m - 1, j_0 - 1) \in S^*$, and this is not possible by the minimality of j_0 . The network is now in a deadlock state, contradicting the fact that, according to Corollary 1, S^* is a DF uniform controller. \square

The theorem's "intuitive" meaning is that if a controller uses the same local information as FC and it permits a move which is not allowed in FC, then this controller is not deadlock-free.

We can similarly prove the following theorem about backward-count controllers:

THEOREM 14.

- (a) *There are no (b, k) DF backward-count uniform controllers for $b \leq k$.*
- (b) *For $b > k$, any (b, k) DF backward-count uniform controller is a subset of BC.*

Proof. Similar to the proof of Theorem 13. \square

3.5. Optimality of FS and BS. As before, we could first define and prove some closure properties of forward and backward-state controllers and then use the results to prove the optimality of FS and BS. These closure properties and their proofs being somewhat more complicated for state controllers than for count controllers, we use a more direct approach instead.

THEOREM 15.

- (a) *There are no (b, k) DF forward-state uniform controllers for $b \leq k$.*
- (b) *For $b > k$, any (b, k) DF forward-state uniform controller is a subset of FS.*

Proof. Suppose S is a (b, k) DF forward-state uniform controller contradicting (a) or (b). We claim that for some tuple $(\mathbf{j} = \langle j_0, j_1, \dots, j_k \rangle, j)$ in S we have

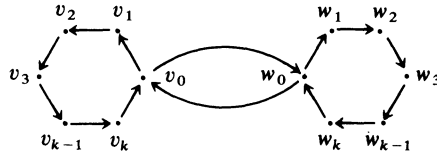
$$(*) \quad b - \sum_{r=i_0}^k j_r \leq i_0 \quad \text{for some } i_0, \quad 0 \leq i_0 \leq j.$$

In fact, if S is a contradiction to (a), then $b \leq k$ and the tuple $(\mathbf{j} = \langle 0, 0, \dots, 0 \rangle, k)$ must be in S , and $(*)$ is satisfied with $i_0 = k$. If S contradicts (b) then, using the definition of FS, it is easy to check that the condition $(*)$ must be satisfied for some tuple (\mathbf{j}, j) in S . Note

that $i_0 > 0$ or else S would contain a useless element. We now defined d as

$$d = \min \{j \mid \text{for some } \mathbf{j} = \langle j_0, j_1, \dots, j_k \rangle \text{ the tuple } (\mathbf{j}, j) \text{ is in } S \text{ and the condition } (*) \text{ for the } (\mathbf{j}, j) \text{ tuple is satisfied}\}.$$

If $d = 0$ then $b - \sum_{r=0}^k j_r \leq 0$. Such a \mathbf{j} -state node has no available free buffers. Therefore $(\mathbf{j}, 0)$ is not in S . We can now assume that $d \geq 1$, and we consider the network G illustrated in Fig. 3. All the vertices have b buffers and packet routes have a maximal length of k (the routes are specified in the figure).



Packet routes	Length l
$v_p \dots \rightarrow v_q$ $w_p \dots \rightarrow w_q$ where $0 \leq p \neq q \leq k$	$1 \leq l = p - q \leq k$
$v_0 \dots \rightarrow w_q$ $w_0 \dots \rightarrow v_q$ where $0 \leq q \leq k - 1$	$1 \leq l = q + 1 \leq k$

FIG. 3. The network G in the proof of Theorem 3.

We now show how we can reach a deadlock state using the controller S in this (b, k) -network G . Since $(\mathbf{j}, d) \in S$ and S is free of useless elements then $\vdash_S^* \mathbf{j}$. By Lemma 2, this implies $\models_S^* \mathbf{j}$ and the state \mathbf{j} can actually be reached independently in both v_0 and w_0 by generating and moving packets along each of the v_0 and w_0 k -cycles. Moreover, when the state \mathbf{j} is reached in v_0 and w_0 , all the other nodes of the network are empty (see the proof of Lemma 2). In what follows, any move taken on the v_0 side is also taken, symmetrically, on the w_0 side.

1. We first reach in the node v_0 the state $\mathbf{j} = \langle j_0, j_1, \dots, j_k \rangle$ as we just described above.

2. Then we generate in v_0 a packet whose route is $v_0 \dots \rightarrow w_{d-1}$. Since (\mathbf{j}, d) is in S , this generation is permissible. The state of v_0 is now $\mathbf{j}^{(1)} = \langle j_0^{(1)}, j_1^{(1)}, \dots, j_k^{(1)} \rangle$ with

$$j_r^{(1)} = j_r \quad \text{for } r \neq d,$$

$$j_d^{(1)} = j_d + 1 \quad \text{for } r = d.$$

3. We successively exit and consume along the $v_0 k$ -cycle all the $0, 1, \dots, (i_0 - 1)$ -state packets which are in v_0 . The state of v_0 is now $\mathbf{j}^{(2)} = \langle j_0^{(2)}, j_1^{(2)}, \dots, j_k^{(2)} \rangle$ with

$$j_r^{(2)} = 0 \quad \text{for } 0 \leq r \leq i_0 - 1,$$

$$j_r^{(2)} = j_r^{(1)} \quad \text{for } i_0 \leq r \leq k;$$

that is,

$$j_r^{(2)} = 0 \quad \text{for } 0 \leq r \leq i_0 - 1,$$

$$j_r^{(2)} = j_r \quad \text{for } i_0 \leq r \leq k \text{ and } r \neq d,$$

$$j_d^{(2)} = j_d + 1 \quad \text{for } i_0 \leq r \leq k \text{ and } r = d.$$

The destination of the remaining packets in v_0 is changed so that packets with state q , for $i_0 \leq q \leq k$, are now directed to the $v_0 \cdots \rightarrow w_{q-1}$ route (their new destination is w_{q-1}).

4. Finally, we successively try to generate in v_0 $d, (d+1), \dots, k$ -state packets whose route are $v_0 \cdots \rightarrow w_{q-1}$ for $d \leq q \leq k$. We stop when we reach in v_0 a state $\mathbf{j}^{(3)}$ such that

$$(**) \quad (\mathbf{j}^{(3)}, q) \notin S \quad \text{for } d \leq q \leq k$$

and these packet generations are not permissible anymore. The state $\mathbf{j}^{(3)} = \langle j_0^{(3)}, j_1^{(3)}, \dots, j_k^{(3)} \rangle$ is such that

$$\begin{aligned} j_r^{(3)} &= j_r^{(2)} && \text{for } 0 \leq r \leq d-1, \\ j_r^{(3)} &\geq j_r^{(2)} && \text{for } d \leq r \leq k; \end{aligned}$$

that is,

$$\begin{aligned} j_r^{(3)} &= 0 && \text{for } 0 \leq r \leq i_0-1, \\ j_r^{(3)} &\geq j_r && \text{for } i_0 \leq r \leq k \text{ and } r \neq d, \\ j_d^{(3)} &\geq j_d + 1 && \text{for } i_0 \leq r \leq k \text{ and } r = d. \end{aligned}$$

Since $j_d^{(3)} \geq 1$, there is at least one packet in v_0 . Since $i_0 \geq 1$, then $j_0^{(3)} = 0$ and all the packets in v_0 are directed toward w_0 . It is now clear that G is in a deadlock state if and only if

$$(***) \quad (\mathbf{j}^{(3)}, q) \notin S \quad \text{for all } q, \quad i_0-1 \leq q \leq k-1.$$

Since $(**)$ holds, it suffices to show that

$$(\mathbf{j}^{(3)}, q) \notin S \quad \text{for all } q, \quad i_0-1 \leq q \leq d-1.$$

The proof is the following. Let q be in the interval $i_0-1 \leq q \leq d-1$. We have

$$b - \sum_{r=i_0-1}^k j_r^{(3)} = b - (j_{i_0-1}^{(3)} + j_{i_0}^{(3)} + \dots + j_k^{(3)}).$$

Since $j_{i_0-1}^{(3)} = 0, j_r^{(3)} \geq j_r$ and $j_d^{(3)} \geq j_d + 1$, we have

$$b - \sum_{r=i_0-1}^k j_r^{(3)} \leq b - (j_{i_0} + \dots + (j_d + 1) + \dots + j_k),$$

so

$$b - \sum_{r=i_0-1}^k j_r^{(3)} \leq \left(b - \sum_{r=i_0}^k j_r \right) - 1;$$

using the $(*)$ inequality, we have

$$b - \sum_{r=i_0-1}^k j_r^{(3)} \leq i_0 - 1.$$

Then for all $q, 0 \leq i_0-1 \leq q \leq d-1$, the condition $(*)$ for the tuple $(\mathbf{j}^{(3)}, q)$ is satisfied; but $q < d$, so, by the minimality of d , $(\mathbf{j}^{(3)}, q)$ cannot be in S . Therefore $(***)$ is proved, and the (b, k) -network G is in a deadlock state, a contradiction to the fact that S is a (b, k) DF uniform controller. \square

We can also prove a similar theorem for the controller BS.

THEOREM 16.

- (a) *There are no (b, k) DF backward-state uniform controllers for $b \leq k$.*
 (b) *For $b > k$, any (b, k) DF backward-state uniform controller is a subset of BS.*
Proof. Similar to the proof of Theorem 15. \square

4. Computational efficiency of the FC, BC, FS and BS controllers. It is clear that DF controllers should be as simple and efficient as possible to prevent overhead in the processors handling the traffic of packets. We consider the computational complexity of the count and state controllers when executing the following two tasks.

(1) *Node state update.* To update a node state when a state change is caused by the arrival or departure of a packet (“bookkeeping”).

(2) *Membership decision.* To determine whether an arbitrary (α, β) tuple is in the set defined by a controller.⁶

With the BC, BS, FC and FS controllers it is clear that only one addition (or subtraction) is needed for node state updates. With the BC and FC count controllers, no more than five comparisons are required for each membership decision. With the BS and FS state controllers, the same task requires at most $k + 1$ comparisons and $k + 1$ additions, i.e., a total of $2(k + 1)$ operations. However, the efficiency of state controllers can usually be improved in the following way. Let n be the number of packets in the buffers of a node. From the definition of the FS and BS state controllers, it is easy to verify that if n is less than $b - k$ then any packet can be accepted in the node, regardless of packet and node states.⁷ Therefore nodes can store and update the value of n , and as long as n is less than $b - k$ they can accept any packet without further checking. In this case, two additions (or subtractions) are needed for each node state update, and only one comparison (and no addition) is required for membership decisions. When $n \geq b - k$, a node is considered to be *congested*, the danger of a deadlock involving this node rises, and membership decisions require up to $2(k + 1)$ operations. In actual network implementations k is usually much smaller than the number of buffers provided to each node; therefore it is hoped that node congestion will generally be avoided, and very efficient one-comparison membership decisions will usually be made.

5. Individual controllers and minimal buffer requirements. Let G be a network and k be the length of the longest route in G . If we want to use any one of the DF controllers we defined it is usually not necessary to provide every node of G with b buffers such that $b > k$. In fact, FC, BC, FS and BS can each be “tailored” to the particular characteristics of any specific node in G , and with these *individual controllers* minimal buffer requirements for the nodes may be significantly smaller than k . We show how to derive an individual forward-count controller from FC, and very similar derivations can be made from FS, BC and BD. Let v_i be a node in the network G , and let $d_0 < d_1 < \dots < d_{k_i}$ ($k_i \leq k$) be the ordered list of all the different distances from v_i . If we assume this list is known to the node, then v_i can define the monotonic bijection $\phi : \phi(d_j) = j$, for $0 \leq j \leq k_i$, between the set of distances $\{d_j | 0 \leq j \leq k_i\}$ and a set $\{j | 0 \leq j \leq k_i\}$ called the set of *internal distance representations*. Since ϕ is a monotonic mapping, then $\phi(d) \geq \phi(d')$ if and only if $d \geq d'$, for any two distances d and d' in the set of distances for the node v_i . If we provide the node v_i with $b_i > k_i$ buffers,⁸ then an

⁶ Obviously, a membership decision is needed each time a packet asks access to a node.

⁷ This property holds also for the FC and BC count controllers. For individual controllers the condition becomes $n < b_i - k_i$ for any node v_i ; the definitions of “individual controllers”, of b_i , and k_i are given in the next section.

⁸ Note that k_i can be significantly smaller than k .

Individual Forward-Count (or IFC) controller for v_i can be derived from the $FC(b_i, k_i)$ controller where distances are substituted with corresponding internal distance representations. The IFC controller for v_i is the set

$$\{(m, j) \mid \phi(j) < m \text{ and } 0 \leq \phi(j) \leq k_i \text{ and } 1 \leq m \leq b_i\}.$$

THEOREM 17. *The IFC controller described above is a DF uniform controller.*

Proof. Thanks to the monotonicity of the bijection ϕ which preserves all the relations needed, this proof is very similar to the proof of Theorem 1. Suppose we reach a deadlock in a network G . Let p_1 be a blocked packet; p_1 is in the node v_1 , and v_1 is distance d_1 -away from p_1 's destination node (note that $d_1 \geq 1$, else p_1 is consumed in v_1). Let v_2 be the next node in p_1 's route; v_2 is provided with b_2 buffers and the cardinality of the distance set of v_2 is $(k_2 + 1) \leq b_2$. Let ϕ be the distance bijection of the node v_2 and m_2 be the number of free buffers in v_2 . Surely $\phi(d_1 - 1) \geq m_2$, else p_1 could be passed to v_2 ; since $b_2 > k_2$ and $k_2 \geq \phi(d_1 - 1)$, then $b_2 > m_2$ and there is at least one blocked packet in v_2 . Let d_2 be the distance from v_2 to the destination of the last packet p_2 to enter the node v_2 . Since p_2 was accepted by v_2 , then $\phi(d_2) < m_2 + 1$; combined with $\phi(d_1 - 1) \geq m_2$ we have $\phi(d_1 - 1) \geq \phi(d_2)$. Since ϕ is monotonic we have $d_1 - 1 \geq d_2$ or, equivalently, $d_2 < d_1$. By reductio ad absurdum we can show that there is a deadlocked packet with zero moves to go; but such a packet would be consumed. \square

With the help of the monotonic bijection ϕ between distances and their internal representations, individual forward-state, backward-count and backward-state DF uniform controllers can be derived similarly from FS, BC and BS, respectively. The proofs follow closely the proofs of Theorems 2, 3 and 4.

6. Adaptive routing. The local information needed by the controllers we presented can be easily obtained and updated under the assumption of fixed routing for packets. We now show how to incorporate adaptive routing procedures [KL] in DF local controllers. With adaptive routing the destination node for any packet p is included in the packet, with each processor in the network responsible for dynamically deducing the next vertex to which the packet is to be passed according to such factors as destination address, channel availability, channel and node congestion, etc.

The conversion of backward controllers to adaptive routing is straightforward. Consider a packet p asking access to a node v . The distance i from the source of p to the node v , along the adaptive route taken so far by the packet, is readily available, and it does not depend on the remaining segment of p 's route. As far as the node v is concerned, the fact that p followed a fixed or a dynamically computed route does not make any difference, and no changes are necessary in the definition of the parameters i and \mathbf{i} used in the BC and BS controllers. The only difficulty created by adaptive routing involves the value of k , which is defined as the length of the longest route taken by a packet in the network. With an adaptive routing procedure we may not know the exact value of k , but with BC and BS our only concern is that the number of buffers in each node must be greater than k .⁹ In actual network implementations this condition will usually be satisfied if reasonable adaptive routing procedures that guarantee bounded finite-length routes are used (several descriptions of adaptive routing algorithms resulting in acyclic routes can be found in the literature [SE1], [SMG], [MES]).

It is slightly more complicated to integrate adaptive routing with forward controllers. Between any two nodes v and w in a network a fixed *emergency route*, i.e. a

⁹ With the individual controllers discussed in the previous section, in each node v_i we should have at least $k_i + 1$ buffers, where $k_i + 1$ is the cardinality of the distance set of v_i .

fixed path in the network connecting the two nodes, is defined. It will become clear later that it is better for the emergency routes to be as short as possible, and it is convenient to think of the emergency route between two nodes in a network G as a shortest path between these two nodes in G .¹⁰ Packets do not necessarily follow the emergency routes when traversing the network. These routes are used occasionally, only when the danger of a deadlock seems imminent and some action must be taken to prevent it. Every processor v keeps a table which lists the following information for each possible destination node w ,

1. the length of the $v \cdots \rightarrow w$ emergency route and
2. the node that follows v in this emergency route.

Let p be a packet with destination w waiting to access a node v . With adaptive routing procedures, the *state of p relative to the node v* is given by the parameter j , where j is redefined to be the length of the $v \cdots \rightarrow w$ emergency route. The length of the longest emergency route in the network is denoted by k , and the *state of a node v* is given by the vector $\mathbf{j} = \langle j_0, j_1, \dots, j_k \rangle$, where j_i , for $0 \leq i \leq k$, is the number of packets in the node v whose emergency route from v to destination is of length i . The definition of the FC and FS controllers is not changed, and their use as DF uniform controllers with adaptive routing is described below. A packet p , with destination w , residing in a node v , can be dynamically directed, by adaptive routing, to any adjacent node u ; the packet p is accepted or rejected by the node u according to the rules of the forward controller chosen (with respect to the emergency routes out of u), exactly as with fixed routing¹¹; but if p does not leave the node v after a certain predefined time interval t_0 , then the danger of a deadlock involving the packet p appears to be imminent, and the packet is directed toward the $v \cdots \rightarrow w$ emergency route. It is now clear that any deadlock would involve packets deadlocked along emergency routes, but such deadlocks are surely prevented by a DF uniform controller applied relative to these routes.

We may note that several obvious generalizations of this scheme can be made. For example, a node v may have a *set* of emergency routes for each destination, and a packet p that is directed to its emergency route may still be passed to any node which accepts p . Some of these generalizations may improve the throughput of particular network implementations.

7. Controllers for particular networks. We might ask, given a number of vertices N , what is the least number of buffers per vertex for which we can select a route for any two nodes and select a deadlock-free controller for this network with these routes. The next theorems give an answer to this question.

THEOREM 18. *Let C be a local controller for a one-buffer network $G = (V, E)$ such that for any vertex v there is a vertex w such that (v, w) is an edge. Then either C allows deadlock or does not permit packets with arbitrary acyclic routes to be generated.*

Proof. C either forbids or allows a packet with a route of length one to be generated into an empty buffer. In the first case, no packet with a route of length one can be generated. In the second case, we may generate into each buffer a packet destined for some other vertex, producing a deadlock. \square

THEOREM 19. *The cycle of N vertices with two buffers per vertex has a local controller that prevents deadlock.*

¹⁰ However this is not a necessary assumption, and any path connecting the two nodes in G can be chosen for an emergency route as well.

¹¹ It should be clear by now that shorter emergency routes correspond to less restrictive conditions for the admission of packets into nodes.

Proof. We discussed the working of this controller in § 1.2. Note that this controller, while local, is not in any of the classes we have studied. \square

A binary tree network is another two-buffer network that has a limited degree, yet for which routes and a controller can be devised to prevent deadlock [MS]. However, these networks have the property that on the order of N^2 routes pass through some of the vertices, which may well be more traffic than one processor can handle. If we wish to limit to d the degree of any vertex in the network, to avoid excessive connections to any processor, then some route must be at least $\log_d N$ long. The problem of finding graphs with limited degree and small *diameter* (maximum shortest path length between any two vertices) has been studied by [AK], [AI], [FR], [STOR], [TS2].

If the maximum path length is k , we can use $b > k$ buffers and the controller FS(b, k) to achieve our ends. For example, the perfect shuffle interconnection of [STONE] allows us to connect N vertices $0, 1, \dots, N-1$, where N is a power of 2, with edges from i to $i+1$ and back, for even i , and an edge from i to $2i \bmod N$. Then there is a path from i to j of length at most $2 \log_2 N - 1$ for any i and j . Therefore, $2 \log_2 N - 1$ buffers suffice to give us total interconnection through fixed routes, with vertices of degree 3 and no more than $2N \log_2 N$ routes passing through any vertex.

The *buffer graphs* technique described in [MS] can also be applied to particular networks to derive deadlock-free controllers. However, there are very few schemes for constructing buffer graphs taking advantage of the specific topology of a particular network. Schemes for tree and mesh networks are given in [MS].

As a final remark, we note that the deadlock-free controllers presented here do not prevent another kind of network failure: livelock, i.e., a situation in which unfair scheduling of packets prevents one or more packets from reaching their destination. A deadlock- and livelock-free controller is described in [T]; it guarantees that every packet reaches its destination within a finite amount of time from the moment of its creation.

REFERENCES

- [AK] S. B. AKERS, *On the construction of (d, k) graphs*, IEEE Trans. Elec. Comp., 14 (1965), 488.
- [AL] B. W. ARDEN AND H. LEE, *A multitree structured network*, Proc. IEEE COMPCON 78, Washington, DC, September 1978, pp. 201–210.
- [FR] H. FRIEDMAN, *A Design for (d, k) graphs*, IEEE Trans. Elec. Comp., 16 (1966), pp. 253–254.
- [G] K. D. GUNTHER, *Prevention of buffer deadlocks in packet switching networks*, IFIP-IIASA Workshop on Data Communications, Ladenberg, Austria, September 1975, g.15-g.19.
- [GD] M. G. GOUDA, *Protocol machines*, Ph.D. thesis, Dept. of Computer Science, Univ. of Waterloo, Waterloo, Ontario, Canada, 1977.
- [GHKP] A. GIESSLER, J. HAENLE, A. KOENIG AND E. PADÉ, *Free buffer allocation—an investigation by simulation*, Computer Networks, 2 (1978), pp. 191–208.
- [KL] L. KLEINROCK, *Queueing Systems Vol. II: Computer Applications*, John Wiley, New York 1976.
- [KN] I. KORN, *On (d, k) graphs*, IEEE Trans. Elec. Comp., 16 (1967), p. 90.
- [KS] S. R. KIMBLETON AND G. M. SCHNEIDER, *Computer communication networks: approaches, objectives, and performance considerations*, Comp. Surv., 7 (1975), pp.129–172.
- [MES] P. M. MERLIN AND A. SEGALL, *A failsafe distributed routing protocol*, IEEE Trans. Comm., 27 (1979), pp. 1280–1287.
- [MS] P. M. MERLIN AND P. J. SCHWEITZER, *Deadlock-avoidance in store-and-forward networks. I: Store-and-Forward Deadlock*, IBM RC 6624 Yorktown Hts., NY.
- [MT] D. E. MORGAN AND D. J. TAYLOR, *Computer network reliability and availability: the state of the art*, Conference Record, 14th IEEE International Conference on Communications, Toronto, June, 1978, Vol. 1, pp. 3.3.1–3.3.5.

- [RH] E. RAUBOLD AND J. HAENLE, *A method of deadlock-free resource allocation and flow control in packet networks*, Proc. ICCS 76, Toronto, Ont., Canada, Aug., 1976, pp. 483–487.
- [SE1] A. SEGALL, *Optimal distributed routing in data-communication networks*, Proc. 10th IEEE Conference Electrical and Electronics Engineers in Israel, Tel-Aviv, Israel, Oct., 1977, pp. 4–6.
- [SMG] A. SEGAL, P. M. MERLIN AND R. G. GALLAGER, *A recoverable protocol for loop-free distributed routing*, Conference Record, 14th IEEE International Conference on Communications, Toronto, June, 1978, Vol. 1, pp. 3.5.1–3.5.5.
- [STONE] H. S. STONE, *Parallel processing with the perfect shuffle*, IEEE Trans. Comput., 20 (1971), pp. 153–161.
- [STOR] R. M. STORWICK, *Improved construction techniques for (d, k) graphs*, IEEE Trans. Comput., 19 (1970), pp. 1214–1216.
- [TS1] S. TOUEG AND K. STEIGLITZ, *Some complexity results in the design of deadlock-free packet switching networks*, this Journal 10 (1981), to appear.
- [TS2] ———, *The design of small-diameter networks by local search*, IEEE Trans. Comput., C-28 (1979), pp. 537–542.
- [T] S. TOUEG, *Deadlock- and livelock-free packet switching networks*, Proc. 12th ACM Symposium on Theory of Computing, Los Angeles, California, April 1980, pp. 94–99.

**CORRIGENDUM:
SOUNDNESS AND COMPLETENESS OF AN AXIOM SYSTEM
FOR PROGRAM VERIFICATION***

STEPHEN A. COOK†

K. R. Apt pointed out to me that Theorem 3 (completeness) is technically false, because of a problem with initializing newly declared variables. For example, the formula

$$\text{true } \{\text{begin begin new } x; x := 1 \text{ end; begin new } x; y := x \text{ end end}\} y = 1$$

is valid according to the semantics given (because the second declaration of x assigns the same register to x as the first), but it is not provable in \mathcal{H} .

Perhaps the simplest way to fix this is to require all newly declared variables to be initialized to some distinguished value $0 \in D$. This would involve changing the first case (that of variable declaration) in the definition of $\text{Comp}(A, s, \delta, \pi)$ on p. 74, so that the computation proceeds with a new state s' . Here s' is the same as s except for $s'(X_{k+1}) = 0$. To make \mathcal{H} complete we would slightly modify Rule 1 (Rule of variable declarations) of the system \mathcal{H} to read

$$\frac{x = 0 \ \& \ P \frac{y}{x} \{\text{begin } D^*; A^* \text{ end}\} Q \frac{y}{x}}{P \{\text{begin new } x; D^*; A^* \text{ end}\} Q}.$$

A second possible fix, suggested in Apt [1], requires no changes in the proof system \mathcal{H} , but changes the semantics so that \mathcal{H} becomes complete. The idea is that each newly declared variable is assigned a register that has never been used before. A state s would be redefined so that it assigns a member of $D \times \{0, 1\}$ to each register X_k instead of simply a member of the domain D . The second component of $s(X_k)$ indicates whether X_k has been assigned previously. We would only consider pairs (s, δ) in the definition of $\text{Comp}(A, s, \delta, \pi)$, $P(s, \delta)$, etc. such that $(s(\delta(x)))_2 = 1$ for each variable x in the domain of δ , indicating that register $\delta(X)$ has been assigned. The first case in the definition of Comp would be changed so that $\delta'(x) = X_k$, where X_k is the first register for which $(s(X_k))_2 = 0$. Also the computation would continue in a new state s' such that $(s'(X_k))_2 = 1$. The other cases of Comp would be unchanged except for minor editing.

REFERENCE

- [1] K. R. APT, *Ten years of Hoare's logic, a survey*, Proceedings of the 5th Scandinavian Logic Symposium, Aalborg University Press, Aalborg, Denmark, 1979, pp. 1-44.

* This Journal, 7 (1978), pp. 70-90.

† Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A7.

EXEGESIS OF SELF-ORGANIZING LINEAR SEARCH*

GASTON H. GONNET†, J. IAN MUNRO‡ AND HENDRA SUWANDA‡

Abstract. We consider techniques for self-organizing linear search, examining the behavior of methods under arbitrary and specific probability distributions.

The notion of moving an element forward after it has been accessed k times in a row is introduced. One implementation performs the transformation after any k identical requests. A second essentially groups requests into batches of k , and performs the action only if all requests of a batch are the same. Adopting as the transformation, the move to front heuristic, the second approach is shown in general to be superior. We show that the batched approach, with $k = 2$, leads to an average search time no greater than 1.21... times that of the optimal ordering. For the more direct approach, a ratio of 1.36... is shown under the same constraints.

The simple move to front heuristic (i.e., $k = 1$) is also examined. It is shown that for a particular distribution this scheme can lead to an average number of probes $\pi/2$ times that of the optimal order. Within an interesting class of distributions, this is shown to be the worst average behavior.

Key words. self organizing files, linear search, move to front, transpose rule, complexity analysis, asymptotic analysis, heuristics

1. Introduction and preliminary results. Suppose we have a file which must be searched sequentially, and that the probabilities of accessing the various elements are fixed and independent, but unfortunately, unknown. The obvious approach to the problem of finding a good ordering for the list is to count requests and dynamically keep the file in decreasing order by request count. Given enough time, by the law of large numbers we will clearly arrive at the best possible ordering. The cost of maintaining such counters is very often prohibitive, and so heuristics for rearranging the file without the use of extra ordering information have been studied (Bitner [3], Hendricks [8], Knuth [9], McCabe [10], Rivest [11], Tanenbaum [12]). The basic approach of such methods is to consider a set of n permutations (for a list of length n), π_1, \dots, π_n . If the element currently in position i is requested, then π_i is applied to the list. We will call such a technique a *memory-free* self-organizing heuristic. The most obvious such heuristic is the *transposition* of the requested element with the one in front of it. Another, more drastic approach is to *move* the requested element to the *front* of the list, and effectively slide the others back one position. This method, while asymptotically not as effective as the transposition rule, has proven more amenable to analysis.

Our contribution is, first, to continue the study of such memory-free heuristics, comparing their behavior with that of an optimally ordered list. Secondly, but perhaps more significantly, we show that the use of even a very small amount of storage, to "remember" the location of records which have been requested, can lead to expected search costs arbitrarily close to that of the optimal ordering. The pre-

Received by the editors January 3, 1980, and in revised form February 2, 1981.

*This work was supported by the Natural Sciences and Engineering Research Council of Canada under grants A8237 and A3353. This paper was typeset at the University of Waterloo.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1.

‡ Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, M5S 1A7.

vious known results most relevant to our problem may be summarized as follows:

- (i) If we count access requests, the law of large numbers will permit us to order the file optimally.
- (ii) Let p_i ($i = 1, n$) denote the probability of a request for the i th most frequently accessed element and hence we observe the convention that $p_i \geq p_{i+1}$. We will say that a distribution \vec{p} is *trivial* if $p_i = p_j$ for all nonzero probabilities or if $p_i = 0$ for $i \geq 3$. Otherwise the distribution is *nontrivial*. Furthermore, if $T(\vec{p})$ denotes (the asymptotic value of) expected number of probes under the transposition rule, $F(\vec{p})$ the expected number under the move to front rule, and $\text{Opt}(\vec{p}) = \sum i p_i$ the expected number under the optimal ordering, then

$$F(\vec{p}) = 2 \sum_{i=1}^n \sum_{j=1}^i \frac{p_i p_j}{p_i + p_j},$$

- and for all nontrivial distributions $T(\vec{p}) < F(\vec{p}) < 2\text{Opt}(\vec{p})$ (Rivest [11]).
- (iii) While the move to front heuristic is known to produce a system with cost no more than twice that of the optimal ordering, the greatest value of $F(\vec{p})/\text{Opt}(\vec{p})$ which has been demonstrated is $2\ln 2$ (≈ 1.386). This occurs under Zipf's law, i.e. $p_i = 1/(iH_n)$ where H_n denotes the n th harmonic number (Knuth [9]).
- (iv) If any single memory-free heuristic has asymptotic behavior at least as good as every other such method for every probability distribution, it is the transposition rule (A. Yao as reported by Rivest [11]).

A casual glance at the results cited above indicates that the measure of effectiveness of a heuristic which has been used primarily is the (asymptotic) ratio of the expected behavior of the heuristic to that of the optimal ordering. This seems appropriate in studying classes of distributions under which the average cost of searching an optimally ordered list is not bounded by a constant as the list becomes longer (e.g. Zipf's law). It is not clear that this is a good measure for specific distributions, such as $p_i = 1/2^i$, under which the expected search time is bounded by a constant. Indeed we feel, and in §3 give "evidence", that this latter case may be the reason that we do not as yet have tight bounds on the behavior of even the move to front heuristic relative to the optimal ordering.

Yao's observation (that if any memory-free heuristic is better than all others over all possible distributions, it is the transposition rule) is perhaps the most intriguing fact known about the self organization of linear files. A definition of optimality for such a class must, however, be made carefully. Rivest [11] informally suggested such a definition, saying that a set of permutations was an optimal heuristic if, over all distributions and all initial orderings of the file, the expected number of probes to perform a search under the heuristic was (asymptotically) no greater than that under any other scheme using the same distribution and initial configuration. This wording is, unfortunately, a bit too strong, in that the "do nothing" heuristic outperforms all others (under most distributions) if it is fortunate enough to find its keys in decreasing order of probability. There is, then, under this definition, no optimal memory-free heuristic. We prefer to make a slight modification saying: A set of permutations is an *optimal memory-free heuristic* if for all probability distributions the maximum over all initial configurations of the asymptotic value of the average search cost is no greater than that for any other such scheme.

A slightly weaker, but equally satisfactory, definition is obtained by insisting that the expected asymptotic behavior be independent of the initial configuration, which appears to have been Rivest's intention. Under either definition the theorems of Rivest and observations of Yao regarding an optimal heuristic hold. With the following definition, we can make a more general statement about such methods.

Let G denote the class of all memory-free self-organizing heuristics for which:

- (i) When the element in position i is requested, that element is moved to position τ_i ($\tau_i < i$ if $i \neq 1$ and $\tau_1 = 1$), and all elements in positions τ_i through $i - 1$ are moved back one position.
- (ii) No other elements are moved.
- (iii) If $i < j$ then $\tau_i \leq \tau_j$.

THEOREM 1.1. *The asymptotic behavior of a reorganization technique in G is independent of the initial configuration.*

Proof. First we ignore the elements with probability 0 of being accessed since they will eventually percolate to the end of the list and be of no concern. We note that any configuration is reachable after at most n^2 accesses (we simply access each element enough times to bring it to position 1 in the reverse of the desired order). The probability of such a sequence of requests is nonzero; consequently, given enough time, each configuration is reachable with probability 1. The theorem then follows. \square

We feel that the class G provides a reasonable framework within which to study memory-free heuristics, and strongly conjecture that if H_1 and H_2 are heuristics in G such that $H_1 \neq H_2$ and $\tau_i(H_1) \leq \tau_i(H_2)$ for all i , then, for all nontrivial distributions, H_1 converges to its asymptotic behavior more quickly than H_2 , but the expected search time under H_2 is, asymptotically, less than that of H_1 .

Next to the asymptotic behavior of a heuristic, we feel that the rate of convergence to this behavior is its most interesting property. We are able to demonstrate the following results on convergence rates of the transposition and move to front heuristics.

THEOREM 1.2. *The transposition rule can take $\Omega(n^2)$ accesses to reach within factor $1 + \epsilon$ of the steady state behavior.*

Proof. To construct such an example, let n be the total number of elements, k of which have accessing probabilities $(1 - \delta)/k$ and the $n - k$ remaining have probability $\delta/(n - k)$.

Let δ be small enough so that $\delta/(n - k) < (1 - \delta)/k$; then the cost of the optimal configuration is

$$\text{Opt}(\vec{p}) = \frac{k(k+1)(1-\delta)}{2k} + \frac{n(n+1)-k(k+1)}{2} \cdot \frac{\delta}{n-k}.$$

The worst case (the reverse of the optimal) has a cost

$$\text{worst-case} = \frac{n(n+1)-(n-k)(n-k+1)}{2} \cdot \frac{1-\delta}{k} + \frac{(n-k)(n-k+1)}{2} \cdot \frac{\delta}{n-k}.$$

Each access to the file either maintains the average cost, or increases or decreases it by $(1 - \delta)/k - \delta/(n - k)$.

Since the steady state cost of the transposition rule is less than twice the optimal, the least number of accesses to reach a factor of $1 + \epsilon$, ($0 < \epsilon < 1$) of the steady state is

$$\begin{aligned} \text{min-accesses} &\geq \frac{\text{worst-case} - 2(1 + \epsilon)\text{Opt}(\vec{p})}{(1 - \delta)/k - \delta/(n - k)} \\ &\geq (n - \frac{1}{2} - \epsilon)k - (3/2 + \epsilon)k^2 + O(\delta) \end{aligned}$$

iff $k = \alpha n$; then, for any α such that $\alpha - 5\alpha^2/2 > 0$ or $0 < \alpha < 2/5$,

$$\text{min-accesses} = \Omega(n^2). \quad \square$$

Following Bitner [3], we define the overwork of a heuristic after t steps as the difference between the expected cost of t searches when the elements are in random order and the asymptotic cost of t searches. For the move to front rule this overwork is shown in [3] to be

$$\text{Ow}(t) = \frac{1}{2} \sum_{1 \leq i < j \leq n} \frac{(p_i - p_j)^2}{p_i + p_j} (1 - p_i - p_j)^t.$$

THEOREM 1.3. *The overwork at time t in the move to front rule is $O(n^2/t)$.*

Proof. Using the fact that for the chosen summation range $p_j \leq p_i$, we can rewrite the above as

$$\text{Ow}(t) \leq \frac{1}{2} \sum_{i=1}^n p_i (1 - p_i)^t (n - i).$$

For a given t , since this expression is maximized for $p_i = (t + 1)^{-1}$,

$$p_i (1 - p_i)^t \leq \frac{1}{t + 1} \left(\frac{t}{t + 1} \right)^t.$$

$(t/(t + 1))^t$ is a monotonically decreasing function of t . Consequently

$$p_i (1 - p_i)^t \leq \frac{1}{2(t + 1)}$$

for $t \geq 1$, and

$$\text{Ow}(t) \leq \frac{(n - 1)n}{8(t + 1)} = O\left(\frac{n^2}{t}\right). \quad \square$$

This bound is tight in the sense that we can find a file such that, for any $\epsilon > 0$, $\text{Ow}(t) = \Omega(n^{1-\epsilon})$ for $t = o(n^{1+\epsilon})$. Consider the table with

$$\begin{aligned} p_i &= \frac{2n - i}{n^{2+\epsilon}}, & 2 < i \leq n/2, \\ p_i &= \frac{2n - i}{n^{3+\epsilon}}, & n/2 < i \leq n \end{aligned}$$

and

$$p_1 = 1 - \sum_{i > 1} p_i,$$

with n large enough such that $p_1 > p_2$. Then

$$\begin{aligned} \text{Ow}(t) &\geq \frac{1}{2} \sum_{i=2}^{n/2} \sum_{j>n/2} \frac{(p_i - p_j)^2}{p_i + p_j} (1 - p_i - p_j)^t \\ &\geq \frac{(n-1)(n-2)}{8} \Theta(n^{-(1+\epsilon)})(1 - \Theta(n^{-(1+\epsilon)}))^t. \end{aligned}$$

For $t = o(n^{1+\epsilon})$ we derive

$$\text{Ow}(t) = \Omega(n^{1-\epsilon}).$$

For this example we also find that

$$\frac{\text{Ow}(t)}{F(\vec{p})} = \Omega(1).$$

2. k in a row heuristics. As we have noted, keeping a count on the number of accesses made on each element does enable us to order a table optimally. The objection, of course, is the storage requirement. In this section we propose a class of heuristics, closely akin to the memory-free techniques which use $(\log n + \log k)$ (for fixed k) extra bits of storage rather than the $\Theta(n)$ or more bits required for counter schemes. These techniques yield near optimal behavior.

The basic approach is simple, we apply the transposition (move to front or any other) heuristic only if the same element is accessed k times in a row. We first analyze the simple k heuristic and later a slight modification of it.

2.1. Simple k heuristics. We first analyze the simple k heuristic with the move to front rule. Let $b_k(j, i)$ denote the (asymptotic) probability that record j precedes record i in the list. This value can be found by considering the Markov chain shown in Fig. 1. State i_r denotes the fact that record i is ahead of record j , but the last r requests have been for record j ; i_0 is the initial state where record i precedes record j . In this state, record j is accessed with probability p_j and causes a transition to state i_1 . Otherwise, with probability $1 - p_j$, we remain in state i_0 . If record j is accessed k times in a row, the move to front rule is applied and we will arrive at state j_0 where record j precedes record i . The probability of record j preceding record i in the list is given by the sum of the probabilities of states j_0, j_1, \dots, j_{k-1} . The probabilities of the states satisfy the following equations as can be seen from the diagram.

$$\sum_{\ell=0}^{k-1} p_{i_\ell} + p_{j_\ell} = 1,$$

$$p_{i_\ell} = p_j^\ell p_{i_0}$$

and

$$p_{j_\ell} = p_i^\ell p_{j_0}, \quad 0 \leq \ell < k,$$

$$p_{i_0} = p_i p_{j_{k-1}} + (1 - p_j) \sum_{\ell=0}^{k-1} p_{i_\ell},$$

$$p_{j_0} = p_j p_{i_{k-1}} + (1 - p_i) \sum_{\ell=0}^{k-1} p_{j_\ell}.$$

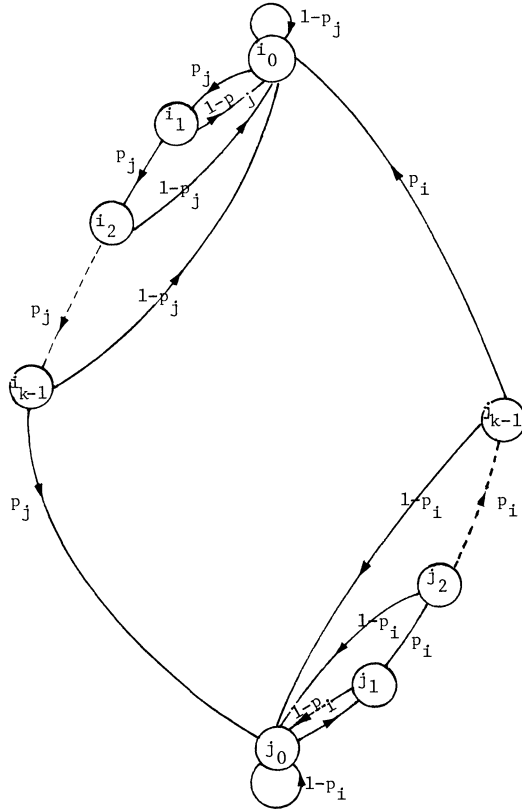


Fig. 1. Markov chain for the simple k heuristic.

The solution of this system of equations is

$$p_{j_m} = \frac{p_j^k p_i^m}{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell + p_i^k \sum_{\ell=0}^{k-1} p_j^\ell},$$

$$p_{i_m} = \frac{p_i^k p_j^m}{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell + p_i^k \sum_{\ell=0}^{k-1} p_j^\ell},$$

as can be easily verified. Thus we present the following lemma:

LEMMA 2.1.1.

$$b_k(j,i) = \frac{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell}{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell + p_i^k \sum_{\ell=0}^{k-1} p_j^\ell}.$$

Let $F_k(\vec{p})$ denote the (asymptotic) value of the expected number of probes necessary to perform a search in a table ordered under the simple k heuristic with the move to front rule. Then

$$F_k(\vec{p}) = \sum_{i=1}^n \sum_{j=1}^i \frac{p_j^k p_i \sum_{\ell=0}^{k-1} p_i^\ell + p_i^k p_j \sum_{\ell=0}^{k-1} p_j^\ell}{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell + p_i^k \sum_{\ell=0}^{k-1} p_j^\ell}.$$

Noting that the average search cost under the optimal ordering is $\text{Opt}(\vec{p}) = \sum_{i=1}^n i p_i$ and that we would like to bound the ratio of these expressions, we consider the ratio of the m th terms of the two expressions, which can be simplified as

$$\frac{F_k(\vec{p})}{\text{Opt}(\vec{p})} \leq \max_m \left\{ \frac{1}{m} \sum_{j=1}^m \frac{p_j^k \sum_{\ell=0}^{k-1} p_m^\ell + p_j p_m^{k-1} \sum_{\ell=0}^{k-1} p_j^\ell}{p_j^k \sum_{\ell=0}^{k-1} p_m^\ell + p_m^k \sum_{\ell=0}^{k-1} p_j^\ell} \right\}.$$

This term is, of course, bounded by the maximum of the terms in the summation. In other words, it is bounded by the maximum of

$$\frac{p^k(1+q+\dots+q^{k-1})+pq^{k-1}(1+p+\dots+p^{k-1})}{p^k(1+q+\dots+q^{k-1})+q^k(1+p+\dots+p^{k-1})},$$

subject to $0 \leq q \leq p \leq 1$.

$$\begin{aligned} \frac{F_k(\vec{p})}{\text{Opt}(\vec{p})} &\leq \frac{p^k + pq^{k-1} \frac{1+p+\dots+p^{k-1}}{1+q+\dots+q^{k-1}}}{p^k + q^k \frac{1+p+\dots+p^{k-1}}{1+q+\dots+q^{k-1}}} \\ &\leq \frac{p^k + pq^{k-1}k}{p^k + q^k k} \\ &= \frac{(p/q)^k + (p/q)k}{(p/q)^k + k} = \frac{x^k + kx}{x^k + k}, \quad \text{where } x = p/q \geq 1. \end{aligned}$$

Since this function is unimodal in the relevant range, it is maximized when x satisfies the equation

$$(1-k)x^k + kx^{k-1} + k = 0.$$

LEMMA 2.1.2. *The polynomial $(1-k)x^k + kx^{k-1} + k$ has a zero at*

$$x = 1 + \frac{1+w(k/e)}{k} + O(k^{-2}(\ln k)^2),$$

where $w(x)$ is the transcendental function defined by $w(x)e^{w(x)} = x$, and hence from 4 $w(x) = \ln x - \ln(\ln x) + o(1)$.

Proof. The proof follows from substituting $x = 1 + (1+w(k/e))/k + aw(k/e)^2/k^2$ in the above polynomial. After some routine asymptotic calculations we find

$$(1-k)x^k + kx^{k-1} + k = (1/2 - a)w(k/e)^2 + O(w(k/e)).$$

For any $a > 1/2$ the polynomial becomes negative for a sufficiently large k and similarly for $a < 1/2$ the polynomial becomes positive. Consequently the root of the polynomial occurs for

$$x = 1 + \frac{1 + w(k/e)}{k} + O(k^{-2}(\ln k)^2). \quad \square$$

By substituting this asymptotic value of the root we have:

THEOREM 2.1.3. *The ratio of the expected number of probes required to perform a linear search on a table ordered by the simple k heuristic with the move to front rule, $F_k(\vec{p})$, to the search cost under the optimal ordering is bounded by*

$$\frac{F_k(\vec{p})}{\text{Opt}(\vec{p})} \leq 1 + \frac{w(k/e)}{k} + O(k^{-2}(\ln k)^2).$$

The analogous substitution of the zeros of the function for small values of k yield what may be considered even more interesting:

THEOREM 2.1.4.

$$F_2(\vec{p}) \leq \frac{\sqrt{3} + 1}{2} \text{Opt}(\vec{p}) = 1.36602\dots \text{Opt}(\vec{p}),$$

$$F_3(\vec{p}) \leq 1.27388\dots \text{Opt}(\vec{p}),$$

and

$$F_4(\vec{p}) \leq 1.22788\dots \text{Opt}(\vec{p}).$$

We will now show that the behavior strictly improves as k is increased.

LEMMA 2.1.5. *For $k > 1$ and $p_i \geq p_j$*

$$\begin{aligned} p_j(1 + p_i + \dots + p_i^{k-1})(1 + p_j + \dots + p_j^{k-2}) \\ \leq p_i(1 + p_j + \dots + p_j^{k-1})(1 + p_i + \dots + p_i^{k-2}). \end{aligned}$$

Proof. The proof is by induction on k . For $k = 2$, $p_j(1 + p_i) \leq p_i(1 + p_j)$. Assume that the inequality holds for k , i.e.,

$$(*) \quad p_j a (b - p_i^{k-1}) \leq p_i b (a - p_j^{k-1}),$$

where $a = 1 + p_i + \dots + p_i^{k-1}$ and $b = 1 + p_j + \dots + p_j^{k-1}$.

We also know that $p_j^k(1 - p_i^k) \leq p_i^k(1 - p_j^k)$ or

$$(**) \quad p_j^k(1 - p_i)a \leq p_i^k(1 - p_j)b.$$

Adding (*) and (**) we obtain

$$p_j a b - a p_i p_j^k \leq p_i a b - b p_i^k p_j$$

or

$$p_j b (a + p_i^k) \leq p_i (b + p_j^k) a. \quad \square$$

COROLLARY 2.1.6. *For $k > 1$ and $p_i \geq p_j$, then*

$$p_j(1 - p_i^k)(1 - p_j^{k-1}) \leq p_i(1 - p_j^k)(1 - p_i^{k-1}).$$

THEOREM 2.1.7. For $k > 1$, $F_k(\vec{p}) \leq F_{k-1}(\vec{p})$. Furthermore, the inequality is strict for nontrivial distributions.

Proof. For $i < j$ and therefore $p_i \geq p_j$, the following is valid. By Corollary 2.1.6, $p_j(1-p_i^k)(1-p_j^{k-1}) \leq p_i(1-p_j^k)(1-p_i^{k-1})$ or

$$p_i^{k-1}p_j^k(1-p_i^k)(1-p_j^{k-1}) \leq p_i^k p_j^{k-1}(1-p_j^k)(1-p_i^{k-1})$$

and hence

$$\begin{aligned} p_j^{2k-1} \frac{1-p_i^k}{1-p_i} \frac{1-p_i^{k-1}}{1-p_i} + p_i^{k-1}p_j^k \frac{1-p_i^k}{1-p_i} \frac{1-p_j^{k-1}}{1-p_j} \\ \leq p_j^{2k-1} \frac{1-p_i^k}{1-p_i} \frac{1-p_i^{k-1}}{1-p_i} + p_i^k p_j^{k-1} \frac{1-p_j^k}{1-p_j} \frac{1-p_i^{k-1}}{1-p_i} \end{aligned}$$

or

$$\frac{p_j^k \frac{1-p_i^k}{1-p_i}}{p_j^k \frac{1-p_i^k}{1-p_i} + p_i^k \frac{1-p_j^k}{1-p_j}} \leq \frac{p_j^{k-1} \frac{1-p_i^{k-1}}{1-p_i}}{p_j^{k-1} \frac{1-p_i^{k-1}}{1-p_i} + p_i^{k-1} \frac{1-p_j^{k-1}}{1-p_j}},$$

and therefore

$$b_k(j,i) \leq b_{k-1}(j,i).$$

The theorem follows from [8, Thm. 2]. \square

Now we turn to the simple k heuristic with transposition rule.

THEOREM 2.1.8. Under the simple k heuristic with transposition rule the stationary probabilities obey:

$$\frac{\text{Prob}[R_{i_1}R_{i_2}\dots R_{i_j}R_{i_{j+1}}\dots R_{i_n}]}{\text{Prob}[R_{i_1}R_{i_2}\dots R_{i_{j+1}}R_{i_j}\dots R_{i_n}]} = \frac{p_{i_j}^k \sum_{\ell=0}^{k-1} p_{i_{j+1}}^\ell}{p_{i_{j+1}}^k \sum_{\ell=0}^{k-1} p_{i_j}^\ell} \quad \text{for } 1 \leq j < n \text{ if } p_\ell \neq 0 \text{ for } 1 \leq \ell \leq n.$$

Proof. Consider the Markov chain given in Fig. 2 which describes the transposition rule under the simple k heuristic from the configuration $R_{i_1}R_{i_2}\dots R_{i_n}$. For the sake of clarity some edges are not drawn. It is not difficult to fill in the edges because of symmetry. The probability of the configuration $R_{i_1}\dots R_{i_n}$ is the sum of the probabilities of each state in the big dotted square. Let $I1$ denote the state representing the initial configuration of $R_{i_1}\dots R_{i_n}$, $I2$ denote the state representing the initial configuration of $R_{i_2}R_{i_1}R_{i_3}\dots R_{i_n}$, etc.. $X1$ is the probability of being in the state $I1$, $X2$ is the probability of being in the state $I2$, etc.. Starting at $I1$, m consecutive requests for R_{i_j} (each with probability p_{i_j}) will lead us to the state $I1_{jm}$. The probability of being in state $I1_{jm}$ is denoted by $X1_{jm}$. Hence, we get the following equations:

$$X1 = p_{i_1} \left(X1 + \sum_{j=1}^{k-1} X1_{2j} + \dots + \sum_{j=1}^{k-1} X1_{nj} \right) + p_{i_1} X2_{1,k-1} + \dots + p_{i_{n-1}} Xn_{n-1,k-1}, \tag{1}$$

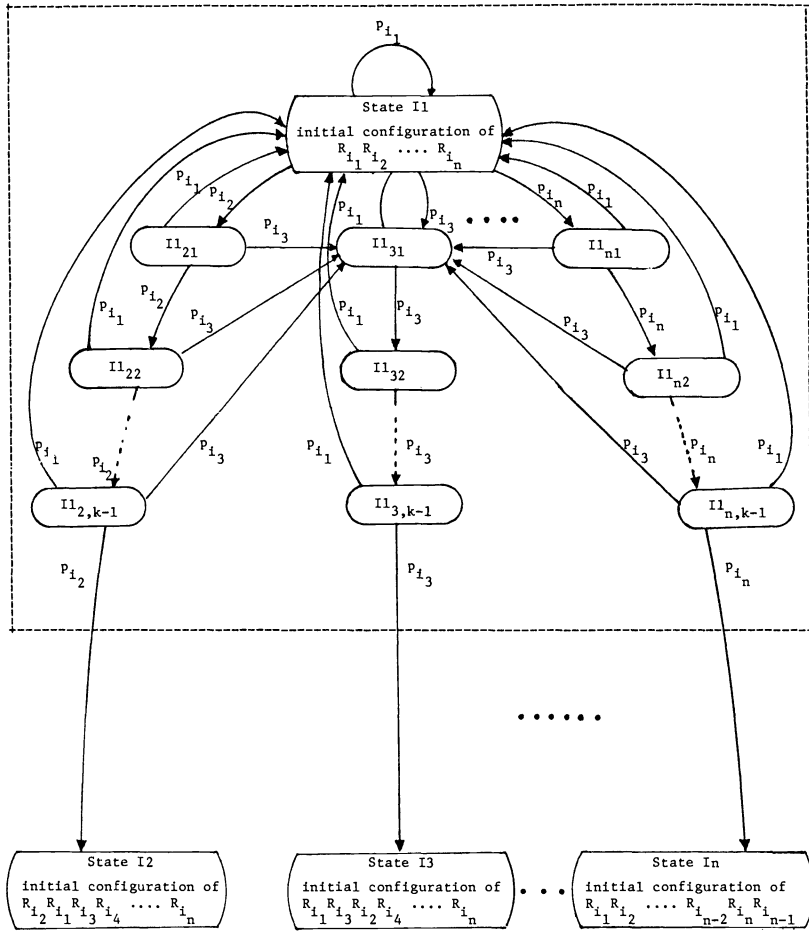


Fig. 2. Markov chain for the transposition rule under the simple k heuristic.

$$(2.2) \quad X1_{21} = p_{i_2} \left(X1 + \sum_{j=1}^{k-1} X1_{3j} + \dots + \sum_{j=1}^{k-1} X1_{n,k-1} \right),$$

⋮

$$(2.n) \quad X1_{n1} = p_{i_n} \left(X1 + \sum_{j=0}^{k-1} X1_{2j} + \dots + \sum_{j=1}^{k-1} X1_{n-1,k-1} \right),$$

$$(3) \quad X1_{jm} = p_{i_j}^{m-1} X1_{j1}, \quad 1 \leq m \leq k-1, \quad 2 \leq j \leq n.$$

From (2.2), ..., (2.n) and (3) we obtain

$$(4) \quad p_{i_j} (1 + \dots + p_{i_m}^{k-1}) X1_{m1} = p_{i_m} (1 + \dots + p_{i_j}^{k-1}) X1_{j1}.$$

From (1), (2.2) and (3) we obtain

$$(5) \quad p_{i_2} X 1 = p_{i_1} \sum_{j=0}^{k-1} p_{i_2}^j X 1_{21} + p_{i_2} p_{i_1}^{k-1} X 2_{11} + p_{i_2}^k X 3_{21} + \dots + p_{i_2} p_{i_{n-1}}^{k-1} X n_{n-1,1}.$$

Notice that $\text{Prob}[R_{i_1} \dots R_{i_n}] = (\sum_{j=0}^{k-1} p_{i_2}^j / p_{i_2}) X 1_{21}$. From (4) and (2.2) we obtain

$$(6) \quad p_{i_2} X 1 = 1 - \left(\sum_{j=0}^{k-1} p_{i_2}^j \right) \left(p_{i_3} \sum_{j=0}^{k-2} p_{i_3}^j + \dots + p_{i_n} \sum_{j=0}^{k-2} p_{i_n}^j \right) / \left(\sum_{j=0}^{k-1} p_{i_n}^j \right) X 1_{21}.$$

Comparing (5) and (6) and noting that $p_{i_1} = 1 - p_{i_2} - \dots - p_{i_3}$ we get

$$\begin{aligned} \frac{X 1_{21}}{X n_{n-1,1}} &= \frac{p_{i_{n-1}}^{k-1} \sum_{j=0}^{k-1} p_{i_2}^j p_{i_2}}{p_{i_n}^k \sum_{j=0}^{k-1} p_{i_2}^j}, \\ &\vdots \\ \frac{X 1_{21}}{X 3_{21}} &= \frac{p_{i_2}^k \sum_{j=0}^{k-1} p_{i_3}^j}{p_{i_3}^k \sum_{j=0}^{k-1} p_{i_2}^j}, \\ \frac{X 1_{21}}{X 2_{11}} &= \left(\frac{p_{i_1}}{p_{i_2}} \right)^{k-1}. \end{aligned}$$

Hence,

$$\frac{\text{Prob}[R_{i_1} \dots R_{i_j} R_{i_{j+1}} \dots R_{i_n}]}{\text{Prob}[R_{i_1} \dots R_{i_{j+1}} R_{i_j} \dots R_{i_n}]} = \frac{p_{i_j}^k \sum_{\ell=0}^{k-1} p_{i_{j+1}}^\ell}{p_{i_{j+1}}^k \sum_{\ell=0}^{k-1} p_{i_j}^\ell}.$$

Following [11, Thm. 2], we note that the stationary probabilities must satisfy the equations stated in the theorem. \square

COROLLARY 2.1.9.

$$\frac{\text{Prob}[R_1 \dots R_{i-1} R_i \dots R_{i+\ell} \dots R_n]}{\text{Prob}[R_1 \dots R_{i-1} R_{i+\ell} \dots R_i \dots R_n]} = \left(\frac{p_i^k \sum_{j=0}^{k-1} p_{i+\ell}^j}{p_{i+\ell}^k \sum_{j=0}^{k-1} p_i^j} \right)^\ell.$$

Now we can show that the transposition rule is better than the move to front rule under the simple k heuristic. Let $T_k(\vec{p})$ denote the asymptotic value of the expected number of probes necessary to perform a search in a table ordered under a simple k heuristic with the transposition rule, then:

THEOREM 2.1.10. $T_k(\vec{p}) \leq F_k(\vec{p})$ for all distributions; furthermore, the inequality is strict for nontrivial distributions.

Proof.

$$\text{Prob [record } j \text{ precedes record } i] = \frac{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell}{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell + p_i^k \sum_{\ell=0}^{k-1} p_j^\ell}$$

for the simple k heuristic with move to front. For the simple k heuristic with the transposition rule we have

$$\text{Prob [} j \text{ precedes } i] = \sum_{\ell=0}^{n-2} \sum_{\alpha} \text{Prob}[\alpha],$$

where α is a configuration for which record j precedes record i , with exactly ℓ items between them or

$$\text{Prob [} j \text{ precedes } i] = \sum_{\ell=0}^{n-2} \left(\frac{p_j^k \sum_{h=0}^{k-1} p_i^h}{p_i^k \sum_{h=0}^{k-1} p_j^h} \right)^{\ell+1} \sum_{\beta} \text{Prob}[\beta],$$

where β is a configuration identical to α except that record j and record i are interchanged or

$$\text{Prob [} j \text{ precedes } i] \leq \frac{p_j^k \sum_{h=0}^{k-1} p_i^h}{p_i^k \sum_{h=0}^{k-1} p_j^h} (1 - \text{Prob [} j \text{ precedes } i]).$$

Thus,

$$\text{Prob [record } j \text{ precedes record } i] \leq \frac{p_j^k \sum_{h=0}^{k-1} p_i^h}{p_i^k \sum_{h=0}^{k-1} p_j^h + p_j^k \sum_{h=0}^{k-1} p_i^h}. \quad \square$$

Although we are not able to compare the average search cost of transposition rule under the simple k heuristic to the optimal one, we can show that the cost decreases as k increases. The idea of the proof is that for a given probability distribution and $i \leq j$ (or $p_i \geq p_j$), if $b(j,i)$, the probability that record j precedes record i , is smaller for heuristic A than for heuristic B, then the average search cost for heuristic A is smaller than the one for heuristic B [8]. For a list of n records, we have $n!$ configurations. Record j precedes record i in $n!/2$ of them and vice versa. Consider a pair of identical configurations except that record j and i are interchanged. There are exactly $n!/2$ pairs of such configurations. Each satisfies the ratio described in Corollary 2.1.9. First we show the following lemma:

LEMMA 2.1.11. For $p_i \geq p_j$ and $k \geq 1$:

$$\frac{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell}{p_i^k \sum_{\ell=0}^{k-1} p_j^\ell} \leq \frac{p_j^{k-1} \sum_{\ell=0}^{k-2} p_i^\ell}{p_i^{k-1} \sum_{\ell=0}^{k-2} p_j^\ell}.$$

Proof. Follows from Lemma 2.1.3. \square

Let $C_1, \dots, C_{n!}$ denote the $n!$ possible configurations. $C_1, \dots, C_{n!/2}$ are the configurations where record j precedes record i , $C_{1+n!/2}, \dots, C_{n!}$ are the corresponding identical configurations with record j and record i interchanged. Note that C_i and $C_{i+n!/2}$ make a pair of configurations as mentioned above. Let $x_1, \dots, x_{n!}$ be the stationary probabilities of $C_1, \dots, C_{n!}$ under heuristic A and $y_1, \dots, y_{n!}$ be the stationary probabilities under heuristic B. $x_1 + \dots + x_{n!/2}$ and $y_1 + \dots + y_{n!/2}$ are the probabilities of record j preceding record i in heuristic A and B respectively.

LEMMA 2.1.12. Let $x_i/x_{i+n!/2} = a_i$ and $y_i/y_{i+n!/2} = b_i$.

If $b_i \leq a_i$ then $\sum_{i=1}^{n!/2} y_i \leq \sum_{i=1}^{n!/2} x_i$.

Proof.

$$\sum_{i=1}^{n!} x_i = 1 = \sum_{i=1}^{n!} y_i, \quad \text{and}$$

$$\sum_{i=1}^{n!/2} (1 + a_i)x_{i+n!/2} = \sum_{i=1}^{n!/2} (1 + b_i)y_{i+n!/2} = 1.$$

Since $a_i \geq b_i$, thus

$$\sum_{i=1}^{n!/2} x_{i+n!/2} \leq \sum_{i=1}^{n!/2} y_{i+n!/2} \quad \text{or} \quad \sum_{i=1}^{n!/2} x_i \geq \sum_{i=1}^{n!/2} y_i. \quad \square$$

Now we are ready to prove the following theorem.

THEOREM 2.1.13. For $k > 1$, $T_k(\vec{p}) \leq T_{k-1}(\vec{p})$. Furthermore, the inequality is strict for nontrivial distributions.

Proof. The proof follows from Corollary 2.1.9 and Lemmas 2.1.11 and 2.1.12. \square

2.2. Batched k heuristics. A slightly different approach is to view requests as (for purposes of reorganization) being batched into groups of k consecutive requests. A reordering permutation is then applied only if all k requests in a batch were for the same element. The effect of such schemes may seem to be equivalent to the simple k approach. However, if an element is accessed and subsequently not moved forward because of the access of some other (unknown) element, the original element has a lower probability of being moved forward than in the case of the basic k in a row scheme. Intuitively these heuristics are better than their simple k heuristics counterparts, because these heuristics perform fewer changes.

For a batched k heuristic with the move to front rule, the probability that record j precedes record i ($b'(j,i)$) can be derived in an analogous manner to that for $b_k(j,i)$ (in §2.1). Indeed, the Markov chain in Fig. 3 describes this process.

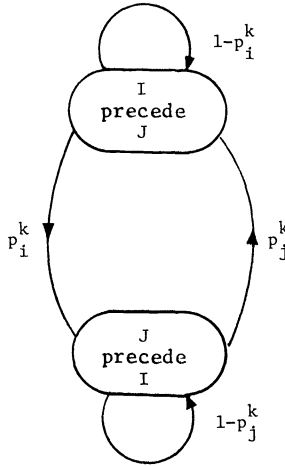


Fig. 3. Markov chain for batched k algorithm.

It can be easily verified from this Markov chain that

$$b_k' = \frac{p_j^k}{p_i^k + p_j^k}.$$

Intuitively, the effect of the batched k heuristics is to raise the probability of access of each record to the power k which after normalization makes the large probabilities larger and the small probabilities even smaller.

Define $F_k'(\vec{p})$ and $T_k'(\vec{p})$ for the batched schemes in the same way that $F_k(\vec{p})$ and $T_k(\vec{p})$ were defined for the simple k schemes.

THEOREM 2.2.1. For $k > 1$, $F_k'(\vec{p}) \leq F_k(\vec{p})$. The inequality is strict for all nontrivial distributions.

Proof. For $i < j$ and thus $p_i \geq p_j$

$$p_j^{2k} \sum_{\ell=0}^{k-1} p_i^\ell + p_i^k p_j^k \sum_{\ell=0}^{k-1} p_j^\ell \leq p_i^k p_j^k \sum_{\ell=0}^{k-1} p_i^\ell + p_j^{2k} \sum_{\ell=0}^{k-1} p_i^\ell.$$

Thus

$$\frac{p_j^k}{p_i^k + p_j^k} \leq \frac{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell}{p_j^k \sum_{\ell=0}^{k-1} p_i^\ell + p_i^k \sum_{\ell=0}^{k-1} p_j^\ell},$$

or

$$b_k'(j,i) \leq b_k(j,i) \quad \text{for } i < j.$$

The theorem follows from [8, Thm. 2]. \square

THEOREM 2.2.2. For $k > 1$, $T_k'(\vec{p}) \leq T_{k-1}'(\vec{p})$. The inequality is strict for all nontrivial distributions.

Proof. Similar to that of Theorem 2.1.13. \square

THEOREM 2.2.3. For $k \geq 1$, $T_k'(\vec{p}) \leq T_k(\vec{p})$. The inequality is strict for all nontrivial distributions.

Proof. For the batched k heuristic with transposition rule

$$\frac{\text{Prob}[\dots R_j \dots R_i \dots]}{\text{Prob}[\dots R_i \dots R_j \dots]} = \frac{p_j^k}{p_i^k} \leq \frac{p_j^k(1 + \dots + p_i^{k-1})}{p_i^k(1 + \dots + p_j^{k-1})}$$

for $p_i \geq p_j$. The theorem follows from Lemma 2.1.12 and [8]. \square

LEMMA 2.2.4. The polynomial $(1-k)x^k + kx^{k-1} + 1$ has a root at $x = 1 + a/k + O(k^{-2})$, where a is the root of $(a-1)e^a = 1$, $a = 1 + w(e^{-1})$.

Proof. Let $x = 1 + a/k + b/k^2 + O(k^{-3})$. Then

$$\ln x = \frac{a}{k} + \frac{2b - a^2}{2k^2} + O(k^{-3}),$$

and

$$\begin{aligned} x^k &= e^a \left(1 + \frac{2b - a^2}{2k} + O(k^{-2}) \right), \\ x^{k-1} &= \frac{x^k}{x} = e^a \left(1 + \frac{2b - a^2 - 2a}{2k} + O(k^{-2}) \right). \end{aligned}$$

Substituting x^k and x^{k-1} in the original polynomial, and using the fact that $e^a = 1/(a-1)$, we obtain

$$(1-k)x^k + kx^{k-1} + 1 = \frac{a(a(a+1) - 2b)}{2(a-1)k} + O(k^{-2}).$$

For $b > a(a+1)/2$, the polynomial will be negative for sufficiently large k , while for $b < a(a+1)/2$ the polynomial will be positive. Consequently the root of the polynomial is located at $x = 1 + a/k + O(k^{-2})$. \square

THEOREM 2.2.5. The ratio of the expected number of probes required to perform a linear search on a table ordered by the batched k heuristic with the move to front rule, $F_k'(\vec{p})$, to the search cost under the optimal ordering is bounded by

$$\frac{F_k'(\vec{p})}{\text{Opt}(\vec{p})} \leq 1 + \frac{a-1}{k} + O(k^{-2}),$$

where a is the solution of $e^a(a-1) = 1$ or $a = 1 + w(e^{-1}) = 1.27846\dots$

Proof.

$$\begin{aligned} b'(j,i) &= \frac{p_j^k}{p_i^k + p_j^k}, \quad \text{and} \\ F_k'(\vec{p}) &= \sum_{i=1}^n p_i \left(1 + \sum_{i \neq j} \frac{p_j^k}{p_j^k + p_i^k} \right) \\ &= \sum_{i=1}^n \sum_{j=1}^i \frac{p_i p_j^k + p_i^k p_j}{p_i^k + p_j^k}. \end{aligned}$$

Noting that the average search cost under the optimal ordering is

$\text{Opt}(\vec{p}) = \sum_{i=1}^n ip_i$ and considering the ratio of the m th terms of the two expressions, we get

$$\frac{F_k'(\vec{p})}{\text{Opt}(\vec{p})} \leq \max_m \left\{ \frac{1}{m} \sum_{j=1}^m \frac{p_j^k + p_m^{k-1} p_j}{p_j^k + p_m^k} \right\}.$$

This term is bounded by the maximum of the terms in the summation. In other words, it is bounded by the maximum of

$$\frac{p^k + q^{k-1} p}{p^k + q^k}, \quad 0 \leq q \leq p \leq 1,$$

or

$$\frac{x^k + x}{x^k + 1}, \quad x \geq 1.$$

This ratio is unimodal in the relevant range and so maximized when the numerator of its derivative $(1-k)x^k + kx^{k-1} + 1$ is 0. By Lemma 2.2.4, this occurs when $x = 1 + a/k + O(k^{-2})$. Thus,

$$\begin{aligned} \frac{x^k + x}{x^k + 1} &= 1 + \frac{x - 1}{x^k + 1} \leq 1 + \frac{\frac{a}{k} + O(k^{-2})}{e^a(1 + O(k^{-1})) + 1} \\ &\leq 1 + \frac{\frac{a}{k}}{e^a + 1} + O(k^{-2}) \\ &= 1 + \frac{a - 1}{k} + O(k^{-2}). \quad \square \end{aligned}$$

In particular, solving the polynomial exactly for $k = 2, 3$ and 4 we obtain THEOREM 2.2.6.

$$\begin{aligned} F_2'(\vec{p}) &\leq \frac{1 + \sqrt{2}}{2} \text{Opt}(\vec{p}) = 1.20710\dots \text{Opt}(\vec{p}), \\ F_3'(\vec{p}) &\leq 1.11843\dots \text{Opt}(\vec{p}), \\ F_4'(\vec{p}) &\leq 1.08302\dots \text{Opt}(\vec{p}). \end{aligned}$$

THEOREM 2.2.7. *Under the batched k heuristic with transposition rule the stationary probabilities obey:*

$$\frac{\text{Prob}[R_{i_1} R_{i_2} \dots R_{i_j} R_{i_{j+1}} \dots R_{i_n}]}{\text{Prob}[R_{i_1} R_{i_2} \dots R_{i_{j+1}} R_{i_j} \dots R_{i_n}]} = \frac{p_{i_j}^k}{p_{i_{j+1}}^k} \quad \text{for } 1 \leq j < n.$$

Proof.

$$\text{Prob}[R_{i_1} \dots R_{i_n}] =$$

$$(1 - p_{i_2}^k - p_{i_3}^k - \dots - p_{i_n}^k) \text{Prob}[R_{i_1} \dots R_{i_n}] + \sum_{1 \leq j < n} p_{i_j}^k \text{Prob}[R_{i_1} \dots R_{i_{j+1}} R_{i_j} \dots R_{i_n}].$$

With the equations stated in the theorem, this implies

$$\text{Prob}[R_1 \dots R_{i_n}] = \text{Prob}[R_{i_1} \dots R_{i_n}].$$

Following [11, Thm. 2], we note that the stationary probabilities must satisfy the equations stated in the theorem. \square

COROLLARY 2.2.8.

$$\frac{\text{Prob}[R_1 \dots R_{i-1} R_i \dots R_{i+\ell} \dots R_n]}{\text{Prob}[R_1 \dots R_{i-1} R_{i+\ell} \dots R_i \dots R_n]} = \left(\frac{p_i^k}{p_{i+\ell}^k} \right)^\ell.$$

THEOREM 2.2.9. $T_k'(\vec{p}) \leq F_k'(\vec{p})$ for all distributions; furthermore, the inequality is strict for nontrivial distributions.

Proof.

$$\text{Prob}[\text{record } j \text{ precedes record } i] = \frac{p_j^k}{p_i^k + p_j^k}$$

for the batched k heuristic with move to front rule. For the batched k heuristic with transposition rule we have

$$\text{Prob}[\text{record } j \text{ precedes record } i] = \sum_{\ell=0}^{n-2} \sum_{\alpha} \text{Prob}[\alpha],$$

where α is a configuration for which record j precedes record i , with exactly ℓ items between them, or

$$\text{Prob}[j \text{ precedes } i] = \sum_{\ell=0}^{n-2} \left\{ \left(\frac{p_j^k}{p_i^k} \right)^{\ell+1} \sum_{\beta} \text{Prob}[\beta] \right\},$$

where β is a configuration identical to α except that record j and record i are interchanged, or

$$\text{Prob}[j \text{ precedes } i] \leq \left(\frac{p_j^k}{p_i^k} \right) (1 - \text{Prob}[\text{record } j \text{ precedes record } i]).$$

Thus

$$\text{Prob}[\text{record } j \text{ precedes record } i] \leq \frac{p_j^k}{p_i^k + p_j^k}. \quad \square$$

2.3. A brief digression to self-organizing binary search trees. In the case of the self-organizing linear list, the k heuristics applied to the move to front rule perform uniformly better than the move to front heuristic. But, in the case of self-organizing binary search trees [2], the batched k scheme applied to the move to root heuristic is not always better than the simple move to root heuristic. For example, if $p_1 = 0.4$, $p_2 = 0.3$ and $p_3 = 0.3$, the cost for move to root is 1.88286... and the cost for the batched 2 scheme applied to the move to root is 1.88306... . The intuitive explanation for this fact is that under the batched scheme applied to the

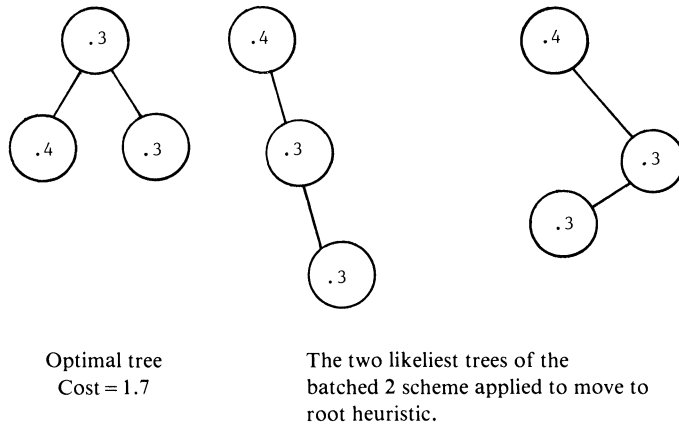


Fig. 4. Three possible trees.

move to root heuristic, the key with the largest probability of access is more inclined to become the root of the tree than in the case for the simple move to root. However, having the most frequently accessed key as the root of the tree does not necessarily produce the best tree. The trees are depicted in Fig. 4.

Furthermore, although we are able to decrease the cost of the batched k scheme as k increases in the case of linear list, the cost of the tree obtained by the batched k scheme applied to the move to root heuristic will not necessarily decrease as k increases. This occurs, for example, in the three key tree noted above and also in the case in which n keys have equal probability of being accessed. In the latter example the move to root heuristic both with and without a batched scheme produce a tree with a cost of about $(2\ln 2)\log n$. We note that this is the worst possible behavior relative to the optimal solution for the simple move to root heuristic.

3. More on the move to front heuristic. In this section our attention returns to memory-free heuristics, and in particular to the move to front rule. We analyze the expected behavior of this rule for a number of distributions and demonstrate what is apparently a rather tight upper bound on the ratio $F(\vec{p})/\text{Opt}(\vec{p})$ for a class of distributions. This leads to the observation that this ratio can be as large, and no larger than, $\pi/2 \simeq 1.57$ for an interesting class of distributions. It also gives us some intuition as to the difficulty of closing the gap between the worst known case of this ratio, and the best known upper bound, 2. Finally, an expression for arbitrary moments of the number of accesses required by the move to front heuristic under any distribution is given.

3.1. Analyses of the costs of the move to front rule under several distributions. Knuth [9] analyzed the expected cost of a search under the move to front rule. We present the results of such an analysis for this and several other interesting distributions. The most interesting observation is that the ratio $F(\vec{p})/\text{Opt}(\vec{p})$ for Lotka's law is $\pi/2$ (H_n denotes the n th harmonic number, $H_n^{(\ell)}$ denotes $\sum_{i=1}^n i^{-\ell}$). We are interested primarily in the case in which n , the number of elements in the list, tends to ∞ .

(i) Zipf's law. Under Zipf's law elements have accessing probabilities $p_i = 1/(iH_n)$. The optimal ordering has a cost $\text{Opt}(\vec{p}) = n/H_n$. The simple move to front rule has a cost [9]

$$F(\vec{p}) = 2\ln 2 \frac{n}{H_n} - \frac{1}{2} + o(1),$$

$$\frac{F(\vec{p})}{\text{Opt}(\vec{p})} \leq 2\ln 2 = 1.38629\dots$$

In this case we can also compute

$$F_2'(\vec{p}) = \frac{1}{2} + \frac{1}{H_n} \sum_{j=1}^n \sum_{i=1}^n \frac{i}{i^2 + j^2}$$

$$= \frac{1}{2} + \frac{1}{H_n} \sum_{j=1}^n j \left(-\frac{1}{2j^2} + \frac{\pi \coth(\pi j)}{2j} - \sum_{i=n+1}^{\infty} \frac{1}{i^2 + j^2} \right)$$

$$= \frac{\pi}{2H_n} \left(\sum_{j=1}^n \coth(\pi j) - \sum_{j=1}^n \tan^{-1}(j/n) \right) + O(1)$$

$$\approx \frac{n}{H_n} \left(\frac{\pi}{4} - \frac{\ln 2}{2} \right) = 1.13197\dots \frac{n}{H_n}.$$

$$\frac{F_2'(\vec{p})}{\text{Opt}(\vec{p})} \leq 1.13197\dots$$

(ii) For Lotka's law the accessing probabilities are $p_i = (i^2 H_n^{(2)})^{-1}$, and the optimal arrangement has cost $\text{Opt}(\vec{p}) = H_n/H_n^{(2)}$,

$$F(\vec{p}) = \frac{1}{2} + \frac{1}{H_n^{(2)}} \sum_{j=1}^n \sum_{i=1}^n \frac{1}{i^2 + j^2}$$

$$= \frac{1}{2} + \frac{1}{H_n^{(2)}} \sum_{j=1}^n \left(-\frac{1}{2j^2} + \frac{\pi \coth(\pi j)}{2j} - \sum_{i=n+1}^{\infty} \frac{1}{i^2 + j^2} \right)$$

$$= \frac{1}{H_n^{(2)}} \sum_{j=1}^n \left(\frac{\pi \coth(\pi j)}{2j} - \frac{\tan^{-1}(j/n)}{j} + O(n^{-2}) \right)$$

$$= \frac{3\ln n + 3\gamma + 6C_1}{\pi} - \frac{6\beta(2)}{\pi^2} + O\left(\frac{\ln n}{n}\right)$$

$$= \frac{3}{\pi} \ln n - 0.00206339\dots + O\left(\frac{\ln n}{n}\right),$$

where $C_1 = -\sum_j \ln(1 - e^{-2\pi j}) = 0.001872\dots$ and $\beta(2) = 0.91596\dots$ is Catalan's constant [1],

$$\frac{F(\vec{p})}{\text{Opt}(\vec{p})} \leq \frac{\pi}{2} = 1.57080\dots$$

(iii) For the exponential distribution we consider that $n \rightarrow \infty$ and $p_i = (1-a)a^{i-1}$. Consequently $\text{Opt}(\vec{p}) = 1/(1-a)$.

$$F(\vec{p}) = \frac{1}{2} + \sum_{j=1}^{\infty} \sum_{i=1}^{\infty} \frac{(1-a)a^i a^j}{a(a^i + a^j)}$$

$$= \frac{1}{2} + \frac{(1-a)}{a} \left(\frac{a}{2(1-a)} + 2 \sum_{j=1}^{\infty} \sum_{i=1}^{\infty} \frac{a^{i+j}}{1+a^j} \right) = 1 + 2 \sum_{j=1}^{\infty} \frac{a^j}{1+a^j}.$$

Using the modified Euler-Maclaurin summation formula [5] we obtain

$$F(\vec{p}) = -\frac{2\ln 2}{\ln a} - \frac{1}{2} - \frac{\ln a}{24} + O(\ln^3 a),$$

$$\frac{F(\vec{p})}{\text{Opt}(\vec{p})} \leq 2\ln 2 = 1.38629\dots$$

(iv) For the wedge distribution $p_i = 2(n + 1 - i)/((n + 1)n)$ and $\text{Opt}(\vec{p}) = n/3 + 2/3$. It is straightforward to compute

$$F(\vec{p}) = \frac{4(1 - \ln 2)}{3}n - H_n + \frac{5(1 - \ln 2)}{3} + O(1),$$

$$\frac{F(\vec{p})}{\text{Opt}(\vec{p})} \leq 4(1 - \ln 2) = 1.22741\dots$$

3.2. Upper bounds on $F(\vec{p})/\text{Opt}(\vec{p})$ for a class of distributions. Rivest [11] has shown that the average cost of the move to front rule is at most twice that of the optimal ordering. It has been conjectured that this bound is not tight, and indeed the worst value known for this ratio is $\pi/2$, as seen in the preceding subsection. In this subsection we derive an upper bound for the ratio, $F(\vec{p})/\text{Opt}(\vec{p})$, for the class of distributions $p_i \propto i^{-\lambda}$. Observing that $\text{Opt}(\vec{p})$ is bounded by a constant when $\lambda > 2$, we will see that over the distributions of the above form for which the average search time is not bounded, Lotka’s law is the distribution which maximizes the ratio $F(\vec{p})/\text{Opt}(\vec{p})$. Our analysis seems very good in the range $0 \leq \lambda \leq 2$, but weakens above this range.

Case (i). $0 \leq \lambda \leq 2$.

$$\frac{F(\vec{p})}{\text{Opt}(\vec{p})} = \frac{2 \sum_{i=1}^n \sum_{j=1}^i \frac{p_i p_j}{p_i + p_j}}{\sum_{i=1}^n i p_i}.$$

Taking the m th term of the numerator and denominator we have

$$\frac{F(\vec{p})}{\text{Opt}(\vec{p})} \leq \frac{2 \sum_{j=1}^m \frac{p_m p_j}{p_m + p_j}}{m p_m} = \frac{2}{m} \sum_{j=1}^m \frac{p_j}{p_m + p_j}.$$

By definition, $p_j/p_m = (m/j)^\lambda$; consequently

$$\frac{F(\vec{p})}{\text{Opt}(\vec{p})} \leq \frac{2}{m} \sum_{j=1}^m \frac{m^\lambda}{m^\lambda + j^\lambda} = \frac{2}{m} \sum_{j=1}^m \frac{1}{1 + (j/m)^\lambda}.$$

From the last expression we can easily verify that the bound is a monotonically increasing function of λ in the given range. Applying the Euler-Maclaurin summation formula and the method proposed in [5] for the evaluation of the constant, we conclude that

$$\frac{F(\vec{p})}{\text{Opt}(\vec{p})} \leq \frac{2}{m} \left(\int_0^m \frac{m^\lambda}{m^\lambda + x^\lambda} dx - \frac{1}{4} + O(m^{-\lambda}) \right)$$

$$\leq 2 \int_0^1 \frac{1}{1 + y^\lambda} dy - \frac{1}{2m} + O(m^{-1-\lambda}).$$

Using well-known definite integrals [7, 3.241, 8.370] and ignoring the negative term we obtain:

THEOREM 3.1. *Over the class of probability distributions of the form $p_i \propto i^{-\lambda}$ for $0 \leq \lambda \leq 2$,*

$$\frac{F(\vec{p})}{\text{Opt}(\vec{p})} \leq \frac{1}{\lambda} \left(\psi\left(\frac{\lambda+1}{2\lambda}\right) - \psi\left(\frac{1}{2\lambda}\right) \right),$$

where $\psi(x) = \Gamma'(x)/\Gamma(x)$ is the psi function [1].

(Note that $\text{Opt}(\vec{p})$ is not bounded in this case.) As we noted before, in the relevant range this bound is a monotonic increasing function of λ and we have

COROLLARY 3.2. *For probability distributions of the form $p_i \propto i^{-\lambda}$ such that $\text{Opt}(\vec{p})$ diverges: $F(\vec{p})/\text{Opt}(\vec{p}) \leq \pi/2$. Equality is achieved when $\lambda = 2$.*

It is interesting to note a few other values for λ and the upper bound on the ratio; see Table 1.

TABLE 1

λ	Upper bound on ratio
0 (Uniform)	1
$\frac{1}{2}$	$4(1 - \ln 2) = 1.2274\dots$
1 (Zipf)	$2\ln 2 = 1.3863\dots$
$\frac{3}{2}$	$\frac{4}{3} \left(\frac{\pi}{\sqrt{3}} - \ln 2 \right) = 1.4942\dots$
2 (Lotka)	$\frac{\pi}{2} = 1.5708\dots$

We note that these bounds are tight for $\lambda = 0, 1$ and 2 , which are the only values in the range for which we have a precise analysis.

Consider now the other case:

Case (ii). $\lambda > 2$.

Our analysis is made easier if we let $n \rightarrow \infty$; then it can be shown for this range that

$$p_i = \frac{1}{\zeta(\lambda) i^\lambda}$$

(where ζ denotes the Riemann zeta function). Thus

$$\text{Opt}(\vec{p}) = \sum_{i=1}^{\infty} i p_i = \frac{\zeta(\lambda-1)}{\zeta(\lambda)},$$

and

$$\begin{aligned} F(\vec{p}) &= \frac{1}{2} + \sum_{j=1}^{\infty} \sum_{i=1}^{\infty} \frac{p_i p_j}{p_i + p_j} \\ &= \frac{2}{\zeta(\lambda)} \sum_{j=1}^{\infty} \sum_{i=j}^{\infty} \frac{1}{i^\lambda + j^\lambda}. \end{aligned}$$

Asymptotic analysis using the Euler-Maclaurin summation formula shows that the fourth term is negative in the range $2 < \lambda < 4$; hence taking the first three terms we find an upper bound:

$$F(\vec{p}) < \frac{2}{\zeta(\lambda)} \sum_{j=1}^{\infty} \left(\frac{1}{j^{\lambda-1}} \int_1^{\infty} \frac{dx}{x^{\lambda+1}} + \frac{1}{4j^{\lambda}} + \frac{\lambda}{48j^{\lambda+1}} + \frac{\lambda(\lambda^2-4)}{5760j^{\lambda+3}} \right),$$

and finally, dividing both sides by $\text{Opt}(\vec{p})$,

$$\begin{aligned} \frac{F(\vec{p})}{\text{Opt}(\vec{p})} < \frac{1}{\lambda} \left[\frac{2\pi}{\sin(\pi/\lambda)} - \psi\left(\frac{\lambda+1}{2\lambda}\right) + \psi\left(\frac{1}{2\lambda}\right) \right] + \frac{\zeta(\lambda)}{2\zeta(\lambda-1)} \\ + \frac{\lambda\zeta(\lambda+1)}{48\zeta(\lambda-1)} + \frac{\lambda(\lambda^2-4)\zeta(\lambda+3)}{5760\zeta(\lambda-1)}. \end{aligned}$$

The limit of this function for $\lambda = 2$ is $\pi/2$. For $\lambda = 2.1$ this value is roughly 1.500. Then it decreases monotonically to $\lambda = 4$, where it is about 1.095. The bound appears to be reasonably good, although we have discarded terms and so it is not tight.

For $\lambda \geq 4$ the summation that defines $F(\vec{p})$ becomes rapidly convergent, and using simple arguments we can bound it by

$$\begin{aligned} F(\vec{p}) &\leq 1 + \frac{2}{\zeta(\lambda)(1+2^\lambda)} + \frac{2}{\lambda-2} 3^{2-\lambda}. \\ &\leq 1.21981\dots \quad \text{for } \lambda \geq 4; \end{aligned}$$

equivalently, since $\text{Opt}(\vec{p}) > 1$,

$$\frac{F(\vec{p})}{\text{Opt}(\vec{p})} \leq 1 + \frac{2}{\zeta(\lambda)(1+2^\lambda)} + \frac{2}{\lambda-2} 3^{2-\lambda}.$$

Consequently, for any $\lambda \geq 0$, if $p_i \propto i^{-\lambda}$ then $F(\vec{p})/\text{Opt}(\vec{p}) \leq \pi/2$. Note that all ‘folklore’ probability distributions (e.g., 80-20, Zipf’s, Bradford) asymptotically coincide with the above for values of λ close to 1 [6, Ch. 8].

3.3. Worst cases of the move to front rule. From the above discussion one wonders which is the worst possible case for the move to front rule. There are two interesting worst cases, one given by the distribution that maximizes $F(\vec{p})/\text{Opt}(\vec{p})$ and the second given by the one that maximizes $F(\vec{p}) - \text{Opt}(\vec{p})$. Let

$$\alpha_k = \sum_{j=1}^n \left(\frac{p_j}{p_k + p_j} \right)^2;$$

then:

THEOREM 3.3.1. *The probability distribution that maximizes the ratio follows:*

$$\frac{\alpha_k - \alpha_1}{k - 1} = \frac{F(\vec{p})}{2\text{Opt}(\vec{p})}.$$

Proof. By taking partial derivatives of $F(\vec{p})/\text{Opt}(\vec{p})$ with respect to p_k . \square

THEOREM 3.3.2. *The probability distribution that maximizes the difference $F(\vec{p}) - \text{Opt}(\vec{p})$ follows*

$$\frac{\alpha_k - \alpha_1}{k - 1} = \frac{1}{2}.$$

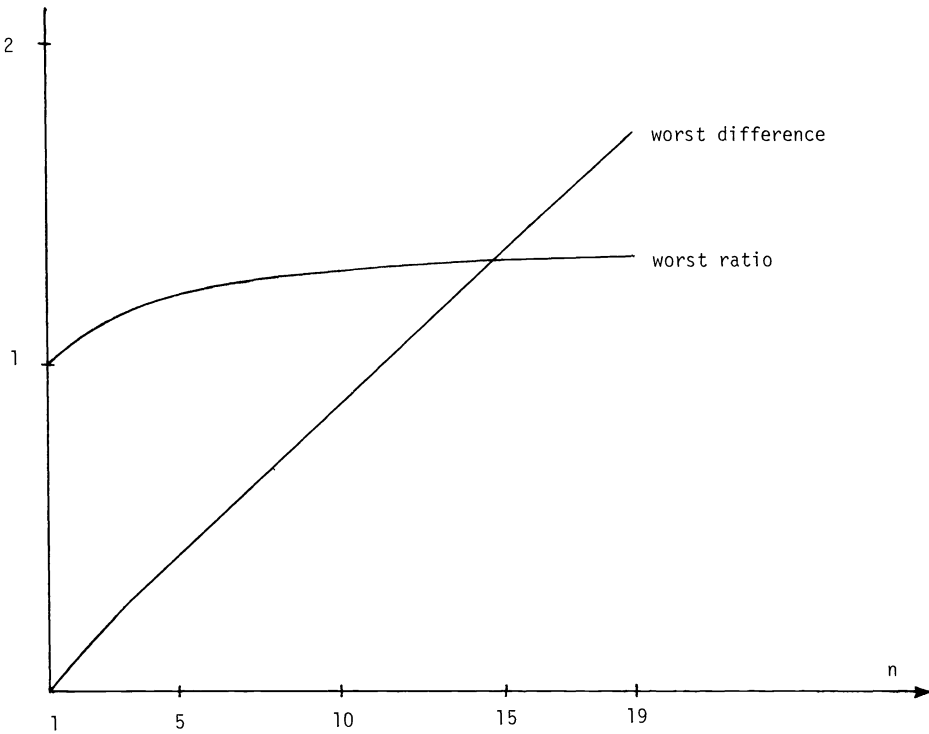


Fig. 5. Worst ratio and worst difference for move to front rule.

Proof. By taking partial derivatives of $F(\vec{p}) - \text{Opt}(\vec{p})$ with respect of p_k . \square

Unfortunately we do not know an explicit form for the worst cases, although we can compute the distributions for small n . The two graphs of Fig. 5 show the worst ratio and the worst difference for small values of n .

3.4. Arbitrary moments of the move to front rule. A closed form for the average cost of the memory-free move to front heuristic is given by Rivest [11] and used extensively in this paper. Also of interest are higher moments of this value and of course the variance, which can be derived as follows.

Let B_{ij} be the random variable defined by

$$B_{ij} = \begin{cases} 1 & \text{if record } i \text{ precedes record } j, \\ 0 & \text{otherwise.} \end{cases}$$

Then let $E[B_{ij}] = b(i, j) = p_i / (p_i + p_j)$,

$$E[(\text{accesses})^m] = E\left[\sum_{j=1}^n p_j (\sum_{i \neq j} B_{ij} + 1)^m\right] = \sum_{j=1}^n p_j E\left[(\sum_{i \neq j} B_{ij} + 1)^m\right].$$

Let $S_i(m) = \sum_{i \geq j} \binom{m}{j}$. We derive

$$E[(\text{accesses})^m] = \sum_{j=1}^n p_j \left[1 + S_1(m) E\left[\sum_{i \neq j} B_{ij}\right] + S_2(m) E\left[\sum_{i, k \neq j} B_{ij} B_{kj}\right] + \dots \right].$$

Finally, using the expression for the probability of a given permutation [8, eq. 3] we conclude

THEOREM 3.3. *The m th moment of the distribution is given by*

$$\begin{aligned}
 E[(\text{accesses})^m] &= 1 + 2S_1(m) \sum_{i < j} \frac{p_i p_j}{p_i + p_j} \\
 &+ 4S_2(m) \sum_{i < k < j} \frac{p_i p_k p_j}{p_i + p_k + p_j} \left(\frac{1}{p_j + p_k} + \frac{1}{p_i + p_j} + \frac{1}{p_i + p_k} \right) + \dots \\
 &+ 2r! S_r(m) \sum_{i_1 < i_2 < \dots < i_r < i_{r+1}} \frac{p_{i_1} p_{i_2} \dots p_{i_{r+1}}}{p_{i_1} + p_{i_2} + \dots + p_{i_{r+1}}} I(p_{i_1}, p_{i_2}, \dots, p_{i_{r+1}}),
 \end{aligned}$$

where

$$I(a_1, a_2, \dots, a_\ell) = \sum \frac{1}{a_{i_1} + a_{i_2}} \frac{1}{a_{i_1} + a_{i_2} + a_{i_3}} \dots \frac{1}{a_{i_1} + a_{i_2} + \dots + a_{i_{\ell-1}}},$$

the summation being over all permutations $i_1, i_2, \dots, i_{\ell-1}$ of the integers $1 \dots \ell$.

In particular, for $m = 1$,

$$F(\vec{p}) = 1 + 2 \sum_{i < j} \frac{p_i p_j}{p_i + p_j},$$

and for $m = 2$,

$$\begin{aligned}
 E[(\text{accesses})^2] &= 1 + 6 \sum_{i < j} \frac{p_i p_j}{p_i + p_j} \\
 &+ 4 \sum_{i < k < j} \frac{p_i p_k p_j}{p_i + p_k + p_j} \left(\frac{1}{p_j + p_k} + \frac{1}{p_i + p_j} + \frac{1}{p_i + p_k} \right).
 \end{aligned}$$

From this we can demonstrate

COROLLARY 3.4. *The variance of the move to front heuristic is*

$$\begin{aligned}
 \text{var}(\vec{p}) &= (2 - F(\vec{p}))(F(\vec{p}) - 1) \\
 &+ 4 \sum_{i < k < j} \frac{p_i p_k p_j}{p_i + p_k + p_j} \left(\frac{1}{p_i + p_j} + \frac{1}{p_j + p_k} + \frac{1}{p_k + p_i} \right).
 \end{aligned}$$

The form of this value is different from the expression for the variance given by McCabe [10].

4. Conclusion. We have demonstrated a technique for maintaining self-organizing linear files in near optimal order without significant memory requirements. Although the analysis given can probably be tightened, it is sufficient to demonstrate the value of the approach. The behavior of the simple move to front heuristic has been analyzed for a class of distributions. It is shown that over all distributions in this class the ratio of the cost of the move to front scheme to that of optimal ordering is at most $\pi/2$, and that this bound can be achieved. We conjecture that this is the maximum value this ratio can achieve over any class of distributions whose optimal search cost is unbounded.

5. Acknowledgments. The authors are grateful to one of the anonymous referees for a detailed and very helpful report, and to Brian Finch for his assistance in typesetting.

REFERENCES

- [1] M. ABRAMOWITZ and I. A. STEGUN, *Handbook of Mathematical Functions*, Dover Publications, New York, 1964.
- [2] B. ALLEN and J. I. MUNRO, *Self-organizing search trees*, J. Assoc. Comput. Mach. , 25(1978), pp. 526-535.
- [3] J. R. BITNER, *Heuristics that dynamically organize data structures*, this Journal, 8(1979), pp. 82-110.
- [4] N. G. DE BRUIJN, *Asymptotic Methods in Analysis*, North Holland, Amsterdam, 1970.
- [5] G. H. GONNET, *Notes on the derivation of asymptotic expressions from summations*, Information Processing Letters, 7(1978), pp. 165-169.
- [6] G. H. GONNET, *A Handbook of Algorithms and Data Structures*, Report CS-80-23, University of Waterloo, May, 1981.
- [7] I. S. GRADSHTEYN and I. M. RYZHIK, *Tables of Integrals, Series and Products*, Academic Press, New York, 1965.
- [8] W. J. HENDRICKS, *An account of self-organizing systems*, this Journal, 5(1976), pp.715-723.
- [9] D. E. KNUTH, *The Art of Computer Programming, Searching and Sorting*, vol. III, Addison-Wesley, Don Mills, Ont., 1973.
- [10] J. MCCABE, *On serial files with relocatable records*, Operations Res., 12(1965), pp. 609-618.
- [11] R. RIVEST, *On self-organizing sequential search heuristics*, Comm. ACM, 19(1976), pp. 63-67.
- [12] A.M. TENENBAUM, *Simulations of dynamic sequential search algorithms*, Comm. ACM, 21(1978), pp. 790-791.

COMPUTING SEQUENCES WITH ADDITION CHAINS*

PETER DOWNEY†, BENTON LEONG‡ AND RAVI SETHI§

Abstract. Given a sequence n_1, \dots, n_m of positive integers, what is the smallest number of additions needed to compute all m integers starting with 1? This generalization of the addition chain ($m=1$) problem will be called the *addition-sequence* problem. We show that the sequence $\{2^0, 2^1, \dots, 2^{n-1}, 2^n-1\}$ can be computed with $n+2.13\sqrt{n}+\log n$ additions, and that $n+\sqrt{n}-2$ is a lower bound. This lower bound result is applied to show that the addition-sequence problem is NP-complete.

Key words. expression evaluation, addition chains, NP-complete problems

1. Introduction. How can one compute x^n from x with fewest multiplications? Since Scholz [8] raised this problem of optimal *addition chains*, it has received considerable attention [1], [3], [5], [6], [7]. The term “addition chain” comes from the observation that computations involving multiplication and a single variable x are isomorphic to computations involving addition and the integer 1.

As yet no efficient general algorithm or expression is known for $l(n)$, the fewest additions needed to compute n starting from 1. It is known that

$$(1) \quad \log n + \log v(n) - 2.13 \leq l(n) \leq \lceil \log n \rceil + v(n) - 1,$$

where $v(n)$ is the number of 1's in the binary representation of n . (All logarithms in this paper are base 2.) The lower bound is given in [9]; the upper bound in [5]. A better upper bound of

$$\log n + (\log n / \log \log n) + o(\log n / \log \log n)$$

is given in [1]; this bound is tight for almost all n [3].

A problem posed by Knuth [5, § 4.6.3, problem 32], called here the *addition-sequence problem*, is to find the optimal number of additions to compute an arbitrary sequence $\{n_1, \dots, n_m\}$ of positive integers. The bounds in (1) have been extended to upper and lower bounds for arbitrary sequences [7].

Dobkin and Lipton [2] have considered the problem of finding addition chains for *particular* sequences of integers, showing that for a class of polynomials $p(x)$, evaluating the sequence $\{p(1), p(2), \dots, p(n)\}$ requires $n+n^{2/3-\epsilon}$ additions. In particular, the sequence $\{1^k, 2^k, \dots, n^k\}$ requires this many additions; for $k=2$ the sequence can be computed in $n+O(n/\sqrt{\log n})$ additions.

This paper contains results on both particular and arbitrary sequences. We show (Theorem 2.1) that the particular sequence $\{2^0, 2^1, \dots, 2^{n-1}, 2^n-1\}$ requires at least $n+\sqrt{n}-2$ additions, and (Theorem 2.2) that it can be computed in $n+2.13\sqrt{n}+\log n$ additions.

We then turn to the question of optimal chains for addition sequences. We show that the problem of fewest additions for an arbitrarily given sequence of integers is NP-complete (Theorems 3.1, 3.2). While there remain interesting problems in

*Received by the editors December 21, 1979, and in revised form October 10, 1980. This paper was typeset at Bell Laboratories, Murray Hill, New Jersey, using the *troff* program running under the UNIX[®] operating system. Final copy was produced on June 1, 1981.

†Department of Computer Science, University of Arizona, Tucson, Arizona 85721.

‡Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

§Bell Laboratories, Murray Hill, New Jersey 07974.

optimally evaluating particular sequences [2], the results in § 3 suggest that no general optimal algorithm will be found.

2. Lower and upper bounds. Let n be a positive integer. An *addition chain* for n is a sequence of integers

$$(2) \quad 1 = a_0, a_1, \dots, a_r = n$$

with the property that

$$(3) \quad a_i = a_j + a_k$$

for some $k \leq j < i$, for all $i = 1, 2, \dots, r$. The *length* of the chain is r .

Since the ordering of an addition chain is somewhat arbitrary, we prefer to regard (2) as represented by a directed acyclic graph (dag) having a *single* leaf node labelled 1 and a distinguished root node labelled n , called the *output node*. Each nonleaf node p will have exactly two successors (its *sons*). The label of p is the sum of its sons' labels. It will be convenient in the sequel to refer to a node by its label.

Such dags can be generalized to compute several numbers simultaneously. An *addition dag* for $\{n_1, \dots, n_m\}$ is a dag D as described above, but having m output nodes labelled n_1, \dots, n_m . The *length* of D is the number of nonleaf nodes. Note that computations may be shared when computing several integers simultaneously.

Example 2.1. The dag representing an optimal addition chain for 15, given by 1,2,3,6,12,15 is depicted in Fig. 2.1(a). 1,2,4,5,10,15 is another optimal addition chain for 15. An optimal addition dag for the *sequence* $\{1,2,4,8,15\}$ is depicted in Fig. 2.1(b). □

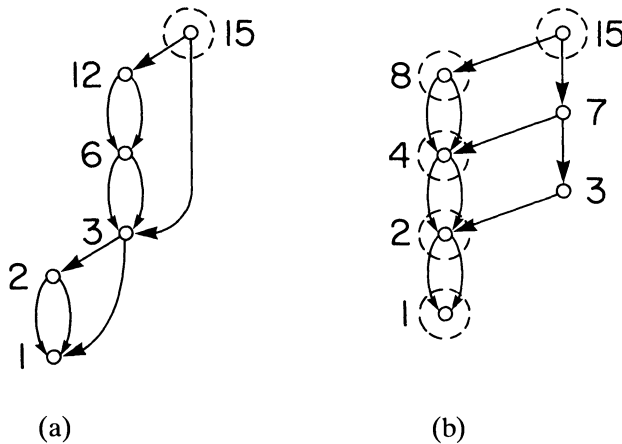


FIG. 2.1. Each nonleaf node represents the result of an addition. Output nodes are circled.

If p is a node, define $\lambda(p) = \lfloor \log p \rfloor$ as the highest power of two contained in p . A node p is a *doubling* if both sons are identical (and then p is twice the label of the son node). Suppose node p has sons q and r . Then p is a *minor node* if $\lambda(p)$ is either $\lambda(q)$ or $\lambda(r)$ and a *major node* if $\lambda(q)$ and $\lambda(r)$ are both less than $\lambda(p)$. Since $p = q+r$, it is clear that every node is either major or minor, and that for a major node p , either $\lambda(q)$ or $\lambda(r)$ must equal $\lambda(p)-1$.

An n -necklace is an addition chain computing 2^n from 2^0 using only doubling nodes. We will need the following observation on n -necklaces.

PROPOSITION 2.1. *Let D be an addition dag computing $2^0, 2^1, \dots, 2^n$, among other integers. Then, without loss of generality, we may assume that D has an n -necklace.*

Proof. Let the largest necklace in D compute 2^s for $s < n$. Since D computes all of $2^0, 2^1, \dots, 2^n$, it computes 2^{s+1} somewhere. Rearrange both edges out of 2^{s+1} to point to 2^s and leave all edges into 2^{s+1} untouched. The number of nodes in the resulting dag D' remains the same, but there is an $(s+1)$ -necklace. Repeat this transformation as often as needed. \square

The following lower bound result may be interpreted as follows: even if we have precomputed all the 1-bit numbers 2^i , computing a number with many 1-bits requires many additions just for assembly.

THEOREM 2.1. *The computation of the sequence $\{2^0, 2^1, \dots, 2^{y-1}, 2^y - 1\}$ requires at least $y + \sqrt{y} - 2$ additions, starting from 1.*

Proof. Consider an addition dag that computes the given sequence. Then the addition dag computes $n = 2^y - 1$, and by Proposition 2.1 we can assume that the dag has a $(y-1)$ -necklace. There are y nodes in the necklace: we will prove the theorem by showing that there are at least $\sqrt{y} - 1$ non-necklace nodes in the subdag below node $n = 2^y - 1$.

The subdag below node n is an addition chain computing n . This addition chain contains certain necklace nodes. If we delete all edges of the subdag which leave necklace nodes and discard all necklace nodes which are thus rendered isolated, we obtain a dag F which we shall call the *off-necklace chain*. The leaves of F are nodes of the necklace. See Fig. 2.2. (We started with a subdag in which the only leaf was labelled 1. By deleting all edges leaving necklace nodes, we have converted every necklace node into a leaf.)

Since all nodes in F are below n , there are at most y bits in the binary numeral for each node. Let us assume that every node is represented by its binary numeral, with leading zeros being supplied to assure a y -bit numeral. For example, node 2 is represented as $0^{y-2}10$.

Define $\text{gap}(b)$ to be the length of the longest substring of zeros in b . For $b = 00101^y-50$, $\text{gap}(b) = 2$. (Note that here the gap is determined by the leading zeros).

A *cut* in F is a set of edges disconnecting n from the necklace. Let C_0 , the *initial cut*, be the set of edges of F entering leaf nodes. For convenience we imagine a final cut C_r containing a special edge entering n .

Given a cut C_i , $0 \leq i \leq r$ define the nodes in F below C_i and above C_i in an obvious manner. The *gap of cut C_i* , written $\text{gap}(C_i)$, is the gap of the binary numeral obtained by taking the logical “or” of all nodes below C_i . That is,

$$\text{gap}(C_i) = \text{gap}(\vee \{z : z \text{ is below } C_i\})$$

Clearly $\text{gap}(C_r) = \text{gap}(1^y) = 0$.

Any traversal of F can be represented as a sequence of cuts starting with C_0 and ending with C_r . A cut C_i is said to *advance to cut C_{i+1} across p* by picking a node p above C_i whose out-edges are in C_i , deleting these edges, and replacing them in C_{i+1} by all edges entering p .

CLAIM 1. *If cut C_i is advanced to cut C_{i+1} , $\text{gap}(C_{i+1}) \geq \text{gap}(C_i) - 1$.*

Proof of claim. Let C_{i+1} result by advancing C_i across node p . Let n_1, n_2 be the sons of p . Then $p = n_1 + n_2$. Now the binary numeral $(n_1 + n_2) \vee (n_1 \vee n_2)$ can differ from $(n_1 \vee n_2)$ only in having (possibly) a single extra 1-bit to the left of each substring of 1's in $(n_1 \vee n_2)$. Thus $\text{gap}(p \vee n_1 \vee n_2) \geq \text{gap}(n_1 \vee n_2) - 1$. Since n_1 and n_2 are

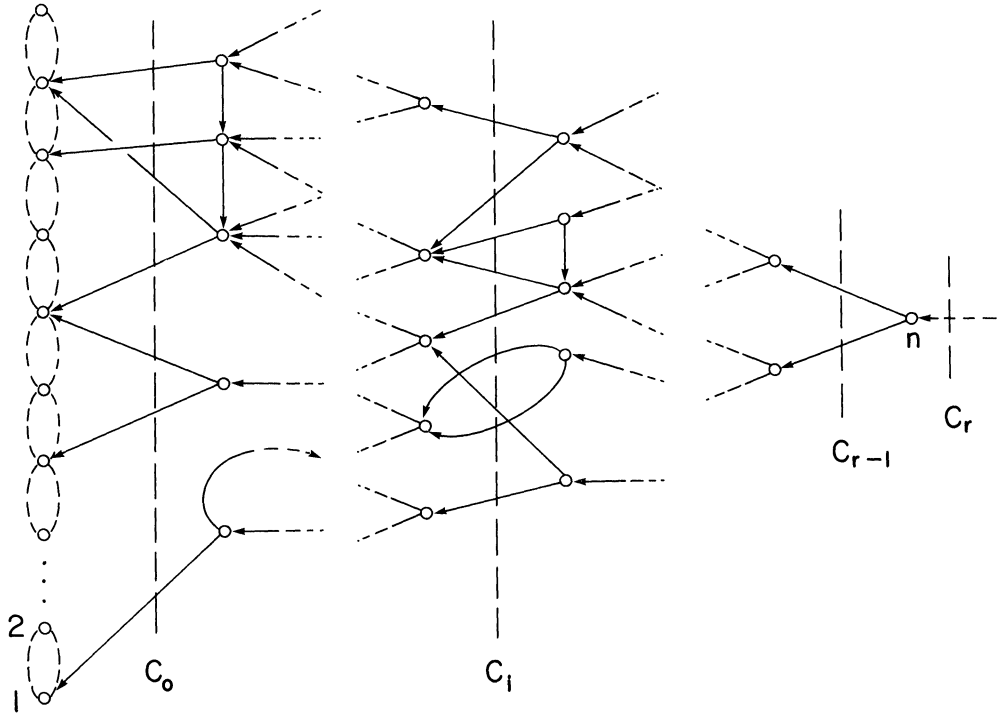


FIG. 2.2. Off-necklace chain F and some cuts.

below C_i , it follows that $\text{gap}(C_{i+1}) \geq \text{gap}(C_i) - 1$. \square

CLAIM 2. Let $g = \text{gap}(C_0)$. Then F has at least $\lfloor y/(g+1) \rfloor + (g-1)$ edges.

Proof of claim. Consider the edges of C_0 . Some edge enters node 2^0 in the necklace, since F computes the integer n which has its lowest bit set to 1. Since $g = \text{gap}(C_0)$, an edge of C_0 enters a necklace node at least every $g+1$ nodes in the sequence $2^0, 2^1, \dots, 2^{y-1}$. Therefore C_0 has $\geq \lfloor y/(g+1) \rfloor$ edges.

Consider a sequence of cuts C_0, C_1, \dots, C_r where C_i advances to C_{i+1} across some node. Each advance, except the last from C_{r-1} to C_r , involves enumerating at least one edge of F that is not in C_0 . Thus there are at least $r-1$ edges of F outside C_0 . Since $\text{gap}(C_r) = 0$ and $\text{gap}(C_0) = g$, Claim 1 implies that $r \geq g$. Thus there are at least $g-1$ edges of F outside C_0 . \square

CLAIM 3. Computing n requires at least $\sqrt{y}-1$ nodes off the necklace.

Proof of claim. By Claim 2, F has $\geq \lfloor y/(g+1) \rfloor + (g-1)$ edges. Note that

$$\lfloor y/(g+1) \rfloor + g - 1 \geq y/(g+1) + g - 1,$$

which is minimized for $g = \sqrt{y}-1$, so that F has at least $2\sqrt{y}-2$ edges. Pairing each off-necklace node in F with the two edges leaving it, we conclude that F has $\geq \sqrt{y}-1$ nodes off the necklace. \square

The last claim concludes the proof of Theorem 2.1. \square

A slight alteration of the the above proof proves:

COROLLARY 2.1. Computation of the sequence

$$\{2^0, \dots, 2^m, 2^{m+1}, \dots, 2^{m+n-1}, 2^m(2^n-1)\}$$

requires at least $m+n+\sqrt{n}-2$ additions. \square

THEOREM 2.2. *The sequence $\{2^0, \dots, 2^{n-1}, 2^n - 1\}$ can be computed with $n + 3\sqrt{n}/2 + \log n$ additions. (Note that $3/\sqrt{2} < 2.13$.) If $n = 2^{2^j}$ for some j , then $n + 2\sqrt{n} + (\log n)/2 - 3$ additions are enough.*

Proof. Compute the sequence $\{2^0, \dots, 2^{n-1}\}$ using an $n - 1$ necklace.

Case $n = 2^{2^j}$ for some j . We suggest how a node computing $2^n - 1$ can be added to the necklace, by showing how a node for $2^{16} - 1$ can be added.

Partition the 2^{2^j} bits into 2^j blocks of 2^j bits each. After $2^j - 1$ additions to set the lowest order bits of each block we get:

$$(4) \quad 0001\ 0001\ 0001\ 0001$$

For $i = 0$ we now have node p_i in (4) consisting of 2^{j+i} blocks of 2^{j-i} bits each, with the lowest order bit of each block set. The idea is to double the number of blocks and halve the number of bits in a block at each step. At the $i + 1$ st step, start with p_i , repeatedly double 2^{j-i-1} times and add the result to p_i , yielding p_{i+1} . From (4) we get

$$01\ 01\ 01\ 01\ 01\ 01\ 01\ 01.$$

The process stops with $p_j = 2^n - 1$.

There are $2^j - 1 = \sqrt{n} - 1$ additions to construct p_0 . The steps contain

$$2^{j-1} + 2^{j-2} + \dots + 2^0 = 2^j - 1 = \sqrt{n} - 1$$

doublings. There is an addition in each of the j steps, leading to a subtotal of $j = (\log n)/2$. The total of these operations is $2\sqrt{n} + (\log n)/2 - 2$, which together with the $n - 1$ additions in the necklace yields the desired upper bound for the case $n = 2^{2^j}$ for some j .

Case $2^{2^{j-1}} < n \leq 2^{2^j}$ for some j . The approach is the same as in the previous case. Let $n = k \cdot 2^j + l_0$, where $2^{j-1} \leq k \leq 2^j$ and $0 \leq l_0 < 2^j$. Start with p_0 containing $k \cdot 2^0$ blocks of 2^{j-0} bits each, and $0 \leq l_0 < 2^{j-0}$. There are $k - 1$ initial additions to set the lowest order bits in each block. For $n = 19$, we start with

$$(5) \quad 000\ 0001\ 0001\ 0001\ 0001.$$

Not only will we double the number of blocks and halve the number of bits in each block at every step, we will check the remainder l_i to see if a new block should be started. Doubling the number of blocks in (5) yields

$$000\ 01\ 01\ 01\ 01\ 01\ 01\ 01\ 01.$$

With $n = 19$, l_0 is 3, so we start a new block and set l_1 to $3 - 2 = 1$:

$$(6) \quad 0\ 01\ 01\ 01\ 01\ 01\ 01\ 01\ 01.$$

We leave it to the reader to formalize the argument.

In this case there are $k - 1$ initial additions; $2^j - 1$ doublings; j additions to split blocks; and at most j additions to start new blocks. We will be careful in bounding $k + 2^j$. Let $k = w \cdot 2^j$. From the definition of k we get $1/2 \leq w \leq 2$. Note that w need not be an integer. Now,

$$k + 2^j = w \cdot 2^j + 2^j = 2^j \sqrt{w} (\sqrt{w} + 1/\sqrt{w}) \leq \sqrt{n} (\sqrt{w} + 1/\sqrt{w}).$$

In the interval $1/2 \leq w \leq 2$, the quantity $(\sqrt{w} + 1/\sqrt{w})$ is maximized for $w = 1/2$ or $w = 2$. Therefore,

$$k + 2^j + 2^j - 2 \leq 3\sqrt{n/2} + \log n,$$

thereby proving the theorem. \square

3. Complexity of the addition-sequence problem. We leave the problem of bounding computation for sequences and consider the problem of actually finding the optimal addition dag, given an arbitrary sequence of integers. This “addition-sequence problem” will be seen to be hard; probably we shall have to settle for tight bounds on computation and not exact optimality (unless, of course, P and NP turn out to be equal).

Consider the following problem, which we shall prove to be NP-complete.

Addition-Sequence (AS)

Instance: A sequence n_1, \dots, n_m of positive integers and a positive integer L .

Question: Does there exist an addition dag D with output nodes n_1, \dots, n_m having length $\leq L$?

THEOREM 3.1. *Problem AS is in NP.*

Proof. Given an instance $I_{AS} = \langle n_1, \dots, n_m, L \rangle$, the upper bound (1) in § 1 tells us that the optimal addition dag for n_1, \dots, n_m has at most

$$B = 1 + \sum_{i=1}^m 2 \log n_i$$

nodes (the 1 is for the single leaf). Since each dag with $\leq B$ nodes has outdegree at most two, we can “guess” a dag for n_1, \dots, n_m in $O(B)$ space and time. Then in time polynomial in B we can traverse the structure from leaf to roots, labelling and counting the nodes. We succeed if the output nodes are labelled n_1, \dots, n_m and the number of nonleaf nodes is $\leq L$. □

THEOREM 3.2. *Problem AS is complete in NP.*

Proof. The proof will be by reduction of *Vertex Cover* to *Addition-Sequence*. The former problem is known to be NP-complete [4], and can be stated as follows:

Vertex Cover (VC)

Instance: A connected undirected graph $G = (V, E)$, without self edges, and a positive integer $K \leq |V|$.

Question: Is there a *vertex cover* of size K or less for G , i.e., a subset $V' \subseteq V$ such that $|V'| \leq K$ and for each edge $\{u, v\} \in E$, at least one of u or v is in V' ?

We describe the reduction of VC to AS. Given an instance of *Vertex Cover*, $I_{VC} = \langle G, K \rangle$, let $V = \{1, 2, \dots, n\}$ and construct an instance of *Addition-Sequence* as follows: the integers to be computed are

$$(7) \quad \{2^0, 2^1, 2^2, \dots, 2^{\sigma n}\} \cup \{1 + 2^{\sigma u} + 2^{\sigma v} : \{u, v\} \in E\},$$

where $\sigma = 9|E|^2$. The integer L is chosen to be $\sigma n + 1 + |E| + K$. Let us call this instance I_{AS} . It is clearly constructable from I_{VC} in time polynomial in $|V| \cdot |E|$ since each of the integers (7) are represented in binary notation.

We wish to prove that finding a dag for I_{AS} is equivalent to finding a cover for I_{VC} . More precisely, we show: I_{AS} has a solution if and only if I_{VC} has a solution.

A dag D is *optimal* for I_{AS} if it has the fewest vertices among all dags for I_{AS} .

Any connected undirected graph has a vertex cover of size $\leq |E| - 1$. Using this fact, we have the following bound.

CLAIM 1. *If D is an optimal dag computing I_{AS} it has at most $\sigma n + 2|E|$ nodes.*

Proof of claim. By explicit construction. Proposition 2.1 guarantees that D contains a σn -necklace (containing $\sigma n + 1$ nodes). Let v_1, v_2, \dots, v_t be any vertex cover. Using the necklace we can construct nodes $2^v + 1$ for each vertex v in the cover. Using these and the necklace nodes, we can compute $2^u + 2^v + 1$ for each edge $\{u, v\}$. By definition of a vertex cover, this can be done in $\sigma n + 1 + |E| + t$ nodes overall. The result follows since t can be chosen $\leq |E| - 1$. □

Henceforth we will assume that the optimal dag D computing I_{AS} has a σn -necklace of $\sigma n + 1$ nodes and no more than $2|E| - 1$ off-necklace nodes.

Consider an optimal dag D computing I_{AS} , and let $n = 2^{\sigma u} + 2^{\sigma v} + 1$ be an output node not on the necklace. If the computation of $2^{\sigma u} + 2^{\sigma v} + 1$ is as in Fig. 3.1, we will say that the computation of the output node is *normalized*.

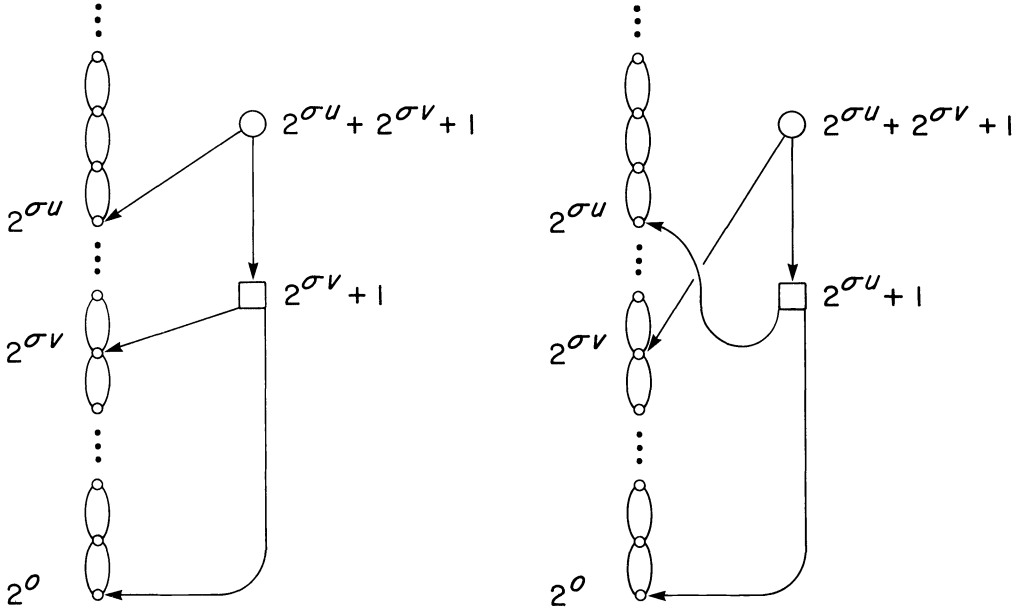


FIG. 3.1. Both output nodes shown are normalized. \circ denotes an off-necklace output node; \square a non-output node.

In Claim 2 we argue that any optimal D for I_{AS} can be transformed so that all output nodes off the necklace have been normalized. This will immediately yield the desired theorem.

CLAIM 2. Let D be an optimal dag computing I_{AS} . Then we can transform D into an optimal dag D' computing I_{AS} having a σn -necklace and with all off-necklace output nodes normalized.

Proof of claim. Let D have a σn -necklace. Claim 1 guarantees without loss of generality that D has $\leq \sigma n + 2|E|$ nodes, and so D has $\leq 2|E| - 1$ off-necklace nodes.

We show how to successively normalize the off-necklace output nodes of D beginning with the largest.

Let p be the largest labelled output node off the necklace which has yet to be normalized. Let it have sons q and r . Let $p = 2^{\sigma u} + 2^{\sigma v} + 1$ where $u > v > 0$.

Case 1. p is a major node. Without loss of generality let $\lambda(q) \leq \lambda(r)$ and $\lambda(r) = \lambda(p) - 1 = \sigma u - 1$. Since $\lambda(r) = \sigma u - 1$, r is not an output node.

Subcase 1.1. Suppose r is not on the necklace. All off-necklace output nodes s_i , $1 \leq i \leq t$, which depend upon r must have $\lambda(s_i) = \sigma u$ since p was chosen maximal. (They are not normalized, since they depend on r .) Refer to Fig. 3.2. We may delete r and replace it by $2^{\sigma u} + 1$, using this node to compute p and the s_i , $1 \leq i \leq t$ in a “normalized” fashion. Call this transformation T .

Subcase 1.2. Suppose r is on the necklace. Then $r = 2^{\sigma u - 1}$ and so $q = p - r = 2^{\sigma u - 1} + 2^{\sigma v} + 1$. Obviously q is not an off-necklace output node, nor can it be on the necklace. All output nodes s_i depending on q have $\lambda(s_i) = \sigma u$, and as in Subcase 1.1 we can delete q in favor of $2^{\sigma u} + 1$.

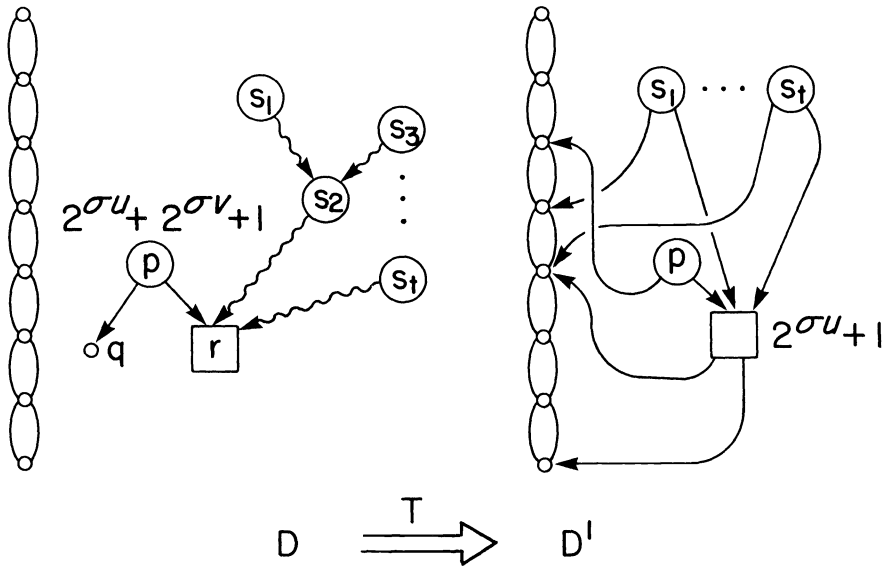


FIG. 3.2. Transformation T .

Case 2. p is a minor node. Without loss of generality let $\lambda(q) \leq \lambda(r)$ and $\lambda(r) = \lambda(p) = \sigma u$. Now r is not a necklace node, for if so then since $\lambda(r) = \sigma u$, $r = 2^{\sigma u}$ and $q = 2^{\sigma v} + 1$, showing that p is already normalized.

We also claim

(*) r is not an off-necklace output node.

To see this, suppose $r = 2^{\sigma u} + 2^{\sigma w} + 1$ is an output node. See Fig. 3.3. Then $q = 2^{\sigma v} - 2^{\sigma w}$, $v > w$. Let $v = w + \delta$, $\delta \geq 1$, so that $q = 2^{\sigma w} (2^{\sigma \delta} - 1)$.

From Corollary 2.1, the subdag reachable from q is an addition chain with at least $\sqrt{\sigma \delta} - 1$ nodes off the necklace. Since $\sigma = 9|E|^2$, D must have $\geq 3|E|\sqrt{\delta} - 1 \geq 3|E| - 1$ nodes off the necklace. This violates Claim 1. This contradiction establishes (*).

By (*), r is not an output node. All output nodes s dependent upon r have $\lambda(s) = \sigma u$. We can perform the transformation T of Case 1, deleting r , inserting $2^{\sigma u} + 1$ and normalizing p and all the nodes s , as in Fig. 3.2.

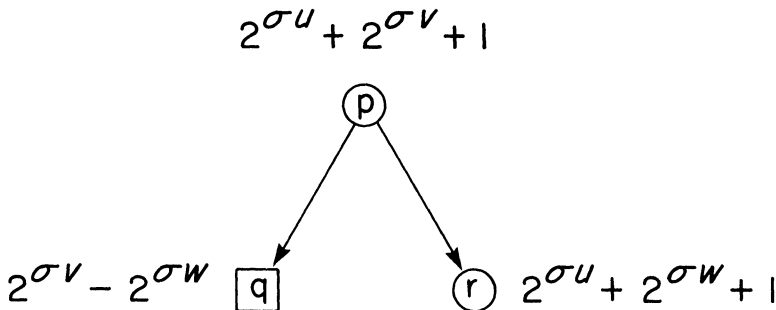


FIG. 3.3. Case 2 in the proof of Theorem 3.2.

Clearly, performing transformation T does not increase the number of nodes and does not affect output nodes already normalized. Thus by repeating T as often as necessary, an optimal normalized dag results. \square

CLAIM 3. I_{AS} has a solution if and only if I_{VC} has a solution.

Proof of claim. (→) If I_{VC} has a cover of size K then repeating the construction of Claim 1 yields a dag for I_{AS} having $\sigma n + 1 + |E| + K$ nodes.

(←) If I_{AS} can be computed by an addition dag in $\sigma n + 1 + |E| + K$ nodes, then let D be the optimal normalized dag computing I_{AS} guaranteed by Claim 2. D has $\sigma n + 1$ necklace nodes, $|E|$ off-necklace output nodes (which are roots of D) and $\leq K$ remaining nodes off the necklace of the form $2^{\sigma v} + 1$. Call this set of nodes C . By construction of I_{AS} from I_{VC} , it is clear that C forms a vertex cover of the edges in E , and thus G has a cover of size $\leq K$. \square

Claim 3 concludes the proof of Theorem 3.2. \square

4. Conclusions. In spite of the complexity results of § 3, there may still be reasons for seeking upper bounds for particular sequences of integers, as noted in [2]. Thus bounds and even explicitly optimal algorithms for sequences having very regular structure can profitably be pursued.

If one extends the notion of “chain” to allow nodes to represent other operations than addition, say addition and multiplication, this leads to a model for the evaluation of arbitrary polynomials. Bounds for this model and others are in [6]. For this model, the complexity of computing arbitrary sequences remains to be settled.

Two problems whose complexity remains open are: computing the value of $l(n)$ (the original “addition chains problem”), and the optimality of addition dags with one output node but many indeterminates at the leaves. The latter problem is closely related to polynomial evaluation over the integers.

REFERENCES

- [1] A. BRAUER, *On addition chains*, Bull. Amer. Math. Soc., 45 (1939), pp. 736-739.
- [2] D. DOBKIN AND R. J. LIPTON, *Addition chain methods for the evaluation of specific polynomials*, SIAM J. Comp., 9, (1980), pp. 121-125.
- [3] P. ERDOS, *Remarks on number theory, III: On addition chains*, Acta Arith., 6 (1960), pp. 77-81.
- [4] 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.
- [5] D. E. KNUTH, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley, New York, 1969, pp. 398-422.
- [6] R. J. LIPTON AND D. DOBKIN, *Complexity measures and hierarchies for the evaluation of integers and polynomials*, Theoret. Comp. Sci., 3 (1976), pp. 349-357.
- [7] N. PIPPENGER, *On the evaluation of powers and related problems*, Proc. IEEE 17th Ann. Symp. on Foundations of Computer Science (Oct 1976), pp. 258-263.
- [8] A. SHOLZ, *Aufgabe 253*, Jahresbericht der deutschen Mathematiker-Vereinigung, 47 (1937), pp. 41-42.
- [9] A. SCHONHAGE, *A lower bound for the length of addition chains*, Theoret. Comp. Sci., 1, (1975), pp. 1-12.

PRESERVING FUNCTIONAL DEPENDENCIES*

C. BEERI† AND P. HONEYMAN‡

Abstract. We show that functional dependency preservation can be tested in polynomial time. We show further that while finding a cover for all embedded dependencies is NP-complete, such a cover can be found in polynomial time if dependencies are preserved.

Key word. Relational database

1. Introduction. One important research area in relational database theory is the development of a logical database design methodology. Briefly, the problem can be stated as follows. Given the attributes of the database and a set of semantic constraints that describe the meaning of the database, select an appropriate database scheme, *i.e.*, a collection of relation schemes. The choice of a database scheme depends, of course, on the given constraints. So far, results have been obtained for the family of constraints known as data dependencies, especially for functional dependencies. For a survey of this area, see [5].

Bernstein [8] presents an algorithm that constructs a relational database scheme from a set of functional dependencies. One important property of the algorithm is that the relation schemes it generates embed a cover of the given functional dependencies. A different approach exploited in [14], [3], [13], [7] is to propose a formalism for the concept of a "good" database and then to find necessary and sufficient conditions for a scheme to be good. Such conditions are presented in [7], [13] for functional, multivalued, and join dependencies. For functional dependencies, the condition turns out to be as follows.

First, the scheme has to be lossless (for an explanation of this condition, see [14], [1]). Second, the relation schemes must embed a cover of the given dependencies. Another approach to database design that requires this condition to be satisfied was presented in [6]. The significance of the embedding condition is obvious; we want updates to preserve the semantic integrity of the database. That is, if a database satisfies the given dependencies, it will continue to satisfy them after being updated. If a cover for the dependencies is embedded in the relation schemes, it suffices to check that each updated relation satisfies the dependencies that are embedded in its scheme.

As remarked above, the scheme produced by Bernstein's algorithm satisfies the embedding condition. It is conceivable, however, that other design methods will produce schemes for which it is not known whether the condition is satisfied. Or we might like to know whether a scheme proposed by a user is a good one. Thus, it would be useful to have an efficient algorithm to test if a cover of a given

*Received by the editors October 3, 1979, and in revised form February 11, 1981. This paper was typeset at Bell Laboratories, Murray Hill, New Jersey, using software developed for the UNIX™ operating system. Final copy was produced on June 10, 1981.

†Department of Computer Science, The Hebrew University, Jerusalem, Israel. The work of this author was partially supported by the U.S.A.—Israel Binational Science Foundation under grant 1849/79.

‡Bell Laboratories, 600 Mountain Avenue, Murray Hill, New Jersey 07974. The work of this author was partially supported by the National Science Foundation under grants MCS 76-15255 and MCS 78-21939 at Princeton University.

set of functional dependencies is embedded in a set of relation schemes. This paper presents such an algorithm.

A naive approach is to compute F^+ , the set of dependencies implied by F , project F^+ onto the database scheme, and determine whether the resulting set of dependencies covers F . Since computing F^+ requires exponential time (and space), we must reject this tactic. A more promising approach is to compute more directly the dependencies in the embedded cover (if it exists). Using this approach we will show that if such a cover exists, then it can be computed in polynomial time.

Suppose the relation schemes do not embed a cover of the set of dependencies. Here, we would like to construct a cover for those dependencies that are embedded. Indeed, given such a cover, we could add relation schemes that embed additional dependencies until a cover for all the given dependencies is obtained. We will show that this approach is probably infeasible since finding a cover of the embedded dependencies is NP-complete.

2. Definitions and background. The reader should be familiar with the relational model of database systems, as expounded by [9], [16]. We follow their notational conventions. We refer to single attributes by upper case italic letters near the beginning of the alphabet, reserving upper case italic letters near the end of the alphabet for sets of attributes. We also drop braces surrounding sets whenever possible and allow concatenation to represent set union. Thus, $A_1A_2 \cdots A_n$, XY , and A refer to $\{A_1, A_2, \cdots, A_n\}$, $X \cup Y$, and $\{A\}$, respectively.

2.1. Relations. We assume that a finite set U of attributes is given. All sets of attributes are assumed to be subsets of U .

Each attribute A_i in U is associated with a domain of admissible values D_i . For a set $R \subseteq U$, an R -value is a map that assigns to each $A_i \in R$ a value in D_i . A *relation* on R is a finite set of R -values. A natural representation of a relation on R is a table with columns labeled by the attributes of R and rows consisting of the respective values of the constituent elements. If r is a relation on R , we say that R is the *relation scheme* of r . Intuitively, a relation scheme is a description of the format of a relation.

In this paper, we use relation scheme and set of attributes interchangeably. The set U is called the *universe*. A *database scheme* is a set of relation schemes $\mathbf{R} = \{R_1, R_2, \cdots, R_n\}$. It is usually assumed that $\bigcup_{i=1}^n R_i = U$, but this is irrelevant for our purposes.

2.2. Dependencies. An important semantic constraint is the *functional dependency*. A functional dependency $X \rightarrow Y$ is a statement that in any relation r with XY a subset of its relation scheme, any two rows with identical values in all columns labeled by X must agree in all columns labeled by Y . If this is the case, then r *satisfies* $X \rightarrow Y$. If $X \rightarrow Y$ is a functional dependency, then X *functionally determines* Y .

Other types of dependencies, e.g., multivalued and join dependencies, have been proposed [10], [15]. In this paper, only functional dependencies will be considered; thus the terms *dependency* and *determine* should be read as functional dependency and functionally determine, respectively.

From a set of dependencies, it may be logically implied that other dependencies must hold. For example, $AB \rightarrow A$ is logically implied by the empty set of

dependencies. A less trivial example is the logical implication of $A \rightarrow C$ from the dependencies $A \rightarrow B$ and $B \rightarrow C$. Formally, a set F of dependencies *implies* a dependency f if f holds in every relation in which F holds. Given a set of dependencies F , the set of all dependencies logically implied by F is called the *closure* of F , denoted F^+ . Different sets of dependencies can have the same closure. A set G is a *cover* of F if $G^+ = F^+$. Note that a cover of F is not necessarily a subset of F .

Example 1. Let $F = \{A \rightarrow B, B \rightarrow A, A \rightarrow C, AB \rightarrow C\}$ and let $G = \{A \rightarrow B, B \rightarrow A, B \rightarrow C\}$. G is a cover of F (and thus *vice versa*), but F and G are incomparable sets.

For a set of attributes X , the *closure* of X (with respect to F), denoted $\text{CLOSURE}_F(X)$, is the set of attributes that depend on X by the dependencies of F . That is,

$$\text{CLOSURE}_F(X) = \{A \mid X \rightarrow A \in F^+\}.$$

It follows directly that $X \rightarrow Y$ iff $Y \subseteq \text{CLOSURE}_F(X)$.

It is useful to be able to answer questions such as “is f implied by F ?”. This can be accomplished using the deduction rules for functional dependencies introduced by Armstrong [2].

- A1. If $Y \subseteq X$ then $X \rightarrow Y$.
- A2. If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$.
- A3. If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$.

These rules were shown to be complete in the sense that any dependency logically implied by a set of dependencies can be deduced using a finite number of applications of the rules. Thus F^+ is also the closure of F under finite application of rules A1, A2, and A3.

A useful device, which models the process of deriving a new dependency from a set of dependencies via Armstrong’s axioms, is the derivation tree [8]. If F is a set of functional dependencies, then the set of *F-based derivation trees* (F -based DT’s) is defined inductively, as follows.

DT1. If A is an attribute, then a node labeled A is an F -based DT.

DT2. If T is an F -based DT with a leaf node labeled A and $B_1 \cdots B_n \rightarrow A$ is a dependency in F , then the tree constructed by adding nodes labeled $B_1 \cdots B_n$ as children of the leaf labeled A is also an F -based DT.

DT3. T is an F -based DT only if it follows from a finite number of applications of rules DT1 and DT2.

The set of leaves of a derivation tree is the *frontier*. The *height* of a derivation tree is the length of the longest path from the root to the frontier. If the frontier of a F -based DT T is labeled by a subset of X , then T is called an *F-based X-DT*. The fundamental importance of derivation trees is the following:

PROPOSITION 1 [8]. $X \rightarrow A \in F^+$ iff there exists an F -based X -DT with root labeled A .

Using Armstrong’s rules, we can generate the dependencies that are implied by a given set of dependencies. In principle, we could decide whether $f \in F^+$ by generating F^+ . However, calculating F^+ is out of the question, since the number of dependencies in F^+ is exponential in the size of the universe. Nonetheless, we can rapidly determine membership in F^+ using the membership algorithm of [4]. This algorithm computes $\text{CLOSURE}_F(X)$ in time linear in $\|F\|$. (The number of elements in a set X is denoted $|X|$, while the number of characters needed to list X is denoted $\|X\|$.)

The algorithm for computing $\text{CLOSURE}_F(X)$ works as follows. We start by setting $\text{CLOSURE}_F(X)$ equal to the given set X . Then if $Y \rightarrow Z$ is in F and $Y \subseteq \text{CLOSURE}_F(X)$, we replace $\text{CLOSURE}_F(X)$ by $\text{CLOSURE}_F(X) \cup Z$. This step is repeated until nothing more can be added to $\text{CLOSURE}_F(X)$.

2.3. Dependency preservation. A dependency $X \rightarrow Y$ is *embedded* in a relation scheme R if $XY \subseteq R$. A set of dependencies F is embedded in a database scheme \mathbf{R} if for each $f \in F$ there is a relation scheme in \mathbf{R} in which f is embedded. A database scheme \mathbf{R} *preserves* a set of dependencies F if some cover of F is embedded in \mathbf{R} . Equivalently, \mathbf{R} preserves F if the dependencies of F^+ that are embedded in \mathbf{R} form a cover of F .

We would like F to be preserved, since this provides for efficient enforcement of constraints given as functional dependencies, as noted in § 1. If F is embedded in \mathbf{R} , determining whether \mathbf{R} preserves F is easy. However, it may be the case that F is preserved by \mathbf{R} by virtue of some unequal cover of F being embedded in \mathbf{R} . In the next section, a polynomial time algorithm is developed for testing whether F is preserved and for generating an embedded cover of F , if one exists.

3. Covers of the embedded dependencies. Our goal, given dependencies F and relation schemes \mathbf{R} , is to find a cover of F that is embedded in \mathbf{R} , if it exists. If such a cover does not exist, we would like to know that. Furthermore, we would like at least to generate a cover for the dependencies of F^+ that are embedded in \mathbf{R} . These problems are treated in this section.

3.1. A closure algorithm. As a first step, we develop an algorithm that determines the set of attributes in the closure of a set X with respect to those dependencies of F^+ that are embedded in \mathbf{R} .

DEFINITION. Let $F = \{X_1 \rightarrow Y_1, \dots, X_k \rightarrow Y_k\}$ be a set of dependencies and let $\mathbf{R} = \{R_1, \dots, R_n\}$ be a database scheme. The set of dependencies in F^+ that are embedded in \mathbf{R} is the *projection of F^+ onto \mathbf{R}* , denoted $F^+[\mathbf{R}]$.

Observe that $(F^+[\mathbf{R}])^+$ may properly contain $F^+[\mathbf{R}]$ (see Example 2 below), but is always contained in F^+ . Observe as well that $F^+[U]$ is equal to F^+ .

Since any set of attributes determines its CLOSURE , the dependency $(X \cap R) \rightarrow \text{CLOSURE}_F(X \cap R)$ is in F^+ , for any $X \subseteq U$ and $R \in \mathbf{R}$. By rules A1 and A3, F^+ also contains

$$(1) \quad (X \cap R) \rightarrow \text{CLOSURE}_F(X \cap R) \cap R.$$

The right side of (1) is the set of all attributes in R that depend on $X \cap R$. Note that this dependency is embedded in R . This observation yields the basic step of the closure algorithm.

Algorithm 1 computes $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$.

ALGORITHM 1.

Input: A database scheme \mathbf{R} , a set of dependencies F , and a set of attributes X .

Output: $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$.

Data structures: *depend* is a subset of U that is known to be in $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$ so far. *old_depend* is a subset of *depend*.

Method: We use the method of [4]. Attributes are added to *depend* by applying dependencies of the form of dependency (1), above. See Figure 1.

```

begin
(1)   depend ← X;
      repeat
          begin
(2)       old_depend ← depend;
(3)       for each  $R \in \mathbf{R}$  do
(4)           depend ← depend  $\cup$   $\left[ \text{CLOSURE}_F(\text{old\_depend} \cap R) \cap R \right]$ 
          end
(5)   until (old_depend = depend);
(6)   print depend
end.

```

FIG. 1. Calculating $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$.

To prove the correctness of the algorithm, we need the following lemmas.

LEMMA 1. *Algorithm 1 always terminates.*

Proof. Except for the last pass through the outer loop, at least one attribute is added to *depend* on each pass. Since *depend* is a subset of U , the loop is executed at most $|U|$ times. \square

LEMMA 2. *The set *depend* is contained in $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$.*

Proof. The proof is by induction on n , the number of iterations of the outer loop.

Basis: $n=0$. Before the first pass, *depend* = X , which is surely contained in the closure of X .

Induction: Suppose that after $n-1$ passes of the outer loop *depend* is contained in $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$, and suppose that some $A \in U$ is added to *depend* on the n th pass. By the inductive hypothesis, *old_depend* is contained in $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$, thus the dependency $X \rightarrow \text{old_depend}$ is in F^+ . Since A is added to *depend*, there is some $R \in \mathbf{R}$ such that A is in $\text{CLOSURE}_F(\text{old_depend} \cap R) \cap R$. Thus the dependency $(\text{old_depend} \cap R) \rightarrow A$ is embedded in R and is a member of F^+ , and thus a member of $F^+[\mathbf{R}]$.

Since $X \rightarrow \text{old_depend} \cap R$ and $\text{old_depend} \cap R \rightarrow A$ are implied by $F^+[\mathbf{R}]$, so is $X \rightarrow A$. Thus A is in $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$. This completes the induction and the proof. \square

LEMMA 3. *If A is in $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$, then A is added to *depend*.*

Proof. Suppose A is in $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$. Let $G = F^+[\mathbf{R}]$. Then there exists a G -based X -DT T with root labeled A . The proof that A is added to *depend* is by induction on the height of T .

Basis: T has height 1. Then $A \in X$ so $A \in \text{depend}$.

Induction: Suppose any attribute that labels the root of a G -based X -DT with height less than n is added to *depend*. Let T be a G -based X -DT with root labeled A and height n . Each node that is a child of the root is itself the root of a G -based X -DT with height less than n . Let Y be the set of labels of the children of the root node. By the inductive hypothesis each attribute in Y is added to *depend*. Furthermore, the dependency $Y \rightarrow A$ is a member of F^+ , and some scheme R contains YA .

On that pass of the outer loop when all of Y is first added to *old_depend* (i.e., the pass following that in which all of Y has been added to *depend*), the set $\text{CLOSURE}_F(\text{old_depend} \cap R) \cap R$ contains A . Therefore, A is added to *depend* on this pass. This completes the induction step and the proof. \square

In analyzing the running time of Algorithm 1 (and succeeding algorithms), set operations are assumed as primitives. This does not affect any polynomial time bound conclusions, and is a reasonable assumption if a bit vector implementation is used.

THEOREM 1. *Algorithm 1 determines $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$ and runs in time $O(\|F\| \cdot |\mathbf{R}| \cdot |U|)$.*

Proof. Termination and correctness follow from Lemmas 1, 2, and 3. For the complexity analysis, observe that the innermost statement (line 4) computes a CLOSURE. Using the algorithm of [4], we can carry out this step in $O(\|F\|)$ time. Line 4 is reached $|\mathbf{R}|$ times during each pass of the outer loop. We showed in the proof of Lemma 1 that the outer loop is traversed no more than $|U|$ times, from which the stated time bound follows. \square

By following the operation of the algorithm, we can infer the dependencies in $F^+[\mathbf{R}]$ used to compute $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$.

3.2. Finding an embedded cover of F . The machinery now exists to determine an embedded cover of F , if it exists. If, for each dependency $X \rightarrow Y$ in F , Y is in $\text{CLOSURE}_{F^+[\mathbf{R}]}(X)$, then $F^+[\mathbf{R}]$ covers F . In this section, we show how to determine a set of dependencies in $F^+[\mathbf{R}]$ that contains enough information to derive each dependency in F .

Example 2. Let $F = \{AB \rightarrow D, B \rightarrow E, E \rightarrow B\}$ and let $\mathbf{R} = \{AED, BE\}$. Using Algorithm 1, we can show that D is in $\text{CLOSURE}_{F^+[\mathbf{R}]}(AB)$, i.e.,

$$(2) \quad AB \rightarrow D \in (F^+[\mathbf{R}])^+.$$

To determine the embedded dependencies that imply (2), consider the application of Algorithm 1 to input AB .

First $AB \cap AED$ is found to be A , the closure of which is just A . At this point, *depend* is still AB . The next step is to determine $\text{CLOSURE}_F(AB \cap BE) = \text{CLOSURE}_F(B)$. Since $\text{CLOSURE}_F(B)$ contains E and $E \in BE$, we add E to *depend*. Observe that we have implicitly used the embedded dependency $B \rightarrow E$.

At this point, we reconsider relation scheme AED . Intersecting AED with *depend* now yields AE , which has $ABDE$ as its closure. Consequently, D is added to *depend*. Here, the embedded dependency $AE \rightarrow D$ has been used.

Since D was added to *depend*, the algorithm has demonstrated the truth of (2). In addition, our analysis has revealed that $B \rightarrow E$ and $AE \rightarrow D$ are the embedded dependencies that imply (2).

By applying Algorithm 1 to the left side of each $f \in F$, an embedded cover can be produced, if it exists.

ALGORITHM 2.

Input: A set of dependencies F , another dependency $f: X \rightarrow Y$, and a database scheme \mathbf{R} .

Output: A subset G of $F^+[\mathbf{R}]$ such that G implies f , if such a G exists. Otherwise abnormal termination results.

Data structures: $depend$, old_depend , and new are sets of attributes in $CLOSURE_{F^+[R]}(X)$. G is a set of dependencies in $F^+[R]$.

Method: We compute $CLOSURE_{F^+[R]}(X)$ using Algorithm 1. Each time a new attribute is added to $depend$, we add the embedded dependency that justifies this addition to G . See Figure 2.

```

begin
   $G \leftarrow \emptyset$ ;
   $depend \leftarrow X$ ;
  repeat
    begin
       $old\_depend \leftarrow depend$ ;
      for each  $R \in R$  do
        begin
           $new \leftarrow (CLOSURE_F(old\_depend \cap R) \cap R) - old\_depend$ ;
          if  $new \neq \emptyset$  then
            begin
               $G \leftarrow G \cup \{old\_depend \cap R \rightarrow new\}$ ;
               $depend \leftarrow depend \cup new$ 
            end
          end
        end
      end
    until ( $old\_depend = depend$ );
    if  $Y \subseteq depend$  then
      print  $G$ 
    else
      abort
    end
  end

```

FIG. 2. Determining an embedded cover

LEMMA 4. Let G be the output of Algorithm 2. If $f: X \rightarrow Y$ is in $F^+[R]$, then f is in G^+ .

Proof. Suppose $f \in F^+[R]$. To show that $f \in G^+$, it suffices to show that we can construct a G -based X -DT with root labeled A , for each $A \in Y$.

Since $f \in F^+[R]$, Y is contained in $CLOSURE_{F^+[R]}(X)$ (by definition). Therefore, by Lemma 3, Y is added to $depend$. It is therefore enough to show that during any iteration of the outer loop, any attribute A added to $depend$ labels the root of a G -based X -DT. This proof is by induction on n , the number of passes through the inner loop.

Basis: $n=0$. Then $A \in X$, thus a single node labeled A is a trivial G -based X -DT.

Induction: Suppose that for any attribute B added to $depend$ during $n-1$ passes of the inner loop, there exists a G -based X -DT with root labeled B . Suppose A is added to $depend$ on the n th pass. Then there exists a relation scheme R such that $old_depend \cap R \rightarrow A$ and $A \in R - old_depend$. Therefore, the dependency $old_depend \cap R \rightarrow A$ is added to G . By the inductive hypothesis, every attribute in $old_depend \cap R$ labels the root of some G -based X -DT. By starting with root labeled A and adding an arc to the root of each of these G -based DT's, we construct a G -based X -DT with root labeled A . This completes the induction step. \square

Observe that Algorithm 2 has the same time complexity as Algorithm 1. By using Algorithm 2 for each f in F , we can now produce an embedded cover of F , if it exists.

ALGORITHM 3.

Input: A set of dependencies F and a set of relation schemes \mathbf{R} .

Output: An embedded cover of F , if it exists.

Method: Algorithm 2 is applied to each $f \in F$. If abnormal termination occurs for any such f , then we terminate abnormally here. Otherwise, the output of each application of Algorithm 2 is produced as output here.

THEOREM 2. *If \mathbf{R} preserves F then Algorithm 3 finds an embedded cover of F in time $O(|F| \cdot \|F\| \cdot |\mathbf{R}| \cdot |U|)$.*

Proof. The complexity analysis is straightforward; correctness follows from Lemma 4. \square

It is interesting to ask what the output of Algorithm 3 would be if the abnormal termination condition were eliminated and F were not preserved. We might hope that the resulting output covers $F^+[\mathbf{R}]$. The next example shows that, in general, this is not the case.

Example 3. Let $F = \{A \rightarrow B, BC \rightarrow D\}$ and let $\mathbf{R} = \{ACD, BD\}$. Then Algorithm 3 produces no output, yet the dependency $AC \rightarrow D$ is embedded in \mathbf{R} . \square

Observe that the left side of any dependency produced as output by Algorithm 3 is contained in the closure of the left side of some dependency in F . Determining a collection of left sides that will lead to a cover of $F^+[\mathbf{R}]$ turns out to be a difficult problem.

3.3. A negative result. Suppose \mathbf{R} is a database scheme that does not preserve F . As noted in § 1, there might be some value in determining a small set of dependencies that is embedded in \mathbf{R} and covers $F^+[\mathbf{R}]$. To show that G covers $F^+[\mathbf{R}]$ we must show that $F^+[\mathbf{R}] \subseteq G^+$. However, the complement of this problem, determining whether there exists a dependency in $F^+[\mathbf{R}]$ not in G^+ , is NP-complete,¹ thus it is unlikely that an efficient algorithm to solve this covering problem can be found.

THEOREM 3. *Let F and G be sets of functional dependencies and let \mathbf{R} be a database scheme. Then determining whether G does not cover $F^+[\mathbf{R}]$ is NP-complete.*

Proof. The problem is easily seen to be in NP. Nondeterministically select a dependency $f: X \rightarrow Y$, verify that $f \in F^+[\mathbf{R}]$, and that $f \notin G^+$. To complete the proof, the hitting set problem will be transformed to the covering problem. Hitting set was first demonstrated to be NP-complete in [12].

Let S_1, \dots, S_n be a set of subsets of a finite set S . H is a hitting set if $|H \cap S_i| = 1$, for each i .

Hitting set is transformed to the covering problem as follows. Let the universe of attributes be $S \cup \{B_1, \dots, B_n, C\}$, where $\{B_1, \dots, B_n, C\} \cap S = \emptyset$. Let $P = \{(A, A') \mid A \text{ and } A' \text{ are in } S_i, \text{ for some } i, A \neq A'\}$. Any set containing a pair from P can not be a hitting set. We describe three sets of dependencies:

F_1 . $A \rightarrow B_i$ for each $A \in S_i$ and for each i ,

F_2 . $Q \rightarrow C$ for every pair Q in P , and

F_3 . $B_1 B_2 \dots B_n \rightarrow C$.

¹ See [11] for an exposition of NP-completeness and related topics.

F is the union of $F_1 - F_3$. The database scheme \mathbf{R} contains the relation scheme $S \cup \{C\}$ and the relation schemes AB_i for every $A \in S_i$ and for each i . Let G be $F_1 \cup F_2$.

The size of the covering problem produced is polynomial in $\|\{S_1, \dots, S_n\}\|$. Observe that since $B_1 \dots B_n \rightarrow C$ is not in $(F^+[\mathbf{R}])^+$, \mathbf{R} does not preserve F .

It remains to be shown that a hitting set exists iff G does not cover $F^+[\mathbf{R}]$. For the only if part, assume that H is a hitting set. By the dependencies in F_1 , $H \rightarrow B_1 \dots B_n$ is in F^+ . Hence, by F_3 , $H \rightarrow C \in F^+$. Since $H \rightarrow C$ is embedded in \mathbf{R} , $H \rightarrow C \in F^+[\mathbf{R}]$.

Any nontrivial G -based DT with root labeled C must correspond to a dependency of F_2 . However, G contains no dependency with A on its right side, so no dependency can be attached to a G -based DT for $Q \rightarrow C$. Thus $H \rightarrow C \notin G^+$, proving the only if part.

For the if part, suppose that some dependency $f: H \rightarrow Z$ is in $F^+[\mathbf{R}]$ but not in G^+ . By Armstrong's rules, $X \rightarrow A_1 A_2 \dots A_n$ holds iff each of the dependencies $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$ holds. We can therefore assume that Z is a single attribute D , i.e., $H \rightarrow D \in F^+[\mathbf{R}] - G^+$. Obviously, $D \notin H$.

Assume D is some $A \in S$. Since F contains no dependencies with A on the right side, the only F -based DT with A as its root is the tree consisting of a single node labeled A . Thus $X \rightarrow A \in F^+$ iff $A \in X$. It follows, since $H \rightarrow D \in F^+[\mathbf{R}]$, that if D is some A then $D \in H$, a contradiction.

Next, assume that D is B_i , for some i . By similar arguments on the DT of $H \rightarrow D$, we can show that for some $A \in S_i$, $A \in H$. But then $H \rightarrow D$ is in G^+ . D must therefore be equal to C .

The $F^+[\mathbf{R}]$ -based DT for $H \rightarrow C$ may contain only dependencies of F_1 and the dependencies of $F^+[S \cup \{C\}]$. As shown above, F^+ contains no nontrivial dependency with right side A . In particular, all nontrivial dependencies of F^+ embedded in $S \cup \{C\}$ are of the form $S' \rightarrow C$, where $S' \subseteq S$. Thus the root dependency of the DT must also be of this form. Further, since F^+ contains no nontrivial dependency with right side A , no other dependency can be attached to the nodes labeled by S' . It follows that the $F^+[\mathbf{R}]$ -based DT for $H \rightarrow C$ corresponds to a single dependency $S' \rightarrow C$, where $S' \subseteq S$. We can assume that $H = S'$. Now, if for some i , $|H \cap S_i| \geq 2$, then $H \rightarrow C \in G^+$. Hence, $|H \cap S_i| \leq 1$, for all i . However, if $|H \cap S_i| = 0$ then $H \rightarrow C$ is not in F^+ . Hence $|H \cap S_i| = 1$ for all i , i.e., H is a hitting set \square .

COROLLARY. *A polynomial time algorithm for constructing a cover of $F^+[\mathbf{R}]$ exists iff $\mathbf{P} = \mathbf{NP}$.*

Proof. Given F , G , and \mathbf{R} as in Theorem 3, we could check if G covers $F^+[\mathbf{R}]$ as follows. First construct a cover H for $F^+[\mathbf{R}]$. Then check if G covers H . Since the second step can be performed in polynomial time [4], the first step can be performed in polynomial time iff $\mathbf{P} = \mathbf{NP}$. \square .

4. Conclusions. Preserving functional dependencies is an important goal in the design of a database scheme. We have shown that determining whether a database scheme preserves dependencies can be efficiently tested. Furthermore, if dependencies are preserved, then an embedded cover can be determined in polynomial time. If dependencies are not preserved, determining whether a set of dependencies covers those that are embedded in the database scheme is NP-complete, making it unlikely that a cover of the embedded dependencies can then be found in polynomial time.

Acknowledgments. The second author gratefully acknowledges the advice and support of J. D. Ullman, and the criticism of A. O. Mendelzon and E. Sciore.

REFERENCES

- [1] A. V. AHO, C. BEERI and J. D. ULLMAN, *The theory of joins in relational databases*, Trans. Database Systems, 4 (1979), pp. 297–314.
- [2] W. W. ARMSTRONG, *Dependency structures of database relationships*, Proc. IFIP Congress, 1974, North Holland, Amsterdam, pp. 580–583.
- [3] A. K. ARORA and C. R. CARLSON, *The information preserving properties of certain relational database transformations*, Proc. 4th ACM Intl. Conf. on Very Large Data Bases, 1978, pp. 352–259.
- [4] C. BEERI and P. A. BERNSTEIN, *Computational problems related to the design of normal form relational schemas*, Trans. Database Systems, 4 (1979), pp. 30–59.
- [5] C. BEERI, P. A. BERNSTEIN and N. GOODMAN, *A sophisticate's introduction to database normalization theory*, Proc. 4th ACM Intl. Conf. on Very Large Data Bases, 1978, pp. 113–124.
- [6] C. BEERI, A. O. MENDELZON, Y. SAGIV and J. D. ULLMAN, *Equivalence of relational database schemes*, this Journal, 10 (1981), pp. 352–370.
- [7] C. BEERI and J. RISSANEN, *Faithful decompositions of relational database schemes*, Res. Rep. RJ2722, IBM Res. Lab, San Jose CA, 1980.
- [8] P. A. BERNSTEIN, *Synthesizing third normal form relations from functional dependencies*, Trans. Database Systems, 1 (1976), pp. 277–298.
- [9] E. F. CODD, *A relational model for large shared data banks*, Comm. ACM 13 (1968), pp. 377–387.
- [10] R. FAGIN, *Multivalued dependencies and a new normal form for relational databases*, Trans. Database Systems, 2 (1977), pp. 262–278.
- [11] M. R. GAREY and D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.
- [12] R. M. KARP, *Reducibility among combinatorial problems*, Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.
- [13] D. MAIER, A. O. MENDELZON, F. SADRI and J. D. ULLMAN, *Adequacy of decompositions of relational databases*, J. Comput. System Sci., 21 (1980), pp. 368–379.
- [14] J. RISSANEN, *Independent components of relations*, Trans. Database Systems, 2 (1977), pp. 317–325.
- [15] ———, *Theory of relations for databases — a tutorial survey*, Proc. 7th Symp. on Mathematical Foundations of Comp. Sci., Lecture Notes in Computer Science, 64, Springer-Verlag, New York, 1978, pp. 536–551.
- [16] J. D. ULLMAN, *Principles of Database Systems*, Computer Science Press, Potomac MD, 1980.

PARALLEL MATRIX AND GRAPH ALGORITHMS*

ELIEZER DEKEL[†], DAVID NASSIMI[‡] AND SARTAJ SAHNI[†]

Abstract. Matrix multiplication algorithms for cube connected and perfect shuffle computers are presented. It is shown that in both these models two $n \times n$ matrices can be multiplied in $O(n/m + \log m)$ time when n^2/m , $1 \leq m \leq n$, processing elements (PEs) are available. When only m^2 , $1 \leq m \leq n$, PEs are available, two $n \times n$ matrices can be multiplied in $O(n^2/m + m(n/m)^{2.61})$ time. It is shown that many graph problems can be solved efficiently using the matrix multiplication algorithms.

Key words. cube connected computer, perfect shuffle computer, parallel algorithm, matrix multiplication, graph algorithm, complexity

1. Introduction. This paper is concerned with the development of general purpose matrix multiplication and graph algorithms for single instruction stream, multiple data stream (SIMD) parallel computers (see Flynn [10] for a classification of computer models). All SIMD computers have the following characteristics:

(1) They consist of p processing elements (PEs). The PEs are indexed $0, 1, \dots, p-1$ and an individual PE may be referenced as in PE (i). Each PE is capable of performing the standard arithmetic and logical operations. In addition, each PE knows its index.

(2) Each PE has some local memory.

(3) The PEs are synchronized and operate under the control of a single instruction stream.

(4) An enable/disable mask can be used to select a subset of the PEs that are to perform an instruction. Only the enabled PEs will perform the instruction. The remaining PEs will be idle. All enabled PEs execute the same instruction. The set of enabled PEs can change from instruction to instruction.

While several SIMD models have been proposed and studied, in this paper we wish to make a distinction between the shared memory model (SMM) and the remaining models, all of which employ an interconnection network and use no shared memory. In the shared memory model, there is a common memory available to each PE. Data may be transmitted from PE (i) to PE (j) by simply having PE (i) write the data into the common memory and then letting PE (j) read it. Thus, in this model it takes only $\theta(1)$ time for one PE to communicate with another PE. Two PEs are not permitted to write into the same word of common memory simultaneously. The PEs may or may not be allowed to simultaneously read the same word of common memory. If the former is the case, then we shall say that read conflicts are permitted.

Most algorithmic studies of parallel computation have been based on the SMM [2], [4], [8], [9], [14], [15], [20], [21], [27]. This model is, however, not very realistic as it assumes that the p PEs can access any p words of memory (1 word per PE) in the same time slot. In practice, however, the memory will be divided into blocks so that no two PEs can simultaneously access words in the same block. If two or more PEs wish to access words in the same memory block, then the requests will get queued. Each PE will be served in a different time slot. Thus, in the worst case $O(p)$ time could be spent

* Received by the editors May 24, 1979, and in revised form October 10, 1980. This research was supported in part by the National Science Foundation under grant MCS 78-15455.

[†] Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455.

[‡] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60201.

transferring data to the p PEs. All the papers cited earlier ignore this and take the time for a simultaneous memory access by all PEs to be $O(1)$.

SIMD computers with restricted interconnection networks appear to be more realistic. In fact, the ILLIAC IV is an example of such a machine. There are several other such machines that are currently being fabricated. The largest of these is the massive parallel processor (MPP) designed by K. Batcher. It has $p = 16K$. A block diagram of a SIMD computer with an interconnection network is given in Fig. 1.1. Observe that there is no shared memory in this model. Hence, PEs can communicate among themselves only via the interconnection network.

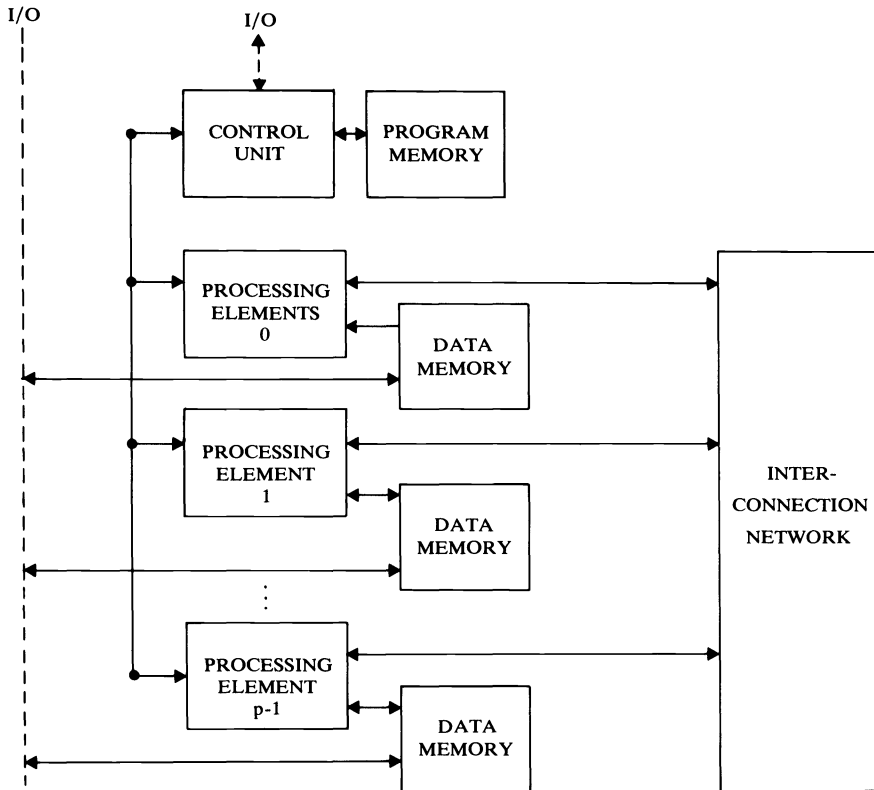


FIG. 1.1. Block diagram of an SIMD computer.

While several interconnection networks have been proposed (see [29]), we shall consider only three interconnection networks in this paper. These are: mesh, cube, and perfect shuffle. The corresponding computer models are described below. Fig. 1.2 shows the resulting interconnection pattern.

(i) *Mesh connected computer (MCC)*. In this model the PEs may be thought of as logically arranged as in a k -dimensional array $A(n_{k-1}, n_{k-2}, \dots, n_0)$, where n_i is the size of the i th dimension and $p = n_{k-1} * n_{k-2} * \dots * n_0$. The PE at location $A(i_{k-1}, \dots, i_0)$ is connected to the PEs at locations $A(i_{k-1}, \dots, i_j \pm 1, \dots, i_0)$, $0 \leq j < k$, provided they exist. Data may be transmitted from one PE to another only via this interconnection pattern. The interconnection scheme for a 16 PE MCC with $k = 2$ is given in Fig. 1.2(a).

(ii) *Cube connected computer* (CCC). Assume that $p = 2^q$ and let $i_{q-1} \dots i_0$ be the binary representation of i for $i \in [0, p - 1]$. Let $i^{(b)}$ be the number whose binary representation is $i_{q-1} \dots i_{b+1} \bar{i}_b i_{b-1} \dots i_0$, where \bar{i}_b is the complement of i_b and $0 \leq b < q$. In the cube model, PE (i) is connected to PE ($i^{(b)}$), $0 \leq b < q$. As in the mesh model, data can be transmitted from one PE to another only via the interconnection pattern. Fig. 1.2(b) shows an 8 PE CCC configuration.

(iii) *Perfect shuffle computer* (PSC). Let p, q, i and $i^{(b)}$ be as in the cube model. Let $i_{q-1} \dots i_0$ be the binary representation of i . Define SHUFFLE (i) and UNSHUFFLE (i) to, respectively, be the integers with binary representation $i_{q-2} i_{q-3} \dots i_0 i_{q-1}$ and $i_0 i_{q-1} \dots i_1$. In the perfect shuffle model, PE (i) is connected to PE ($i^{(0)}$), PE (SHUFFLE (i)) and PE (UNSHUFFLE (i)). These three connections are, respectively, called *exchange*, *shuffle*, and *unshuffle*. Once again, data transmission from one PE to another is possible only via the connection scheme. An 8 PE PSC configuration is shown in Fig. 1.2(c).

It should be noted that the MCC model requires $2k$ connections per PE, the CCC model requires $\log p$ (all logarithms in this paper are base 2) and the PSC model requires only three connections per PE. The SMM requires a large (and impractical) amount of PE to memory connections to permit simultaneous memory access by several PEs. It should also be emphasized that in any time instance, only one unit of data (say one word) can be transmitted along an interconnection line. All lines can be busy in the same time instance, however.

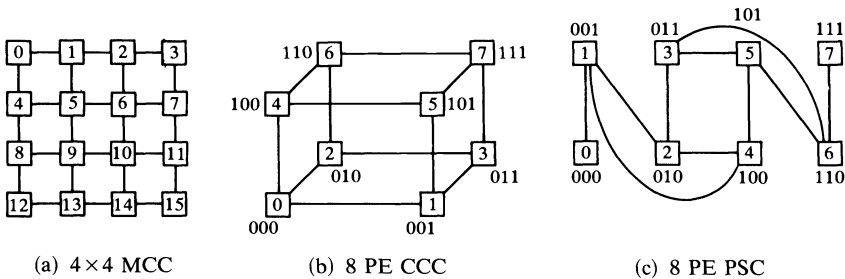


FIG. 1.2. Boxes represent PEs.

Each of the four models (including the SMM) described above has received much attention in the literature. Agerwala and Lint [2], Arjomandi [3], Csanky [8], Eckstein [9] and Hirschberg [14] have developed algorithms for certain matrix and graph problems using a SMM. Hirschberg [15], Muller and Preparata [20] and Preparata [27] have considered the sorting problem for the SMM. The evaluation of polynomials on the SMM has been studied by Munro and Paterson [21], while arithmetic expression evaluation has been considered by Brent [4] and others. Efficient algorithms to sort and perform data permutations on an MCC can be found in Thompson and Kung [32], Nassimi and Sahni [22], [23] and Thompson [31]. Thompson's algorithms [31] can also be used to perform permutations on a CCC and a PSC. Lang [18], Lang and Stone [19] and Stone [30] show how certain permutations may be performed using shuffles and exchanges. Nassimi and Sahni [25] develop fast sorting and permutation algorithms for a CCC and a PSC.

Gentleman [13] and Abelson [1] have studied the inherent complexity of parallel matrix computations. Gentleman [13], for example, has shown that at least $0.35N$ routing steps are needed to compute the product of two $N \times N$ matrices on an MCC. Also, $O(\log N)$ routes are needed to do this on a CCC or PSC.

In this paper, we are primarily concerned with the problem of multiplying two $n \times n$ matrices. The matrix multiplication problem is a very important and fundamental computational problem. The reason for this is that many other problems are best solved using matrix multiplication. In the case of parallel computations even some problems that are normally not solved using matrix multiplication turn out to be efficiently solved using matrix multiplication (or a variation of matrix multiplication). Some examples of problems in this category are:

- (a) Find the shortest distance between all pairs of vertices in a graph. This problem is traditionally solved using a dynamic programming algorithm [16].
- (b) Transitive closure. This is traditionally also solved using dynamic programming [16].
- (c) Radius, diameter and centers of a graph.
- (d) Breadth-first spanning tree. This is traditionally found using a breadth-first search.
- (e) Topological sort. This is normally done using a special graph traversal [16].
etc.

These and other graph problems that are efficiently solved using matrix multiplication are described in greater detail in § 4. In almost all of the cases considered in § 4, the application of matrix multiplication is quite straightforward. Nonetheless, we describe how matrix multiplication (or a variation) may be used in each case. This serves to highlight the difference in the thought pattern needed to arrive at an efficient parallel algorithm versus an efficient single processor algorithm.

The matrix multiplication problem for SMMs and MCCs has been extensively studied. Savage [28] presents an $O(\log n)$ algorithm to multiply two $n \times n$ matrices on a SMM with $n^3/\log n$ PEs. The problem may be easily solved in $O(n)$ time when only n^2 PEs are available. Both these algorithms are obtained from the classical $O(n^3)$ matrix multiplication algorithm. Chandra [6] has obtained an $O(\log n)$ implementation of Strassen's multiplication algorithm. His implementation runs in $O(n^{\log 7}/p)$ time on a p PE SMM computer when $p \leq n^{\log 7}/\log n$. When $p = n^{\log 7}/\log n$, the complexity of Chandra's parallel algorithm becomes $O(\log n)$. This is easily seen to be optimal as it takes $O(\log n)$ time to add n numbers optimally in parallel. Agerwala and Lint [2] have obtained an efficient algorithm to multiply Boolean matrices. Their algorithm is an implementation of the well-known four Russians algorithm.

For MCCs, Van Scoy [33] and Flynn and Kosaraju [11] have developed $O(n)$ matrix multiplication algorithms for the case when n^2 PEs are available. A very simple $O(n)$ algorithm for MCCs can be obtained from the algorithm developed by Cannon [5] for 2 dimensional MCCs with wraparound. A two dimensional MCC with wraparound is an MCC as defined here with the following additional PE connections:

- (i) Let PE (i, j) denote the PE in position (i, j) of the $n \times n$ PE array. PE $(i, 0)$ is connected to PE $(i, n - 1)$, $0 \leq i < n$.
- (ii) PE $(0, j)$ is connected to PE $(n - 1, j)$, $0 \leq j < n$.

Cannon's algorithm can be easily adapted to run in $O(n)$ time on an $n \times n$ MCC without wraparound.

We begin, in § 2, by developing the matrix multiplication algorithms for CCCs. First, algorithms are presented for the case of n^3 and n^2 PEs. The algorithm for the case of $n^2 m$ PEs, $1 \leq m \leq n$ is then obtained by combining together the algorithms for the cases of n^3 and n^2 PEs. The resulting algorithm is of complexity $O(n/m + \log m)$. Note that when $m = n/\log n$, $n^3/\log n$ PEs are used and the complexity is $O(\log n)$. When $m = 1$, n^2 PEs are used and the complexity is $O(n)$. Next, we discuss how two $n \times n$ matrices may be multiplied in $O(n^3/m^2)$ time when m^2 PEs, $1 \leq m \leq n$, are available.

This complexity is easily lowered to $O(n^2/m + m(n/m)^{2.61})$ if Pan's [26] algorithm is used in place of the classical matrix multiplication algorithm. In § 3, these results are extended to obtain PSC algorithms of the same complexity. Also, in § 3, we present an interesting algorithm to compute $(A * B)^T$ on an n^3 PE PSC. This algorithm is faster than the corresponding PSC algorithm to compute $A * B$.

As mentioned earlier, the matrix multiplication algorithms obtained can be used to obtain efficient algorithms for several graph problems. These are discussed in § 4.

2. Cube connected computer. In presenting our algorithms, we shall make use of the following conventions and notation.

(i) " \leftarrow " will be used to denote an assignment requiring data movement between PEs that are directly connected. " $:=$ " will be used to denote an assignment in which all variables in the left- and right-hand side of " $:=$ " are local to the same PE or to the control unit.

(ii) We shall use i_r to denote bit r in the binary representation of i . Thus the binary representation of i is $i_{u-1}i_{u-2} \cdots i_0$ for some u . $i^{(b)}$ will denote the number with binary representation $i_{u-1}i_{u-2} \cdots i_{b+1}\bar{i}_b i_{b-1} \cdots i_0$, where $\bar{i}_b = 1 - i_b$.

(iii) PE (k) will denote the PE with index k . Similarly, $A(k)$, $B(k)$, $C(k)$, etc., will denote memory locations or registers in PE (k).

(iv) PEs may be enabled by providing a selectivity function. This function can be placed after any statement. Operations are performed only on enabled PEs. For example, the statement

$$A(i) := B(i) + C(i), \quad (i_2 = 0)$$

has the selectivity function ($i_2 = 0$). As a result, the addition and assignment is carried out only in PEs whose index i has bit 2 equal to 0. The statement

$$A(i^{(b)}) \leftarrow B(i), \quad (i_0 = 1)$$

specifies a data route. Data from the B register of PEs with index i and $i_0 = 1$ is routed to the A register of corresponding PEs with bit b equal to $1 - i_b$.

(v) The complexity of an algorithm is the sum of the PE time needed and the communication time (i.e., the time needed to route data from PE to PE). A *unit-route* is a data transmission from a PE to another PE to which it is directly connected. In a unit-route all PEs can transmit one word of data each to one of the PEs to which they are directly connected.

2.1. CCCs with n^3 PEs. Consider a CCC with $n^3 = 2^{3q}$ PEs. Conceptually, these PEs may be regarded as arranged in an $n \times n \times n$ array pattern. If we assume, that the PEs are indexed in row major order, the PE, PE (i, j, k) in position (i, j, k) of this array has index $in^2 + jn + k$ (note that array indices are in the range $[0, n - 1]$). Hence, if $r_{3q-1} \cdots r_0$ is the binary representation of the PE position (i, j, k) then $i = r_{3q-1} \cdots r_{2q}$, $j = r_{2q-1} \cdots r_q$ and $k = r_{q-1} \cdots r_0$. Using $A(i, j, k)$, $B(i, j, k)$ and $C(i, j, k)$ to represent memory locations in PE (i, j, k), we can describe the initial condition for matrix multiplication as

$$\left. \begin{aligned} A(0, j, k) &= a_{jk} \\ B(0, j, k) &= b_{jk} \end{aligned} \right\}, \quad 0 \leq j < n, \quad 0 \leq k < n.$$

a_{jk} and b_{jk} are the elements of the two matrices to be multiplied. The desired final configuration is

$$C(0, j, k) = c_{jk}, \quad 0 \leq j < n, \quad 0 \leq k < n,$$

where

$$(2.1) \quad c_{jk} = \sum_{l=0}^{n-1} a_{jl}b_{lk}.$$

Our algorithm computes the product matrix C by directly making use of (2.1). The algorithm has three distinct phases. In the first, elements of A and B are distributed over the n^3 PEs so that we have $A(l, j, k) = a_{jl}$ and $B(l, j, k) = b_{lk}$. In the second phase the products $C(l, j, k) = A(l, j, k) * B(l, j, k) = a_{jl}b_{lk}$ are computed. Finally, in the third phase the sums $\sum_{l=0}^{n-1} C(l, j, k)$ are computed. The details are spelled out in procedure CCM1 (Algorithm 2.1). In this procedure all PE references are by PE index (recall that the index of PE (i, j, k) is $in^2 + jn + k$). Lines 1–10 implement phase 1. The loop of lines 1 to 4 copies the data initially in PE $(0, j, k)$ to the PEs, PE (i, j, k) , $1 \leq i < n$ (recall that bits $3q-1, \dots, 2q$ of a PE index yield the i value). Following this loop, we have

$$\left. \begin{array}{l} A(i, j, k) = a_{jk} \\ B(i, j, k) = b_{jk} \end{array} \right\}, \quad 0 \leq i < n.$$

Note that $A(i, j, i) = a_{ji}$. In lines 5 to 7, $A(i, j, i)$ is replicated over $A(i, j, k)$, $0 \leq k < n$ with the result that $A(i, j, k) = a_{ji}$, $0 \leq k < n$. The loop of lines 8–10 replicates $B(i, i, k) = b_{ik}$ over $B(i, j, k)$, $0 \leq j < n$. In line 11, the product $C(i, j, k) = A(i, j, k) * B(i, j, k) = a_{ji}b_{ik}$ is computed in PE (i, j, k) , $0 \leq i < n$, $0 \leq j < n$ and $0 \leq k < n$. Finally, the loop of lines 12–14 computes the sum

$$C(0, j, k) = \sum_{i=0}^{n-1} C(i, j, k) = \sum_{i=0}^{n-1} a_{ji}b_{ik} = c_{jk}.$$

The analysis of CCM1 is quite straightforward. The processor communication time in each of the last three **for** loops is that for q unit-routes. The first **for** loop requires $2q$ unit-routes. Hence, the total communication time is that for $5q = O(\log n)$ unit-routes. The PE time is clearly seen to be $O(\log n)$.

ALGORITHM 2.1.

```

line  procedure CCM1 ( $A, B, C$ ) //CCC multiplication algorithm using  $n^3$  PEs//
1      for  $l := 3q - 1$  down to  $2q$  do
2           $A(j^{(l)}) \leftarrow A(j), (j_l = 0)$ 
3           $B(j^{(l)}) \leftarrow B(j), (j_l = 0)$ 
4      end
5      for  $l := q - 1$  down to  $0$  do //replicate  $A(i, j, i)$  into  $A(i, j, *)$ //
6           $A(j^{(l)}) \leftarrow A(j), (j_l = j_{2q+l})$ 
7      end
8      for  $l := 2q - 1$  down to  $q$  do //replicate  $B(i, i, k)$  into  $B(i, *, k)$ //
9           $B(j^{(l)}) \leftarrow B(j), (j_l = j_{q+l})$ 
10     end
11      $C(j) := A(j) * B(j)$ 
12     for  $l := 2q$  to  $3q - 1$  do //add terms//
13          $C(j) \leftarrow C(j) + C(j^{(l)})$ 
14     end
15 end CCM1

```

2.2. CCCs with n^2 PEs. Two $n \times n$ matrices can be multiplied in $O(n)$ time on a CCC with n^2 PEs. It is instructive to first look at Cannon's algorithm [5] for MCCs with wraparound. This algorithm is given as procedure MCCMULT (Algorithm 2.2). $A(i, j)$,

$B(i, j)$ and $C(i, j)$ refer to memory locations (or registers) A , B and C in PE (i, j) , $0 \leq i < n$ and $0 \leq j < n$. Initially, $A(i, j) = a_{ij}$ and $B(i, j) = b_{ij}$. The algorithm consists of two identifiable phases. In phase 1 the A and B values are aligned so that the product $A(i, j) * B(i, j)$ gives one of the n terms in the sum for c_{ij} (recall that $c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$). Lines 1–4 produce the desired alignment. The elements in the i th row of A are shifted left circularly i times while the elements in the j th column of B are shifted up circularly j times. The net result is that, following this loop, $A(i, j) = a_{i, (j+i) \bmod n}$ and $B(i, j) = b_{(i+j) \bmod n, j}$. Note that $A(i, j) * B(i, j)$ is a valid product pair for $c(i, j)$.

In the second phase the A values in each row are shifted left circularly one position and B values in each column are shifted up circularly one position. This shift retains compatibility in A and B . The A and B values in each PE are multiplied to get a new term of C . Repeating this shift-multiply operation $n - 1$ times, all the terms in c_{ij} can be generated and added to obtain the product matrix (see lines 6–10 of Algorithm 2.2). The total communication time is that for $4(n - 1)$ unit-routes. The overall algorithm complexity is $O(n)$.

ALGORITHM 2.2.

```

line  procedure MCCMULT ( $A, B, C$ )
1      for  $l := 1$  to  $n - 1$  do //align  $A$  and  $B$ //
2           $A(i, j) \leftarrow A(i, (j + 1) \bmod n)$ , ( $i \geq l$ )
3           $B(i, j) \leftarrow B((i + 1) \bmod n, j)$ , ( $j \geq l$ )
4      end
5       $C(i, j) := A(i, j) * B(i, j)$  //initialize  $C$ //
6      for  $l := 1$  to  $n - 1$  do
7           $A(i, j) \leftarrow A(i, (j + 1) \bmod n)$ 
8           $B(i, j) \leftarrow B((i + 1) \bmod n, j)$ 
9           $C(i, j) := C(i, j) + A(i, j) * B(i, j)$ 
10     end
11 end MCCMULT

```

Cannon's algorithm is easily adapted to run in $O(n)$ time on an $n \times n$ MCC without wraparound. The left circular shifts of rows of A and the upward circular shifts of columns of B are easily simulated. First, we reverse and shift right the first half of each row of A to obtain

$$A'(i, j) = A(i, n - 1 - j), \quad \frac{n}{2} \leq j \leq n - 1, \quad 0 \leq i \leq n - 1.$$

Next, the first half of each column of B is reversed and shifted down to yield

$$B'(i, j) = B(n - 1 - i, j), \quad \frac{n}{2} \leq i \leq n - 1, \quad 0 \leq j \leq n - 1.$$

Each of these reversal-shift operations can be accomplished using $3n/2 - 2$ unit-routes (see Nassimi and Sahni [23] for details).

Now, a left circular shift of A is accomplished by the following instructions:

$$\begin{aligned}
 T(i, n/2) &:= A(i, n/2), \\
 A(i, j) &\leftarrow A(i, j + 1), \quad 0 \leq j \leq n - 2, \\
 A(i, n - 1) &:= A'(i, n - 1), \\
 A'(i, j) &\leftarrow A'(i, j - 1), \quad n/2 + 1 \leq j \leq n - 1, \\
 A'(i, n/2) &:= T(i, n/2).
 \end{aligned}$$

An upward circular shift of the columns of B can be obtained similarly. Since each shift takes two unit-routes, MCCMULT can be run in $8(n-1)+2*(3n/2-2)=11n-12$ unit-routes on an MCC with no wraparound.

Our CCC multiplication algorithm has the same two phases as does Cannon's. However, the initial alignment and later shifting pattern are substantially different. We assume that $n^2 = 2^{2q}$ PEs are available. Conceptually, these PEs may be viewed as in an $n \times n$ array (see Fig. 2.1). Once again, we shall use two different notations to refer to the j th PE. PE index is hardware determined. $A(j), B(j), C(j)$ will refer to memory locations or registers in PE (j). In the $n \times n$ array, we assume that the index of the PE in position (i, j) is the same as the row-major index of the position. Thus if PE (k) is at position (i, j) of the array then $k = in + j$. So, we can, without confusion, refer to the PE in position (i, j) as PE (i, j) . Similarly, we can use the notation $A(i, j), B(i, j)$ and $C(i, j)$. Our, algorithm, however, will only use the notation $A(j), B(j), C(j)$.

$i \backslash j$	0	1	2	3
0	0 0000	1 0001	2 0010	3 0011
1	4 0100	5 0101	6 0110	7 0111
2	8 1000	9 1001	10 1010	11 1011
3	12 1100	13 1101	14 1110	15 1111

FIG. 2.1. Array view of a 16 PE CCC. Each square represents a PE. The number in a square is the PE index (both decimal and binary representations are provided).

Procedure CCM2 (Algorithm 2.3) is a formal statement of our multiplication algorithm. It is assumed that A and B have been initialized so that $A(i, j) = a_{ij}$ and $B(i, j) = b_{ij}$, $0 \leq i < n, 0 \leq j < n$. The algorithm computes $C(i, j) = c_{ij} = \sum_{k=0}^{n-1} a_{ik}b_{kj}$, $0 \leq i < n, 0 \leq j < n$. Lines 1–5 obtain the initial alignment. Lines 6–13 form the product matrix by repeated shift-multiply-add steps. The loop of lines 1–5 results in the following configuration (Lemma 2.1):

$$(2.2) \quad \begin{aligned} A(i, t) &= a_{i, i \oplus t}, \\ B(i, t) &= b_{i \oplus t, t}. \end{aligned}$$

$i \oplus t$ denotes an *exclusive or* of the binary representations of i and t . Hence, the initial product of line 6 computes $C(i, t) = a_{i, i \oplus t} * b_{i \oplus t, t}$, which is one of the terms in the sum for c_{it} (see (2.1)). FUNC (line 8) is a function that controls the shifting of A and B values. It returns a bit index in the range $[0, q-1]$ (recall that $n = 2^q$). Note that the q least significant bits in a PE index (using the one-dimensional indexing scheme) determines the column index for the two-dimensional indexing scheme while the q most significant bits determine the row index. Keeping this in mind, we easily see that line 10 moves the elements in A along rows while line 11 moves elements of B along columns. Thus in line 12 we always have $A(i, t) = a_{i,p}$ and $B(i, t) = b_{r,t}$ for some p and r . Lemma 2.2 shows that $r = p$ always so that we get a valid pairing in $A(i, t)$ and $B(i, t)$. From the construction of

FUNC (yet to be specified) it will follow that no a or b value can show up in the same PE more than once.

ALGORITHM 2.3.

```

line  procedure CCM2 ( $A, B, C$ )
        //multiply two  $n \times n$  matrices on a CCC with  $n^2$  PEs//
        //obtain initial alignment//
    1      for  $l := 0$  to  $q - 1$  do
    2           $k := q + l$ 
    3           $A(j) \leftarrow A(j^{(l)}), (j_k = 1)$ 
    4           $B(j) \leftarrow B(j^{(k)}), (j_l = 1)$ 
    5      end
    6       $C(j) := A(j) * B(j)$ 
    7      for  $k := 1$  to  $n - 1$  do //shift-multiply//
    8           $l := \text{FUNC}(k)$  //get a bit index  $l \in \{0, 1, \dots, q - 1\}$ //
    9           $m := q + l$ 
   10          $A(j) \leftarrow A(j^{(l)})$ 
   11          $B(j) \leftarrow B(j^{(m)})$ 
   12          $C(j) := C(j) + A(j) * B(j)$ 
   13     end
   14 end CCM2

```

Hence, each time the product $A(j) * B(j)$ is computed in line 12, a new term of (2.1) is formed. This implies that when the loop of lines 7–13 terminates, the product matrix will have been computed.

LEMMA 2.1. *When the loop of lines 1–5 of CCM2 terminates, (2.2) correctly represents the values for $A(i, t)$ and $B(i, t)$, $0 \leq i < n$, $0 \leq t < n$.*

Proof. Using the two-dimensional notation, lines 3–4 of CCM2 may be written as

$$A(i, t) \leftarrow A(i, t^{(l)}), (i_l = 1),$$

$$B(i, t) \leftarrow B(i^{(l)}, t), (t_l = 1).$$

From this, the correctness of (2.2) is easily established. \square

LEMMA 2.2. *Whenever line 12 of CCM2 is reached, $A(i, t) = a_{ir}$ and $B(i, t) = b_{rt}$ for some r (note that $j = in + t$), $0 \leq i < n$ and $0 \leq t < n$.*

Proof. Let $A^k(j) = A^k(i, t)$ be the value of $A(j) (= A(i, t))$ when line 12 is reached on iteration k of the loop of lines 7–13. Let l_k be the value assigned to l (line 8) in this iteration. From line 10 it follows that $A^k(j) = A^{k-1}(j^{(l_k)}), k \geq 1$. From Lemma 2.1, we know that $A^0(j) = A^0(i, t) = a_{i,t'}$ where $t' = i \oplus t$. Consequently, $A^k(i, t) = a_{i,r}$ where $r = t' \oplus z^{l_k, \dots, l_1}$ and z^{l_k, \dots, l_1} is obtained from 0 by successively complementing bits l_k, l_{k-1}, \dots, l_1 in the binary representation of 0 (recall that the specification of FUNC requires that $0 \leq \text{FUNC}(\cdot) < q$). The same argument shows that $B^k(i, t) = b_{s,t}$ where $s = t' \oplus z^{l_k, \dots, l_1}$ ($B^k(\cdot)$ is defined analogous to $A^k(\cdot)$). Therefore, $r = s$ for all $k, k \geq 1$. \square

We now proceed to specify FUNC. We are looking for a function that will generate a sequence of $n - 1$ integers in the range $[0, q - 1]$ such that, by successively complementing along each bit in this sequence, all elements of a row will pass through each PE in that row (and so by adding q to each element in the sequence we can get every element in a column to go through each PE in that column). In other words, in terms of the notation of Lemma 2.2, we want the sequence of integers

$$\{z^{l_1}, z^{l_2, l_1}, \dots, z^{l_{n-1}, \dots, l_1}\}$$

to be some permutation of $\{1, 2, \dots, n - 1\}$. It is very easy to arrive at a suitable FUNC if we think recursively. Divide a row into two halves. All PEs in the left half have bit $q - 1 = 0$ while those in the right half have bit $q - 1 = 1$. If we have a bit sequence S_{q-2} that can be used when the number of PEs in a row is 2^{q-1} then this sequence will cause all elements in each half of a 2^q PE row to go through each PE in that half. By complementing on bit $q - 1$ we can exchange halves and then reuse S_{q-2} to make each element go through the PEs it has not already gone through. So, we obtain

$$S_{q-1} = S_{q-2}, q - 1, S_{q-2} \quad \text{and} \quad S_0 = 0.$$

When $q = 4$, the sequence to use is

$$\begin{aligned} S_3 &= S_2, 3, S_2 \\ &= S_1, 2, S_1, 3, S_1, 2, S_1 \\ &= S_0, 1, S_0, 2, S_0, 1, S_0, 3, S_0, 1, S_0, 2, S_0, 1, S_0 \\ &= 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0. \end{aligned}$$

Observe that the recursion sequence results in a gray code and is similar to that used in the towers-of-Hanoi problem. It should be easy to see how FUNC may be incorporated into CCM2, inline, using only $O(q) = O(\log n)$ control unit memory. The time needed to generate the entire sequence is $O(n)$.

As far as the complexity of CCM2 is concerned, exactly $2(q + n - 1)$ unit-routes are needed. Both the PE and communication times are $O(n)$.

	column					
	row					
		00	01	10	11	
00		00	01	02	03	A
		00	01	02	03	B
01		10	11	12	13	A
		10	11	12	13	B
10		20	21	22	23	A
		20	21	22	23	B
11		30	31	32	33	A
		30	31	32	33	B

		00	01	10	11
00		00	01	02	03
		00	11	22	33
01		11	10	13	12
		10	01	32	23
10		22	23	20	21
		20	31	02	13
11		33	32	31	30
		30	21	12	03

(a) Initial

(b) After lines 1-5

01	00	03	02
10	01	32	23
10	11	12	13
00	11	22	33
23	22	21	20
30	21	12	03
32	33	30	31
20	31	02	13

(c) $k = 1, l = 0$

03	02	01	00
30	21	12	03
12	13	10	11
20	31	02	13
21	20	23	22
10	01	32	23
30	31	32	33
00	11	22	33

(d) $k = 2, l = 1$

02	03	00	01
20	31	02	13
13	12	11	10
30	21	12	03
20	21	22	23
00	11	22	33
31	30	33	32
10	01	32	23

(e) $k = 3, l = 0$

FIG. 2.2. Data movement in CCM2 when $n = 4$.

Example 2.1. We illustrate the working of procedure CCM2 when $n = 4$. Fig. 2.2 shows the data movement caused by CCM2. Fig. 2.2(a) gives the initial configuration. The square in row i and column j denotes PE $(i, j) = PE(4i + j)$. The first entry in each PE is its A value and the second is its B value. The example matrices have $a_{ij} = b_{ij} = ij$. For convenience, we have labeled the columns and rows by their index in binary. Figure 2.2(b) gives the $A(i, j)$ and $B(i, j)$ values following the initial alignment. As can be seen, $A(i, j) = ii \oplus j$ and $B(i, j) = i \oplus jj$ (note that by $ii \oplus j$ we mean a number with first digit i and second digit $i \oplus j$). In the shift loop (lines 7–13) the sequence of l values is 0, 1, 0. Figures 2.2(c), (d) and (e) show the new A and B values following lines 11 and 12 for each of the three iterations of the shift loop. As is evident, for each PE (i, j) the four $A(i, j), B(i, j)$ pairings in Figs. 2.2(b) to 2.2(e) yield a distinct term in sum for c_{ij} (see (2.1)). \square

2.3. CCCs with $n^2m, 1 \leq m \leq n$ PEs. By combining together algorithms CCM1 and CCM2 one can obtain an efficient parallel multiplication algorithm for CCCs with $n^2m, 1 \leq m \leq n$, PEs. We shall assume that n and m are powers of 2. So, m divides n . The matrices, A and B , to be multiplied, can each be partitioned into $m^2 n/m \times n/m$ equal sized submatrices A_{ij} and $B_{ij}, 0 \leq i < m$ and $0 \leq j < m$ (see Fig. 2.3). The product matrix C may be similarly partitioned. It is easy to see that C_{ij} is given by:

$$(2.3) \quad C_{ij} = \sum_{k=0}^{m-1} A_{ik}B_{kj}, \quad 0 \leq i < m, \quad 0 \leq j < m.$$

A_{00}	A_{01}	A_{02}	A_{03}
A_{10}	A_{11}	A_{12}	A_{13}
A_{20}	A_{21}	A_{22}	A_{23}
A_{30}	A_{31}	A_{32}	A_{33}

FIG. 2.3. Partitioning of A into 16 submatrices.

The partitioned matrices may be viewed as an $m \times m$ matrix with each element being an $n/m \times n/m$ matrix. The n^2m PEs in the CCC may be viewed as forming an $m \times n \times n$ array PE $(i, j, k), 0 \leq i < m, 0 \leq j < n$ and $0 \leq k < n$. For each fixed i , the PEs in this array form an $n \times n$ array which can be partitioned into $m^2 n/m \times n/m$ arrays $PE_{i,j,k}, 0 \leq j < m, 0 \leq k < m$. The initial configuration for matrix multiplication is given by $A_{0,j,k} = A_{jk}$ and $B_{0,j,k} = B_{jk}, 0 \leq j < m, 0 \leq k < m$. $A_{0,i,k}$ and $B_{0,i,k}$ denote the A and B registers of the PE partition $PE_{0,i,k}$. A and B can be multiplied using (2.3), CCM1 and CCM2. First, the partitioned submatrices of A and B are replicated as in lines 1 to 10 of CCM1 to obtain $A_{i,j,k} = A_{ji}$ and $B_{i,j,k} = B_{ik}, 0 \leq i < m, 0 \leq j < m$ and $0 \leq k < m$. This replication requires only $O(\log m)$ time. The product of line 11 of CCM1 is now a product of two $n/m \times n/m$ matrices A_{ji} and B_{ik} . The number of PEs available to carry out each product $A_{ji} * B_{ik}$ is n^2/m^2 (i.e., the PE partition $PE_{i,j,k}$). So, CCM2 can be used to implement line 11 of CCM1 in $O(n/m)$ time. Finally the addition loop of lines 12 to 14 (which perform the summation of (2.3)) is easily implemented in $O(\log m)$ time. The resulting algorithm is procedure CCM3. The correspondence between CCM3 and the two earlier algorithms is easily seen. Lines 1–10 of CCM3 correspond to lines 1–10 of CCM1, lines 11–23 of CCM3 correspond to lines 1–13 of CCM2, and lines 24–26 of CCM3 correspond to lines 12–14 of CCM1. CCM3 requires $O(n/m + \log m)$ unit-routes and PE time. One readily sees that when $m = 1$, CCM3 works exactly like CCM2 and when $m = n$, CCM3 makes the same data moves as does CCM1. An interesting

special case is when $m = n/\log n$. Now, only $n^3/\log n$ PEs are available and the complexity of CCM3 is still $O(\log n)$.

ALGORITHM 2.4.

```

procedure CCM3 ( $A, B, C, n, m$ ) //multiply two  $n \times n$  matrices on a CCC with
                                                     $n^2 m$  PEs//
0       $q := \log n$ ;  $h := \log m$ 
1      for  $l := 2q + h - 1$  down to  $2q$  do //copy  $A, B$ //
2           $A(j^{(l)}) \leftarrow A(j), (j_l = 0)$ 
3           $B(j^{(l)}) \leftarrow B(j), (j_l = 0)$ 
4      end
5      for  $l := q - 1$  down to  $q - h$  do //copy  $A$  submatrices//
6           $A(j^{(l)}) \leftarrow A(j), (j_l = j_{q+h+l})$ 
7      end
8      for  $l := 2q - 1$  down to  $2q - h$  do //copy  $B$  submatrices//
9           $B(j^{(l)}) \leftarrow B(j), (j_l = j_{h+l})$ 
10     end
           //Now multiply two  $n/m \times n/m$  matrices in each
                                                     $1 \times n/m \times n/m$  PE partition//
11     for  $l := q - h - 1$  down to  $0$  do //Initial alignment//
12          $k := q + l$ 
13          $A(j^{(l)}) \leftarrow A(j), (j_k = 1)$ 
14          $B(j^{(k)}) \leftarrow B(j), (j_l = 1)$ 
15     end
16      $C(j) := A(j) * B(j)$ 
17     for  $k := 1$  to  $n/m - 1$  do //shift-multiply-add//
18          $l := \text{FUNC}(k)$  //get a bit index  $l \in \{0, 1, \dots, q - h - 1\}$ //
19          $r := q + l$ 
20          $A(j) \leftarrow A(j^{(l)})$ 
21          $B(j) \leftarrow B(j^{(r)})$ 
22          $C(j) := C(j) + A(j) * B(j)$ 
23     end
           //Final add//
24     for  $l := 2q + h - 1$  down to  $2q$  do
25          $C(j) \leftarrow C(j) + C(j^{(l)})$ 
26     end
end CCM3

```

2.4. CCCs with m^2 PEs, $1 \leq m \leq n$. The partitioning strategy of § 2.3 can also be used when only m^2 , $1 \leq m \leq n$ PEs are available. In this case A and B are partitioned into A_{ij} and B_{ij} , $0 \leq i < m$ and $0 \leq j < m$ as before. The PEs are viewed as forming an $m \times m$ array PE_{ij} , $0 \leq i < m$ and $0 \leq j < m$. Initially, PE_{ij} contains the n^2/m^2 elements of A_{ij} as well as the n^2/m^2 elements of B_{ij} . The matrix product can be formed proceeding essentially as in procedure CCM2 with n and q respectively replaced by n/m and $\log(n/m)$. The routes of lines 3, 4, 10 and 11 of CCM2 now involve n^2/m^2 elements each and the product of line 6 (or 12) is a product of two $n/m \times n/m$ matrices. This product takes $O(n^3/m^3)$ time using the classical matrix multiplication algorithm. If Pan's [26] matrix multiplication algorithm is used then line 6 takes $O((n/m)^\beta)$ time where $\beta \approx 2.61$. Hence the overall complexity of CCM2 when adapted to the case of m^2 PEs is

$$O\left(\frac{n^2}{m} + m\left(\frac{n}{m}\right)^\beta\right) = O\left(\frac{n^2}{m} + \frac{n^{2.61}}{m^{1.61}}\right).$$

The interesting special case of n PEs falls into the case just discussed when $m = \sqrt{n}$ (provided of course n is a perfect square). Actually, for the case of n PEs there exists a very straightforward $O(n^2)$ algorithm for matrix multiplication. The algorithm uses the classical matrix multiplication algorithm and starts with column i of A and B in PE (i). We leave it to the reader to fill in the details of this algorithm.

3. Perfect shuffle computer.

3.1. PSC with n^3 PEs. An $O(\log n)$ multiplication algorithm for an $n^3 = 2^{3q}$ PSC can be arrived at by simulating the routes in procedure CCM1 using the technique used by Nassimi and Sahni [25]. The resulting algorithm will require $13 \log n$ unit-routes. A slightly faster algorithm (i.e., $10 \log n$ unit-routes) can be obtained for PSCs. We shall use the same three-dimensional view of PEs and the same initial configuration as was used in § 2.1. The following unit-route statements will be used:

- (i) $A(j^{(0)}) \leftarrow A(j) \cdots$ routing along the EXCHANGE connection.
- (ii) $A(\text{SHUFFLE}(j)) \leftarrow A(j) \cdots$ routing along the SHUFFLE connection.
- (iii) $A(\text{UNSHUFFLE}(j)) \leftarrow A(j) \cdots$ routing along the UNSHUFFLE connection.

The basic steps in the PSC multiplication algorithm are the same as those used in CCM1. Their implementation is somewhat different. Procedure PSCM1 multiplies two $n \times n$ matrices on a PSC with n^3 PEs. It uses only $10 \log n$ routes. The algorithm is self explanatory and we shall not explain it further. The initial configuration is $A(0, i, t) = a_{i,t}$ and $B(0, i, t) = b_{i,t}$.

ALGORITHM 3.1.

```

line  procedure PSCM1 ( $A, B, C$ ) //PSC multiplication algorithm,  $n^3$  PEs//
1      for  $b := 0$  to  $q - 1$  do //copy  $A, B$ //
2           $A(\text{SHUFFLE}(i)) \leftarrow A(i)$ 
3           $B(\text{SHUFFLE}(i)) \leftarrow B(i)$ 
4           $A(j^{(0)}) \leftarrow A(j), (j_0 = 0)$ 
5           $B(j^{(0)}) \leftarrow B(j), (j_0 = 0)$ 
6      end //now we have  $A(i, t, p) = a_{i,t}$  and  $B(i, t, p) = b_{i,t}$ //
7      for  $b := 0$  to  $q - 1$  do
8           $A(\text{SHUFFLE}(i)) \leftarrow A(i)$ 
9      end //now we have  $A(i, t, p) = a_{p,i}$  and  $B(i, t, p) = b_{i,t}$ //
10      $C(j) := A(j) * B(j)$  //now  $C(i, t, p) = a_{p,i} * b_{i,t}$ //
11     for  $b := 0$  to  $q - 1$  do
12          $C(\text{SHUFFLE}(i)) \leftarrow C(i)$ 
13          $C(j) \leftarrow C(j) + C(j^{(0)})$ 
14     end //now  $C(t, p, i) = c_{p,t}, 0 \leq i < n$ // //In particular,  $C(t, p, t) = c_{p,t}$ //
15     for  $b := 0$  to  $q - 1$  do
16          $C(\text{SHUFFLE}(i)) \leftarrow C(i)$ 
17          $C(j^{(0)}) \leftarrow C(j), (j_0 = j_q = 1)$ 
18     end //now  $C(p, t, 0) = c_{p,t}$ //
19     for  $b := 0$  to  $q - 1$  do
20          $C(\text{UNSHUFFLE}(i)) \leftarrow C(i)$ 
21     end //now  $C(0, p, t) = c_{p,t}$ //
22     end PSCM1

```

It is important to note that PSCM1 uses only twice as many unit-routes as does CCM1. Recall that each PE in a PSC is connected to only 3 PEs while in a CCC each PE is connected to $\log n$ other PEs.

We now momentarily deviate from the matrix multiplication problem and consider the related problem of computing $C = (A * B)^T$ (i.e., the transpose of the product). Surprisingly, we can compute $(A * B)^T$ using fewer unit-routes (only $8 \log n$) than used by PSCM1 to compute $A * B$. The algorithm to compute $(A * B)^T$ can be obtained from PSCM1 by replacing lines 15–21 by the following code:

```

for  $b := 0$  to  $q - 1$  do
   $C(\text{UNSHUFFLE}(i)) \leftarrow C(i)$ 
end

```

3.2. PSCs with $n^2 m$, $1 \leq m \leq n$ PEs. Procedure CCM3 can be adapted to run on an $n^2 m$ PE PSC in time $O(n/m + \log n)$, $1 \leq m \leq n$. Lines 0–16 and 24–26 are easily implemented using $O(q + h) = O(\log n)$ shuffles, unshuffles, and exchanges (see the simulation method described in [25]). A straightforward adaptation of lines 17 to 23 would require $O(n/m \log n)$ unit-routes as the data routing of lines 20 and 21 would require $O(\log n)$ shuffles and unshuffles, and one exchange each time. A faster implementation of lines 17 to 23 can, however, be obtained. This requires us to rearrange the A s and B s so that bits l and r (lines 20 and 21 of CCM3) are adjacent. Define $\beta(j)$ to be the following permutation of the binary bits j_{2q+h-1}, \dots, j_0 in the representation of any PE index j :

$$\begin{aligned} \beta(j) &= \beta(j_{2q+h-1} \dots j_0) \\ &= j_{2q+h-1} \dots j_{2q-h} j_{q-1} \dots j_{q-h} j_{2q-h-1} j_{q-h-1} j_{2q-h-2} j_{q-h-2} \dots j_q j_0. \end{aligned}$$

Observe that line 20 of CCM3 exchanges A values only along bits $q - h - 1, \dots, 0$ while line 21 exchanges B values only along bits $2q - h - 1, \dots, q$. If preceding the loop of lines 17 to 23, A and B are permuted according to β , then lines 20 and 21 can be replaced by

$$\begin{aligned} A(j^{(2l)}) &\leftarrow A(j), \\ B(j^{(2l+1)}) &\leftarrow B(j). \end{aligned}$$

When the loop of lines 17 to 23 terminates, the C values will have to be permuted according to β^{-1} to get back the configuration currently obtained by CCM3. So, an equivalent code for lines 17 to 23 is

```

17.1'   $A(\beta(j)) \leftarrow A(j)$ 
17.2'   $B(\beta(j)) \leftarrow B(j)$ 
18'    for  $k := 1$  to  $n/m - 1$  do
19'       $l := \text{FUNC}(k)$ 
20'       $A(j^{(2l)}) \leftarrow A(j)$ 
21'       $B(j^{(2l+1)}) \leftarrow B(j)$ 
22'       $C(j) := C(j) + A(j) * B(j)$ 
23'    end
23.1'   $C(j) \leftarrow C(\beta(j))$ 

```

Define the initial *state* of any register to be 0. If a shuffle is performed on a register its state becomes $q - 1$. At this time bit $q - 1$ is in position 0 (a shuffle transforms from $j_{q-1} \dots j_0$ to $j_{q-2} \dots j_0 j_{q-1}$). Another shuffle changes the state to $q - 2$. The state $q - 1$ can be restored now by an unshuffle. Let POSITION($\langle A, B, C \rangle, i, j$) be a PSC algorithm that sets registers A , B , and C in state j if they were originally in state i . Clearly, this can be done using $\min\{|i - j|, 2q + h - |i - j|\}$ shuffles or unshuffles. Lines 17.1' to 23.1' are now simulated on a PSC by the following code:


```

17   $A(\beta(j)) \leftarrow A(j)$ 
18   $B(\beta(j)) \leftarrow B(j)$ 
19  STATE := 0
20  for  $k := 1$  to  $n/m - 1$  do //shift-multiply-add//
21       $l := \text{FUNC}(k)$ 
22      call POSITION ( $\langle A, B, C \rangle$ , STATE,  $2l$ )
23       $A(i^{(0)}) \leftarrow A(i)$  //simulate 20'//
24       $A(\text{UNSHUFFLE}(i)) \leftarrow A(i)$ 
           $B(\text{UNSHUFFLE}(i)) \leftarrow B(i)$ 
           $C(\text{UNSHUFFLE}(i)) \leftarrow C(i)$  //state becomes  $2l + 1$ //
25       $B(i^{(0)}) \leftarrow B(i)$  //simulate 21'//
26       $C(j) := C(j) + A(j) * B(j)$ 
27      STATE :=  $2l + 1$ 
28  end
29  call POSITION ( $C$ , STATE, 0)
30   $C(j) \leftarrow C(\beta(j))$ 

```

It is easy to see that lines 17 to 30 above will have the same effect on a PSC as will lines 17.1'-23.1' on a CCC. We now examine the complexity of lines 17 to 30. The permutations β and β^{-1} used in lines 17, 18, and 30 fall in the class of bit-permute-complement (BPC) permutations considered by Nassimi and Sahni [24]. All BPC permutations on an n^2m PE PSC can be performed in $O(\log n^2m) = O(\log n)$ time for $1 \leq m \leq n$. In the loop of lines 20 to 28 only line 22 takes more than $O(1)$ time. So, let's concentrate on this line. The sequence of bit indices generated by FUNC has the property

$$S_{t-1} = S_{t-2}, t-1, S_{t-2}.$$

Since S_{t-2} ends in 0, when $l = t-1$ in the loop, STATE = 0. Hence, the number of unshuffles, needed by the call to POSITION from line 22 is $2(t-1)$ (this assumes that A , B , and C can be routed in one step). On the next iteration $l = 0$ and STATE = $2(t-1) + 1 = 2t-1$. So, $2t-1$ shuffles are needed at line 22. Let $R(t-1)$ be the total number of shuffles and unshuffles needed in all calls to POSITION from line 22. From the preceding discussion we obtain the recurrence

$$R(t-1) = \begin{cases} 2R(t-2) + 4t - 3, & t > 1, \\ 0, & t = 1. \end{cases}$$

We may solve for R using one of the standard methods to solve recurrence equations. The solution to our recurrence equation is $R(t-1) = q * 2^{t-1} - 4(t-1) - 9 = O(2^t)$. Since, in our case, $t = q-h$, the time spent in the loop of lines 20-28 is $O(2^{q-h}) = O(n/m)$. Hence, the overall complexity of the resulting PSC algorithm is $O(n/m + \log n)$.

By choosing a slightly different function for FUNC, the number of shuffles and unshuffles needed by line 22 can be reduced by a factor of almost 2. This new FUNC generates the sequence given by

$$\begin{aligned}
S_0 &= 0, \\
S_1 &= 1, 0, 1 \quad (\text{instead of } 0, 1, 0), \\
S_{t-1} &= S_{t-2}, t-1, S_{t-2}, \quad t > 2.
\end{aligned}$$

3.3. PSCs with m^2 , $1 \leq m \leq n$ PEs. From the discussions of §§ 2.4 and 3.2 it should be clear how to obtain a PSC algorithm of complexity $O(n^2/m + m(n/m)^{2.61})$ for the case when m^2 PEs are available, $1 \leq m \leq n$.

4. Applications to graph problems. Efficient parallel algorithms for several graph problems may be obtained using the matrix multiplication algorithms of §§ 2 and 3. We discuss some of these in the following subsections. Analyses are provided only for the case of n^3 PEs. It holds for $n^3/\log n$ PEs too if CCM3 is used. The following discussion assumes we are dealing with n vertex graphs. Central to many of these applications is the repeated squares method of computing the transitive closure of an $n \times n$ boolean matrix (i.e., $A^* = (A + I)^n = (((A + I)^2)^2 \cdots)^2$). Since this requires $\log n$ matrix multiplications, transitive closures may be determined in $O(\log^2 n)$ time on n^3 PE PSCs and CCCs.

4.1. All-pairs shortest-paths. The all-pairs shortest-path matrix A is an $n \times n$ matrix such that $A(i, j)$ is the length of a shortest path from i to j in a weighted n -vertex graph. Let $A^k(i, j)$ denote the length of a shortest path from i to j going through at most k intermediate vertices. Clearly, $A(i, j) = A^n(i, j)$. Let $A^0(i, j)$ be the length of the edge $\langle i, j \rangle$ if $\langle i, j \rangle$ is in the graph. Let $A^0(i, j)$ be $+\infty$ if $\langle i, j \rangle$ is not in the graph and 0 if $i = j$. It should be easy to see that

$$A^k(i, j) = \min_l \{A^{k/2}(i, l) + A^{k/2}(l, j)\}.$$

Hence, A^n may be computed from A^0 by computing $A^2, A^4, A^8, \dots, A^n$. A^k may be computed from $A^{k/2}$ using the matrix multiplication algorithm with $+$ substituted for $*$ and \min substituted for $+$. The complexity of the resulting algorithm is $O(\log^2 n)$ on an n^3 PE PSC or CCC.

4.2. Radius, diameter and centers. The radius, diameter and centers of a graph may be trivially computed in $O(\log n)$ time (on an n^3 PE PSC or CCC) from the all-pairs shortest-path matrix (which itself requires $O(\log^2 n)$ to compute).

4.3. Median and median length. Let $d(i, j)$ be the length of a shortest path from i to j . Let $h(i)$ be the weight of vertex i . Vertex v is a *weighted median* [12] of the graph iff

$$\sum_{j=1}^n h(j)d(v, j) \leq \sum_{j=1}^n h(j)d(k, j), \quad 1 \leq k \leq n.$$

When $h(j) = 1$, $1 \leq j \leq n$, vertex v is simply a *median* of the graph. $\sum_{j=1}^n h(j)d(v, j)$ is called the *weighted median length* of the graph. For any graph these quantities can be easily computed once the shortest-path matrix A has been determined. The sum $\sum_{j=1}^n h(j)d(k, j) = \sum_{j=1}^n h(j)A(k, j)$ can be computed for all k in $O(\log n)$ time on an n^3 PE PSC and CCC. The minimum of these can also be found in this much time.

4.4. Shortest-path, breadth-first, minimum depth and least median spanning trees. A shortest-path spanning tree with root v is a spanning tree of the given graph. Its root is vertex v and the distance from v to each vertex j equals $d(v, j)$. ($d(v, j)$ is the shortest distance from v to j in the graph.) Let $R(i, j)$ be a vertex index such that $R(i, j) = 0$ or $A(i, R(i, j)) + A(R(i, j), j) = A(i, j)$. $R(i, j) = 0$ iff the shortest i to j path has no intermediate vertices. Otherwise $R(i, j)$ gives a vertex that is halfway along the i to j path (i.e., if there are p intermediate vertices then $R(i, j)$ is either the $\lfloor p/2 \rfloor$ th or $\lceil p/2 \rceil$ th vertex). Clearly, $R(i, j)$ can be computed in $O(\log^2 n)$ time along with the computation of $A(i, j)$. A shortest-path spanning tree rooted at v may be obtained

from $R(i, j)$ in $O(\log^2 n)$ time. If $R(v, j) = 0$ then j is a child of v and we may set $\text{PARENT}(j) = v$. If $R(i, j) = k$ then j is a descendent of k and we need to examine $R(k, j)$. $R(k, j)$ can be routed to PE (i, j) in $O(\log n)$ time. If $R(k, j) = 0$ then set $\text{PARENT}(j) = k$. Otherwise j is a descendent of $R(k, j)$. Since $R(i, j)$ is midway from i to j and $R(k, j)$ midway from $R(i, j)$ to j , etc., we will have to follow down at most $\log n$ R values before discovering the parent of j . Hence, all nodes will know their parents in the spanning tree in $O(\log^2 n)$ time plus the $O(\log^2 n)$ time needed to compute R initially.

A breadth-first spanning tree can be obtained in the same way as above by starting with $A^0(i, j) = 1$ if (i, j) is an edge in the graph, $A^0(i, j) = \infty$ if (i, j) not in the graph and $A^0(i, j) = 0$ if $i = j$. Note that $A(v, j)$ is the depth (less one) of node j in the breadth-first spanning tree with root v . Hence, the depth $D(v)$ of the spanning tree rooted at vertex v is $\max_j \{A(v, j)\} + 1$. A minimum depth spanning tree may be obtained in $O(\log^2 n)$ time by first computing A as for a breadth-first spanning tree, then computing $D(j)$ for all j (this takes only $O(\log n)$ time). Next, $\min(D(j))$ is computed (again only $O(\log n)$ time is needed). The j value with minimum $D(j)$ is the root of the minimum depth spanning tree. Now find the breadth-first spanning tree with root j . A shortest median spanning tree may be found similarly.

4.5. Max gain. Let G be a directed acyclic graph. Let $w(i, j) \geq 0$ be the weight of edge $\langle i, j \rangle$ (assume $w(i, j) = 0$ if $\langle i, j \rangle$ not in G). The gain on an i, j path is the product of the edge weights on that path. W is the max gain matrix iff $W(i, j)$ is the maximum gain for every pair i, j . Clearly, this matrix may be computed in $O(\log^2 n)$ time using an approach similar to that used for the all-pairs shortest-path matrix.

4.6. Topological sort and critical paths. These problems are described in [16]. Given an acyclic directed graph $G = (V, E)$, we are required to list the vertices in topological order. This can be done in $O(\log^2 n)$ time using matrix multiplication. Let $A^0(i, j) = 1$ if $\langle i, j \rangle \in E(G)$, $A^0(i, j) = -\infty$ if $\langle i, j \rangle \notin E(G)$ and $A^0(i, i) = 0$ if $0 \leq i \leq n - 1$. Use the all pairs shortest path algorithm to compute the lengths of the longest paths. This requires using max in place of the min used on the all-pairs shortest-path algorithm. Set $A(0, i) = 0$ for all i with the property that $A(t, i) \leq 0$, $0 \leq t < n$. Note that now $A(0, i) = 0$ for exactly those vertices in G that have no predecessors. Also note that $A(i, i) > 0$ for i iff G has a directed cycle. This is not possible as G is acyclic. Let i_1, i_2, \dots, i_k be the values of i for which $A(0, i) = 0$. For each j for which $A(0, j) \neq 0$, compute $A(0, j) = \max_{1 \leq p \leq k} \{A(i_p, j)\}$. This can clearly be done in $O(\log n)$ time on both an n^3 PE PSC and CCC. $A(0, j)$ gives the length of the longest path from any node with no predecessors to node j . Now sort the pairs $[A(0, j), j]$, $0 \leq j < n$ into non-decreasing order of $A(0, j)$. This can be done in $O(\log n)$ time on an n^3 PSC or CCC using the algorithm of [25]. If the pair $[A(0, j), j]$ ends up in PE $(0, k)$ then vertex j is the k th vertex in the topological order. The correctness of this statement is easily verified.

The strategy just described can also be used to determine early and late start times for tasks in an activity on edge network (see [16]). In such a network each edge represents an activity and has a length associated with it. The early start time for any activity $\langle i, j \rangle$ is the length of the longest path in the given directed acyclic graph from the start vertex, 0, to vertex i . The early start times can easily be found in $O(\log^2 n)$ time using the first part of the above strategy. Once the early start times are known, the earliest time the project can be finished is known. From this the latest start times (i.e., the time by which an activity must start so that the project length doesn't increase) can be computed in another $O(\log^2 n)$ steps. With both the early and late start times known the critical activities and critical paths are readily obtainable.

From the discussions of the preceding six subsections it should be clear that, as far as parallel computing is concerned, matrix multiplication is a very central problem. Several graph problems may be efficiently solved by a very straightforward application of matrix multiplication. In the examples we have given, the parallel way to solve a graph problem is quite different from the methods used in the best known sequential algorithms. This difference doesn't always show up. For example, the algorithm of [7] for the bottleneck (or maximum flow matrix) problem directly translates into an $O(\log^2 n)$ parallel algorithm. Similarly the algorithms obtained in [17] for the shortest cycle problem and the problem of determining if a graph has a triangle directly translate to $O(\log^2 n)$ and $O(\log n)$ matrix multiplication based parallel algorithms. Several other graph problems can be trivially solved in $O(\log^2 n)$ time on a parallel machine using matrix multiplication. We shall refrain from listing these here.

REFERENCES

- [1] H. ABELSON, *Lower bounds on information transfer in distributed computations*, Proc. 19th IEEE Symposium on Foundations of Computer Science, 1978, pp. 151–158.
- [2] T. AGERWALA AND B. LINT, *Communication in parallel algorithms for Boolean matrix multiplication*, Proc. 1978 IEEE Int. Conference on Parallel Processing, 1978, pp. 146–153.
- [3] E. ARJOMANDI, *A study of parallelism in graph theory*, Ph.D. thesis, Dept. of Computer Science, University of Toronto, December 1975.
- [4] R. P. BRENT, *The parallel evaluation of general arithmetic expressions*, J. Assoc. Comput. Mach., 21 (1974), pp. 201–206.
- [5] L. E. CANNON, *A cellular computer to implement the Kalman filter algorithm*, Ph.D. thesis, Montana State University, 1969.
- [6] A. CHANDRA, *Maximal parallelism in matrix multiplication*, IBM Tech. Rept. R C 6193, Sept. 1976.
- [7] I-NGO CHEN, *A new parallel algorithm for network flow problems*, Proc. 1974 Sagamore Computer Conference, 1974, pp. 306–307.
- [8] L. CSANSKY, *Fast parallel matrix inversion algorithms*, Proc. 6th IEEE Symposium on Foundations of Computer Science, October, 1975, pp. 11–12.
- [9] D. ECKSTEIN, *Parallel graph processing using depth-first search and breadth-first search*, Ph.D. thesis, University of Iowa, 1977.
- [10] M. J. FLYNN, *Very high speed computing systems*, Proc. IEEE, 54 (1966), pp. 1901–1909.
- [11] M. FLYNN AND R. KOSARAJU, *Processes and their interactions*, Kybernetics, 5 (1976), pp. 159–163.
- [12] H. FRANK AND I. T. FRISCH, *Communication, Transmission and Transportation Networks*, Addison-Wesley, Reading, MA, 1971.
- [13] W. M. GENTLEMAN, *Some complexity results for matrix computations on parallel processors*, J. Assoc. Comput. Mach., 25 (1978), pp. 112–115.
- [14] D. S. HIRSCHBERG, *Parallel algorithms for the transitive closure and the connected component problems*, Proc. 8th ACM Symposium on Theory of Computing, May, 1976, pp. 55–57.
- [15] D. S. HIRSCHBERG, *Fast parallel sorting algorithms*, Comm. A.C.M., 21 (1978), pp. 657–661.
- [16] E. HOROWITZ AND S. SAHNI, *Fundamentals of Data Structures*, Computer Science Press, Potomac, MD, 1976.
- [17] A. ITAI AND M. RODEH, *Finding a minimum circuit in a graph*, this Journal, 7 (1978), pp. 413–423.
- [18] T. LANG, *Interconnections between processors and memory modules using the shuffle-exchange network*, IEEE Trans. Comput., C-25 (1976), pp. 496–503.
- [19] T. LANG AND H. STONE, *A shuffle exchange network with simplified control*, Ibid., C-25, (1976), pp. 55–65.
- [20] D. E. MULLER AND F. P. PREPARATA, *Bounds to complexities of networks for sorting and for switching*, J. Assoc. Comput. Mach., 22 (1975), pp. 195–201.
- [21] I. MUNRO AND M. PATERSON, *Optimal algorithms for parallel polynomial evaluation*, JCSS, 7 (1973), pp. 189–198.
- [22] D. NASSIMI AND S. SAHNI, *Bitonic sort on a mesh-connected parallel computer*, IEEE Trans. Comput., C-28, (1979), pp. 2–7.
- [23] ———, *An optimal routing algorithm for mesh-connected parallel computers*, J. Assoc. Comput. Mach., 27 (1980), pp. 6–29.

- [24] ———, *A self routing Benes network and parallel permutation algorithms*, IEEE Trans. Comput., to appear.
- [25] ———, *Parallel permutation and sorting algorithms and a new generalized connection network*, J. Assoc. Comput. Mach., to appear.
- [26] V. YA. PAN, *New methods for the acceleration of matrix multiplications*, Proc. 20th IEEE Symposium on Foundations of Computer Science, 1979, pp. 28–38.
- [27] F. P. PREPARATA, *New parallel-sorting schemes*, IEEE Trans. Comput., C-27 (1978), pp. 669–673.
- [28] C. SAVAGE, *Parallel algorithms for graph theoretic problems*, Ph.D. thesis, University of Illinois, Urbana, August 1978.
- [29] H. SIEGAL, *A model of SIMD machines and a comparison of various interconnection networks*, IEEE Trans. Comput., C-28 (1979), pp. 907–917.
- [30] H. STONE, *Parallel processing with the perfect shuffle*, Ibid., C-20 (1971), pp. 153–161.
- [31] C. D. THOMPSON, *Generalized connection networks for parallel processor interconnection*, IEEE Trans. Comput., C-27 (1978), pp. 1119–1125.
- [32] C. D. THOMPSON AND H. T. KUNG, *Sorting on a mesh-connected parallel computer*, Comm. A.C.M., 20 (1977), pp. 263–271.
- [33] F. VAN SCOY, *Parallel algorithms in cellular spaces*, Ph.D. dissertation, University of Virginia, 1976.

ON A GREEDY HEURISTIC FOR COMPLETE MATCHING*

EDWARD M. REINGOLD[†] AND ROBERT E. TARJAN[‡]

Abstract. Finding a minimum weighted complete matching on a set of vertices in which the distances satisfy the triangle inequality is of general interest and of particular importance when drawing graphs on a mechanical plotter. The "greedy" heuristic of repeatedly matching the two closest unmatched points can be implemented in worst-case time $O(n^2 \log n)$, a reasonable savings compared to the general minimum weighted matching algorithm which requires time proportional to n^3 to find the minimum cost matching in a weighted graph. We show that, for an even number n of vertices whose distances satisfy the triangle inequality, the ratio of the cost of the matching produced by this greedy heuristic to the cost of the minimal matching is at most $\frac{4}{3}n^{\lg 2} - 1$, $\lg 2 \approx 0.58496$, and there are examples that achieve this bound. We conclude that this greedy heuristic, although desirable because of its simplicity, would be a poor choice for this problem.

Key words. graph algorithms, matching, greedy heuristic, analysis of algorithms

Introduction. We begin with some motivation, the connection of which to our central topic will become clear later.

The problem of drawing a graph $G = (V, E)$ on a mechanical plotter with pre-specified vertex locations arises in numerous applications [7]. For example, in the solution of shock wave propagation problems by the finite element method [2] it is necessary to plot meshes of thousands of nodes in order to check them visually. Other applications include the drawing of maps, PERT charts, electrical networks, etc. To draw the graph efficiently we must minimize wasted plotter-pen movement, i.e., movement with the pen off the paper so that no line is drawn. The wasted pen movement can be significant; in [5] the use of a naive algorithm resulted in excessively long plotting times.

If the graph G contains an Eulerian cycle or path, then it can be drawn with no wasted pen movement. Moreover, since a graph contains an Eulerian cycle if and only if it contains no vertices of odd degree and an Eulerian path if and only if it contains two vertices of odd degree, it is easy to determine whether either case applies and if so then to use a simple depth-first search algorithm to find the cycle or path [see [10, p. 399]]. If the graph does not contain an Eulerian cycle or path then it contains an even number $n > 2$ of vertices of odd degree. In this case, as a simple consequence of the triangle inequality, the minimum wasted pen movement is achieved by finding a minimum weighted complete matching of the n vertices of odd degree, and drawing the graph by traversing the Eulerian cycle that exists when the edges of the minimum matching are added to the original graph; these edges are traversed with the pen off the paper during the drawing.

The currently known best algorithm for finding a minimum complete matching in a weighted graph requires time proportional to n^3 [8]. For our application that is too inefficient, since n can be fairly large in practice, and we do not want to sacrifice much (relatively expensive) computer time to save (relatively inexpensive) plotter time.

* Received by the editors December 7, 1978, and in revised form October 2, 1980. This research was supported in part by the National Science Foundation under grants NSF MCS 77 22830 and NSF MCS 75-22870, by the Office of Naval Research under contract N00014-76-C-0688, and by a Guggenheim Fellowship.

[†] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

[‡] Department of Computer Science, Stanford University, Stanford, California 94305. Current address: Bell Laboratories, 600 Mountain Avenue, Murray Hill, New Jersey 07974.

However, it would certainly be worthwhile to be able to economize somewhat on plotter time if it could be done without an excessive amount of computer time.

Thus we arrive at the following problem: Can we find near minimum complete matchings of n vertices in the Euclidean plane in, say, time $O(n \log n)$ or time $O(n^2)$ as was done for the traveling salesman problem in [11]?

A greedy heuristic. An obvious heuristic is the following “greedy” algorithm: Repeatedly match the two closest unmatched remaining vertices, resolving any ties in an arbitrary fashion. For n vertices this can be done in worst-case time $O(n^2 \log n)$ by sorting the n^2 distances. This time bound represents reasonable savings for the moderately large values of n encountered in practice, but we must consider how far from minimum the resulting matching will be. The average behavior of this heuristic has been analyzed in [1].

In examining this question we quickly arrive at the sequence of examples in Fig. 1. For 2^t vertices ($t \geq 1$), we have an example in which the minimum matching has cost

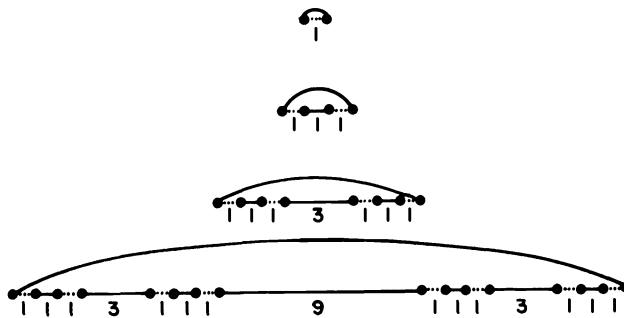


FIG. 1. Examples in which the greedy heuristic produces matchings (shown in solid lines) costing $\frac{4}{3}n^{\lg \frac{3}{2}} - 1$ times as much as the minimal matching (shown in dotted lines) for $n = 2^t$. Comparable examples are easy to construct for N even but not a power of 2.

2^{t-1} , while the cost of the solution produced by the greedy algorithm can be as bad as $2 \cdot 3^{t-1} - 2^{t-1}$ (the solution of the recurrence relation $g_1 = 1, g_{t+1} = 2g_t + 2 \cdot 3^{t-1}$). The ratio of the cost of the matching found by the greedy algorithm to the cost of the minimum matching is

$$\frac{2 \cdot 3^{t-1} - 2^{t-1}}{2^{t-1}} = 2 \cdot \left(\frac{3}{2}\right)^{t-1} - 1 = \frac{4}{3}(2^t)^{\lg \frac{3}{2}} - 1.$$

This tells us that the ratio can be as bad as $\frac{4}{3}n^{\lg \frac{3}{2}} - 1$ for n vertices ($\lg \frac{3}{2} \approx 0.58496$). We now prove that it can be no worse.

THEOREM. *Given an even number n of vertices, the distances between which satisfy the triangle inequality, the ratio of the cost of the matching found by the greedy algorithm to the cost of the minimum matching is at most*

$$\left(\frac{4}{3}n^{\lg \frac{3}{2}}\right) \frac{3^\theta}{2^{\theta+1} - 1} - 1,$$

where $\theta = \lceil \lg n \rceil - \lg n$.

Before proving this theorem we observe that the function $3^\theta / (2^{\theta+1} - 1)$ is close to 1 throughout the interval $0 \leq \theta < 1$. Its maxima occur at the endpoints $\theta = 0$ and $\theta = 1$ when it is exactly 1; its minimum of approximately 0.94650 occurs at $\theta = \lg$

$\log_2 3 - 1 \approx 0.43803$. The value of $\theta = 0$ corresponds precisely to the examples in Fig. 1, where the number of vertices is a power of 2. For numbers of vertices that are even but not a power of 2, the bound corresponds precisely to examples analogous to those in Fig. 1. The bound of this theorem is thus tight for all even n .

Proof. Observe that the union of any two matchings is a collection of disjoint cycles, the edges of which alternate between the two matchings. (An edge that is in both matchings forms a “double edge” in the union, that is, a cycle of length two.) Consider the collection of such cycles that results from taking the union of the minimum matching and the matching produced by the greedy algorithm for an arbitrary set of $n = 2k$ vertices. Let these cycles be C_1, C_2, \dots, C_m , and let M_i be the sum of the lengths of the edges from the minimum matching in C_i and G_i be the corresponding sum of the edges in C_i from the matching produced by the greedy algorithm. Clearly M_i is the cost of a minimum matching on vertices of C_i and G_i is the cost of the matching resulting from applying the greedy algorithm to the vertices of C_i . We want to bound

$$\begin{aligned} \frac{G_1 + G_2 + \dots + G_m}{M_1 + M_2 + \dots + M_m} &= \frac{M_1}{M_1 + M_2 + \dots + M_m} \frac{G_1}{M_1} + \frac{M_2}{M_1 + M_2 + \dots + M_m} \frac{G_2}{M_2} \\ &\quad + \dots + \frac{M_m}{M_1 + M_2 + \dots + M_m} \frac{G_m}{M_m} \\ &= \alpha_1 \frac{G_1}{M_1} + \alpha_2 \frac{G_2}{M_2} + \dots + \alpha_m \frac{G_m}{M_m}, \end{aligned}$$

where

$$\alpha_i = \frac{M_i}{M_1 + M_2 + \dots + M_m},$$

$\alpha_i > 0, \sum \alpha_i = 1$. Thus

$$\frac{G_1 + G_2 + \dots + G_m}{M_1 + M_2 + \dots + M_m}$$

is a weighted average of $G_1/M_1, G_2/M_2, \dots, G_m/M_m$ and hence less than the largest G_i/M_i . It therefore suffices to consider the ratio of the two costs when the union of the two matchings is a single cycle.

Consider the cycle shown in Fig. 2, in which the edge AD is the last edge added by the greedy algorithm and the edge BC is the penultimate. Linearly scale the edge

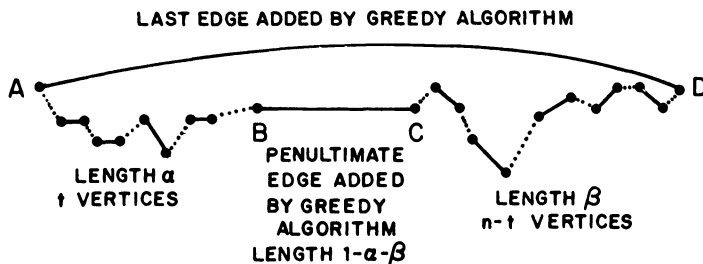


FIG. 2. Proof of the theorem.

lengths so that the sum of the lengths of the edges along the path A to B to C to D is 1. Now, define $f(n)$ to be the smallest fraction of this unit length that can consist of edges of the minimum matching of the n vertices, for any set of n vertices with the property that

the union of the optimal match and the greedy match is a single cycle. Before continuing with an analysis of $f(n)$, observe that the ratio of the matching produced by the greedy algorithm to the minimal matching is bounded above by

$$\frac{\text{length of } AD + (1 - f(n))}{f(n)}.$$

Since the length of AD is at most 1 by the triangle inequality, this ratio is at most

$$(1) \quad \frac{2}{f(n)} - 1.$$

We get a lower bound on $f(n)$ by developing and solving a recurrence relation. Let α be the (scaled) length of the sum of the edges from A to B and let β be the (scaled) length of the sum of the edges from C to D . Thus the edge BC has length $1 - \alpha - \beta$. For the greedy algorithm to choose BC in preference to AB or CD as the penultimate edge selected, we must have $\alpha \geq 1 - \alpha - \beta$ and $\beta \geq 1 - \alpha - \beta$, respectively. Since the vertices B and C are distinct, we also have that $1 - \alpha - \beta > 0$. Letting $t = 2l$ be the number of vertices along the path from A to B and $n - t$ the number along the path from C to D , we have

$$(2) \quad f(n) = \min_{\substack{2 \leq t \leq n-2 \\ \alpha \geq 1 - \alpha - \beta > 0 \\ \beta \geq 1 - \alpha - \beta > 0}} \{ \alpha f(t) + \beta f(n - t) \}$$

and, obviously,

$$f(2) = 1.$$

Since the extremum of a linear function on a polyhedron must occur at a vertex of the polyhedron [3, p. 154], (2) reduces to

$$(3) \quad f(n) = \min \{ f(2), f(4), \dots, f(n - 2), \\ \frac{1}{3}f(2) + \frac{1}{3}f(n - 2), \frac{1}{3}f(4) + \frac{1}{3}f(n - 4), \dots, \frac{1}{3}f(n - 2) + \frac{1}{3}f(2) \},$$

because the vertices of the polyhedron defined by $\alpha \geq 1 - \alpha - \beta \geq 0$ and $\beta \geq 1 - \alpha - \beta \geq 0$ are $(0, 1)$, $(\frac{1}{3}, \frac{1}{3})$, and $(1, 0)$. We now show by induction that the solution to (3) with $f(2) = 1$ is

$$(4) \quad f(2k) = 1 - \sum_{i=2}^k \frac{1}{3^{\lceil \lg i \rceil}}.$$

This is clearly true for $k = 1$. Suppose (4) holds for all $k < t$. Using (4) for $k < t$, we find the minimum occurs at

$$f(2t) = \begin{cases} \frac{2}{3}f(t), & t \text{ even,} \\ \frac{1}{3}f(t+1) + \frac{1}{3}f(t-1), & t \text{ odd.} \end{cases}$$

We consider only the case when t is even; the case of t odd is similar. Let $t = 2u$. From (4) we have

$$f(2t) = \frac{2}{3}f(t) = \frac{2}{3}f(2u) = \frac{2}{3} \left(1 - \sum_{i=2}^u \frac{1}{3^{\lceil \lg i \rceil}} \right) = 1 - \frac{1}{3} - \frac{2}{3} \sum_{i=2}^u \frac{1}{3^{\lceil \lg i \rceil}}.$$

Substituting the identity

$$\frac{1}{3} = \frac{1}{3} \sum_{i=2}^u \frac{1}{3^{\lceil \lg i \rceil}} + \sum_{i=u+1}^{2u} \frac{1}{3^{\lceil \lg i \rceil}},$$

we get

$$\begin{aligned} f(2t) &= 1 - \frac{1}{3} \sum_{i=2}^u \frac{1}{3^{\lceil \lg i \rceil}} - \sum_{i=u+1}^{2u} \frac{1}{3^{\lceil \lg i \rceil}} - \frac{2}{3} \sum_{i=2}^u \frac{1}{3^{\lceil \lg i \rceil}} \\ &= 1 - \sum_{i=2}^{2u} \frac{1}{3^{\lceil \lg i \rceil}} \\ &= 1 - \sum_{i=2}^t \frac{1}{3^{\lceil \lg i \rceil}}, \end{aligned}$$

as claimed.

For $n = 2^l - 2x$, $2^{l-2} > x \geq 0$, (4) becomes

$$\begin{aligned} f(n) &= 1 - \sum_{i=2}^{2^{l-1}} \frac{1}{3^{\lceil \lg i \rceil}} + \frac{x}{3^{l-1}} \\ &= \left(\frac{2}{3}\right)^{l-1} + \frac{x}{3^{l-1}}, \end{aligned}$$

and it follows that if $\theta = \lceil \lg n \rceil - \lg n$ then

$$f(n) = \frac{2^\theta - \frac{1}{2}}{3^{\theta-1}} n^{\lg 3}.$$

From (1) the ratio of interest is at most

$$\left(\frac{4}{3} n^{\lg 3}\right) \frac{3^\theta}{2^{\theta+1} - 1} - 1,$$

as desired.

Conclusions. Together with the examples of Fig. 1, the theorem tells us that the performance of this greedy heuristic is disappointing. Since a similar greedy heuristic for the traveling salesman problem results in tours costing at most twice the cost of the optimal tour when the triangle inequality holds [11], we might have hoped for comparable results for this matching problem. Considering results in [13], we conclude that this greedy heuristic would in general be a poor choice for the matching problem.

It would be interesting to investigate other heuristics. For example, it may be useful to construct the Voronoi diagram in $O(n \log n)$ time and restrict our attention to its straight line dual; Drysdale [4] has shown that this approach will not guarantee an optimal matching, but perhaps it results in a near optimal one. It may also be possible to develop partition algorithms in the style of [6] or [9] that have good average-case behavior. Related results can be found in [1] and [13].

Acknowledgments. We are grateful to Klaus Ecker for pointing out a flaw in an earlier version of the recurrence relation (2).

REFERENCES

- [1] D. AVIS, *Two greedy heuristics for the weighted matching problem*, Proc. Ninth Southeastern Conf. on Combinatorics, Graph Theory, and Computing, 1978, pp. 65–76.
- [2] C. J. CONSTANTINO, *Two dimensional wave propagation through nonlinear media*, J. Comp. Phys., **4** (1969), pp. 147–170.
- [3] G. B. DANTZIG, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.
- [4] S. DRYSDALE, personal communication.
- [5] R. W. GOLLAND, E. M. REINGOLD, R. R. ROBINSON AND A. WACHOWSKI, *Stress waves in layered arbitrary media slam code free-field study*, vol. IV, SAMSO TR 68-181, Department of the Air Force, July 1968.
- [6] R. M. KARP, *Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane*, Math. Oper. Res., **2** (1977), pp. 209–224.
- [7] R. KOPPE, *Automatische Abbildung eines planaren Graphen in die Ebene mit beliebig vorgebbaren Örttern der Knotenbilder*, Computing, **20** (1978), pp. 61–73.
- [8] C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, in press.
- [9] M. O. RABIN, *Probabilistic Algorithms*, in Algorithms and Complexity, Academic Press, New York, 1976.
- [10] E. M. REINGOLD, J. NIEVERGELT AND N. DEO, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.
- [11] D. J. ROSENKRANTZ, R. E. STEARNS AND P. M. LEWIS, *An analysis of several heuristics for the traveling salesman problem*, this Journal, **6** (1977), pp. 563–581.
- [12] M. I. SHAMOS AND D. HOEY, *Closest-point problems*, Proc. 16th IEEE Symposium on Foundations of Computer Science, 1975, pp. 151–162.
- [13] K. J. SUPOWIT, D. A. PLAISTED AND E. M. REINGOLD, *Heuristics for weighted perfect matching*, Proc. 12th ACM Symposium on Theory of Computing, 1980, pp. 398–419.

FAST, EFFICIENT PARALLEL ALGORITHMS FOR SOME GRAPH PROBLEMS*

CARLA SAVAGE† AND JOSEPH JA'JA‡

Abstract. Algorithms for solving graph problems on an unbounded parallel model of computation are considered. Parallel algorithms of time complexity $O(\log^2 n)$ are described for finding biconnected components, bridges, minimum spanning trees and fundamental cycles. In the algorithms for finding minimum spanning trees, bridges, and fundamental cycles, the number of processors used is small enough that the parallel algorithm is efficient in comparison with the best sequential algorithms for these problems. Several other algorithms are presented which are especially suitable for processing sparse graphs.

Key words. parallel algorithm, graph algorithm, minimum spanning tree, biconnected component, bridge, fundamental cycle, computational complexity, graph theory, multiprocessing

1. Introduction. Recent interest in parallel computers has motivated the development of parallel algorithms to solve many types of problems. Several different models of computations are commonly used (e.g., [7], [14], [16]). We focus here on the unbounded parallel model.

The unbounded parallel model assumes the presence of an unlimited number of identical processors with independent control; each processor is capable of performing typical operations such as arithmetic and boolean, and each is identified by a unique label. These processors have access to a common main memory which contains the instructions of a program; an instruction may contain a reference to a processor label. We make the assumption that different processors can obtain the content of one memory location at the same time; they may store information into different memory locations simultaneously but no two processors should attempt to change the content of the same memory location at the same time. We further assume that all the processors are synchronous in the sense that if a set of instructions is executed in parallel, then each must be allowed to finish before the next set of instructions is started. This model conforms with that of the SIMD computer of [7].

Algorithms have been designed, based on this unbounded parallel model, to solve such problems as evaluating an arithmetic expression [3], [13], [15]; computing matrix operations [4], [5], [9] and sorting [2], [17]. Recently, graph problems have been considered. In [10] and [19], algorithms to find the transitive closure of a directed graph (or Boolean matrix) in time $O(\log^2 n)^1$ using $O(n^3)$ processors are presented. Hirschberg [10] also shows that the connected components of an undirected graph can be found in time $O(\log^2 n)$ using $O(n^2)$ processors. This algorithm can be used to find the transitive closure of an undirected graph in time $O(\log^2 n)$ using $O(n^2)$ processors. Other algorithms for finding connected components are found in [19].

In this paper we are concerned with finding parallel graph algorithms which are not only "fast," but also "efficient" in comparison with sequential algorithms. A parallel algorithm which solves a given problem in time $T(n)$ using $P(n)$ processors will yield a sequential algorithm to solve the problem in time $T(n) \cdot P(n)$. Thus

* Received by the editors May 25, 1979 and in final revised form January 7, 1981.

† Computer Science Department, North Carolina State University, Raleigh, North Carolina, 27612. Part of this work was done at the University of Illinois at Urbana-Champaign, where it was supported by NSF Grant CS 76-17321, and at the University of Texas, where the first author was an IBM Postdoctoral Fellow.

‡ Computer Science Department, Pennsylvania State University, University Park, Pennsylvania 16802. The work of this author was supported by NSF Grant MCS 78-06118.

¹ All logarithms are base two.

$T(n) \cdot P(n)$ is bounded below by the time complexity of the best known sequential algorithm for the given problem. It is to be expected that, for many problems, there are parallel algorithms in which $T(n)$ is very small; we show here that $T(n) = O(\log^2 n)$ for several graph problems. It is more difficult to find parallel algorithms in which we also have $T(n) \cdot P(n)$ very close to the best sequential time complexity of a given problem. We present techniques for obtaining some parallel graph algorithms, with $T(n) = O(\log^2 n)$, in which the ratio of $T(n) \cdot P(n)$ to the best sequential time is no more than a power of $\log n$.

Let $G = (V, E)$ be an undirected graph with $|V| \geq 2$. Define the *diameter* d of G to be $d = \max\{d(v, w), 2\}$, $v, w \in V$, where $d(v, w)$ is the length of a shortest path between v and w , if any, $-\infty$ otherwise. Note that $d \leq |V| - 1$. Throughout this paper we use n, m and d to mean $n = |V|$, $m = |E|$ and d the diameter of G .

In § 2, we consider two different algorithms to find the connected components of an undirected graph; the first runs in time $O(\log n \log d)$ and uses $O(n^3/\log n)$ processors. The second, more suitable for processing sparse graphs, runs in time $O(\log^2 n)$ with only $O(m + n \log n)$ processors. Section 3 is devoted to developing efficient algorithms for finding minimum spanning trees in time $O(\log^2 n)$ using $O(n^2)$ processors and fundamental cycles in time $O(\log^2 n)$ with $O(n^3)$ processors. In § 4, we develop two different algorithms to find the biconnected components of a graph; the first is faster with time complexity $O(\log^2 n)$ using $O(n^3/\log n)$ processors, while the second runs in time $O(\log^2 n \log k)$ where k is the number of biconnected components, with $O(mn + n^2 \log n)$ processors. A parallel sorting algorithm of Preparata [17] is used to obtain an efficient parallel bridge algorithm in § 5.

2. Connected components. Let $G = (V, E)$ be an undirected graph. We are interested in developing an efficient algorithm to find the connected components of G in the case when G is dense, i.e., d is small. The algorithm is simple and based on the following strategy: collapse each node with all of its neighbors and repeat until no changes occur between two successive stages. It is easy to see by induction that at the k th step of the above procedure, for each node x , the set $N(x)$ of all nodes collapsed with x is given by

$$N(x) = \{y \mid d(x, y) \leq 2^{k-1}\}.$$

Therefore we need at most $\lceil \log d \rceil + 1$ stages and each stage can be implemented to run in time $O(\log n)$ using only $O(n^3/\log n)$ processors [12]. Therefore we have the following.

THEOREM 2.1. *Let G be an undirected graph with diameter d . Then it is possible to find the connected components of G in time $O(\log n \log d)$ using $O(n^3/\log n)$ processors.*

The same strategy can be applied to finding the transitive closure or the strongly connected components of a graph with the same time and processor bounds.

We will now present another algorithm based on Hirschberg's algorithm and which uses only $O(m + n \log n)$ processors. Before, let's introduce some terms. A *tree-loop* [10], [11], is a tree having directed edges leading from vertices in the tree to their respective fathers (i.e., directed from the leaves toward the root) with one additional edge from the root to one of its descendants. A *club* is a tree-loop in which all vertices other than the root are sons of the root, i.e., the tree has depth one.

Let the elements of V be labelled by the numbers $1, \dots, n$. The main idea of Hirschberg's algorithm is to partition each connected component into clubs such that the root of each club is the vertex of minimal label in that club. At each step of the

algorithm, the number of clubs is decreased by at least a factor of two. Hirschberg's algorithm [11] is given below.

HIRSCHBERG'S ALGORITHM.

Input: adjacency matrix $A[n \text{ by } n]$.

Output: vector D of length n such that $D[x]$ is the smallest node y reachable from node x .

1. **for all** x **do** $D[x] \leftarrow x$
2. **for** $\lceil \log n \rceil$ iterations **do**
 begin
3. **for all** x **do** $C[x] \leftarrow \min \{D[y] \mid A[x, y] = 1 \text{ and } D[y] \neq D[x]\}$
 if none then $D[x]$
4. **for all** x **do** $C[x] \leftarrow D[x] \leftarrow \min \{C[y] \mid D[y] = x \text{ and } C[y] \neq x\}$
 if none then $D[x]$
5. **for** $\lceil \log n \rceil$ iterations **do**
 for all x **do** $D[x] \leftarrow D[D[x]]$
6. **for all** x **do** $D[x] \leftarrow \min \{D[x], C[D[x]]\}$
- end.**

At each step of the main loop (2–6), all the nodes in a club will have the same D value, and this value is the smallest node in this club.

One remark about the above algorithm is that we don't really have to run the main loop $\lceil \log n \rceil$ times. We can run it until $D[x]$ does not change value in two successive iterations, in which case we stop. One can easily check that the running time of the corresponding algorithm is $O(\log n \cdot \min \{\log n, d/2\})$ which is faster than the original algorithm if $d < 2 \log n$.

We now turn our attention to the number of processors we need. Instead of using an adjacency matrix to represent the given graph, we can use an *adjacency list matrix* [6], R , which is an $n \times (n-1)$ matrix such that, for $1 \leq i \leq n$, row i consists of the vertices adjacent to vertex i in G (the remaining entries are not initialized). Associated with the adjacency list matrix is an n -vector of *end markers* EM , where $EM[i]$ contains the index j of the last vertex in the i th row of R . Given x , we can find $\{y \mid A[x, y] = 1\} = \{y \mid y \text{ is in the } x\text{th row of } R\}$ in constant time using precisely $EM[x]$ processors. Therefore, step 3 of the above algorithm can be implemented in time $O(\log n)$ using $O(\sum_x EM[x]) = O(m)$ processors.

Step 4 of the above algorithm is a bit harder to implement efficiently since finding $\{y \mid D[y] = x\}$ for all x seems to require $O(n^2)$ processors. Note that, if x is not a root of a club, then this step does not change the value of $D[x]$. If x is a root, then the set $\{y \mid D[y] = x\}$ is precisely the set of vertices in the club with root x . To reduce the number of processors we do the following. Let B be a copy of D and let I be an n -vector with $I[x] = x$, for all x . We can use Preparata's algorithm to sort B such that, whenever we switch the i th and j th entries of B , we also switch the i th and j th entries of I (index sort). This can be done in $O(\log n)$ time with $O(n \log n)$ processors. We now know that the vertices of each club are grouped together in I . It is easy to see how to modify a copy R' of the matrix R such that the x th row of R' contains the vertices in the club with root x in time $O(\log n)$ with $O(n)$ processors. We can also construct efficiently the end marker EM such that, for each root x , $EM[x]$ is the number of vertices in the club with root x . Using these data structures, one can implement step 4 in $O(\log n)$ time using $O(m)$ processors. Therefore, we have the following theorem.

THEOREM 2.2. *Let G be an undirected graph. Then it is possible to find the connected components of G in time $O(\log^2 n)$ using $O(m + n \log n)$ processors.*

3. Minimum spanning trees and fundamental cycles. Let $G = (V, E)$ be a connected, undirected, weighted graph with $V = \{1, \dots, n\}$ and weight function $\omega : E \rightarrow \mathbb{R}$. A *minimum spanning tree* (MST) of G is a spanning tree $T = (V, E')$ of G with the property that the sum of the weights of the edges in T is minimal over all spanning trees for G . Assume for simplicity that the weights of the edges of G are all distinct. We use an algorithm due to Sollin [1] and the technique of Hirschberg to find a MST of G in time $O(\log^2 n)$ with $O(n^2)$ processors.

If the weights of the edges of G are all distinct, G has a unique MST. Sollin's algorithm can be described in the following way. Let G_0 be the graph (V, \emptyset) . Assume that for some $t \geq 0$, the graph G_t has been constructed. If G_t is connected, then G_t is the MST of G and the algorithm terminates. Otherwise, for each connected component K of G_t , find the minimum weight edge of G joining a vertex of K to a vertex of a distinct component of G_t . These new edges are added to G_t to form the graph G_{t+1} .

Let t^* be the smallest value of t for which G_t is an MST of G . For $0 \leq t < t^*$, each connected component of G_{t+1} is the union of at least two connected components of G_t . The graph G_0 has n components, thus for $0 \leq t \leq t^*$, G_t has at most $n/2^t$ components. Since G_{t^*} is connected, $n/2^{t^*} \geq 1$ so that $t^* \leq \lceil \log n \rceil$. Sollin's algorithm, then, consists of, at most, $\lceil \log n \rceil$ stages. We show below how each stage can be implemented in time $O(\log n)$ using $O(n^2)$ processors.

Assume inductively that for some $t \geq 0$, the graph G_t has been constructed and that for each component K of G_t a vertex of K is chosen as the component representative. Assume further that we have computed a function $r_t : V \rightarrow V$ so that $r_t(i)$ is the representative of the connected component of G_t to which i belongs. (For $t = 0$, we define $r_0 : V \rightarrow V$ so that $r_0(i) = i$.) Construct G_{t+1} as follows:

1. If $r_t(i) = r_t(j)$ for all i, j in V , G_t is the MST of G and the process terminates. This can be checked in time $O(\log n)$ using $O(n)$ processors.

2. Compute a function $e : V \rightarrow V$ so that for each vertex i , $e(i) = j$ iff $\omega(i, j)$ is minimal among all vertices j with $r_t(j) \neq r_t(i)$. For each vertex this involves finding the minimum of at most n numbers, which can be done in time $O(\log n)$ using $O(n)$ processors. Thus, Step 2 can be done in time $O(\log n)$ using $O(n^2)$ processors.

3. For each component K of G_t find the vertex i of K for which $\omega(i, e(i))$ is minimal over all vertices of K . The edges $(i, e(i))$ found in this way are the edges to be added to G_t to form G_{t+1} . As in step 2, this can be done in time $O(\log n)$ with $O(n^2)$ processors.

4. Find a vertex representative for each component of G_{t+1} and compute the function $r_{t+1} : V \rightarrow V$ where $r_{t+1}(i)$ is the representative of the connected component of G_{t+1} to which i belongs. This can be done in time $O(\log n)$ with $O(n^2)$ processors, using the same technique of Hirschberg as shown in § 2.

THEOREM 3.1. *A minimum spanning tree of a graph G can be found in time $O(\log^2 n)$ using $O(n^2)$ processors.*

The MST algorithm here is relatively efficient compared to the best sequential algorithms which require time $O(n^2)$ [18] or $O(|E| \log \log n)$, [24] for this problem.

It is important to consider the representation of the MST of G found in the algorithm above. We can easily represent it by its adjacency matrix. Initially the matrix contains only zeros; whenever an MST edge is found, the corresponding matrix entry is changed to 1. This, however, is not the most convenient representation of the MST if our aim is to apply this algorithm to obtain other efficient graph algorithms. For example, to execute an instruction of the form "for each edge in the MST do X," we must use $O(n^2)$ processors to look at each entry in the adjacency matrix to determine which entries correspond to MST edges. This is a waste of processors since only $n - 1$

edges of G will be MST edges. A more concise representation of the MST is the following. Let a vertex r of the MST, T , of G be chosen as the root of T . Define a function $F: V \rightarrow V$ so that $F(r) = r$ and if $i \neq r$, $F(i)$ is the father of i in the tree T rooted at r . The pair (F, r) defines T and can be computed by minor modifications of the MST algorithm above [21]. Now an instruction of the form, "for each edge in T do X ," can be executed using $O(n)$ processors, one for each edge $(i, F(i))$ of T .

If the weights of the edges of the graph G are not all distinct, Sollin's algorithm can be modified as shown in [21] so that an MST can still be found in time $O(\log^2 n)$ using $O(n^2)$ processors. In particular, a spanning tree of a connected graph can be found by considering each edge to have weight one.

We now turn our attention to the problem of finding a set of fundamental cycles of a connected, undirected graph $G = (V, E)$. If $H = (V_H, E_H)$ and $K = (V_K, E_K)$ are subgraphs of G , the *symmetric difference* of H and K , written $H + K$, is the subgraph $G' = (V', E')$ of G where

$$E' = \{e \in E_H \cup E_K \mid e \notin E_H \cap E_K\}$$

and

$$V' = \{v \in V \mid v \text{ is incident with some edge of } E'\}.$$

A *set of fundamental cycles* of G is a collection Ω of cycles of G with the property that any cycle C of G can be written as $C = C_1 + C_2 + \cdots + C_k$ for some subcollection of cycles $C_1, C_2, \dots, C_k \in \Omega$.

Let $T = (V, E_T)$ be a spanning tree of G . Every edge $e \in E - E_T$ will create a cycle C_e if it is added to T . It can be shown that the collection $\{C_e \mid e \in E - E_T\}$ is a set of fundamental cycles of G [20]. In this section we use this fact as the basis for an algorithm which will find a set of fundamental cycles for G in time $O(\log^2 n)$ with $O(n^3)$ processors.

We can apply the MST algorithm to G to obtain a spanning tree T which is represented by a pair (F, r) as described above. The vertex r is considered as the root of T , and $F: V \rightarrow V$ is defined so that $F(r) = r$ and for $i \neq r$, $F(i)$ is the father of i in T . For $i, j \in V$, define the *youngest common ancestor* of i and j , $yca(i, j)$, to be the vertex $v \in V$ such that v is an ancestor in T of both i and j and v is a descendant of any other vertex $u \in V$ which is an ancestor of both i and j . Then for $i, j \in V$, if (i, j) is an edge of G which creates a cycle when added to T , C consists of (i, j) and the two paths in T joining i and j to $yca(i, j)$. Thus a parallel algorithm for finding fundamental cycles can be outlined below.

FUNDAMENTAL CYCLE ALGORITHM.

1. Find a spanning tree T of G .
2. For each pair $i, j \in V$, find $yca(i, j)$.
3. For each pair $i, j \in V$, if (i, j) is an edge in G but not in T , find the two paths joining i and j to $yca(i, j)$ in T .

As discussed before, step 1 can be done in time $O(\log^2 n)$ with $O(n^2)$ processors; step 3 basically involves computing powers of F (composition) and it is shown in [21] that it can be done in time $O(\log n)$ with $O(n^3)$ processors. Step 2 can be implemented in the following way. Let H be the *directed* graph with vertex set V and edges of the form $(i, F(i))$ where $i \in V$ and $i \neq r$. Let M be the adjacency matrix of H .

YCA ALGORITHM.

1. Construct M . This can be done in constant time with $O(n^2)$ processors, given (F, r) .

2. Find M^* , the transitive closure of M . ($O(\log^2 n)$ time, $O(n^3)$ processors, as discussed in § 1.)

3. Let R be the relation on V defined by M^* , that is for $i, j \in V$, iRj iff $M^*(i, j) = 1$. Then R is a partial ordering of V . Compute for each pair $i, j \in V$ the minimum, with respect to R , of the set $\{k \mid iRk \text{ and } jRk\}$. This is $yca(i, j)$. Since the minimum of n elements from an ordered set can be found in time $O(\log n)$ with $O(n)$ processors, $yca(i, j)$ can be computed for all i, j in time $O(\log n)$ with $O(n^3)$ processors.

THEOREM 3.2. A fundamental set of cycles of a graph G can be found in time $O(\log^2 n)$ using $O(n^3)$ processors.

This algorithm is relatively efficient in comparison with the best sequential algorithm, which takes time $O(n^3)$ for this problem [20].

4. Biconnected components. Given a connected undirected graph $G = (V, E)$, a vertex $a \in V$ is called an *articulation point* of G if there are vertices i and j in V , distinct from a , such that every path in G joining i and j contains a . The graph G is *biconnected* if it contains no articulation points.

For each vertex i in V , let G_i be the graph obtained from G by removing i . The procedure below will determine from the adjacency list matrix R of G , whether G is biconnected.

Biconnectivity.

1. For $i = 1, \dots, n$, let R_i be the matrix obtained from R by substituting a sequence of zeros in the i th row and by replacing each occurrence of i by zero. This can be done in constant time with $O(m)$ processors for each i .

2. For $i = 1, \dots, n$, determine from R_i , if G_i is connected. One of the algorithms of § 2 could be used to execute this step. G_i is connected iff i is not an articulation point.

3. Fan-in the results of the n computations of step 2. This can be done in time $O(\log n)$ with $O(n)$ processors. G is biconnected iff each G_i is connected.

Therefore, it is possible to check whether a graph is biconnected in time $O(\log^2 n)$ with $O(mn + n^2 \log n)$ processors.

A *biconnected component* of G is a biconnected subgraph, H , of G which is not properly included in any biconnected subgraph of G . For a subset V' of V , let

$$E(V') = \{(i, j) \in E \mid i, j \in V'\}.$$

Note that if $B = (V', E')$ is a biconnected component of G , then $E' = E(V')$. Thus, the set of vertices of a biconnected component of G determine it completely.

We will present two different algorithms to find the biconnected components of a graph. The first, more elegant, runs in time $O(\log^2 n)$ with $O(n^3/\log n)$ processors. The second, more suitable for processing sparse graphs, runs in time $O(\log^2 n \log k)$, where k is the number of biconnected components, using $O(mn + n^2 \log n)$ processors.

We start by describing the first algorithm. Define a relation R on V by iRj if and only if i and j are in a common biconnected component of G . Note that iRj iff, for all $k \in V$ distinct from i and j , there is a path between i and j in G_k . Therefore, we have the following lemma [21].

LEMMA 4.1. Let $G_k^* = (V, E_k^*)$ be the transitive closure of the graph $G_k = (V, E_k)$. For distinct vertices i, j in V , iRj if and only if $(i, j) \in E_k^*$ for all $k \in V$ distinct from i and j .

The graphs G_i^* can be computed, from G , in time $O(\log^2 n)$ with $O(m + n \log n)$ processors for each G_i . The relation R can be computed from the graphs G_i^* and this can be done in time $O(\log^2 n)$ using $O(n/\log n)$ processors for each pair $i, j \in V$.

(This can be done by partitioning the vertices into $\lfloor n/\log n \rfloor$ sets and letting each processor do the checking corresponding to each set.) Therefore, the construction of the relation R requires $O(\log^2 n)$ time with $O(n^3/\log n)$ processors. In order to find the biconnected components, we make use of the following lemma [21].

LEMMA 4.2: *Let (i, j) be an edge of G . The set*

$$V_{(i, j)} = \{k \in V \mid iRk \text{ and } kRj\},$$

is the vertex set of the biconnected component of G which contains (i, j) .

Using Lemma 4.2, it is easy to see that for each $(i, j) \in E$, we can find the vertices of the biconnected components containing (i, j) in constant time using $O(n)$ processors. To make this step more efficient, we find a spanning tree T of G ; since every biconnected component contains an edge of T , it is only necessary to compute for each edge of T the biconnected component of G containing it. Therefore this step could be done with $O(n^2)$ processors. We have proved the following.

THEOREM 4.3. *Let $G = (V, E)$ be a connected graph. It is possible to find the biconnected components of G in $O(\log^2 n)$ time using $O(n^3/\log n)$ processors.*

We now describe the second algorithm. We assume that the graph contains no bridges, otherwise we can use the algorithm of the next section to find the bridge connected components of a graph efficiently. This assumption is for simplicity only; the algorithm could be modified to handle this case. The main idea of the algorithm is based on the observation that, if we remove the articulation points from G , then the vertices of each connected component are in the same biconnected component by Lemma 4.1. Let C be the vertex set of such a connected component and let B be the vertex set of the biconnected component containing C . The algorithm will first find the biconnected components with one or two articulation points. We handle this case by first joining all vertices (in this case at most two) adjacent to vertices in C and then merging any two such sets which intersect in more than one articulation point (for more details, see [12]).

We may have one problem, namely the case when there are two articulation points u and v incident upon C and yet $C \cup \{u, v\}$ is properly contained in the vertex set of a biconnected component. The following lemma handles this problem.

LEMMA 4.4. *Let C and B be defined as above and suppose u and v are two articulation points incident upon vertices in C . Then $C \cup \{u, v\}$ is the vertex set of a biconnected component iff removing all edges between u and vertices in C disconnects G and the same is true for v .*

Proof. It is clear that C and $\{u, v\}$ are in a common biconnected component. If $x \notin C \cup \{u, v\}$ is a node which belongs to B , then removing u should not disconnect x from C . \square

Based upon the above observations, it is easy to see how to find the biconnected components with one or two articulation points. We can then remove these biconnected components and repeat the same process on the connected components of the resulting graph. There will be at most $\log k$ iterations, where k is the number of biconnected components, as the following lemma shows.

LEMMA 4.5. *Given any graph, there are at least as many biconnected components with zero, one or two articulation points as half the total number of biconnected components of G .*

Proof. We use the notion of the *block-cutpoint graph* of a graph G [8], denoted by $bc(G)$. Each biconnected component and each articulation point of G is represented by a vertex of $bc(G)$. Two vertices u and v of $bc(G)$ are adjacent iff u corresponds to a biconnected component containing v or vice versa. Note that $bc(G)$ is acyclic.

It is easy to see that the leaves of $bc(G)$ correspond to those biconnected components with one articulation point. Without loss of generality, we will assume that G is connected. Let 0 be the set of nodes of $bc(G)$ of degree 2. It is easy to see that the number of leaves l satisfies, $l \geq |\{v \in bc(G) : \deg v \geq 3\}|$. It follows that l is at least as large as the number of biconnected components with more than two articulations. The result follows from this observation. \square

It is straightforward to check that the above algorithm can be implemented using adjacency list matrices to run in time $O(\log^2 n \log k)$ using only $O(mn + n^2 \log n)$ processors. We only note that we have to use index sorting at each step to group the nodes of each connected component together.

THEOREM 4.6. *Given any graph $G = (V, E)$ with k biconnected components, it is possible to find the biconnected components of G in time $O(\log^2 n \log k)$ using $O(mn + n^2 \log n)$ processors.*

5. An efficient bridge algorithm. Let $G = (V, E)$ be a connected, undirected graph, where $V = \{1, \dots, n\}$. An edge e of G is a bridge of G if the graph $G_e = (V, E - \{e\})$ is not connected. We show here how to use the sorting algorithm of Preparata to obtain an algorithm for finding the bridges of G in $O(\log^2 n)$ time with $O(n^2 \log n)$ processors. The best sequential algorithms for this problem have time complexity $O(|E| + n)$ [23]. Since the product of time and processor complexities of the parallel bridge algorithm presented here is $O(n^2 \log^3 n)$, it is relatively efficient compared with the sequential algorithm, especially if $|E| = O(n^2)$.

Assume that for a given graph $G = (V, E)$, where $V = \{1, \dots, n\}$, we have found a spanning tree T represented by a pair (F, r) as described in § 3. The following lemma gives a way to test each edge $(i, F(i))$ of T , where $i \in V$ and $i \neq r$, to determine whether it is a bridge of G .

LEMMA 5.1. *For a vertex $i \in V$ with $i \neq r$, the edge $(i, F(i))$ of T is a bridge of G if and only if $(i, F(i))$ is the only edge of G which joins a descendent of i in T with a nondescendent of i in T . (Note that i is a descendent of i .)*

Proof. See [21] or [23].

For each $i \in V$, it is not difficult to see that this test could be implemented using $O(n^2)$ processors to look at all edges joining descendents of i with nondescendents of i . Then $O(n^3)$ processors would be needed to test for all bridges. The following procedure is more efficient.

BRIDGE ALGORITHM.

1. Find a spanning tree T , given by (F, r) , of G .
2. Construct the adjacency matrix M of the directed graph

$$H = (V, \{(i, F(i)) | i \in V, i \neq r\})$$

discussed in § 3. Compute M^* . Although we can use the transitive closure algorithm for directed graphs to compute M^* , we can do it more efficiently (in time $O(\log n)$ with $O(n^2)$ processors) by computing powers of F . See [21] for details.

3. For each $i \in V$, define $b_i: V \rightarrow \{0, 1\}$ for $j \in V$ by

$$b_i(j) = \begin{cases} 1 & \text{if } (i, j) \text{ is an edge of } G, j \neq F(i) \\ & \text{and } j \text{ is not a descendent of } i \text{ in } T, \\ 0 & \text{otherwise.} \end{cases}$$

If $b_i(j) = 1$, then for some ancestor k of i , the edge (i, j) is potentially an edge joining a descendent and an ancestor of k .

4. For each $i \in V$, define $f_i: V \rightarrow \{0, 1\}$ for $j \in V$ by

$$f_i(j) = \begin{cases} 1 & \text{if } b_k(j) = 1 \text{ for some descendent } k \text{ of } i, \\ 0 & \text{otherwise.} \end{cases}$$

If $f_i(j) = 1$, there is some k for which the edge (k, j) is potentially an edge joining a descendent and a nondescendent of i .

5. For each $i \in V$, $(i, F(i))$ is a bridge of G iff for every non-descendent j of i , $f_i(j) = 0$.

Steps 1 and 2 can be done in time $O(\log^2 n)$ using $O(n^2)$ processors. Step 3 can be done in constant time and step 5 in $O(\log n)$ time using $O(n^2)$ processors. It remains to show how to implement step 4. We show below how to do this in time $O(\log^2 n)$ using $O(n^2 \log n)$ processors. Thus, the algorithm can be done in time $O(\log^2 n)$ using $O(n^2 \log n)$ processors.

In step 4, we must solve, for each vertex, a problem of the form: given a function $b: V \rightarrow \{0, 1\}$, compute a function $g: V \rightarrow \{0, 1\}$, where for $v \in V$

$$g(v) = \begin{cases} 1 & \text{if } b(u) = 1 \text{ for some descendent } u \text{ of } v \text{ in } T, \\ 0 & \text{otherwise.} \end{cases}$$

We show how to compute, given b and F , a sequence of functions $g_0, g_1, \dots, g_{\lceil \log(n-1) \rceil}: V \rightarrow \{0, 1\}$, where for $v \in V$ and $0 \leq t \leq \lceil \log(n-1) \rceil$ we have $g_t(v) = 1$ if and only if there is a vertex $u \in V$ with $b(u) = 1$ and $F^k(u) = v$, where $0 \leq k < 2^t$. Since the longest path in T has length at most $n-1$, we will have $g = g_{\lceil \log(n-1) \rceil}$. We show g can be computed in this way in time $O(\log^2 n)$ using $O(n \log n)$ processors and, thus, Step 4 of the bridge algorithm can be done in time $O(\log^2 n)$ using $O(n^2 \log n)$ processors.

Define $g_0: V \rightarrow \{0, 1\}$ for $v \in V$ by $g_0(v) = b(v)$. Define $Y_0: V \rightarrow V$ by $Y_0(v) = F(v)$.

Assume that for some t , with $0 < t \leq \lceil \log(n-1) \rceil$, the functions $g_{t-1}: V \rightarrow \{0, 1\}$ and $Y_{t-1}: V \rightarrow V$ have been computed so that $Y_{t-1}(v) = F^{2^{t-1}}(v)$. Compute g_t and y_t as follows.

a. Let y_t be a $1 \times n$ array where for $v \in V$,

$$y_t(v) = \begin{cases} Y_{t-1}(v) & \text{if } g_{t-1}(v) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

(Then a vertex u of V appears in the array y_t if and only if there is a vertex w in V with $g_{t-1}(w) = 1$ and $F^{2^{t-1}}(w) = u$.) (Note that for each vertex u which appears in y_t , we would like to store a 1 in location $g_t(u)$. However, for distinct i and j we may have $y_t(i) = y_t(j)$. Since we do not allow two values to be stored simultaneously in the same memory location, we eliminate duplication in the array y_t in steps b and c and then compute g_t in step d .)

b. Sort the array $y_t(1), \dots, y_t(n)$ in decreasing order to obtain an array $z_t(1), \dots, z_t(n)$.

c. Let s_t be a $1 \times n$ array where $s_t(1) = z_t(1)$ and for $i = 1, \dots, n-1$,

$$s_t(i+1) = \begin{cases} 0 & \text{if } z_t(i) = z_t(i+1), \\ z_t(i+1) & \text{otherwise.} \end{cases}$$

(All nonzero entries in the array s_t are distinct.)

d. For $v \in V$, let $g_t(v) = g_{t-1}(v)$. Then if $s_t(v) \neq 0$, replace $g_t(s_t(v))$ by 1. (Note that $g_t(v) = 1$ iff $g_{t-1}(v) = 1$ or v appears in the array s_t .)

e. For $v \in V$ let $Y_t(v) = Y_{t-1}(Y_{t-1}(v))$.

The following lemma can be proved by induction to see that the steps above do compute the function g , as claimed.

LEMMA 5.2. *For $t = 0, \dots, \lceil \log(n-1) \rceil$ and $v \in V$, the function g_t defined inductively above, has the property that $g_t(v) = 1$ if and only if there is a vertex $u \in V$ with $b(u) = 1$ and $F^k(u) = v$, where $0 \leq k < 2^t$.*

To compute the function g_t from g_{t-1} and Y_{t-1} , all steps except step b can be done in constant time with $O(n)$ processors. As for step b, Preparata has shown in [17] that n numbers can be sorted in time $O(\log n)$ using $O(n \log n)$ processors. Thus, $g = g_{\lceil \log(n-1) \rceil}$ can be computed in time $O(\log^2 n)$ with $O(n \log n)$ processors.

REFERENCES

- [1] C. BERGE AND A. GHOUILA-HOURI, *Programming, Games and Transportation Networks*, John Wiley, New York, 1965, p. 179.
- [2] K. E. BATCHER, *Sorting networks and their applications*, Proc. AFIPS, Spring Joint Computer Conf., 32, 1968, pp. 307–314.
- [3] R. P. BRENT, *The parallel evaluation of general arithmetic expressions*, J. Assoc. Comput. Mach., 21 (1974), pp. 201–206.
- [4] A. K. CHANDRA, *Maximal parallelism in matrix multiplication*, IBM Report RC6193, Yorktown Heights, N.Y., September 1976.
- [5] L. CSANKY, *Fast parallel matrix inversion algorithms*, this Journal, 5 (1976), pp. 618–623.
- [6] D. M. ECKSTEIN AND D. A. ALTON, *Parallel graph processing using depth-first-search*, Proceedings of Symposium on Theoretical Computer Science, 1977.
- [7] M. J. FLYNN, *Very high-speed computing systems*, Proc. IEEE, 54 (1966), pp. 1901–1909.
- [8] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
- [9] D. HELLER, *A survey of parallel algorithms in numerical linear algebra*, Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, February 1976.
- [10] D. S. HIRSCHBERG, *Parallel Algorithms for the Transitive Closure and the Connected Components Problems*, Proc. 8th Annual ACM Symposium on Theory of Computing, Hershey, PA, 1976, pp. 55–57.
- [11] D. S. HIRSCHBERG, A. K. CHANDRA AND D. V. SARWATE, *Computing connected components on parallel computers*, Comm. ACM, 22 (1979), pp. 461–464.
- [12] J. JA'JA', *Graph connectivity problems on parallel computers*, Technical Report, CS-78-05, Department of Computer Science, Pennsylvania State University, University Park, PA, 1978.
- [13] D. J. KUCK AND K. MARUYAMA, *Time bounds on the parallel evaluation of arithmetic expression*, this Journal, 4 (1975), pp. 147–162.
- [14] K. N. LEVITT AND W. H. KAUTZ, *Cellular arrays for the solution of graph problems*, Comm. ACM, 15 (1972), pp. 789–801.
- [15] D. E. MULLER AND F. P. PREPARATA, *Restructuring arithmetic expressions for parallel evaluation*, J. Assoc. Comput. Mach., 23 (1976), pp. 534–543.
- [16] V. R. PRATT, M. D. RABIN AND L. J. STOCKMEYER, *A characterization of the power of vector machines*, Proc. 6th Annual ACM Symposium on Theory of Computing, Seattle, Washington, 1974, pp. 122–134.
- [17] F. P. PREPARATA, *Parallelism in sorting*, presented at the International Conference on Parallel Processing, Bellair, Michigan, Aug. 1977.
- [18] R. C. PRIM, *Shortest connection networks and some generalizations*, Bell Syst. Tech. J., 36 (1957), pp. 1389–1401.
- [19] E. REGHBATI AND D. G. CORNEIL, *Parallel computations in graph theory*, this Journal, 2 (1978), pp. 230–237.
- [20] E. M. REINGOLD, J. NIEVERGELT AND N. DEO, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.
- [21] C. SAVAGE, *Parallel algorithms for graph theoretic problems*, Ph.D. Thesis, CSL Report ACT-4, Coordinated Science Laboratory, University of Illinois, August, 1977.
- [22] R. TARJAN, *Depth-first search and linear graph algorithms*, this Journal, 1 (1972), pp. 146–160.
- [23] ———, *A note on finding the bridges of a graph*, Inform. Proc. Let., 2 (1974), pp. 160–161.
- [24] A. C. YAO, *An $O(|E| \log \log |V|)$ algorithm for finding minimum spanning trees*, Inform. Proc. Let., 4 (1975), pp. 21–23.

INFORMATION DISSEMINATION IN TREES*

P. J. SLATER,† E. J. COCKAYNE‡ AND S. T. HEDETNIEMI§

Abstract. In large organizations there is frequently a need to pass information from one place, e.g., the president's office or company headquarters, to all other divisions, departments or employees. This is often done along organizational reporting lines. Insofar as most organizations are structured in a hierarchical or treelike fashion, this can be described as a process of information dissemination in trees. In this paper we present an algorithm which determines the amount of time required to pass, or to broadcast, a unit of information from an arbitrary vertex to every other vertex in a tree. As a byproduct of this algorithm we determine the broadcast center of a tree, i.e., the set of all vertices from which broadcasting can be accomplished in the least amount of time. It is shown that the subtree induced by the broadcast center of a tree is always a star with two or more vertices. We also show that the problem of determining the minimum amount of time required to broadcast from an arbitrary vertex in an arbitrary graph is NP-complete.

Key words. algorithm, broadcast center, broadcasting, graph information dissemination, NP-complete, tree

1. Introduction. Most large organizations are structured in a hierarchical or treelike fashion (cf. Fig. 1), from the highest levels at the top to lower and lower levels as one proceeds down the tree. If there is a need to pass some information, e.g., concerning a new company policy, from one office to all others, then it is natural for this to take place along the organizational reporting lines. Thus, a president might inform each vice-president, each of whom in turn informs all subordinate division heads, who in turn inform their subordinate department heads, etc. In this paper we study the amount of time it takes for this kind of information dissemination to take place in such treelike structures.

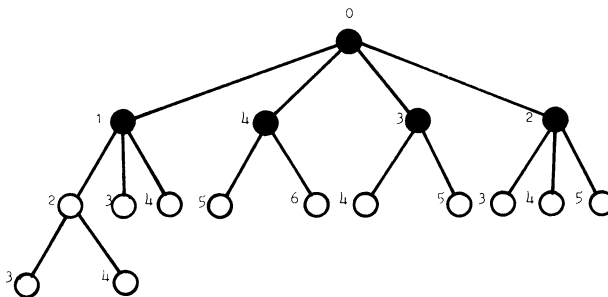


FIG. 1. Broadcasting in a tree.

More specifically, we define *broadcasting from a vertex u* to be the process of passing one unit of information from u to every other vertex in a connected graph $G = (V, E)$. This is accomplished by a series of phone calls over the edges of G , subject to the following constraints:

- (i) each phone call requires one unit of time (to convey the unit of information);

* Received by the editors June 5, 1979, and in revised form, December 22, 1980.

† Sandia Laboratories, Albuquerque, New Mexico 87185. This work was supported by the U.S. Energy Research and Development Administration (ERDA) under contract AT(29-1)-789. By acceptance of this article, the publisher and/or recipient acknowledges the U.S. Government's right to retain a nonexclusive, royalty-free license in and to any copyright covering this paper.

‡ Department of Mathematics, University of Victoria, Victoria, British Columbia, V8W 2Y2. This work was supported by the National Research Council of Canada under grant A-7544.

§ Department of Computer and Information Science, University of Oregon, Eugene, Oregon 97403.

- (ii) a vertex can only call an adjacent vertex; and
- (iii) a vertex can participate in only one call per unit of time.

We define *the broadcast number of u in G* , denoted $b(u; G)$ or simply $b(u)$, to equal the minimum number of time units required to broadcast from u . For example, the broadcast number of the topmost vertex in the tree in Fig. 1 is six; the integer label by each vertex indicates the time period during which it receives the information in one possible calling scheme. We define *the broadcast number of a connected graph G* , $b(G)$, to equal the minimum broadcast number of any vertex in G , i.e.,

$$b(G) = \min_{u \in V(G)} \{b(u)\}.$$

The broadcast center of G , $BC(G)$, is the set of all vertices having minimum broadcast number, i.e.,

$$BC(G) = \{u | b(u) = b(G)\}.$$

In this paper we present an $O(N)$ algorithm for determining the broadcast center of any tree with N vertices, a by-product of which determines the broadcast number of any vertex in the tree. It can be seen, incidentally, that the broadcast center of the tree in Fig. 1 consists of the darkened vertices.

We will also show that the problem of determining $b(u)$ for an arbitrary vertex in an arbitrary graph is NP-complete.

2. An algorithm for determining the broadcast center of a tree. For a given vertex u in tree T there is not necessarily a unique calling scheme to broadcast from u to all other vertices in $b(u)$ time units. We can, however, make the following observations. If (u, v) is an edge in tree T then $T(u, v)$ and $T(v, u)$ will denote the subtrees of T consisting of the components of $T - (u, v)$ containing u and v , respectively. Let v_1, v_2, \dots, v_k denote the vertices adjacent to u in T , and assume they are labeled so that $b(v_1; T(v_1, u)) \geq b(v_2; T(v_2, u)) \geq \dots \geq b(v_k; T(v_k, u))$. Since $b(v_i; T(v_i, u))$ is the amount of time it will take to pass the information from v_i to the other vertices in $T(v_i, u)$ after a call from u to v_i , one expects an optimal calling sequence from u to consist of first calling v_1 , then v_2 , then v_3 , etc. This is, in fact, the case.

The following algorithm, Algorithm BROADCAST, will identify a vertex v in $BC(T)$. For any other vertex x in $V(T)$, let x' denote the vertex adjacent to x on the path from x to v . Algorithm BROADCAST will assign a value $t(x)$ to each vertex x with $t(x) = b(x; T(x, x'))$. The final label $t(v)$ for this v in $BC(T)$ will equal $b(v) = b(G)$. We start by letting $t(x) = 0$ for every endvertex x in T . Subsequently we choose one vertex u at a time to label. At each stage the vertex u chosen to be labeled will be the one whose label is going to be smallest among the remaining vertices.

As indicated, Algorithm BROADCAST proceeds by iteratively assigning to each vertex u in a tree T a value $t(u)$, which equals the minimum time required to broadcast from u to every vertex in a subtree T_u . The subtree T_u in question is the largest subtree of T rooted at u consisting of vertices w which have previously been assigned values $t(w)$. Having initially labeled with zero all endvertices of T , the algorithm proceeds to move "inward" and assigns increasing labels to vertices, all but one of whose neighbors have already been labeled.

ALGORITHM BROADCAST. To determine the broadcast center $BC(T)$ and the broadcast number $b(T)$ of a tree T . At any point in this algorithm U is the set of labeled vertices which have been removed from T , and W is the set of labeled vertices which have not been removed from T .

- Step 0.* (Does T have only one or two vertices?)
If $|V(T)| \leq 2$
then $BC(T) \leftarrow V(T)$;
if $|V(T)| = 2$ **then set** $b(T) \leftarrow 1$
else set $b(T) \leftarrow 0$ **fi**;
STOP
fi.
- Step 1.* (Label the endvertices of T with 0.)
Let U be the set of endvertices of T ;
for each $u \in U$ **do set** $t(u) \leftarrow 0$ **od**;
set $T' \leftarrow T - U$.
- Step 2.* (Label the endvertices of T' .)
Let W be the set of vertices of T' with degree in T' at most one;
for each $w \in W$ **do** let u_1, u_2, \dots, u_k be the labeled vertices in U adjacent to w , ordered so that $t(u_1) \geq t(u_2) \geq \dots \geq t(u_k)$;
set $t(w) \leftarrow \max_{1 \leq i \leq k} \{t(u_i) + i\}$
od.
- Step 3.* (Select the next vertex to be deleted and the next vertex to be labeled, until there is only one vertex left.)
While $|V(T')| \geq 2$ **do**
- Step 4.* (Select the next vertex w .)
Let $w \in W$ satisfy $t(w) = \min \{t(w_i) | w_i \in W\}$;
let v be the vertex adjacent to w in T' .
- Step 5.* (Delete w from W and T' , and add it to U .)
Set $W \leftarrow W - \{w\}$;
set $U \leftarrow U \cup \{w\}$;
set $T' \leftarrow T' - \{w\}$. (Note, if T' now has one vertex, it is considered to be an endvertex.)
- Step 6.* (Label the next vertex.)
If v is now an endvertex of T'
then let v be adjacent to labeled vertices u_1, u_2, \dots, u_k in U ordered so that $t(u_1) \geq t(u_2) \geq \dots \geq t(u_k)$;
set $t(v) \leftarrow \max \{t(u_i) + i | 1 \leq i \leq k\}$;
set $W \leftarrow W \cup \{v\}$ **fi.**
od
- Step 7.* (There is one vertex left.)
Let v be the one vertex of T' ;
set $b(T) \leftarrow t(v)$;
let the neighbors of v in T be u_1, u_2, \dots, u_k , where $t(u_1) \geq t(u_2) \geq \dots \geq t(u_k)$;
let j be the smallest integer such that
 $t(u_j) + j = \max \{t(u_i) + i | 1 \leq i \leq k\}$;
set $BC(T) \leftarrow \{v, u_1, u_2, \dots, u_j\}$;
STOP

We observe that, if T is not a star, then the final vertex v identified in step 7 will have been labeled twice—once when it became an endpoint of T' , and again in step 6 when T' has only the one vertex v . The second label for v is no smaller than the first.

Figure 2 illustrates the application of algorithm BROADCAST to a tree T . In Fig. 2(a) U is the set of endvertices of T , each with label “0,” and each of the vertices in W has been labeled. The first vertex which will be moved from W to U (that is, the “ W ” in

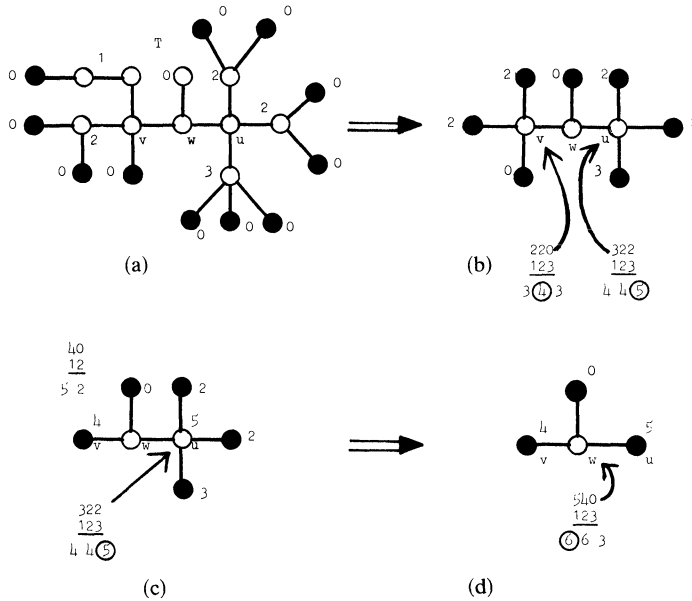


FIG. 2. Algorithm BROADCAST applied to tree T .

step 4) is the one with label “1.” In Fig. 2(b) we see that $t(v) = 4$ and $t(u) = 5$, and so v is the next vertex moved to U . In Fig. 2(c) vertex u receives the value $t(u) = 5$. Finally, in Fig. 2(d) vertex w receives the value 6, and, according to step 7, the broadcast center consists of vertices u and w .

3. Proof of correctness. The following analysis establishes the correctness of Algorithm BROADCAST. With v the root of a subtree T_v of T , $b(v, T_v)$ denotes the broadcast number of v in T_v . Note that, at each step of Algorithm BROADCAST, the subgraph induced by the unlabeled vertices is connected (i.e., is a subtree of T). In what follows we assume that $|V(T)| \geq 3$. The case where $|V(T)| \leq 2$ is easily treated in step 1 of Algorithm BROADCAST.

THEOREM 1. *Let v_1, v_2, \dots, v_k be the neighbors of a vertex w in a tree T , and let $T_i = T(v_i, w)$, for $i = 1, 2, \dots, k$. Suppose further that $b(v_i, T_i) \geq b(v_{i+1}, T_{i+1})$, for $i = 1, 2, \dots, k - 1$. Then,*

$$b(w, T) = \max \{b(v_i, T_i) + i \mid 1 \leq i \leq k\}.$$

Proof. Suppose that v_i receives the information from w at time $\pi(i)$, $i = 1, 2, \dots, k$, where π is an element of the symmetric group S_k of all permutations of $\{1, 2, \dots, k\}$. Then every vertex in T_i may be called by time $b(v_i, T_i) + \pi(i)$, and we deduce that

$$b(w, T) = \min_{\pi \in S_k} \left\{ \max_{1 \leq i \leq k} \{b(v_i, T_i) + \pi(i)\} \right\}.$$

It is clear that the permutation $\pi(i) = i$ minimizes this expression. \square

LEMMA 2. *Suppose in step 5 that Algorithm BROADCAST deletes vertex w from W and adds it to U . If v is adjacent to w in T' , then $t(w) = b(w, T(w, v))$.*

Proof. This follows immediately from Theorem 1 and the observation that if u is an endvertex of T adjacent to vertex w , then $t(u) = 0 = b(u, T(u, w))$. \square

LEMMA 3. Let x_1 and x_2 be adjacent vertices in T , and assume $b(x_1, T(x_1, x_2)) \leq b(x_2, T(x_2, x_1))$. Then

- 1) $b(x_1, T) = 1 + b(x_2, T(x_2, x_1))$, and
- 2) $b(x_2, T(x_2, x_1)) \leq b(x_2, T) \leq 1 + b(x_2, T(x_2, x_1))$.

Proof. The obvious proof is omitted.

LEMMA 4: If w_1, w_2, \dots, w_{m-1} is the sequence of vertices selected in step 4, and w_m is the one remaining vertex v of step 7, then $t(w_1) \leq t(w_2) \leq \dots \leq t(w_m)$.

Proof. Suppose $1 \leq k < h \leq m$. It suffices to show that $t(w_k) \leq t(w_h)$. Vertex w_k is chosen during the k th execution of step 4. If w_h has received the label $t(w_h)$ during one of the first $k - 1$ executions of step 6, then $t(w_k) \leq t(w_h)$ by definition of how w_k is selected in step 4. If w_h receives label $t(w_h)$ during the j th execution of step 6, where $j \geq k$, it suffices to note that $t(w_h) > t(w_j)$. \square

LEMMA 5. At any stage after step 1 in Algorithm BROADCAST, some vertex in T' is in the broadcast center of T .

Proof. Let w be a vertex selected in step 4 to be removed from the current T' and W and added to the current U , and let v be the vertex adjacent to w in T' . It suffices to show that $b(v, T) \leq b(w, T)$.

First we show that $b(w, T(w, v)) \leq b(v, T(v, w))$. If $V(T') = \{w, v\}$, then the labels on w and v are $t(w) = b(w, T(w, v))$ and $t(v) = b(v, T(v, w))$ and $t(w) \leq t(v)$. Hence $b(w, T(w, v)) \leq b(v, T(v, w))$. Suppose that T' has an endpoint $x \notin \{v, w\}$, and let x' be the vertex in T' adjacent to x . By Lemma 4, $b(x, T(x, x')) \geq b(w, T(w, v))$. But $T(x, x')$ is a subtree of $T(v, w)$ implies $b(v, T(v, w)) \geq b(x, T(x, x'))$.

Now, by Lemma 3, $b(w, T) = 1 + b(v, T(v, w)) \geq b(v, T)$. \square

LEMMA 6. If v and w are adjacent vertices in a tree T and $b(w, T(w, v)) \leq b(v, T(v, w))$, then for all vertices x of $T(w, v) - w$ one has $b(x, T) > b(v, T)$.

Proof. By Lemma 3, $b(v, T) \leq 1 + b(v, T(v, w))$. Hence $b(x, T) \geq 2 + b(v, T(v, w)) > 1 + b(v, T(v, w)) \geq b(v, T)$, as required. \square

THEOREM 7. Let v be the single remaining vertex in step 7 of Algorithm BROADCAST, and let u_1, u_2, \dots, u_k be the neighbors of v ordered so that $t(u_i) \geq t(u_{i+1})$, $i = 1, \dots, k - 1$. Let j be the smallest integer such that

$$t(u_j) + j = \max_{1 \leq i \leq k} \{t(u_i) + i\}.$$

Then $BC(T) = \{v, u_1, u_2, \dots, u_j\}$ and $b(T) = t(u_j) + j = t(v)$.

Proof. By Lemma 5 we know that $v \in BC(T)$, and from Theorem 1 and Lemma 2,

$$b(T) = \max_{1 \leq i \leq k} \{t(u_i) + i\} = t(u_j) + j = t(v).$$

Let s be a vertex other than v which is not adjacent to v , and let w be the vertex adjacent to v on the path from v to s . Since w is labeled, by Lemma 4 we have

$$b(w, T(w, v)) \leq b(v, T(v, w)).$$

Hence, by Lemma 6,

$$b(s, T) > b(v, T),$$

and s is not in $BC(T)$. Hence, every vertex in $BC(T)$ is adjacent to v .

We next show that for all h , $1 \leq h \leq j$, $u_h \in BC(T)$.

Let T_i be defined as in Theorem 1. In an optimal series of phone calls originating from vertex v , Theorem 1 asserts that for $i = 1, \dots, k$, vertex u_i is called at time i . Now

suppose that vertex u_h is the originator and v is called first. Vertex v then calls $\{u_i: i = 1, \dots, h-1, h+1, \dots, k\}$ in this sequential order, i.e., vertices u_1, u_2, \dots, u_{h-1} are called by v at times $2, 3, \dots, h$, respectively, and vertices u_{h+1}, \dots, u_k are called at times $h+1, \dots, k$, respectively. Therefore, for $1 \leq i \leq h-1$, all of the vertices in T_i may be called by time $t(u_i) + i + 1 \leq t(u_j) + j$, by definition of h . For $h+1 \leq i \leq k$, the vertices in T_i may be called by time $t(u_i) + i \leq t(u_j) + j$, and the vertices in T_h may be called by time $t(u_h) + 1 \leq t(u_h) + h \leq t(u_j) + j$. Hence,

$$b(u_h, T) \leq t(u_j) + j = b(t) \quad \text{and } u_h \in BC(T),$$

as asserted.

It still remains to show that for $j+1 \leq h \leq k$, $u_h \notin BC(T)$. If u_h is the originator, let the times at which u_1, \dots, u_j are called from vertex v be t_1, \dots, t_j , respectively. We note that each $t_i \geq 2$. By Theorem 1, in order to call all the vertices in $\cup_{1 \leq i \leq j} T_i$ as quickly as possible from vertex u_h , we can choose $t_i = i + 1, i = 1, \dots, j$. Hence, $b(u_h, T) \geq 1 + j + b(u_j, T_j) = 1 + j + t(u_j) > j + t(u_j) = t(v) = b(v, T)$. \square

COROLLARY 8. *For any tree T , $BC(T)$ consists of a star with at least two vertices.*

An interesting by-product of Algorithm BROADCAST is that it can also find the broadcast number $b(v, T)$ of any vertex v in a tree T .

THEOREM 9. *Let v be a vertex in a tree T which is not in the broadcast center of T , and let the shortest distance from v to a vertex x in $BC(T)$ be k . Then $b(v, T) = k + b(x, T) = k + b(T)$.*

Proof. Clearly $b(v, T) \leq k + b(x, T)$.

For the converse, consider the path in T from v to x , and let the vertex adjacent to x on this path be w (where $v = w$ is possible). Clearly $b(w, T(w, x)) \leq b(x, T) - 1$. This implies that if $b(x, T(x, w)) < b(x, T)$ then, since one can broadcast from w by first calling x , $b(w, T) \leq 1 + \max\{b(x, T(x, w)), b(w, T(w, x))\} \leq 1 + \max\{b(x, T) - 1, b(x, T) - 1\} = b(x, T)$. Since this implies that w is also in $BC(T)$, which is a contradiction, we have $b(x, T(x, w)) = b(x, T)$. But $b(x, T(x, w)) = b(x, T)$ implies that $b(v, T) \geq k + b(x, T(x, w)) = k + b(x, T)$.

Consequently $b(v, T) = k + b(x, T) = k + b(T)$. \square

4. Complexity analysis of Algorithm BROADCAST. In this section we will show that Algorithm BROADCAST has a worst-case time (and space) complexity of $O(N)$ for a tree T with N vertices.

Clearly steps 0, 1 and 2 require at most $O(N)$ time. The endvertices of the trees T and T' in steps 1 and 2 can be found in $O(N)$ time, and the value of the label $t(w)$ assigned in step 2 is simply the number of endvertices in T adjacent to w . This number can be determined by making one pass over the list of vertices adjacent to w . The total time spent examining such adjacent vertices, for all endvertices of T' is $O(N)$, since T' has $O(N)$ edges.

Since at least one vertex w is deleted in every execution of step 5, the test in step 3 is executed in $O(N)$ time. We must therefore show that the total amount of time spent in the iteration involving steps 4, 5 and 6 is $O(N)$.

In order to do this we will create two useful data structures. The first data structure is an array L of N pointers, the i th of which corresponds to a linked list of vertices w of T for which $t(w) = i$. After the value for $t(v) = k$ is determined in step 6, we can, in constant time, add v to the end of the k th list.

The second data structure is another list NBR of N pointers, the i th of which corresponds to a linked list of those vertices w which are adjacent to vertex i and which have been given a value $t(w)$. Elements can be added to this list in constant time during

the execution of step 4. If vertex w is selected in step 4, it can be added to the *front* of the $\text{NBR}(v)$ list for the vertex v adjacent to w in T' . It is important to note two things about this addition. First, the value $t(w)$ will be greater than or equal to the value of any other vertex which has previously been added to v 's list; this is guaranteed by Lemma 4. Second, we can use the value $t(w)$ to form an updated value of $t(v) = \max_{1 \leq i \leq k} \{t(w_i) + i\}$ in constant time. For each vertex v , we maintain a value $\text{MAX}(v)$. As a new vertex w , with value $t(w)$, is added to v 's list, the updated value of $\text{MAX}(v)$ is determined by

$$\text{MAX}(v) \leftarrow \max \{ \text{MAX}(v) + 1, t(w) + 1 \}.$$

For example, if the current values of vertices adjacent to v are 2 2 0, and the current maximum is 4 from $\begin{pmatrix} 2 & 2 & 0 \\ 1 & 2 & 3 \end{pmatrix}$, and if the next value added to v 's list is 3, then the new maximum is

$$\max \{ 4 + 1, 3 + 1 \} = 5 \text{ from } \begin{pmatrix} 3 & 2 & 2 & 0 \\ 1 & 2 & 3 & 4 \end{pmatrix}.$$

Returning to step 4 we see that in order to select the next vertex w , all we need to do is either move a pointer to the next element on list $L(i)$ or find the first nonzero element on the next list $L(i + 1), L(i + 2)$, etc. The total time spent moving these pointers is clearly bounded by $O(N)$. The total time spent finding the vertices v adjacent to vertices w is also bounded by $O(N)$ since all we must do is examine, once for each vertex, the vertices adjacent to it. Thus the total time spent in step 4 is $O(N)$.

Since the set W in step 5 is essentially the set of unexamined items on the lists $L(i)$, the process of deleting w from W and T' simply involves moving a pointer to the next vertex after w on some list. The process of adding w to the set U is simply a matter of adding w to the front of the list $\text{NBR}(v)$. Thus the total time spent in step 5 is $O(N)$.

In step 6 the process of deciding if v is an endvertex of T' is simply a matter of knowing whether all, or all but one, of v 's neighbors appear on the list $\text{NBR}(v)$. This can easily be determined in constant time by a simple updating process. The process of determining the value of $t(v)$ is simply a matter of getting the current value of $\text{MAX}(v)$; and the process of adding v to W involves adding v to the end of list $L(\text{MAX}(v))$. Thus the total time spent per vertex in step 6 is bounded by a constant.

Consequently, the total time spent executing steps 3, 4, 5 and 6 is $O(N)$.

Finally, we must show that the time spent executing step 7 is bounded by $O(N)$. The process of getting the last vertex v remaining in T' is simply that of moving a pointer to the last nonzero element on a list $L(i)$. The process of determining the smallest integer j such that $t(u_j) + j = \max_{1 \leq i \leq k} \{t(u_i) + i\}$ is simply one of moving a pointer down the list $\text{NBR}(w)$ to the first place where the value $\text{MAX}(u_i) + i$ equals $\text{MAX}(v)$. This clearly requires at most $O(N)$ time.

Thus, Algorithm BROADCAST has worst-case time complexity of $O(N)$. The space requirements are also $O(N)$ since the lists $L(i)$ and $\text{NBR}(w)$ require $O(N)$ words each, and the tree T can be stored via linked lists also requiring $O(N)$ words.

5. NP-completeness. D. S. Johnson [10] has shown that the problem of determining $b(u)$ for an arbitrary vertex u in an arbitrary graph G is NP-complete. It is with his permission that we present this result here. To begin, we state a more general problem:

BROADCAST TIME. Given a graph $G = (V, E)$ with a specified set of vertices $V_0 \subseteq V$ and a positive integer k , is there a sequence

$$V_0, E_1, V_1, E_2, V_2, \dots, E_k, V_k,$$

where $V_i \subseteq V, E_i \subseteq E, E_i$ consists only of edges with exactly one vertex in V_{i-1} ,

$$V_i = V_{i-1} \cup \{v : uv \in E_i\},$$

and $V_k = V?$ (V_i is the set of vertices who are informed at time i with calls along the edges in E_i .)

It is easy to show that this general problem reduces to our special case when $|V_0| = 1$. We will first show that the general problem is NP-complete.

We will now relate this broadcast problem to the three-dimension matching (3DM) problem which has been shown (cf. Garey and Johnson [8]) to be NP-complete.

3DM. Let $X = x_1, \dots, x_m, Y = y_1, \dots, y_m, Z = z_1, \dots, z_m$ and let $M \subseteq X \times Y \times Z$. Does there exist a subset of M of size m such that each pair of elements of the subset disagree in all three coordinates?

We demonstrate that a solution to the problem 3DM is equivalent to a solution of the BROADCAST TIME problem with $k = 4$ in a certain graph G which can be constructed from the set M in time polynomial in m . The graph G is illustrated in Fig. 3. The independent sets V_0, M are equal in size and the bipartite subgraph induced by these sets is complete. If $(x_i, y_j, z_k) \in M$ then the corresponding vertex of M in G is joined to the vertices x_i of X, y_j of Y and z_k of Z . (For illustrative purposes (x_1, y_2, z_3) is assumed to be an element of M .) All other edges are precisely as depicted.

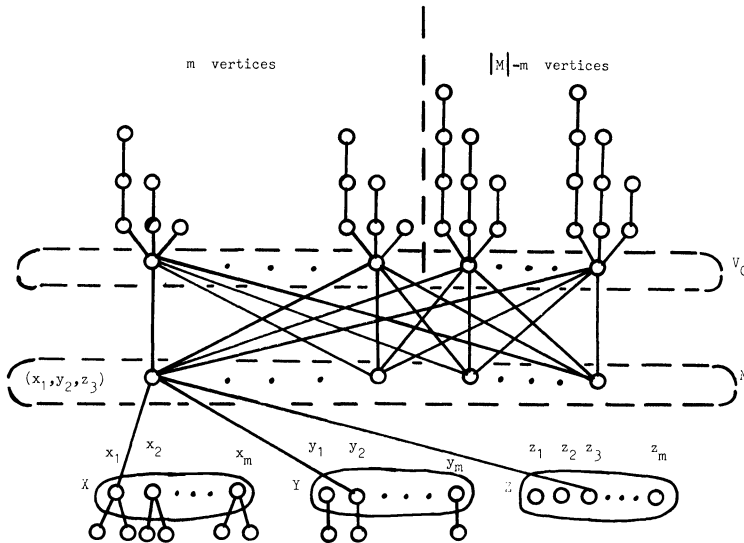


FIG. 3. The graph G corresponding to the problem 3DM.

Suppose that we have a solution to BROADCAST TIME in G . In order that the top line of vertices of G may be informed by time 4, $|M| - m$ vertices of V_0 must broadcast in an upward direction during the first time interval. In order that X, Y, Z and the bottom line may be informed by time 4, the remaining m vertices of V_0 must broadcast to an m -subset S of M at time 1. The vertices in S must be able to broadcast to distinct elements of X, Y, Z at times 2, 3, 4 respectively. This is possible if and only if S is a solution to 3DM.

To show that problem 3DM is reducible to the problem of determining $b(u)$ for an arbitrary vertex u in an arbitrary graph, we further extend the graph G to the graph H in Fig. 4. A solution to the broadcast time problem with $k = 4$ and V_0 the independent

vertex set of size $m \geq 2$ in G is equivalent to determining if $b(u) = m + 5$ in H , where H is obtained from G (in time polynomial in m) as follows. Add to G an independent set $U = \{u_1, u_2, \dots, u_m\}$ and a vertex u and edges (u, u_i) for $1 \leq i \leq m = |V_0|$. Append to u_1 paths of length 6, 7, \dots , $m + 4$, to u_2 paths of length 6, 7, \dots , $m + 3$, \dots , to u_{m-2} paths of length 6, 7, and to u_{m-1} a path of length 6. Finally, add m edges creating a matching of U to V_0 .

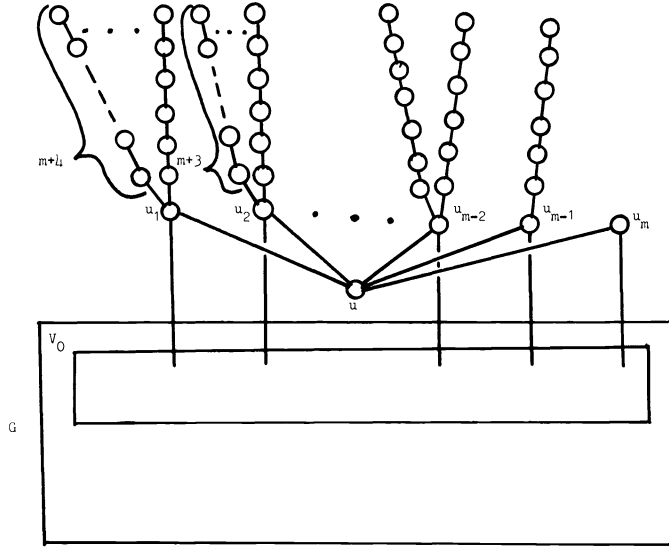


FIG. 4. Supergraph H of G .

6. Conclusions. Algorithm BROADCAST for finding the broadcast center of a tree raises a number of questions about the concept of broadcasting, or information dissemination, in communication networks. In particular, how much faster can broadcasting take place if one permits either “conference” calls, instead of two-person calls, or “long distance” calls instead of local, next-neighbor calls? Another class of problems concerns multiple-message broadcasting.

We initiated the study of the problem of transmitting information from one point to all points of a graph as quickly as possible in 1977 [4] when we presented most of the results in this paper. Subsequently much work has been done concerning this problem [3], [5], [6], [7], [9], [12]. In [4] we used the terms “gossip number” and “gossip center” rather than broadcast number and broadcast center. We prefer “broadcast” to “gossip” since gossip problems (for example [1], [2], [11]) involve every vertex (not just one) initiating a piece of information, and one wants to know how many calls are required and/or how long it will take before everyone knows everything.

Acknowledgments. The authors extend their thanks to the referees for many helpful comments and to Terry Beyer for discussions concerning the data structures and time complexity of Algorithm BROADCAST. We also thank Sandra Mitchell for helpful early discussions about broadcasting (and two of the authors congratulate the other on his marriage to her).

REFERENCES

[1] B. BAKER AND R. SHOSTAK, *Gossips and telephones*, Discrete Math., 2 (1972), pp. 191–193.
 [2] G. BERMAN, *The gossip problem*, Discrete Math., 4 (1973), p. 91.

- [3] E. J. COCKAYNE AND S. T. HEDETNIEMI, *A conjecture concerning broadcasting in m -dimensional grid graphs*, CS-TR-78-14, Computer Science Department, University of Oregon, submitted.
- [4] E. J. COCKAYNE, S. T. HEDETNIEMI AND P. J. SLATER, *The gossip center of a tree* (abstract), in Proc. 8th Southeastern Conference on Combinatorics, Graph Theory and Computing, 1977, p. 658.
- [5] A. M. FARLEY, *Minimal broadcast networks*, Networks, 9 (1979), pp. 313–332.
- [6] A. M. FARLEY AND S. T. HEDETNIEMI, *Broadcasting in grid graphs*, Proc. 9th Southeastern Conference on Combinatorics, Graph Theory and Computing, 1978, pp. 275–288.
- [7] A. M. FARLEY, S. T. HEDETNIEMI, S. L. MITCHELL AND A. PROSKUROWSKI, *Minimum broadcast graphs*, Discrete Math., 25 (1979), pp. 189–193.
- [8] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. F. Freeman, San Francisco, 1979.
- [9] S. T. HEDETNIEMI AND S. L. MITCHELL, *A census of minimum broadcast graphs*, J. Combinatorics, Inform. Systems Sci., 5 (1980), pp. 119–129.
- [10] D. S. JOHNSON, Private communication, May, 1978.
- [11] W. KNODEL, *New gossips and telephones*, Discrete Math., 13 (1975), p. 95.
- [12] A. PROSKUROWSKI, *Minimum broadcast trees*, IEEE Trans. Comput., to appear.

SOME COMPLEXITY RESULTS IN THE DESIGN OF DEADLOCK-FREE PACKET SWITCHING NETWORKS*

SAM TOUEG† AND KENNETH STEIGLITZ‡

Abstract. Deadlocks are very serious system failures and have been observed in existing packet switching networks (PSN's). Several problems related to the design of deadlock-free PSN's are investigated here. Polynomial-time algorithms are given for some of these problems, but most of them are shown to be NP-complete or NP-hard, and therefore polynomial-time algorithms are not likely to be found.

Key words. packet switching network, deadlock, complexity, NP-complete, flow control, routing, computer network, communication network

1. Basic definitions. A *packet switching network* (or PSN) is a directed graph $G = (V, E)$; the vertices V represent processors, and the edges E represent communication links. We assume messages, called *packets*, are to be passed between processors. Each vertex v_i has an associated constant b_i , the number of *buffers* at this vertex; a buffer can hold exactly one packet. Associated with each packet is an *acyclic route* v_1, v_2, \dots, v_q , which is a path in G . Vertex v_1 is the *source*, and v_q is the *destination* vertex for the packet. We assume a *fixed routing procedure* [KL], where a packet's route is determined at the source node. We may also assume that the route of a packet is included as part of the message in the packet, although in practice the packet could hold only the source and destination, with each processor in the network responsible for deducing the next vertex to which the packet is to be passed.

The *moves* made by the network are of three types:

1. *Generation.* A vertex v creates a packet which is placed in an empty buffer of v .
2. *Passing.* A vertex v transfers a packet in one of its buffers to an empty buffer of vertex w , where $v \rightarrow w$ is an edge, and the route for the packet has w following v . The buffer of v holding the packet becomes empty.
3. *Consumption.* A packet in a buffer of v , such that the destination for the packet is v , is removed from that buffer and the buffer is made empty.

2. Flow control procedures. A *flow control procedure* (or *controller*) for a network is an algorithm that permits or forbids various moves in the network. One of the key problems in packet switching is preventing *deadlock states*, which are situations in which one or more packets can never make a move. Deadlock states have been observed in existing packet switching networks [KL]; they tend to occur under near-saturation input load [GHKP]. For example, in the network of Fig. 1, if all physically possible moves are permitted by the controller, v_1 generates b_1 packets with destination v_2 , v_2 generates b_2 packets with destination v_3 , and v_3 generates b_3 packets with destination v_1 , then all buffers of all vertices will be full, no consumption moves can take place without a pass move, and no generation can take place. It is not hard to see that the network is deadlocked.

* Received by the editors July 11, 1979, and in final form October 29, 1980. This work was supported in part by the National Science Foundation under grant GK-42048, and in part by the U.S. Army Research Office, under grant DAAG29-75-0192.

† Department of Computer Science, Cornell University, Ithaca, New York 14853.

‡ Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey 08544.

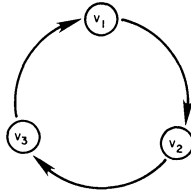


FIG. 1. A network exhibiting deadlock with a trivial controller.

However, if we use a controller that simply prohibits the generation (but not passing) of a packet into the last empty buffer at a vertex, then we can show that at least one empty buffer must exist somewhere in the network. Hence it is always possible to pass or consume some packet if there are any packets in the network, and this controller is deadlock-free.

In what follows, deadlock is assumed to occur with respect to some controller; that is, a controller is *deadlock-free* (or DF) for a given network if it does not permit this network to enter a state in which one or more packets can never make a move permitted by that controller.

3. Fundamental questions. We have assumed that each packet is generated with a fixed route to travel. There are still several options left to us:

1. Are we looking for a DF *uniform controller*, one that is deadlock-free for all networks? The design of optimal DF uniform controllers (optimal in the sense that they put the least restriction on moves) was investigated in [TU], [TO].
2. Are we designing a particular network with its flow control procedure such that this network is DF?

We investigate here the complexity of two deadlock-exposure problems related to the latter approach.

4. Deadlock-exposure problems. We consider the following two problems.

Problem 1. Given a network G and a set of source-destination routes in G , is the network *exposed to deadlock* (i.e., is there a deadlock state in G)?

Problem 2. Given a network G and a set of source-destination pairs of nodes in G , is there a corresponding set of routes in G such that the network is *not* exposed to deadlock?

The complexity of each of these two problems depends on the given buffer configuration of G and on the given flow control procedure applied in G . There are two possible buffer configurations for a network $G = (V, E)$. With a *general buffer configuration*, each node $v_i \in V$ has an individual buffer capacity b_i . With the *regular buffer configuration*, the buffer capacity b is the same for all the nodes $v \in V$. We consider four types of flow control procedures. The last three are commonly used in existing packet switching networks to avoid performance degradation under near-saturation input load.

(1) *Unrestricted flow control.* A node v accepts a packet p provided it has at least one empty buffer available for storing p . There are no other restrictions on packet moves.

(2) *Isarithmic flow control* [DA], [PR], [PRH]. With this form of flow control we have an additional restriction on the total number of packets that might be contained in the network at any given time. A packet can be generated in a node if this node has at least one empty buffer and the total number of packets in the network is less than a certain constant K .

(3) *Individual end-to-end window flow control* [GHKP], [KL], [PRH]. As in (1) with the following additional constraint. Each (v_i, v_j) source-destination pair has a constant k_{ij} associated with it; k_{ij} is an upper limit on the number of packets with source v_i and destination v_j . Such a packet can be generated in the node v_i provided v_i has at least one empty buffer and there are fewer than k_{ij} such packets in the network.

(4) *Regular end-to-end window flow control* [KL], [GHKP], [[RH]. As in (3) but k_{ij} is the same constant k for all the (v_i, v_j) source-destination pairs.

The complexity of Problem 1 and Problem 2 under the different buffer configurations and different flow control procedures¹ is summarized in Table 1 and Table 2.

TABLE 1
Time complexity of Problem 1.

Flow control applied	unrestricted	isarithmic: K	individual end-to-end: k_{ij}	regular end-to-end: k
General buffer conf.: b_i	Polynomial: $O(E)$	Polynomial: $O(V^3)$	NP-complete	NP-complete
Regular buffer conf.: b	Polynomial: $O(E)$	Polynomial: $O(V^3)$	NP-complete	NP-complete

TABLE 2
Time complexity of Problem 2.

Flow control applied	unrestricted	isarithmic: K	individual end-to-end: k_{ij}	regular end-to-end: k
General buffer conf.: b_i	NP-complete	NP-complete	NP-hard (in NP?)	NP-hard (in NP?)
Regular buffer conf.: b	NP-complete	NP-complete	NP-hard (in NP?)	NP-hard (in NP?)

THEOREM 1. *If the network $G = (V, E)$ has a general buffer configuration and the unrestricted flow control is applied, then Problem 1 is solvable in $O(|E|)$ time.*

Proof. It is easy to verify that such a network has a deadlock state if and only if the routes in this network form at least one cycle. A simple breadth-first-search along the routes of G may be used to detect such a cycle. \square

COROLLARY 1. *If the network $G = (V, E)$ has a regular buffer configuration and the unrestricted flow control is applied, then Problem 1 is solvable in $O(|E|)$ time.*

Proof. Immediate consequence of Theorem 1. \square

THEOREM 2. *If the network $G = (V, E)$ has a general buffer configuration and the isarithmic flow control (with constant K) is applied, then Problem 1 is solvable in $O(|V|^3)$ time.*

Proof. In such a network G there is a deadlock state if and only if the routes of G form at least one cycle such that the sum of the buffer capacities of the nodes along this cycle is at most K . A slight modification of Warshall's algorithm can be used to determine the *minimal cycle* formed by the routes in G (minimal in the sense that the

¹ In the formulation of these problems both the buffer configuration and the flow control procedure, with the corresponding constants, are given as part of the input.

sum of the buffer capacities of the nodes along this cycle is minimal with respect to all the other cycles). \square

COROLLARY 2. *If the network $G = (V, E)$ has a regular buffer configuration and an isarithmic flow control is applied, then Problem 1 is solvable in $O(|V|^3)$ time.*

Proof. This is a particular case of Theorem 2. \square

In [VA] an exponential-time algorithm is given to solve Problem 1 when G has a general buffer configuration and an individual end-to-end window flow control is applied. We now prove that, in this case, Problem 1 is NP-complete and, consequently, a polynomial-time algorithm is not likely to be found.

THEOREM 3. *If G has a general buffer configuration and an individual end-to-end window flow control is applied, then Problem 1 is NP-complete.*

Proof. Let $G = (V, E)$ be a network with a general buffer configuration and an individual end-to-end window flow control. In Problem 1 we ask if such a network has a deadlock state. This problem is in NP; we can guess a deadlock state and check its consistency with the given buffer configuration and end-to-end window flow control in polynomial time. We now show how to reduce (in polynomial time) the CNF-satisfiability problem [AHU] to Problem 1. Let $\mathbf{F} = \mathbf{F}_1 \mathbf{F}_2 \cdots \mathbf{F}_q$ be an expression in CNF, where the \mathbf{F}_j are the factors. Let x_1, x_2, \dots, x_n be the literals in \mathbf{F} . We construct the following network $G = (V, E)$. The nodes V of G are given by the set

$$V = \{v\} \cup \{F_j \mid \text{for } 1 \leq j \leq q\} \cup \{x_i, \bar{x}_i, w_i, \bar{w}_i \mid \text{for } 1 \leq i \leq n\}.$$

The edges of G are given by the following set

$$\begin{aligned} E = & \{(v, F_j) \mid \text{for } 1 \leq j \leq q\} \\ & \cup \{(w_i, v), (\bar{w}_i, v) \mid \text{for } 1 \leq i \leq n\} \\ & \cup \{(x_i, w_i), (\bar{x}_i, \bar{w}_i), (w_i, \bar{x}_i) \mid \text{for } 1 \leq i \leq n\} \\ & \cup \{(F_j, x_i) \mid x_i \text{ is a variable in } F_j\} \\ & \cup \{(F_j, \bar{x}_i) \mid \bar{x}_i \text{ is a variable in } F_j\}. \end{aligned}$$

The buffer capacity is $b = 1$ for all the nodes except for the node v which has q buffers. The routes in the network are the following:

1. $\langle v, F_j \rangle$ for $1 \leq j \leq q$.
2. $\langle x_i, w_i, \bar{x}_i, \bar{w}_i \rangle, \langle w_i, v \rangle$ and $\langle \bar{w}_i, v \rangle$ for $1 \leq i \leq n$.
3. $\langle F_j, x_i \rangle$ if x_i is a variable in F_j .
4. $\langle F_j, \bar{x}_i \rangle$ if \bar{x}_i is a variable in F_j .

The upper limit on the number of packets in each of these routes is set to one (this is a case of regular end-to-end window flow control with $k_{ij} = k = 1$). We claim that this network has a deadlock state if and only if \mathbf{F} is satisfiable. Suppose \mathbf{F} is satisfiable. In each node F_j we generate a packet whose destination is the node x_i or \bar{x}_i where $x_i = 1$ or $\bar{x}_i = 1$ is a variable assignment that makes the factor \mathbf{F}_j true. In each such x_i or \bar{x}_i node we put one packet whose route is $\langle x_i, w_i, \bar{x}_i, \bar{w}_i \rangle$. The next node in the route of this packet is either w_i or \bar{w}_i , and in such a node we generate a packet whose destination is the node v . Finally, in v we generate q packets, one in each of the $\langle v, F_j \rangle$ routes. A variable assignment that satisfies \mathbf{F} cannot include both $x_i = 1$ and $\bar{x}_i = 1$; therefore no $\langle x_i, w_i, \bar{x}_i, \bar{w}_i \rangle$ route has a packet in both the x_i and \bar{x}_i nodes; this is consistent with the end-to-end window flow control upper limit of one packet per route. It is easy to check that the network is in a deadlock state. Suppose now that the network has a deadlock state. Then there is at least one cycle of deadlocked packets in this state. Therefore, the node v must be in the deadlocked set of nodes, and there are q packets

filling all the buffers of v . Since we may have at most one packet in each of the $\langle v, F_j \rangle$ routes it must be that each $\langle v, F_j \rangle$ route has exactly one packet stored at the source node v . Therefore, each node F_j must also be deadlocked and must contain one packet whose destination is a node of the form x_i or \bar{x}_i . Then, this x_i or \bar{x}_i node must also be deadlocked and its buffer must contain a packet in the $\langle x_i, w_i, \bar{x}_i, \bar{w}_i \rangle$ route. Since the upper limit on the number of packets in this route is one, it is not possible that both x_i and \bar{x}_i are in the deadlocked set of nodes (therefore there can be no two F_j and F_k nodes with packets whose respective destination is x_i and \bar{x}_i). Let X be the set of x_i and \bar{x}_i destination nodes for the packets in the F_j nodes. It is now clear that assigning the value 1 to all the variables in X is a consistent variable assignment which satisfies all the F_j factors. \square

An example of a CNF Boolean expression, $F = (\mathbf{x}_1 + \mathbf{x}_2)(\bar{\mathbf{x}}_1 + \mathbf{x}_3)$, of the corresponding network, of a variable assignment X satisfying F , and of the corresponding deadlock state are shown in Fig. 2.

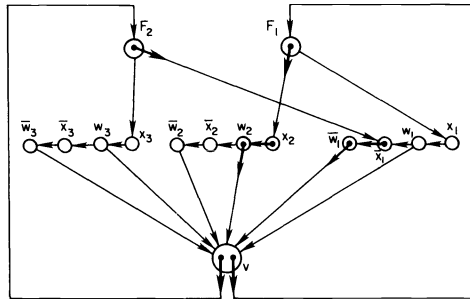


FIG. 2. Network deadlock corresponding to a satisfiable CNF Boolean expression.

In this case $F_1 = (\mathbf{x}_1 + \mathbf{x}_2)$ and $F_2 = (\bar{\mathbf{x}}_1 + \mathbf{x}_3)$. The corresponding network has the following routes:

1. $\langle F_1, x_1 \rangle, \langle F_1, x_2 \rangle, \langle F_2, \bar{x}_1 \rangle$ and $\langle F_2, x_3 \rangle$.
2. $\langle x_i, w_i, \bar{x}_i, \bar{w}_i \rangle, \langle w_i, v \rangle$ and $\langle \bar{w}_i, v \rangle$ for $i = 1, 2$ and 3 .
3. $\langle v, F_j \rangle$ for $i = 1$ and 2 .

All the nodes have a buffer capacity of one packet except the node v which has a buffer capacity of two packets. All the routes have an end-to-end window flow control of one packet per route. A variable assignment satisfying F is given by $X = \{\mathbf{x}_2, \bar{\mathbf{x}}_1\}$, (i.e., $\mathbf{x}_2 = \bar{\mathbf{x}}_1 = 1$). A corresponding deadlock state is the following. We have two packets in v with destination F_1 and F_2 , a packet in F_1 with destination x_2 , a packet in F_2 with destination \bar{x}_1 , a packet in x_2 with destination w_2 , a packet in \bar{x}_1 with destination \bar{w}_1 , a packet in w_2 with destination v and a packet in \bar{w}_1 whose destination is also v .

COROLLARY 3. *If the network G has a general buffer configuration and a regular end-to-end window flow control is applied, then Problem 1 is NP-complete.*

Proof. In the proof of Theorem 3, the window flow control constant was globally set to $k_{ij} = k = 1$. \square

THEOREM 4. *If the network G has a regular buffer configuration and a regular end-to-end window flow control is applied, then Problem 1 is NP-complete.*

Proof. The reduction of the CNF-satisfiability problem to Problem 1 given in Theorem 3 can be modified in the following way. Each node in G is now provided with q buffers, there is an additional set of $q - 1$ nodes, $\{y_j | \text{for } 1 \leq j \leq q - 1\}$, and except for the node v , all the nodes $w \in V$ have an additional set of $q - 1$ edges directly connecting

them to the y_j nodes forming $q - 1$ $\langle w, y_j \rangle$ new routes. Also, for each node y_j ($1 \leq j \leq q - 1$) there is an additional set of q edges directly connecting y_j to the nodes F_k , for $1 \leq k \leq q$, thus forming q $\langle y_j, F_k \rangle$ new routes. The end-to-end window flow control upper limit for all the routes is still one packet per route. We first prove that if \mathbf{F} is satisfiable then the network has a deadlock state. We begin by constructing the network state described in the first part of Theorem 3. Then, in each node w ($w \neq v$) that contains exactly one packet in this state, we generate $q - 1$ additional packets, one packet for each of the new $\langle w, y_j \rangle$ ($1 \leq j \leq q - 1$) routes. Also, in each node y_j ($1 \leq j \leq q - 1$) we generate q packets, one for each one of the new $\langle y_j, F_k \rangle$ ($1 \leq k \leq q$) routes. This is a deadlock state. We now show that if the network has a deadlock state then \mathbf{F} is satisfiable. If the network has a deadlock state then there is at least one cycle of deadlocked packets. Therefore, a node

$$w \in \{v\} \cup \{y_j \mid 1 \leq j \leq q - 1\}$$

must be in the deadlocked set of nodes, and there are q packets filling all the buffers of this node. Since we may have at most one packet in each of the $\langle w, F_j \rangle$ routes ($1 \leq j \leq q$) it must be that each $\langle w, F_j \rangle$ route has exactly one packet stored at the source node w . Therefore, each node F_j must also be deadlocked and must contain q packets. Each one of the $\langle F_j, y_k \rangle$ routes ($1 \leq k \leq q - 1$) can have at most one packet, so F_j must contain at least one packet whose destination is a node of the form x_i or \bar{x}_i . Then this x_i or \bar{x}_i node must also be deadlocked and its buffers contain q packets; at least one of these packets must be in the $\langle x_i, w_i, \bar{x}_i, \bar{w}_i \rangle$ route. From here the proof is identical to the last part of the proof of Theorem 3. \square

COROLLARY 4. *If the network G has a regular buffer configuration and an individual end-to-end window flow control is applied, then Problem 1 is NP-complete.*

Proof. Theorem 3 shows that the problem is in NP and Theorem 4 shows that the problem is NP-hard. \square

We now consider the complexity of finding deadlock-free routes in a network under several buffer and flow configurations.

THEOREM 5. *If a network G has a regular or general buffer configuration and the unrestricted flow control is applied, then Problem 2 is NP-complete.*

Proof. We are given a network $G = (V, E)$ and a set of source-destination pairs of nodes with an unrestricted flow control, and we ask if there is a corresponding set of routes such that G does not have a deadlock state. This is equivalent to the following question. Is there a corresponding set of routes that do not form a cycle in G ? This latter problem is NP-complete. In fact, we can guess a cycle-free set of routes and check the correctness of our guess in polynomial time, and therefore the problem is in NP. We now show that the CNF-satisfiability problem can be reduced to it in polynomial time. Let $\mathbf{F} = \mathbf{F}_1 \mathbf{F}_2 \cdots \mathbf{F}_q$ be a CNF expression, and $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ be the literals in \mathbf{F} . We construct the following network $G = (V, E)$. The nodes of G are given by the set

$$V = \{s\} \cup \{F_j \mid \text{for } 1 \leq j \leq q\} \cup \{x_i, \bar{x}_i, w_i, \bar{w}_i \mid \text{for } 1 \leq i \leq n\}.$$

The edges of G are given by the set

$$\begin{aligned} E = & \{(s, x_i), (s, \bar{x}_i) \mid \text{for } 1 \leq i \leq n\} \\ & \cup \{(x_i, w_i), (\bar{x}_i, \bar{w}_i) \mid \text{for } 1 \leq i \leq n\} \\ & \cup \{(w_i, \bar{x}_i), (\bar{w}_i, x_i) \mid \text{for } 1 \leq i \leq n\} \\ & \cup \{(w_i, F_j) \mid x_i \text{ is a variable in } F_j\} \\ & \cup \{(\bar{w}_i, F_j) \mid \bar{x}_i \text{ is a variable in } F_j\}. \end{aligned}$$

The source-destination pairs are the following:

1. $s - F_j$ for $1 \leq j \leq q$.
2. $w_i - \bar{x}_i$ and $\bar{w}_i - x_i$ for $1 \leq i \leq n$.

We claim that \mathbf{F} is satisfiable if and only if these source-destination pairs have a corresponding cycle-free set of routes in G . We first note that the only routes connecting w_i with \bar{x}_i and \bar{w}_i with x_i are the direct ones using the (w_i, \bar{x}_i) and (\bar{w}_i, x_i) edges. Suppose \mathbf{F} is satisfiable, and let X be a consistent set of variables that, when all are set to 1, satisfy \mathbf{F} . We can give the following cycle-free set of routes. The routes for the $s - F_j$ pairs are $\langle s, x_i, w_i, F_j \rangle$ if $x_i \in X$ and x_i is in F_j , or $\langle s, \bar{x}_i, \bar{w}_i, F_j \rangle$ if $\bar{x}_i \in X$ and \bar{x}_i is in F_j . The routes for $w_i - \bar{x}_i$ and $\bar{w}_i - x_i$ are respectively $\langle w_i, \bar{x}_i \rangle$ and $\langle \bar{w}_i, x_i \rangle$. Suppose this set of routes forms a cycle; the only cycles in G are of the form $\langle x_i, w_i, \bar{x}_i, \bar{w}_i, x_i \rangle$, which includes both the (x_i, w_i) and (\bar{x}_i, \bar{w}_i) edges. Then, both x_i and \bar{x}_i must be in the set of variables X contradicting the consistency of X . Conversely, let R be a cycle-free set of routes corresponding to the given set of source-destination pairs. We showed that R must contain the (w_i, \bar{x}_i) and (\bar{w}_i, x_i) edges for $1 \leq i \leq n$. Since R is cycle-free, it cannot contain both the (x_i, w_i) and (\bar{x}_i, \bar{w}_i) edges for any $1 \leq i \leq n$. For each destination F_j ($1 \leq j \leq q$) there must be a variable x_i or \bar{x}_i (and a corresponding (x_i, w_i) or (\bar{x}_i, \bar{w}_i) edge) such that the route from the source s to F_j passes through this variable. Let X be the set of these variables when j ranges from 1 to q . It is now clear that X is a consistent set of variables that satisfies all the \mathbf{F}_j factors. \square

Let \mathbf{F} be the satisfiable CNF Boolean expression and X be the set of variables defined in the example given for Theorem 3. The corresponding network G and the cycle-free set of routes are illustrated in Fig. 3. The routes are $\langle s, x_2, w_2, F_1 \rangle$, $\langle s, \bar{x}_1, \bar{w}_1, F_2 \rangle$, $\langle w_i, \bar{x}_i \rangle$ and $\langle \bar{w}_i, x_i \rangle$ for $i = 1, 2, 3$.

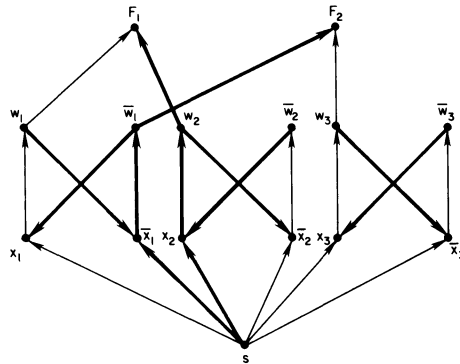


FIG: 3. The network G and a cycle-free set of routes.

THEOREM 6. *If the network G has a regular or a general buffer configuration and the isarithmic flow control (with constant K) is applied, then Problem 2 is NP-complete.*

Proof. The unrestricted flow control is equivalent to an isarithmic flow control where the constant K is larger than the total buffer capacity of the nodes of the network. Then, Problem 2 with an unrestricted flow control can be reduced (in polynomial time) to Problem 2 with an isarithmic flow control, and therefore this latter problem must be NP-hard. Also, if we are given a network G with the isarithmic constant K and a set of source-destination pairs, we can guess a corresponding deadlock-free set of routes in G and then check in polynomial time (using Warshall's algorithm) that the sum of the buffer capacities of the nodes along the minimal cycle is larger than K . Therefore, Problem 2 with an isarithmic flow control is in NP. \square

THEOREM 7. *If the network G has a regular or general buffer configuration and a regular or individual end-to-end flow control is applied, then Problem 2 is NP-hard.*

Proof. The unrestricted flow control is equivalent to a regular or individual end-to-end window flow control where the corresponding upper limits k or k_{ij} are large enough (for example if they exceed the total buffer capacity of the network). Then, Problem 2 with the unrestricted flow control (shown to be NP-complete in Theorem 5) reduces in polynomial time to Problem 2 with end-to-end window flow control. \square

We do not know whether the problems shown to be NP-hard in Theorem 7 are in NP or not.

5. State reachability, reachable deadlock states. Let G be a network and F be the flow control procedure applied in G . We say that a network state S of G is *reachable with respect to F* if, starting with an empty network G , there is a sequence of network moves allowed by F that results in the state S in G . We may be interested only in deadlock states that are reachable from an initially empty network G and redefine the notion of “exposure to deadlock” as follows. A network G is *exposed to deadlock* if G has a reachable deadlock state. An example of a non-reachable deadlock state S is the following.

Consider the network $G = (V, E)$, where

$$V = \{v_1, v_2, v_3\}, \quad E = \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\},$$

and each node has only one buffer. The routes in G are $r_1: \langle v_1, v_2, v_3 \rangle$, $r_2: \langle v_2, v_3, v_1 \rangle$ and $r_3: \langle v_3, v_1, v_2 \rangle$, and the unrestricted flow control is used in G . Let S be the following network state. A packet p_1 is in the node v_2 along the route r_1 , a packet p_2 is in the node v_3 along the route r_2 and a packet p_3 is in the node v_1 along the route r_3 . It is easy to check that S is a deadlock state which is not reachable from an initially empty network G .

We investigated the complexity of Problem 1 and Problem 2 under the new definition of deadlock exposure; the results, quite similar to previous ones, are summarized in Table 3 and Table 4.

TABLE 3
Time complexity of Problem 1 with the reachable deadlock definition.

Flow control applied	unrestricted	isarithmic: K	individual end-to-end: k_{ij}	regular end-to-end: k
General buffer conf.: b_i	Polynomial: $O(E)$?	NP-hard (in NP?)	NP-hard (in NP?)
Regular buffer conf.: b	Polynomial: $O(E)$?	NP-hard (in NP?)	NP-hard (in NP?)

TABLE 4
Time complexity of Problem 2 with the reachable deadlock definition.

Flow control applied	unrestricted	isarithmic: K	individual end-to-end: k_{ij}	regular end-to-end: k
General buffer conf.: b_i	NP-complete	NP-hard (in NP?)	NP-hard (in NP?)	NP-hard (in NP?)
Regular buffer conf.: b	NP-complete	NP-hard (in NP?)	NP-hard (in NP?)	NP-hard (in NP?)

Let S be a state of network $G = (V, E)$. We define the *precedence graph* G' of G relative to S as follows. $G' = (V, E')$ is a directed graph where (v, w) is an edge in E' if and only if when the network is in state S , there is a packet in w whose route passes through the node v before reaching the node w .

LEMMA 1. *With the unrestricted flow control procedure, if the precedence graph G' of a network G relative to a state S is acyclic then the state S is reachable.*

Proof. The algorithm given in Fig. 4 shows how the state S can be reached in G .

begin

topologically sort the directed acyclic graph G' ;

comment After this node sorting if (v_i, v_j) is an edge of G' then $i < j$;

for $j \leftarrow |V|$ **step** -1 **until** 1 **do**

begin

successively generate and move along their respective routes, from their source node to the node v_j , all the packets that are in the buffers of v_i when the state S is reached;

end

end

FIG. 4. Algorithm for reaching the state S .

Consider the inner loop of the algorithm. Suppose a packet with destination v_j cannot be generated or passed to an intermediary node v_i along its route, it must be that

1. the buffers of v_i are full of packets, so $i > j$;
2. (v_i, v_j) must be an edge of the precedence graph G' , therefore $i < j$; the contradiction is obvious. \square

A *cyclic deadlock state* of a network G is a state in which there exists a cycle of nodes in G such that the buffers of each node in the cycle are full of packets waiting to be passed to the next node in the cycle, and the nodes which are not in this cycle are empty.

LEMMA 2. *With the unrestricted flow control procedure, if a network G has a deadlock state S then it has a reachable cyclic deadlock state.*

Proof. If G has a deadlock state S then the routes of G contain a cycle, and therefore G has a cyclic deadlock state S_0 . According to Lemma 1, if S_0 is not a reachable state then the precedence graph G' of G relative to S_0 must have at least one cycle $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_1$. We can construct a cyclic deadlock state in the following way (without regard for reachability). We begin with the empty network G and we consider the edge (v_1, v_2) of the cycle. This edge is in the precedence graph and, by definition, there must be a packet in v_2 whose route r passes through the node v_1 before v_2 . We successively fill the buffers of v_1 and of all the intermediary nodes along the route r up to (but not including) v_2 with packets with route r . We do the same with the edge (v_2, v_3) and the successive edges of the cycle until we reach a node whose buffers are already full of packets² and a cyclic deadlock S_1 is thus formed. Note that S_1 involves a subset of the routes in S_0 , and the deadlocked packets in each such route are one step nearer to the source of the route than they were in the state S_0 . If S_1 is also not a reachable deadlock state then the corresponding precedence graph contains a cycle and the same process can be repeated to yield a new cyclic deadlock state S_2 (and so on). This process eventually results in a reachable deadlock state. In fact, let l be the

² This eventually occurs since $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_1$ is a cycle; note that it may occur before all the edges in the cycle are considered.

length of the longest route in G , then after at most l new nonreachable cyclic deadlock states generated by this process we have a cyclic deadlock state where every deadlocked packet is at the source of its route. Such a deadlock is reachable (the corresponding precedence graph is acyclic). \square

THEOREM 8. *If the network $G = (V, E)$ has a general or regular buffer configuration and the unrestricted flow control is applied, then Problem 1 (with the new definition of deadlock exposure) is solvable in $O(|E|)$ time.*

Proof. If such a network G has a reachable deadlock state then the routes in G form at least one cycle. If the routes of G form a cycle then G has a cyclic deadlock state and, by Lemma 2, it has a reachable deadlock state. So, G has a reachable deadlock state if and only if the routes in G form at least one cycle. An $O(|E|)$ time breadth-first-search along the routes of G can be used to detect such a cycle. \square

THEOREM 9. *If the network $G = (V, E)$ has a regular buffer configuration and a regular end-to-end window flow control is applied, then Problem 1 (with the new definition of deadlock exposure) is NP-hard.³*

Proof. It is easy to check that the polynomial time reductions of the CNF-satisfiability problem to Problem 1 given in the proofs of Theorem 3 and Theorem 4 involve reachable deadlock states. \square

COROLLARY 5. *If the network $G = (V, E)$ has a general or regular buffer configuration and an individual or regular end-to-end window flow control is applied then Problem 1 (with the new definition of deadlock exposure) is NP-hard.*

Proof. Immediate consequence of Theorem 9. \square

Note that, at the present time, nothing is known about the complexity of Problem 1 (with the new definition of deadlock exposure) when an isarithmic flow control procedure is applied.

THEOREM 10. *If a network has a regular or general buffer configuration and the unrestricted flow control is applied then Problem 2 (with the new definition of deadlock exposure) is NP-complete.*

Proof. We are given a network $G = (V, E)$ and a set of source-destination pairs of nodes with an unrestricted flow control, and we want to determine if there is a corresponding set of routes such that G does not have any reachable deadlock state. In the proof of Theorem 8 we showed that, with any set of routes, G does not have a reachable deadlock state if and only if these routes do not form a cycle. Then our problem is to determine if there is a set of routes that do not form a cycle. This problem was shown to be NP-complete in the proof of Theorem 5. \square

COROLLARY 6. *If the network $G = (V, E)$ has a general or regular buffer configuration and an isarithmic or an end-to-end window flow control is applied then Problem 2 (with the new definition of deadlock exposure) is NP-hard.*

Proof. The NP-complete problem of Theorem 10 can be easily reduced to the problems stated in the corollary: isarithmic and end-to-end window flow control with large constants are equivalent to unrestricted flow control. \square

REFERENCES

- [AHU] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [DA] D. W. DAVIES, *The control of congestion in packet switching networks*, IEEE Trans., COM-20 (1973), pp. 546-550.

³ Note that it is not known whether these problems are in NP.

- [G] K. D. GUNTHER, *Prevention of buffer deadlocks in packet switching networks*, IFIP-IIASA Workshop on Data Communications, Ladenberg, Austria, pp. g.15–g.19.
- [GHKP] A. GIESSLER, J. HAENLE, A. KOENIG AND E. PADÉ, *Free buffer allocation—an investigation by simulation*, *Computer Networks*, 2 (1978), pp. 191–208.
- [KL] L. KLEINROCK, *Queueing Systems vol. II: Computer Applications*, John Wiley, New York, 1976.
- [PR] W. L. PRICE, *Simulation studies of an isarithmically controlled store-and-forward data communications network*, Proc. IFIP Congress 74, Stockholm, August 1974, pp. 151–154.
- [PRH] W. L. PRICE AND J. D. HAENLE, *Some comments on a simulated datagram store-and-forward network*, *Computer Networks*, 2, (1978), pp. 70–73.
- [RH] E. RAUBOLD AND J. HAENLE, *A method of deadlock-free resource allocation and flow control in packet networks*, Proc. ICC 76, Toronto, Ont., Canada, August, 1976, pp. 483–487.
- [TO] S. TOUEG, *Design of deadlock- and livelock-free packet switching networks*, Ph.D. Thesis, Dept. of EECS, Princeton University, Princeton, NJ, 1979.
- [TU] S. TOUEG AND J. D. ULLMAN, *Deadlock-free packet switching networks*, Proc. 11th ACM Symposium on the Theory of Computing, Atlanta, GA, May, 1979, pp. 89–98.
- [VA] V. AHUJA, *Determining deadlock exposure for a class of store and forward communication networks*, *IBM J. Res. Dev.*, 24 (1980), pp. 49–55.

THE NP-COMPLETENESS OF SOME EDGE-PARTITION PROBLEMS*

IAN HOLYER†

Abstract. We show that for each fixed $n \geq 3$ it is NP-complete to determine whether an arbitrary graph can be edge-partitioned into subgraphs isomorphic to the complete graph K_n . The NP-completeness of a number of other edge-partition problems follows immediately.

Key words. computational complexity, NP-complete problems, edge-partition problems

1. Introduction. Many graph theory problems have been shown to be NP-complete and so are believed not to have polynomial time algorithms. Garey and Johnson [1] give an account of the theory of NP-completeness, a list of known NP-complete problems and a bibliography of the subject. In particular, they list several NP-complete vertex-partition problems [1, p. 193] including vertex-partition into cliques [2] and vertex-partition into isomorphic subgraphs [3].

In this paper, we consider some similar problems for edge-partitions. We define the edge-partition problem EP_n as follows. Given a graph $G = (V, E)$, the problem is to determine whether the edge-set E can be partitioned into subsets E_1, E_2, \dots in such a way that each E_i generates a subgraph of G isomorphic to the complete graph K_n on n vertices. Our main result is that the problem EP_n is NP-complete for each $n \geq 3$. From this we deduce that a number of other edge-partition problems are NP-complete.

In order to show that EP_n is NP-complete, we will exhibit a polynomial reduction from the known NP-complete problem 3SAT which is defined as follows. A set of clauses $C = \{C_1, C_2, \dots, C_r\}$ in variables u_1, u_2, \dots, u_s is given, each clause C_i consisting of three literals $l_{i,1}, l_{i,2}, l_{i,3}$ where a literal $l_{i,j}$ is either a variable u_k or its negation \bar{u}_k . The problem is to determine whether C is satisfiable, that is, whether there is a truth assignment to the variables which simultaneously satisfies all the clauses in C . A clause is satisfied if one or more of its literals has value "true".

2. The main theorem. Our first task is to find a graph which can be edge-partitioned into K_n 's in exactly two distinct ways. Such a graph can be used as a "switch" to represent the two possible values "true" and "false" of a variable in an instance of 3SAT.

For each $n \geq 3$ and $p \geq 3$ we define a graph $H_{n,p} = (V_{n,p}, E_{n,p})$ by

$$V_{n,p} = \left\{ \mathbf{x} = (x_1, \dots, x_n) \in \mathbb{Z}_p^n : \sum_{i=1}^n x_i \equiv 0 \right\},$$

$$E_{n,p} = \{ \mathbf{xy} : \text{there exist } i, j \text{ such that } y_k \equiv x_k \text{ for } k \neq i, j \text{ and } y_i \equiv x_i + 1, y_j \equiv x_j - 1 \}$$

where the equivalences are modulo p . Note that $H_{n,p}$ can be regarded as embedded in the $(n-1)$ -dimensional torus $T^{n-1} = S^1 \times S^1 \times \dots \times S^1$, and that the local structure of $H_{n,p}$ is the same for each p (see Fig. 1). The properties of $H_{n,p}$ are given in the following lemma.

LEMMA. *The graph $H_{n,p}$ has the following properties:*

- (i) *The degree of each vertex is $2 \binom{n}{2}$.*

* Received by the editors July 18, 1979, and in final form January 7, 1981.

† University Computer Laboratory, University of Cambridge, Cambridge, England. This work was supported by the British Science Research Council.

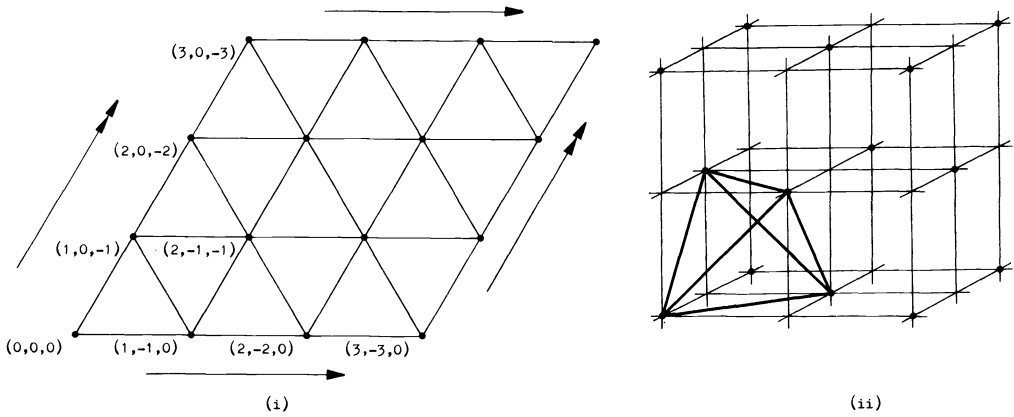


FIG. 1. (i) $H_{3,3}$ embedded in the (2-dimensional) torus. Opposite sides are identified as shown.
 (ii) The local structure of $H_{4,p}$. The edges of a single K_4 are shown.

- (ii) The largest complete subgraph is K_n , and any K_3 is contained in a unique K_n .
- (iii) The number of K_n 's containing a particular vertex is $2n$.
- (iv) Each edge occurs in just two K_n 's.
- (v) Each two distinct K_n 's are either edge-disjoint or have just one edge in common.
- (vi) There are just two distinct edge-partitions of $H_{n,p}$ into K_n 's.

Proof. (i) By translational symmetry we need only consider $\mathbf{0} = (0, \dots, 0)$. This is adjacent to $(1, -1, 0, \dots, 0)$ and the distinct points obtained from it by permuting its coordinates $(0, 1, -1)$ are distinct modulo p as $p \geq 3$. There are clearly $2\binom{n}{2}$ of these.

(ii) By translation and coordinate permutation we may assume that a largest complete subgraph contains the vertices $\mathbf{0} = (0, \dots, 0)$, $(1, -1, 0, \dots, 0)$ and $(1, 0, -1, 0, \dots, 0)$. It is then forced to be the *standard* K_n , which we call K and whose vertices are

$$\begin{aligned}
 &(0, 0, 0, \dots, 0) \\
 &(1, -1, 0, \dots, 0) \\
 &(1, 0, -1, \dots, 0) \\
 &\dots \\
 &(1, 0, 0, \dots, -1)
 \end{aligned}$$

(iii) The K_n 's containing $\mathbf{0}$ are obtained from K and its inverse $-K$ by cyclic permutation of the coordinates. Thus there are $2n$ of them.

(iv) We need only consider a particular edge containing the vertex $\mathbf{0}$ and check that it is contained in just two of the K_n 's given in (iii).

(v) If two K_n 's are not disjoint, we may assume that they have vertex $\mathbf{0}$ in common. We may then use (iii) to check that they have just one more vertex in common.

(vi) The edges containing $\mathbf{0}$ can be partitioned in at most two ways, and these extend to the whole of $H_{n,p}$. All the K_n 's are obtained from K or $-K$ by translation. One edge-partition consists of the translates of K , and the other consists of the translates of $-K$.

We now make the following definitions. The *T-partition* of $H_{n,p}$ (corresponding to logical value "true") consists of the translates of K , and the *F-partition* (corresponding to "false") consists of the translates of $-K$. Two K_n 's in $H_{n,p}$ are called *neighbors*

if they have a common edge. A *patch* is a subgraph of $H_{n,p}$ consisting of the vertices and edges of a particular K_n and of its neighbors. It is a *T-patch* if the central K_n belongs to the *T*-partition, and it is an *F-patch* otherwise. Two patches P_1, P_2 in $H_{n,p}$ are called *noninterfering* if the distance $d(\mathbf{x}, \mathbf{y})$ in $H_{n,p}$ between vertices $\mathbf{x} \in V(P_1)$ and $\mathbf{y} \in V(P_2)$ is always at least 10, say.

THEOREM. *The edge-partition problem EP_n is NP-complete for each $n \geq 3$.*

Proof. The problem EP_n is clearly in NP. Suppose we have an instance $C = \{C_1, C_2, \dots, C_r\}$ of 3SAT in s variables u_1, u_2, \dots, u_s where each C_i consists of literals $l_{i,1}, l_{i,2}$ and $l_{i,3}$. We reduce this instance of 3SAT to an instance $G_n = (V_n, E_n)$ of EP_n as follows.

Choose p sufficiently large so that up to $3r$ noninterfering patches can be chosen in $H_{n,p}$, say $p = 100r$. Take a copy U_i of $H_{n,p}$ to represent each variable u_i and copies $C_{i,1}, C_{i,2}$ and $C_{i,3}$ of $H_{n,p}$ to represent each clause C_i .

Join these copies of $H_{n,p}$ together as follows. If literal $l_{i,j}$ is u_k , then identify an *F-patch* of $C_{i,j}$ with an *F-patch* of U_k . If $l_{i,j}$ is \bar{u}_k , then identify an *F-patch* of $C_{i,j}$ with a *T-patch* of U_k as indicated for $n = 3$ in Fig. 2.

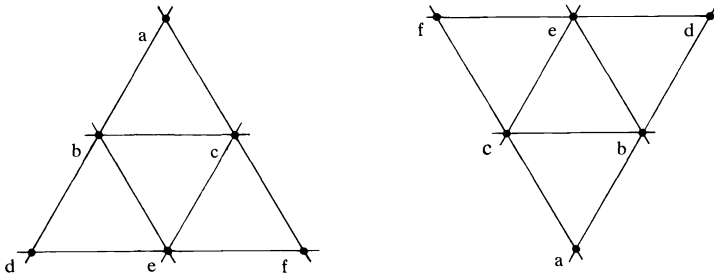


FIG. 2. The identification of an *F-patch* with a *T-patch* when $n = 3$. Similarly labeled vertices (and the edges between them) are identified.

Also join $C_{i,1}, C_{i,2}$ and $C_{i,3}$ for each i by identifying one *F-patch* from each and then removing the edges of the central K_n (see Fig. 3).

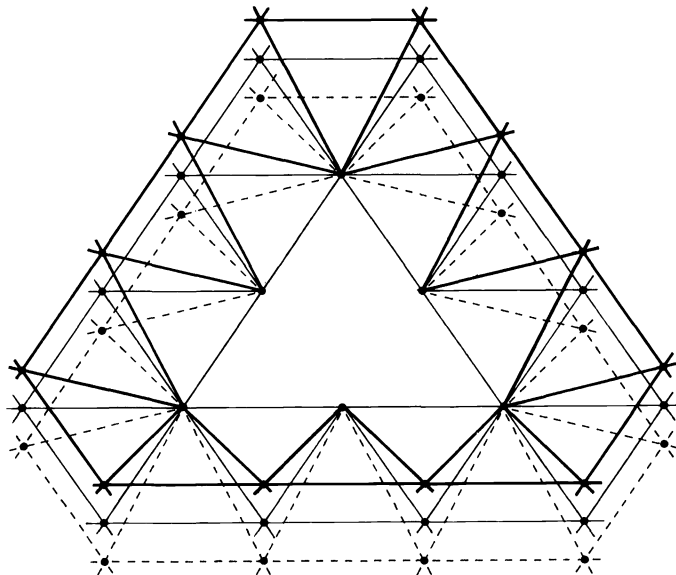


FIG. 3. The join between $C_{i,1}, C_{i,2}$ and $C_{i,3}$ when $n = 3$.

Choose all those patches which occur in a single copy of $H_{n,p}$ to be noninterfering.

Denote by $G_n = (V_n, E_n)$ the graph obtained in this way. We now show that there is an edge-partition of G_n into K_n 's if and only if the instance C of 3SAT is satisfiable.

Suppose that there is an edge-partition of G_n into a set S of K_n 's, and consider a particular copy H of $H_{n,p}$ involved in the construction of G_n . Take a K_n in S , say A , which is in H , but not near any join. Using the properties in the lemma, we see that the neighbors of A do not belong to S , the neighbors of the neighbors of A do belong to S , and so on. Continuing in this way, we deduce that all the edges of H , except perhaps those involved in joins, are T -partitioned, or all F -partitioned. Thus we may say that H is T -partitioned or F -partitioned.

Now suppose $l_{i,j}$ is u_k and consider the join between $C_{i,j}$ and U_k . We claim that the edges in the vicinity of this join can be edge-partitioned into K_n 's if and only if at least one of $C_{i,j}$, U_k is T -partitioned. If (say) $C_{i,j}$ is T -partitioned, this accounts for all the edges of $C_{i,j}$ near the joining patch except for those of the patch itself. The patch can then be regarded as belonging to U_k , which can then be locally partitioned in either way. If on the other hand both $C_{i,j}$ and U_k are F -partitioned, the argument of the previous paragraph shows that the edges of the patch not belonging to the central K_n are forced to belong to the F -partitions of both $C_{i,j}$ and U_k , which is a contradiction.

Similarly if $l_{i,j}$ is \bar{u}_k , then either $C_{i,j}$ is F -partitioned or U_k is T -partitioned.

Now consider the join between $C_{i,1}$, $C_{i,2}$ and $C_{i,3}$. We claim that the edges in the vicinity of this join can be edge-partitioned into K_n 's if and only if exactly one of $C_{i,1}$, $C_{i,2}$, $C_{i,3}$ is F -partitioned. The argument is the same as above, except that now, as the central K_n is missing, the remaining edges of the patch must be claimed by an F -partition in exactly one of $C_{i,1}$, $C_{i,2}$, $C_{i,3}$.

Thus if G_n can be edge-partitioned into K_n 's, then there is a truth assignment to u_1, \dots, u_s which satisfies C , namely u_k has value "true" if and only if U_k is T -partitioned.

If C is satisfiable, we partition G_n by partitioning U_k according to the truth of u_k in a satisfying assignment, choosing one "true" literal $l_{i,j}$ for each i , and F -partitioning the corresponding $C_{i,j}$.

It should be clear that the above reduction from 3SAT to EP_n can be carried out using a polynomial time algorithm, and so the proof of the theorem is complete. \square

3. Deductions. The following problems are now easily seen to be NP-complete.

- (i) Find the maximum number of edge-disjoint K_n 's in a graph ($n \geq 3$).
- (ii) Find the maximum number of edge-disjoint maximal cliques in a graph.
- (iii) Edge-partition a graph into the minimum number of complete subgraphs.
- (iv) Edge-partition a graph into maximal cliques.
- (v) Edge-partition a graph into cycles C_m of length m .

For (i) we use the same construction as for EP_n . For (ii), (iii) and (iv) we use the same construction as for EP_3 . Note that G_3 contains no K_4 's, and every edge K_2 is in a K_3 , so the maximal cliques coincide with the K_3 's.

For (v) we alter the construction for EP_3 in the following way. Note that the edges in $H_{3,p}$ occur in three distinct directions, say **a**, **b** and **c**, and that the joins in the construction of G_3 are made so that edges which are identified have the same direction. In G_3 , replace each edge with direction **a** (say) by a path of $m - 2$ edges.

We conjecture that the problem of edge-partitioning a graph into subgraphs isomorphic to a fixed graph H is NP-complete for all graphs H with at least 3 edges. The problem is polynomial if H has at most 2 edges, and it is easy to show that the

problem is NP-complete for a number of particular small, connected graphs H . The NP-completeness of the problem seems difficult to prove if H is disconnected, e.g., if $H = 3K_2$, that is, H has 6 vertices and 3 independent edges.

REFERENCES

- [1] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.
- [2] R. M. KARP, *Reducibility among combinatorial problems*, in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, eds., Plenum, New York, 1972, pp. 85–103.
- [3] D. G. KIRKPATRICK AND P. HELL, *On the complexity of a generalized matching problem*, in *Proc. 10th Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, 1978, pp. 240–245.

THE NP-COMPLETENESS OF EDGE-COLORING

IAN HOLYER†

Abstract. We show that it is NP-complete to determine the chromatic index of an arbitrary graph. The problem remains NP-complete even for cubic graphs.

Key words. computational complexity, NP-complete problems, chromatic index, edge-coloring

1. Introduction. The chromatic index of a graph is the number of colors required to color the edges of the graph in such a way that no two adjacent edges have the same color. By Vizing's theorem [1], the chromatic index is either d or $d + 1$, where d is the maximum vertex degree.

We prove the conjecture (Garey and Johnson [2, p. 286]) that it is NP-complete to determine the chromatic index of an arbitrary graph. In fact, we prove the stronger result that it is NP-complete to determine whether the chromatic index of a cubic graph is 3 or 4. Thus this problem probably has no polynomial time algorithm.

The terminology and results of NP-completeness are given in [2]. It is clear that the chromatic index problem is in the class NP. To prove that the problem is NP-complete, we exhibit a polynomial reduction from the known NP-complete problem 3SAT which is defined as follows. A set of clauses $C = \{C_1, C_2, \dots, C_r\}$ in variables u_1, u_2, \dots, u_s is given, each clause C_i consisting of three literals $l_{i,1}, l_{i,2}, l_{i,3}$, where a literal $l_{i,j}$ is either a variable u_k or its negation \bar{u}_k . The problem is to determine whether C is satisfiable, that is, whether there is a truth assignment to the variables which simultaneously satisfies all the clauses in C . A clause is satisfied if one or more of its literals has value "true".

2. The parity condition. We will use the following lemma given in Isaacs [3].

LEMMA. *Let G be a cubic, 3-edge-colored graph and $V' \subseteq V(G)$ a set of vertices of G . Let $E' \subseteq E(G)$ be the set of edges of G which connect V' to the remainder of the graph. If the number of edges of color i in E' is k_i ($i = 1, 2, 3$), then*

$$k_1 \equiv k_2 \equiv k_3 \pmod{2}.$$

Proof. If E_{12} is the set of edges of G which are colored with color 1 or 2, then E_{12} consists of a collection of cycles. Thus E_{12} meets E' in an even number of edges, and so $k_1 + k_2 \equiv 0 \pmod{2}$ which gives $k_1 \equiv k_2 \pmod{2}$. Similarly $k_2 \equiv k_3 \pmod{2}$. \square

3. The components used in the construction. Given an instance C of the problem 3SAT, we will show how to construct a cubic graph G which is 3-edge-colorable if and only if C is satisfiable. The graph G will be put together from pieces or "components" which carry out specific tasks. Information will be carried between components by pairs of edges. In a 3-edge-coloring of G , such a pair of edges is said to represent the value T ("true") if the edges have the same color, and to represent F ("false") if the edges have distinct colors.

The inverting component is shown with its symbol in Fig. 1. It was used by Loupekiné (see [4]) to construct a large family of cubic graphs with chromatic index 4. Using the parity condition above, it may be checked that if this component is 3-edge-colored, one of the pairs of connecting edges marked a, b or c, d must have equal colors and the remaining 3 edges must have distinct colors. There is no further

* Received by the editors January 31, 1980, and in final form January 7, 1981.

† University Computer Laboratory, University of Cambridge, Cambridge, England. This work was supported by the British Science Research Council.

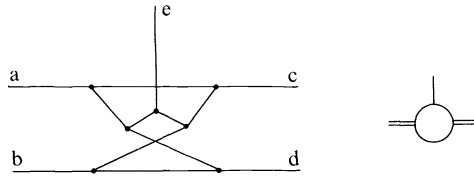


FIG. 1. The inverting component and its symbolic representation.

restriction on the possible colors of the five connecting edges. Regarding the pair of edges a, b as the input and the pair c, d as the output, the component changes a representation of T to one of F and vice versa.

The truth or falsity of each variable u_i will be represented by a variable-setting component such as that shown in Fig. 2. The component shown has 4 pairs of output edges, but in general the component representing u_i should have as many output pairs as there are appearances of u_i or \bar{u}_i among the clauses of C . It may be checked that in any 3-edge-coloring of a variable-setting component, all the output pairs must represent the same value.

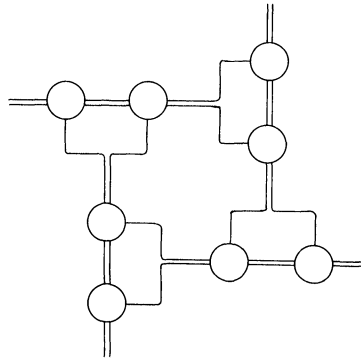


FIG. 2. The variable-setting component made from 8 inverting components and having 4 output pairs of edges. More generally it is made from $2n$ inverting components and has n output pairs.

The truth of each clause c_j will be tested by a satisfaction-testing component as shown in Fig. 3. This component can be 3-edge-colored if and only if the three input pairs of edges do not all represent F . The remaining connecting edges will be discussed later.

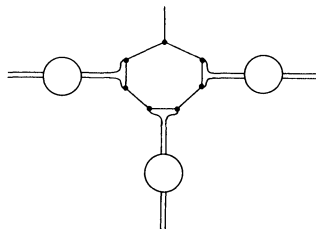


FIG. 3. The satisfaction-testing component.

4. The main theorem. We are now in a position to prove the following theorem.

THEOREM. *It is NP-complete to determine whether the chromatic index of a cubic graph is 3 or 4.*

Proof. The problem is clearly in the class NP. We exhibit a polynomial reduction from the problem 3SAT. Consider an instance C of 3SAT and construct from it a graph G as follows.

For each variable u_i take a variable-setting component U_i with one output pair of edges associated with each appearance of u_i or \bar{u}_i among the clauses of C . Take also a satisfaction-testing component C_j for each clause c_j . Suppose literal $l_{j,k}$ in clause c_j is the variable u_i . Then identify the k th input pair of C_j with the associated output pair of U_i . If, on the other hand, $l_{j,k}$ is \bar{u}_i , then insert an inverting component between the k th input pair of C_j and the associated output pair of U_i . The resulting graph H still has some connecting edges unaccounted for. The cubic graph G is formed from two copies of H by identifying the remaining connecting edges in corresponding pairs.

The graph G has a 3-edge-coloring if and only if the collection C of clauses is satisfiable, as can be verified using the properties of the components developed above. Moreover, the graph G can be produced from C using a polynomial time algorithm, so we have the result. \square

5. Comments. The above theorem may give some insight into the difficulty in classifying graphs according to their chromatic index. At any rate, it probably excludes the possibility of a polynomially checkable criterion, and it indicates that the restriction to cubic graphs is no easier.

Acknowledgments. I would like to thank M. Garey and D. Johnson for suggesting a simplification in the proof.

REFERENCES

- [1] S. FIORINI AND R. J. WILSON, *Edge-colourings of Graphs*, Pitman, London, 1977.
- [2] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.
- [3] R. ISAACS, *Infinite families of non-trivial trivalent graphs which are not Tait colorable*, Amer. Math. Monthly, 82 (1975), pp. 221–239.
- [4] ———, *Loupekine's snarks: A bifamily of non-Tait-colorable graphs*, unpublished.

OPTIMAL RETRIEVAL ALGORITHMS FOR SMALL REGION QUERIES*

AZAD BOLOUR†

Abstract. Hashing, or address calculation, has traditionally been associated with the scattering of close records, and therefore with inefficiency for retrieval of range queries. But for answering small range queries it is possible to compromise between randomization and the togetherness of close records by hashing many intervals of keys—rather than many isolated keys—into each hash value. And for answering small region queries, i.e., conjunctions of small range queries, such “interval hashing” may be applied to each attribute of a record in a multiple-key hashing procedure. The resulting technique is called “box-array hashing,” since it maps an orthogonal array of boxes from the space of all possible records into each hash value.

This paper analyzes the achievable efficiency of hashing techniques for answering small region queries, and presents strong analytic evidence suggesting that box-array hashing provides about the most efficient procedure for answering small region queries, among all hashing procedures of comparable randomization power.

Key words. region queries, range queries, hashing, multiple-key hashing, secondary key retrieval

1. Introduction. The use of secondary indexes or inverted files is perhaps the most common method of speeding up the examination of a file in answer to a Boolean combination of range queries. But as the number of records needed by a query increases, retrieval algorithms based on secondary indexing can become quite slow for answering such queries from external files [18], [19]. In fact, unless the file itself is arranged in such a way that close records appear close together, consulting an index for the given query may be of little use in limiting the number of secondary storage accesses to retrieve the needed records, when there are many such records. This problem is further aggravated by the useless retrieval of so-called “false-drops,” when there is no explicit index for a given query and indexes are available only for some of its parts.

A recent approach to the region retrieval problem is the use of tree structures similar to “tries” [11, § 6.3], known as multidimensional search trees [2], [9], [14]. These tree structures can be quite efficient for answering region queries, but their maintenance upon insertions and deletions generally requires rather involved procedures. Other data structures for region retrieval known generally as pyramidal structures [6], [13], [15], [22] lead to very fast access but entail considerable redundancy in storage.

For fast partial-match retrieval, Rothnie and Lozano [19], and Rivest [16], [17] have independently suggested the use of an alternative storage structure known as *multiple key hashing*. To store a record using a multiple key hashing procedure, each attribute of the record is first hashed separately via a hashing function of its own. The ordered sequence of these hash values is then used to identify the bucket where the given record is to be stored. Thus, if $\mathbf{x} = (x_1, \dots, x_n)$ represents a generic n -attribute record, and h_1, \dots, h_n are the hashing functions on the 1st, \dots , n th attribute spaces respectively, then the vector $(h_1(x_1), \dots, h_n(x_n))$ would determine the bucket address for \mathbf{x} . That is, there would be a one-to-one correspondence between such vectors and bucket addresses.

This scheme is the basis of the approach proposed here for the efficient retrieval of records in answer to small region queries. When dealing with region queries, of course,

* Received by the editors October 30, 1978, and in final revised form December 11, 1980. This work was supported by the Naval Electronics System Command under contract N00039-76-C-0022, and was prepared under the National Library of Medicine grant 2-404917 31529. The work is based in part on the author's PhD thesis, University of California, Berkeley, California.

† Medical Information Science Section, University of California, San Francisco, California 94143.

one can no longer use traditional hashing functions that scatter very close records into separate buckets. But for queries specifying small ranges, it is possible to compromise between the conflicting demands of randomization, for obtaining an even distribution of records into buckets, and “togetherness,” for efficiency in answering region queries. The idea is to use, for each attribute, a “hashing” function that is piecewise constant, but that maps many different pieces of the attribute space into a single hash value. Thus, we may “hash” an attribute by a composite function of the form hh' , where h' is order-preserving but h is randomizing. (This type of address calculation for single key storage and retrieval was first suggested to me by Eugene Wong.) Putting this together with the idea of multiple key hashing, we obtain address calculation functions which, for an n -attribute record $\mathbf{x} = (x_1, \dots, x_n)$, look like

$$h(\mathbf{x}) = h(x_1, \dots, x_n) = \bar{h}(h_1(h'_1(x_1)), \dots, h_n(h'_n(x_n))),$$

where \bar{h} is a one-to-one function, h'_1, \dots, h'_n are order-preserving functions, and h_1, \dots, h_n are randomizing functions. We call such functions *box-array addressing functions*, since they map n -dimensional arrays of boxes from the universe of records into each bucket.

The principal contribution of this paper is analytic. Results of Rivest [16], [17], and later of Bolour [7], suggest that in order to find a hashing function that is most efficient for partial-match retrieval, one can usually restrict one’s attention to multiple key hashing functions (assuming that the functions considered are balanced, that is, they partition the universe of records into equal parts). In this paper, these results are extended to include region queries. The objective is to demonstrate that in a large class of reasonable addressing functions, a box-array function can often be found that is near-optimal for answering a given mix of small region queries.

To provide for “randomization” in the functions considered, the universe of records is divided into a number of similar local regions, and the functions are constrained to divide each of these regions equally between the given buckets. Subject to this constraint, we show how an approximate lower bound may be obtained on the average number of buckets that must be examined to answer region queries, under moderate assumptions on the probability of these queries. Computational results then show that this lower bound can usually be approximated by a box-array addressing function.

The remainder of this paper is organized as follows. Section 2 presents a brief survey of the available search strategies for region queries with a more thorough introduction to box-array addressing functions. Sections 3 and 4 develop the problem of finding optimal addressing functions for region queries, and §§ 5 and 6 analyze this problem. Section 7 concludes the paper by indicating how a near-optimal box-array addressing function may be determined in a given situation. The organization and the results to follow parallel those of an earlier paper [7] dealing with partial-match queries only.

2. Retrieval algorithms for region queries. For the purpose of comparison, a brief survey of the major available techniques for region retrieval is presented in this section. All of the techniques to be discussed, including address calculation, are useful both for internal and for external searching. But since the emphasis of this paper is on external searching, the discussion here is specialized to retrieval from direct access secondary storage media. The number of secondary storage accesses necessary to answer a query is then a reasonable measure of the work required for the query. This cost can be broken up into two parts: those accesses (if any) that are made to *search for* and *locate* the

buckets of the main file that (may) contain needed records, and those accesses that are subsequently made to the buckets of the main file in order to retrieve the needed records. The comparison of the different methods is based on these costs as well as on their storage and update costs. For a more general survey of techniques for region retrieval see Bentley and Friedman [4].

Searching algorithms based on *file inversion* or *secondary indexing* are most effective when the number of records needed by a query is small. Their performance, however, quickly deteriorates as the number of needed records increases. Assuming a random distribution of these records in b buckets, and more than a few records per bucket, the average number of buckets needed to retrieve r records (after they have been identified through an inverted file) can be approximated by $b(1 - (1 - (1/b))^r) \approx b(1 - e^{-r/b})$ (Yao [24]). So when the number of records needed by a query is comparable to or exceeds the total number of buckets, most buckets will on the average have to be accessed in answering a query, and an inverted list of needed records will be of little use. On the other hand, when the number of needed records is somewhat less than the total number of buckets, $b(1 - e^{-r/b}) \approx r$, and on the average about one bucket is accessed for each needed record. This rapid rise in the number of main file accesses as a function of the number of needed records is a more serious drawback of inversion techniques than the overhead cost of intersecting inverted lists in the process of answering conjunctive queries. Secondary indexes also require extra storage space, and they lead to increased update costs since an update must be reflected in each index.

In situations where many records are generally needed by a query, it becomes necessary to partition the main file into the given buckets in such a way that records needed by a typical query can be found in as few buckets as possible. One strategy that immediately suggests itself is to divide the record space into identical rectangular boxes and to try to store the records in each such box in a corresponding bucket (Fig. 1). (A "box" here is a Cartesian product of intervals.) Let us call this a *regular boxlike partition*.

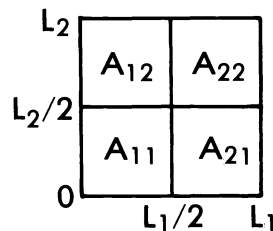


FIG. 1. *Regular boxlike partitioning.*

Knuth [11, p. 554] suggested keeping inverted lists of record identifiers for each box in a regular boxlike partition. Here, we have in mind the storage of records in their entirety in buckets corresponding to the boxes thus formed. Since this will usually lead to unequally sized buckets, we would need a table, a multidimensional array, to keep track of the location of each bucket. The retrieval algorithm would compute the indexes of those boxes that intersect a given query region, and locate the corresponding buckets by table lookup.

In order to avoid having to manage variably sized buckets and incurring their worst case performance, it is possible to use an "irregular" boxlike partitioning of a record space into boxes that may vary in shape but that contain the same number of records, and that are indexed not by an array but by a data structure generally known as a *multidimensional search tree*. A number of authors [2], [9], [14] have explored the use of

these data structures which are similar in nature to tries (Knuth [11, § 6.3]). A typical example for storing an n -dimensional file is a tree structure of kn levels, whose nodes at the 1st, $k + 1$ st, \dots , $(k - 1)n + 1$ st levels partition the values of the first attribute into successively finer intervals; those at the 2nd, $k + 2$ nd, \dots , $(k - 1)n + 2$ nd levels similarly partition the values of the second attribute, and so on. The partitions at each node are so chosen that the given records will be evenly distributed among the external nodes of the tree. Each node in such a tree structure corresponds to a box in the n -dimensional space of all possible records. To answer a region query, only those nodes whose associated boxes intersect the region specified in the query must be examined.

The general worst case performance analysis of multidimensional search trees yields $O(jN^{1-1/n} + r)$ retrieval time for a region query specifying j attribute ranges [12], where N is the total number of records. Their average performance analysis yields $O(\log N + r)$ and $O(r)$ retrieval times in the limit of small and large queries respectively [4]. Here we use a simple analysis of the average performance of multidimensional search trees for external searching, based on the assumption of uniform distribution of records. In that case a multidimensional search tree defines a regular boxlike partition, and the analysis of the two methods can be combined.

Consider an n -dimensional record space in which each attribute space is "normalized" to the interval $[0, 1)$. For simplicity, let the queries specify ranges of the same length, a , for each of j ($\leq n$) attributes (these queries will be referred to as a^j -queries), and suppose that each attribute is partitioned into an equal number, $(b^{1/n})$, of parts (the resulting partitioning of the record space will be referred to as a *cubical* partitioning). It is simple but tedious to show that if a is small compared to one, then on the average about $(b^{1/n})a + 1$ of the $(b^{1/n})$ parts of an attribute intersect a range of length a . So on the average about $((b^{1/n})a + 1)^n$ boxes of the boxlike partition of the record space intersect an a^n -query (assuming, of course, that the compounded approximation errors stay small). And in general it is easy to see that on the average about $(b^{1/n})^{n-i}((b^{1/n})a + 1)^i$ boxes of the cubical partition intersect an a^i -query.

The cost of accessing the directory nodes in the process of locating the needed main file nodes results in at least a logarithmic term in the complexity of retrieval from multidimensional search trees. For external tree structures with large fanouts, however, this cost can be expected to be negligible for many queries and most practical file sizes. But when a small range of values is specified for every attribute, the retrieval algorithm would require an average of close to one access to a main file bucket, but several accesses to directory buckets.

It is possible to make multidimensional search trees dynamic, i.e., capable of supporting insertions and deletions, at some cost. Saxe and Bentley's "binary transform" [21], for example, can be applied to multidimensional search trees to create a data structure that is capable of supporting both region retrieval and record insertion. The resulting increases in the query cost and the cost of building the structure are shown to be no larger than a factor of $\log N$. Willard [23] describes a complicated scheme for handling both insertions and deletions when using multidimensional search trees. It has a worst case complexity of $O((\log N)^2)$ for insertions and deletions and uses moderate storage redundancy.

In recent years, many authors have proposed the use of so-called "pyramidal structures" for range searching (Bentley and Shamos [6], Willard [22], Lueker [15], Lee and Wong [12]). In their simplest forms, these structures can be defined recursively as follows. A one-dimensional pyramidal structure is simply a search tree for a one-dimensional key space. Note that each node of this structure defines a range of values of the key space: the values that may appear as its descendants. An n -dimensional

pyramidal structure consists of a tree structure on the first attribute, in which the descendants of each node are stored redundantly in an $(n - 1)$ -dimensional pyramidal structure on the last $n - 1$ attributes. Thus, each node of the tree structure for the first attribute contains an extra pointer which points to the root of the corresponding $(n - 1)$ -dimensional pyramidal structure on the last $n - 1$ attributes. (Bentley and Maurer [5] also discuss similar schemes called “ k -ranges.”) The retrieval algorithm begins by finding the minimal set of nodes of the tree structure on the first attribute whose associated ranges can be put together to form the specified range of the first attribute. Then the algorithm is called recursively on each of the $(n - 1)$ -dimensional pyramidal structures of these nodes.

While these schemes lead to very efficient algorithms for range searching, they generally require considerable redundancy (except in two dimensions), and are therefore of little practical interest for files of modest size and many dimensions. Their general complexity analysis yields worst case retrieval times of $O((\log N)^n + r)$ at the expense of a factor of $O((\log N)^{n-1})$ storage redundancy. (A more complex scheme by Willard [22] uses somewhat more storage but yields $O((\log N)^{n-1} + r)$ retrieval time.) The average retrieval performance of these schemes for external region searching can be expected to be significantly better than the performance of the other schemes discussed here. But once again accesses to the internal nodes of a pyramid necessary for locating the needed external nodes may become significant for small queries.

An *address calculation function*, or an addressing function for short, is a function that assigns to each possible record the bucket wherein it is to be stored. The buckets are of fixed size. The function should be easily computable, and should have the property that for many situations it maps approximately equal numbers of records into each bucket. When more records are mapped into a bucket than there is room to hold them, the overflow records are stored in other buckets (which in the case of external searching may be specifically designated for overflow), and are chained to their addressed buckets. By using a good address calculation function, and by allowing somewhat more space in each bucket than the space needed by the average number of records mapped into a bucket, overflow can be kept down to a very small fraction of the file if buckets are reasonably large. This means an average of slightly more than two accesses for insertions and deletions in files of approximately constant size. When the file grows or shrinks, it has to be reorganized periodically into a larger or smaller number of buckets. But the amortized cost of reorganization per insertion or deletion may be kept down to a few accesses, e.g., by reorganizing when overflow or underflow is, say, $\frac{1}{3}$ the size of the allocated primary storage area for the file. The near-minimal average cost of insertions and deletions is one advantage of address calculation over multidimensional directories. Address calculation also eliminates the need to make accesses to secondary storage in searching for the addresses of the main file buckets needed for a query.

Usually addressing functions achieve the essential property of uniformity with respect to addresses by scattering close records into different buckets. This, of course, is exactly what should be avoided in partitioning a file for the efficient retrieval of region queries. When ranges specified in a query are small, however, we can use a compromise procedure that maps most small local regions into single buckets, but that also maps many different local regions distributed across the record space into each bucket.

To illustrate, consider the one-dimensional key space of all nonnegative integers less than a million, and assume that we are only concerned with range queries specifying ranges of width less than or equal to ten. Here we should like to have a function that maps most ranges of width ten into single buckets, but that also maps diverse regions of the space $0, \dots, 999999$ into each bucket. To do this we may first divide the key space

into 10,000 equal intervals of width 100, which will be treated as indivisible. Then, a randomizing hashing function can be used to map a number of these intervals into each bucket. For example, if 100 buckets numbered 0 to 99 are used, the entire address calculation function may take the form, $h: [0, 999999] \rightarrow [0, 99]$, $h(x) = \lfloor x/100 \rfloor \bmod 100$, where for a real number y , $\lfloor y \rfloor$ denotes the greatest integer less than or equal to y . The set of points mapped into each bucket by an addressing function is commonly known as a *cluster* of the given function. Each cluster of the above function is an *array* of 100 *intervals* distributed across the space $[0, 999999]$. Because such a cluster samples points from different regions of the key space, one may expect a reasonable degree of uniformity with respect to addresses from the corresponding addressing function. Also, most range queries specifying a range of width less than or equal to ten can be answered by inspecting a single bucket; and the remainder need only two buckets for their answers. We call such functions *piecewise constant hashing functions*. (Another example of such hashing functions appears in § 7.)

For multidimensional key spaces the compromise can be achieved by using the multiple-key hashing strategy of Rothnie [19], and Rivest [17] with component hashing functions that are piecewise constant. For example, to map the points $(x_1, x_2) \in [0, 999999]^2$ into 100 buckets we can use two piecewise constant hashing functions $h_1, h_2: [0, 999999] \rightarrow [0, 9]$ to obtain hash vectors $(h_1(x_1), h_2(x_2))$ which then *uniquely* identify buckets. Thus, two points in $[0, 999999]^2$ will be mapped into the same bucket if and only if their first coordinates are mapped into the same hash value and their second coordinates mapped into the same hash value. The one-to-one mapping of hash vectors into bucket numbers can be accomplished by lexicographic ordering. So by letting $h_1(x) = h_2(x) = \lfloor x/100 \rfloor \bmod 10$ the entire addressing function becomes:

$$(1) \quad h(x_1, x_2) = 10h_1(x_1) + h_2(x_2) = 10\left(\left\lfloor \frac{x_1}{100} \right\rfloor \bmod 10\right) + \left\lfloor \frac{x_2}{100} \right\rfloor \bmod 10.$$

Each cluster of this function is a 1000×1000 *two-dimensional array* of 100×100 squares distributed throughout the key space (Fig. 2). Therefore, one may again expect a reasonable degree of uniformity with respect to addresses.

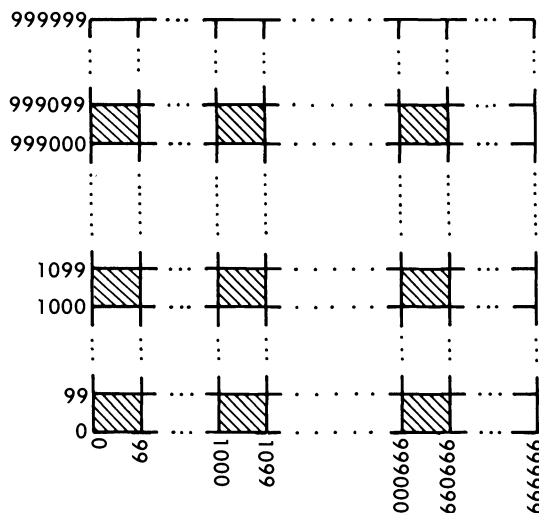


FIG. 2. A cluster of a box array addressing function in two dimensions. $h(x_1, x_2) = 10(\lfloor x_1/100 \rfloor \bmod 10) + \lfloor x_2/100 \rfloor \bmod 10$.

To retrieve the records stored in this way, and belonging to a given region $[a_1, a'_1] \times [a_2, a'_2]$, we need to inspect the set of buckets $h([a_1, a'_1] \times [a_2, a'_2])$, where h here represents the mapping from subsets of the key space into subsets of buckets, induced by the given addressing function. For the example function (1) this is

$$10 \times \left(\left[\left[\frac{a_1}{100} \right], \left[\frac{a'_1}{100} \right] \right] \bmod 10 \right) + \left[\left[\frac{a_2}{100} \right], \left[\frac{a'_2}{100} \right] \right] \bmod 10,$$

where the arithmetic operations of addition, multiplication by 10, and taking the remainder upon division by 10 are interpreted as acting on subsets of integers, and as producing subsets of integers, in an obvious way. If the specified ranges are small, the computation of such an expression is trivial, and will in most cases yield a single bucket address. For instance, the set of buckets that should be examined to answer the query, "retrieve all records (x_1, x_2) such that $5150 \leq x_1 \leq 5159$ and $134 \leq x_2 \leq 140$ " is $h([5150, 5159] \times [134, 140]) = 10([51, 51] \bmod 10) + [1, 1] \bmod 10 = 10[1, 1] + [1, 1] = \{11\}$.

What about range queries on a single attribute? For example, "retrieve all records (x_1, x_2) such that $a_1 \leq x_1 \leq a'_1$." The query region here is the rectangle $[a_1, a'_1] \times [0, 999999]$, and the set of needed buckets is therefore $10([a_1/100], [a'_1/100]) \bmod 10 + [0, 9]$. For a majority of small ranges $[a_1, a'_1]$ here, $[a_1/100]$ would be equal to $[a'_1/100]$, and hence 10 buckets would have to be inspected to answer the query. Similarly, the set of needed buckets for a range $[a, a'_2]$ of the second attribute is $10([0, 9] + [a_2/100], [a'_2/100]) \bmod 10$, and for a majority of small ranges $[a_2, a'_2]$, $[a_2/100]$ would be equal to $[a'_2/100]$ and the expression again yields 10 buckets. How does this compare with the performance of a retrieval algorithm based on boxlike partitioning? Assuming that the storage structure of the boxlike partitioning were also chosen to be symmetric with respect to the two attributes, we would again need ten accesses to main file buckets to answer most small range queries.

These ideas can be generalized in an obvious way to the storage of higher dimensional files, and to the retrieval of region queries specifying ranges of values for all, or for some of the attributes in a record.

For a simple approximate analysis of the performance of box-array addressing for a^i -queries, we may assume that each cluster is an $M \times M \times \dots \times M$ array of cubes spaced at regular intervals along each dimension in a normalized record space, $[0, 1]^n$. An analysis similar to that of a simple boxlike partitioning then shows that for a somewhat smaller than $1/M$, the approximate average number of buckets required to answer an a^i -query by using this scheme is $(b^{1/n})^{n-i} (1 + Ma(b^{1/n}))^i$. This performance is, of course, always worse than that of boxlike partitioning. But the two schemes become comparable for $jMa(b^{1/n}) < 1$. In that case, the approximate query time of box-array addressing for a^i -queries is roughly $(1 + jMa(b^{1/n}))b^{1-i/n}$ (and is in any case no larger than $eb^{1-i/n}$), whereas the approximate query time for boxlike partitioning is $(1 + ja(b^{1/n}))b^{1-i/n}$ (and is in any case no smaller than $b^{1-i/n}$).

How does the performance of box-array addressing compare with that of secondary indexing? Consider again the number of accesses to main file buckets required to answer an a^i -query. Let m be the number of records per bucket, so that the total number of records is bm . Assuming a uniform distribution of the records within $[0, 1]^n$, bma^i records would be needed by an a^i -query. The number of main file buckets needed to answer the query via secondary indexing is then no larger than bma^i . On the other hand, it is easy to see that the average number of main file buckets needed to answer an a^i -query via box-array addressing is no less than $bM^i a^i$ for a somewhat smaller

than $1/M$ and is equal to b for $a \geq 1/M$. Thus, when $m \leq M^i$, secondary indexing can be expected to outperform box-array addressing. When $m > M^i$ the relative efficiency of the two schemes depends on query size. In the region $a > 1/(m^{1/j})$, the number of records needed by the query is larger than the total number of buckets, and so most buckets of the main file are needed by a retrieval scheme based on secondary indexing. Box-array addressing, then, cannot do much worse than secondary indexing in this case. In the region $a \leq 1/(m^{1/j})$, the retrieval time of secondary indexing, $b(1 - e^{-r/b})$, can be roughly approximated by, $r = mba^i$. Comparing this with the retrieval time of box-array addressing, it can be shown that as the query size increases, the query time of secondary indexing deteriorates relative to that of box-array addressing. The critical point beyond which box-array addressing becomes more efficient is $\hat{a} = (1(b^{1/n}))/((m^{1/j}) - M)$.

The upshot of all this then is that for large buckets, large files, and queries specifying few ranges, box-array addressing can be expected to outperform secondary indexing, except when the specified ranges are very small.

3. Problem statement. To simplify the analysis, the universe of records is modeled as the unit n -cube $[0, 1]^n$ in n -dimensional Euclidean space. The development for discrete record spaces [8] is somewhat more cumbersome and is omitted. An addressing function $h : [0, 1]^n \rightarrow \{0, 1, \dots, b-1\}$ is called a b -way addressing function. We denote by H_i the i th cluster of an addressing function h , $H_i = h^{-1}(i)$, $0 \leq i < b$. The partition $\{H_0, \dots, H_{b-1}\}$ of $[0, 1]^n$ induced by h^{-1} will be denoted by \mathcal{H} . The addressing function and the partition are said to be *balanced* if all the associated clusters H_0, \dots, H_{b-1} have equal volume (Lebesgue measure).

While the property of balance appears to be necessary for an even distribution of records into buckets in a broad range of actual files, it is not sufficient. One consequence of the analytic results to follow is that often the most efficient *balanced* partition of a record space for answering region queries is a regular boxlike partition. But since this will seldom lead to an even distribution of records into buckets, we further restrict the class of functions we are going to consider to exclude such partitions.

In order to guard against partitions with contiguous, localized parts, each cluster of an admissible addressing function should be required to sample points from many different regions of the space $[0, 1]^n$. This can be done by dividing $[0, 1]^n$ into a number of local regions of equal volume and by requiring that each of these regions be divided equally between the clusters, or, equivalently, that each cluster be divided equally between these regions. Perhaps the simplest way to define these local regions is to partition $[0, 1]^n$ into a number of subcubes, say subcubes of dimensions $1/M$, for some integer $M > 1$. But since the space $[0, 1]^n$ actually represents a continuous approximation to a discrete record space whose various dimensions may contain widely differing numbers of elements, a partition into subcubes seems rather arbitrary. Instead we require a partition of $[0, 1]^n$ into identical rectangular local regions, or boxes. The dimensions of these boxes will be supplied by the user based on a knowledge of the attribute spaces and on the degree of randomization required. We shall say more about the choice of these dimensions later. For the moment let these dimensions be $1/M_1, \dots, 1/M_n$ respectively, for some integers M_1, \dots, M_n , so that $[0, 1]^n$ is initially partitioned into $M_1 M_2 \dots M_n$ such boxes. We call such a partition the boxlike partition of characteristic $\mathbf{M} = (M_1, \dots, M_n)$, and we restrict our attention to those addressing functions that divide each box of this partition equally into the given clusters. Equivalently, the admissible addressing functions are those whose clusters are divided equally between the $M_1 M_2 \dots M_n$ boxes. Such clusters will be known as *uniformly scattered clusters* with respect to \mathbf{M} , and the associated addressing functions (or

partition) will be known as a uniformly scattering function (or partition) with respect to \mathbf{M} (Fig. 3).

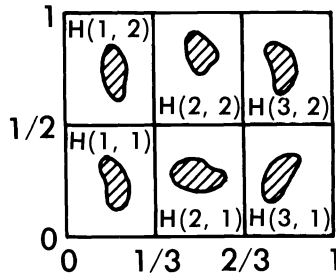


FIG. 3. A uniformly scattered cluster. $M_1 = 3, M_2 = 2$.

The word “query” or the notation Q will be used alternately for the condition specified in a query, and for the region in $[0, 1]^n$ defined by this condition, as well as for the request to retrieve all records satisfying the given condition. To answer a query Q , the retrieval algorithm must inspect a bucket if and only if the cluster associated with that bucket intersects Q . The number of clusters of an addressing function that intersects Q , then, is used to measure the efficiency of the corresponding retrieval algorithm for answering Q . For a set of clusters \mathcal{H} , and a query Q , this number will be denoted by $N_{\mathcal{H}}(Q)$.

What is a reasonable way of aggregating this measure over all region queries of interest? A simple average of $N_{\mathcal{H}}(Q)$ over all such queries seems inadequate, since the contribution of the larger regions to this average could swamp the contribution of the smaller ones. We therefore distinguish between region queries on the basis of the widths of the ranges specified in them. We say two region queries are *similar* if their corresponding regions are translations of each other. In other words, similar region queries define isomorphic boxes in $[0, 1]^n$ (Fig. 4). Similarity is an equivalence relation that partitions region queries into equivalence classes, referred to here as *similarity*

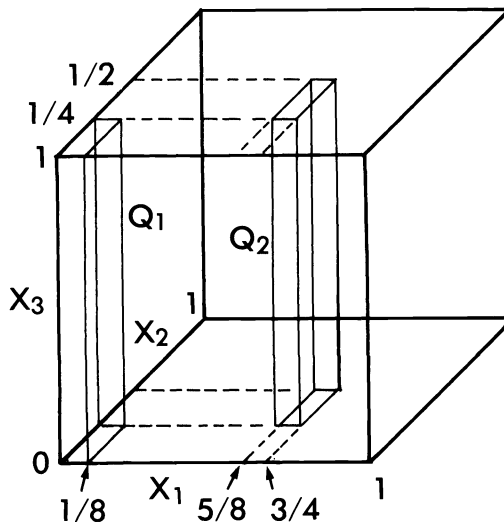


FIG. 4. Similarity of region queries. $Q_1: 0 \leq x_1 \leq \frac{1}{8}, 0 \leq x_2 \leq \frac{1}{4}$. $Q_2: \frac{5}{8} \leq x_1 \leq \frac{3}{4}, \frac{1}{4} \leq x_2 \leq \frac{1}{2}$.

classes. Within each similarity class S , $N_{\mathcal{H}}(Q)$ is aggregated by taking a simple, unweighted, average, denoted by $\bar{N}_{\mathcal{H}}[S]$. But across all similarity classes, we aggregate $\bar{N}_{\mathcal{H}}[S]$ by averaging with respect to an arbitrary probability measure, say $P_{\mathcal{S}}$, where \mathcal{S} denotes the set of similarity classes of queries. We denote this latter average by $\bar{N}_{\mathcal{H}}$. (This, then, is the average number of buckets that must be examined to answer a region query, given the probability measure $P_{\mathcal{S}}$, and assuming a uniform probability distribution over each similarity class.)

In order to find a b -way partition of $[0, 1)^n$ leading to the most efficient retrieval algorithm, $\bar{N}_{\mathcal{H}}$ should be minimized over all balanced b -way uniformly scattering partitions of $[0, 1)^n$.

4. The cost function.

4.1. Definitions and notation. The following notation and definitions are needed in the subsequent development. A generic subset (combination) $\{i_1, \dots, i_j\}$ of the attributes $\{1, \dots, n\}$ is denoted by c ; its complement $\{1, \dots, n\} - c$ is denoted by \bar{c} . To distinguish between the different attribute spaces, the n -dimensional Euclidean space R^n is thought of as $R_1 \times \dots \times R_n$, where R_1, \dots, R_n are distinct copies of R . Given a subset c of $\{1, \dots, n\}$, R_c will denote the subspace of R^n defined by R_{i_1}, \dots, R_{i_j} , $R_c = R_{i_1} \times \dots \times R_{i_j}$. The shorthand \mathbf{x}_c is used to designate a point $(x_{i_1}, \dots, x_{i_j})$ in R_c , and when a particular n -vector $\mathbf{x} = (x_1, \dots, x_n) \in R_1 \times \dots \times R_n$ is implied in the context, \mathbf{x}_c will designate the projection of \mathbf{x} on R_c . In particular, $\mathbf{1}$ and $\mathbf{0}$ will designate n -vectors of all ones and all zeros respectively, and $\mathbf{0}_c$ and $\mathbf{1}_c$ will designate the projections of these vectors on R_c , for each $c \subseteq \{1, \dots, n\}$. Given a set V_c in R_c , we let $|V_c|$ be the volume of V_c in R_c . If $\mathbf{b}_c = (b_{i_1}, \dots, b_{i_j})$ and $\mathbf{b}'_c = (b'_{i_1}, \dots, b'_{i_j})$ are vectors in R_c such that $\mathbf{b}_c \leq \mathbf{b}'_c$, then $[\mathbf{b}_c, \mathbf{b}'_c]$ will denote the box $[b_{i_1}, b'_{i_1}] \times \dots \times [b_{i_j}, b'_{i_j}]$ in R_c . The similarity class of queries specified by a condition $\mathbf{v}_c \leq \mathbf{x}_c \leq \mathbf{v}_c + \mathbf{a}_c$, $\mathbf{v}_c, \mathbf{v}_c + \mathbf{a}_c \in R_c$, \mathbf{a}_c fixed, is designated $S_c(\mathbf{a}_c)$.

DEFINITION 1 (Projection). The *projection* of a set $X \subseteq R^n$ on the subspace R_c will be denoted by $\pi_c(X)$, $\pi_c(X) = \{\mathbf{t}_c \mid \exists \mathbf{x} \in X, \mathbf{x}_c = \mathbf{t}_c\}$.

DEFINITION 2 (Sweep). The *sweep* of a set $X \subseteq R^n$ through a distance a_i parallel to the i th axis is the subset of R^n swept out by a translation of X through a distance a_i parallel to the i th axis in the negative direction. It will be denoted by $\leftarrow^{a_i} X$:

$$\leftarrow^{a_i} X = \{(x_1, \dots, x_{i-1}, x_i - b_i, x_{i+1}, \dots, x_n) \mid (x_1, \dots, x_n) \in X, 0 \leq b_i \leq a_i\}.$$

More generally, given a nonnegative vector $\mathbf{a}_c \in R_c$, the sweep of a set X through \mathbf{a}_c is the set

$$\leftarrow^{a_{i_j}} (\dots (\leftarrow^{a_{i_2}} (\leftarrow^{a_{i_1}} X)) \dots),$$

and will be represented by $\leftarrow^{\mathbf{a}_c} X$. (It is important to note here that for any attribute combination s containing c these operations can be carried out on subsets of R_s .)

The $M_1 M_2 \dots M_n$ boxes of the underlying partition can be indexed as if they were elements of an n -dimensional array by using vectors $\mathbf{k} = (k_1, \dots, k_n) \in \{0, \dots, M_1 - 1\} \times \dots \times \{0, \dots, M_n - 1\}$. We let $L_{\mathbf{M}} = \{0, \dots, M_1 - 1\} \times \dots \times \{0, \dots, M_n - 1\}$, and $L_{\mathbf{M}}^{-1} = \{0, 1/M_1, \dots, 1 - 1/M_1\} \times \dots \times \{0, 1/M_n, \dots, 1 - 1/M_n\}$. Given a cluster H , the part of H belonging to the k th box of the underlying partition will be known as the k th *subcluster* of H , and will be denoted by $H(\mathbf{k})$, $\mathbf{k} \in L_{\mathbf{M}}$ (Fig. 3).

For each $c = \{i_1, \dots, i_j\}$ the underlying partition of $[0, 1]^n$ induces a partition of $[0_c, \mathbf{1}_c]$ into $M_{i_1} \times \dots \times M_{i_j}$ boxes. A subset V_c of $[0_c, \mathbf{1}_c]$ is divided into $M_{i_1} \times \dots \times M_{i_j}$ parts by these boxes, and these parts can be indexed by vectors $\mathbf{k}_c = (k_{i_1}, \dots, k_{i_j}) \in \{0, \dots, M_{i_1-1}\} \times \dots \times \{0, \dots, M_{i_j-1}\}$. The set $\{0, \dots, M_{i_1-1}\} \times \dots \times \{0, \dots, M_{i_j-1}\}$ will be denoted by L_{M_c} .

Let X and Y be subsets of R^n . The notation $X + Y$ will be used to denote the set $\{\mathbf{x} + \mathbf{y} = (x_1 + y_1, \dots, x_n + y_n) \mid \mathbf{x} = (x_1, \dots, x_n) \in X, \mathbf{y} = (y_1, \dots, y_n) \in Y\}$. Similarly, for $\mathbf{x} \in R^n$, and $Y \subseteq R^n$, $\mathbf{x} + Y$ will denote the set $\{\mathbf{x} + \mathbf{y} \mid \mathbf{y} \in Y\}$. A uniformly scattered cluster H will be called a *box array* if there is a box $B = [\mathbf{b}, \mathbf{b}']$, and a rectangular array of points $X = \{x_{11}, \dots, x_{1M_1}\} \times \dots \times \{x_{n1}, \dots, x_{nM_n}\}$ such that $H = B + X$. A uniformly scattered cluster H is said to be *periodic* if $H = H(0, \dots, 0) + L_M^{-1}$. Note that the subclusters of a periodic cluster are translations of one another, and that any one of them determines the entire cluster. Given a cluster H that is not necessarily periodic, let $\text{copy}(H(\mathbf{k}))$ be the periodic cluster determined by $H(\mathbf{k})$, $\text{copy}(H(\mathbf{k})) = H(\mathbf{k}) - (k_1/M_1, \dots, k_n/M_n) + L_M^{-1}$ ($\mathbf{k} \in L_M$).

In § 3 the notations $N_{\mathcal{H}}(Q)$, $\bar{N}_{\mathcal{H}}[S]$, and $\bar{N}_{\mathcal{H}}$ were introduced to denote the number of clusters of \mathcal{H} intersecting a query Q , the average number of clusters of \mathcal{H} intersecting a query in a similarity class S , and the overall average number of clusters of \mathcal{H} intersecting a query. Extending this notation to each cluster H_i of \mathcal{H} , we define $N_{H_i}(Q)$ to be one if the query Q intersects H_i , and zero otherwise, $\bar{N}_{H_i}[S]$ to be the probability that a query in S intersects H_i , and \bar{N}_{H_i} to be the probability that a region query intersects H_i .

4.2 Expressing the cost. We are now in a position to express the cost function $\bar{N}_{\mathcal{H}}$ in terms of the clusters H_0, \dots, H_{b-1} and the given probability measure $P_{\mathcal{Q}}$ on classes of similar queries. Consider a similarity class S of region queries. The quantities $N_{H_0}(Q), \dots, N_{H_{b-1}}(Q)$ and $N_{\mathcal{H}}(Q)$ can be thought of as random variables over the probability space of all queries in S (with the uniform probability distribution). Then, $\bar{N}_{H_i}[S]$ is the expected value of $N_{H_i}(Q)$ in this space, $0 \leq i \leq b-1$, and similarly $\bar{N}_{\mathcal{H}}[S]$ is the expected value $N_{\mathcal{H}}(Q)$. But by the definitions of $N_{H_i}(Q)$, $0 \leq i \leq b-1$, and $N_{\mathcal{H}}(Q)$,

$$(2) \quad N_{\mathcal{H}}(Q) = \sum_{i=0}^{b-1} N_{H_i}(Q).$$

By taking expectations in (2), and by using the fact that expectation distributes over summation, we can then obtain

$$(3) \quad \bar{N}_{\mathcal{H}}[S] = \sum_{i=0}^{b-1} \bar{N}_{H_i}[S].$$

In a similar way we can equate the expectations (with respect to $P_{\mathcal{Q}}$) of the two sides of (3) to obtain

$$(4) \quad \bar{N}_{\mathcal{H}} = \sum_{i=0}^{b-1} \bar{N}_{H_i} = \sum_{i=0}^{b-1} \int_{\mathcal{Q}} \bar{N}_{H_i}[S] dP_{\mathcal{Q}}(S).$$

In order to express $\bar{N}_{H_i}[S]$ in terms of H and S , consider first a similarity class $S(\mathbf{a}) = S(a_1, \dots, a_n)$ whose queries specify ranges for every attribute. Each query in this class specifies a region $[\mathbf{v}, \mathbf{v} + \mathbf{a}] \subseteq [0, 1]^n$, and can be identified by a vector

$\mathbf{v} \in [\mathbf{0}, \mathbf{1} - \mathbf{a}]$. A subset of these queries then defines a subset of vectors in R^n , and the size of this query subset can be measured by the volume of the corresponding subset of vectors. To determine $\bar{N}_H[S]$, that is, the fraction of queries in S that intersect a cluster H , we need to compute the volume of the set of vectors $\mathbf{v} \in [\mathbf{0}, \mathbf{1} - \mathbf{a}]$ whose associated query regions $[\mathbf{v}, \mathbf{v} + \mathbf{a}]$ intersect the cluster H , as a fraction of the volume of all such vectors. But $[\mathbf{v}, \mathbf{v} + \mathbf{a}]$ intersects H , if and only if there exists a point $\mathbf{x} \in H$ such that $\mathbf{v} \leq \mathbf{x} \leq \mathbf{v} + \mathbf{a}$, or $\mathbf{x} - \mathbf{a} \leq \mathbf{v} \leq \mathbf{x}$, that is, if and only if $\exists \mathbf{x} \in H, \mathbf{v} \in \overset{\mathbf{a}}{\leftarrow} \{\mathbf{x}\}$. So the set of points $\mathbf{v} \in [\mathbf{0}, \mathbf{1} - \mathbf{a}]$ that represent queries in S intersecting H is given by

$$(5) \quad \bigcup_{\mathbf{x} \in H} (\overset{\mathbf{a}}{\leftarrow} \mathbf{x}) \cap [\mathbf{0}, \mathbf{1} - \mathbf{a}] = (\overset{\mathbf{a}}{\leftarrow} H) \cap [\mathbf{0}, \mathbf{1} - \mathbf{a}] = \bigcup_{\mathbf{k} \in L_M} ((\overset{\mathbf{a}}{\leftarrow} H(\mathbf{k})) \cap [\mathbf{0}, \mathbf{1} - \mathbf{a}]).$$

Now if $a_i < 1/M_i, 1 \leq i \leq n$, and if M_1, \dots, M_n are large, then for most subclusters $H(\mathbf{k}), \overset{\mathbf{a}}{\leftarrow} H(\mathbf{k}) \subseteq [\mathbf{0}, \mathbf{1} - \mathbf{a}]$. In that case (5) can be approximated by $\bigcup_{\mathbf{k} \in L_M} (\overset{\mathbf{a}}{\leftarrow} H(\mathbf{k}))$, and

$$(6) \quad \bar{N}_H[S(\mathbf{a})] \approx \left| \bigcup_{\mathbf{k} \in L_M} (\overset{\mathbf{a}}{\leftarrow} H(\mathbf{k})) \right| / |[\mathbf{0}, \mathbf{1} - \mathbf{a}]|.$$

For a general similarity class of queries, $S_c(\mathbf{a}_c)$, the above development can be retraced in R_c to yield

$$(7) \quad \bar{N}_{H_i}[S_c(\mathbf{a}_c)] \approx \left| \bigcup_{\mathbf{k}_c \in L_{M_c}} \overset{\mathbf{a}_c}{\leftarrow} \pi_c(H_i)(\mathbf{k}_c) \right| / |[\mathbf{0}, \mathbf{1}_c - \mathbf{a}_c]|.$$

By substituting (7) into (4) we obtain an expression for the objective function $\bar{N}_{\mathcal{H}}$ in terms of the partition $\mathcal{H} = (H_0, \dots, H_{b-1})$ and the probability measure $P_{\mathcal{H}}$. But because of the possible overlap between the sweeps $\overset{\mathbf{a}_c}{\leftarrow} \pi_c(H_i)(\mathbf{k}_c), \mathbf{k}_c \in L_{M_c}$, the resulting expression is difficult to work with analytically. In what follows we use an approximate cost function in which these overlaps are ignored. Then, the size of the union of these sweeps can be approximated by the sum of the sizes of the individual sweeps. Letting $\bar{N}'_{H_i}[S_c(\mathbf{a}_c)]$ be the resulting approximation to the average number of times a query in $S_c(\mathbf{a}_c)$ intersects H_i , we obtain

$$(8) \quad \bar{N}'_{H_i}[S_c(\mathbf{a}_c)] = \sum_{\mathbf{k}_c \in L_{M_c}} \left| \overset{\mathbf{a}_c}{\leftarrow} \pi_c(H_i)(\mathbf{k}_c) \right| / |[\mathbf{0}_c, \mathbf{1}_c - \mathbf{a}_c]|$$

as the right side of (7). Note that this approximation does not allow us to consider the subclusters of a cluster independently of each other. If $c \subset \{1, 2, \dots, n\}$, then $\pi_c(H_i)(\mathbf{k}_c)$ is the union of the projections of $\prod_{l \in c} M_l$ subclusters of H_i , and it is not treated as a disjoint union. Also, for this approximation to be valid the extra volumes produced by the sweeps $\overset{\mathbf{a}_c}{\leftarrow} \pi_c(H_i)(\mathbf{k}_c), \mathbf{k}_c \in L_{M_c}$, need not be small; only the overlap between these sweeps should be small. In fact, a sufficient condition for the approximation to be exact is that for each cluster H and each coordinate $i \in c$, the sets $\pi_{\{i\}}(H)(k_i), 0 \leq k_i \leq M_i - 1$ be separated by distances of at least a_i .

Of course, it cannot be claimed that any optimal partition with respect to this approximation will be near-optimal with respect to $\bar{N}_{\mathcal{H}}$ (except by considering the limiting case as the ranges go to zero width, which is already analyzed in [7]). But the approximation can be used in the following suboptimal strategy for designing addressing functions for small region queries.

Let the approximate cost function obtained by substituting from (8) into (4) be denoted by $\bar{N}'_{\mathcal{R}}$, and let the contribution of a cluster H to this cost be denoted by \bar{N}'_H :

$$(9) \quad \bar{N}'_{\mathcal{R}} = \sum_{i=0}^{b-1} \bar{N}'_{H_i},$$

$$\bar{N}'_{H_i} = \int_{\mathcal{G}} \sum_{\mathbf{k}_c \in L_{\mathbf{M}_c}} (|\leftarrow^{\mathbf{a}_c} \pi_c(H_i)(\mathbf{k}_c)| / |[\mathbf{0}_c, \mathbf{1}_c - \mathbf{a}_c]|) dP_{\mathcal{G}}[S_c(\mathbf{a}_c)].$$

We shall first derive a lower bound on this approximate cost function and see that this lower bound can be approximated by box-array addressing functions. We then choose, among the resulting box-array addressing functions that approximate this lower bound, one that minimizes the actual cost function $\bar{N}_{\mathcal{R}}$. We conjecture that this suboptimal process will usually yield addressing functions that are close to optimal.

5. Optimality result. Let $\bar{N}^* = \inf \{\bar{N}'_H \mid H \subset [0, 1]^n, |H| = 1/b, H \text{ uniformly scattered with respect to } \mathbf{M}\}$, and let us call clusters achieving this infimum *optimal clusters*. Then from (9) $\bar{N}'_{\mathcal{R}} \geq b\bar{N}^*$, and the lower bound is achievable if $[0, 1]^n$ can be partitioned into b optimal clusters. (This lower bounding strategy is due to Rivest [16].) In [7] we proved that for partial-match queries an optimal cluster of a balanced addressing function is a Cartesian product, which may be thought of as an n -dimensional rectangular array of *points*. Here we would like to show that for region queries, an optimal cluster of a balanced uniformly scattered addressing function is an n -dimensional rectangular array of *boxes*, a *box-array*:

THEOREM 1. *The infimum \bar{N}^* is achieved by a box-array.*

Theorem 1 is proved by using the following lemmas.

LEMMA 1. *Let H be a uniformly scattered cluster with respect to \mathbf{M} . There is a subcluster $H(\mathbf{k}^*)$ of H for which*

$$\bar{N}'_{\text{copy}(H(\mathbf{k}^*))} \leq \bar{N}'_H.$$

LEMMA 2. *Let $\mathbf{a}_c = (a_{i_1}, \dots, a_{i_n}) \geq \mathbf{0}_c$, and H be a bounded subset of \mathbb{R}^n . Then*

$$|\leftarrow^{\mathbf{a}_c} H| \geq \sum_{c' \subseteq c} \left[\prod_{i \in c'} a_i \right] |\pi_{c'}(H)|.$$

LEMMA 3 ([7, Thm. 1]). *Given a set $H \subseteq \mathbb{R}^n$, there exists a box $B(H) \subset \mathbb{R}^n$ which has the same volume as H , and whose projected volume on each R_c , $c \subseteq \{1, \dots, n\}$, is no larger than the corresponding projected volume of H . That is,*

$$|\pi_c(B(H))| \leq |\pi_c(H)| \quad \text{for all } c \subseteq \{1, \dots, n\},$$

and $|B(H)| = |H|$.

Lemmas 1 and 2 will be proved later in this section; Lemma 3 is proved in [7].

Proof of Theorem 1 (based on Lemmas 1, 2 and 3). The proof is a simple consequence of Lemma 1 and the following corollaries to Lemmas 2 and 3.

COROLLARY TO LEMMA 3. *Let H belong to a box $[\mathbf{b}, \mathbf{b}'] \subset \mathbb{R}^n$. Then there is a box $B(H) \subseteq [\mathbf{b}, \mathbf{b}']$ that satisfies the conditions of Lemma 3.*

Proof. Since the one-dimensional projections of the box $B(H)$ are no larger than the corresponding projections of H , the box can be translated into $[\mathbf{b}, \mathbf{b}']$ (projected volume is invariant under translation). \square

COROLLARY TO LEMMAS 2 AND 3. *For the box $B(H)$ in Lemma 3 and its corollary,*

$$|\leftarrow^{\mathbf{a}_c} \pi_c(B(H))| \leq |\leftarrow^{\mathbf{a}_c} \pi_c(H)|.$$

Proof. The proof follows by noting that the inequality of Lemma 2 is tight for a box. Thus, for $c = \{1, \dots, n\}$,

$$\begin{aligned} |\overleftarrow{B}(H)| &= \sum_{c' \subseteq \{1, \dots, n\}} \left[\prod_{i \in c'} a_i \right] |\pi_{c'}(B(H))| \\ &\leq \sum_{c' \subseteq \{1, \dots, n\}} \left[\prod_{i \in c'} a_i \right] |\pi_{c'}(H)| \quad (\text{by Lemma 3}) \\ &\leq |\overleftarrow{H}| \quad (\text{by Lemma 2}). \end{aligned}$$

For $c \subset \{1, \dots, n\}$, a similar proof can be used in R_c . \square

Now by Lemma 1 we can without loss of generality restrict attention to periodic clusters in trying to achieve the infimum \bar{N}^* . Let $H_0 + L_{\mathbf{M}}^{-1}$, $H_0 \in [0, 1/M_1) \times \dots \times [0, 1/M_n)$, be a periodic cluster. Then it follows from (8) that

$$(10) \quad \bar{N}'_{H_0 + L_{\mathbf{M}}^{-1}}[S_c(\mathbf{a}_c)] = \left(\prod_{i \in c} M_i \right) |\overleftarrow{a_c} \pi_c(H_0)| / \left(\prod_{i \in c} (1 - a_i) \right).$$

Let $B(H_0)$ be the box associated with H_0 in the corollary to Lemma 3 (where the enclosing box is $[0, 1/M_1) \times \dots \times [0, 1/M_n)$). By using the corollary to Lemmas 2 and 3 in (10) we obtain

$$(11) \quad \begin{aligned} \bar{N}'_{H_0 + L_{\mathbf{M}}^{-1}}[S_c(\mathbf{a}_c)] &\geq \left(\prod_{i \in c} M_i \right) |\overleftarrow{a_c} \pi_c(B(H_0))| / \left(\prod_{i \in c} (1 - a_i) \right) \\ &= \bar{N}'_{B(H_0) + L_{\mathbf{M}}^{-1}}[S_c(\mathbf{a}_c)]. \end{aligned}$$

Since (11) is true for every similarity class $S_c(\mathbf{a}_c)$, it follows by integrating the two sides of (11) that

$$\bar{N}'_{H_0 + L_{\mathbf{M}}^{-1}} \geq \bar{N}'_{B(H_0) + L_{\mathbf{M}}^{-1}}.$$

Hence in searching for the infimum \bar{N}^* it suffices to restrict attention to box-arrays. \square

Proof of Lemma 1. Let $H(\mathbf{k}^*)$ minimize $\bar{N}'_{\text{copy}(H(\mathbf{k}))}$ over all subclusters of H :

$$\bar{N}'_{\text{copy}(H(\mathbf{k}^*))} = \min_{\mathbf{k} \in L_{\mathbf{M}}} (\bar{N}'_{\text{copy}(H(\mathbf{k}))}).$$

We will prove that

$$(12) \quad \bar{N}'_H \geq \bar{N}'_{\text{copy}(H(\mathbf{k}^*))}.$$

Let a vector $\mathbf{k} \in L_{\mathbf{M}}$ be represented as $(\mathbf{k}_c; \mathbf{k}_{\bar{c}})$ where \mathbf{k}_c and $\mathbf{k}_{\bar{c}}$ are the projections of \mathbf{k} on $L_{\mathbf{M}_c}$ and $L_{\mathbf{M}_{\bar{c}}}$ respectively. Also, for notational brevity, let $E_{\mathcal{G}}$ denote integration with respect to the measure $P_{\mathcal{G}}(S_c[\mathbf{a}_c]) / \prod_{i \in c} (1 - a_i)$, let $m_c = \prod_{i \in c} M_i$, and $m = \prod_{i=1}^n M_i$. The proof relies on the fact that the average of a set of real numbers is bounded above and below by the maximum and the minimum values in the set:

$$\begin{aligned} \bar{N}'_H &= E_{\mathcal{G}} \left(\sum_{\mathbf{k}_c \in L_{\mathbf{M}_c}} |\overleftarrow{a_c} \pi_c(H)(\mathbf{k}_c)| \right) \quad (\text{by (9)}) \\ &= E_{\mathcal{G}} \left(\sum_{\mathbf{k}_c \in L_{\mathbf{M}_c}} \left| \bigcup_{\mathbf{k}_{\bar{c}} \in L_{\mathbf{M}_{\bar{c}}}} (\overleftarrow{a_c} \pi_c(H(\mathbf{k}_c; \mathbf{k}_{\bar{c}}))) \right| \right) \\ &\geq E_{\mathcal{G}} \left(\sum_{\mathbf{k}_c \in L_{\mathbf{M}_c}} \max_{\mathbf{k}_{\bar{c}} \in L_{\mathbf{M}_{\bar{c}}}} |\overleftarrow{a_c} \pi_c(H(\mathbf{k}_c; \mathbf{k}_{\bar{c}}))| \right) \end{aligned}$$

$$\begin{aligned} &\geq E_{\mathcal{G}}\left(\sum_{\mathbf{k}_c \in L_{M_c}} \frac{1}{m_{\bar{c}}} \sum_{\mathbf{k}_{\bar{c}} \in L_{M_{\bar{c}}}} |\leftarrow^{\mathbf{a}_c} \pi_c(H(\mathbf{k}_c; \mathbf{k}_{\bar{c}}))|\right) \\ &= E_{\mathcal{G}}\left(\frac{1}{m} \sum_{\mathbf{k} \in L_M} m_c |\leftarrow^{\mathbf{a}_c} \pi_c(H(\mathbf{k}))|\right) \quad (\text{since } m = m_c m_{\bar{c}}) \\ &= \frac{1}{m} \sum_{\mathbf{k} \in L_M} E_{\mathcal{G}}(m_c |\leftarrow^{\mathbf{a}_c} \pi_c(H(\mathbf{k}))|) \\ &\geq \min_{\mathbf{k} \in L_M} E_{\mathcal{G}}(m_c |\leftarrow^{\mathbf{a}_c} \pi_c(H(\mathbf{k}))|). \end{aligned}$$

But it is easy to see that

$$\tilde{N}'_{\text{copy}}(H(\mathbf{k})) = E_{\mathcal{G}}(m_c |\leftarrow^{\mathbf{a}_c} \pi_c(H(\mathbf{k}))|),$$

and (12) follows. \square

Proof of Lemma 2.

SUBLEMMA 2.1 (Lemma 2 for $n = 1$). *Let H be a bounded subset of R . Then $|\leftarrow^a H| \geq a + |H|$.*

Proof. Without loss of generality let $\inf(H) \in H$. Then $|\leftarrow^a H| \geq |\leftarrow^a \inf(H)| + |H|$ since H and $\leftarrow^a \inf(H)$ are disjoint except for $\inf(H)$ (a set of measure zero) and they both belong to $|\leftarrow^a H|$. But $|\leftarrow^a \inf(H)| = a$. \square

SUBLEMMA 2.2. *For a bounded subset H of R^n ,*

$$|\leftarrow^{a_1} H| \geq a_1 |\pi_{\{2, \dots, n\}}(H)| + |H|.$$

Proof. For $(t_2, \dots, t_n) \in R_2 \times \dots \times R_n$ let $H/(t_2, \dots, t_n)$ denote the set of points in H on the line of constant (t_2, \dots, t_n) , and treated as a subset of R_1 : $H/(t_2, \dots, t_n) = \{t_1 | (t_1, t_2, \dots, t_n) \in H\}$. By Sublemma 2.1 the inequality holds for each of these lines: $|\leftarrow^{a_1} (H/(t_2, \dots, t_n))| \geq a_1 + |H/(t_2, \dots, t_n)|$. Sublemma 2.2 follows from integrating the two sides of this inequality over all points $(t_2, \dots, t_n) \in \pi_{\{2, \dots, n\}}(H)$, since the sweep of H over a_1 is the disjoint union of the sweeps of its intersections with lines of constant (t_2, \dots, t_n) (lines parallel to the R_1 axis). \square

Lemma 2 can now be proved by induction on the dimension of the sweep. Sublemma 2.2 is both the basis and the principal relation used in the induction step. We also need the following relations in the induction step.

R1. For $c' \subseteq c$, $\leftarrow^{\mathbf{a}_{c'}} \pi_c(H) = \pi_c(\leftarrow^{\mathbf{a}_{c'}} H)$, $H \subseteq R^n$.

R2. For $c' \subseteq c$, $\pi_{c'}(\pi_c(H)) = \pi_{c'}(H)$.

Suppose the lemma is valid for all sweeps of dimension $j - 1$, $2 \leq j \leq n$. To prove the lemma for some \mathbf{a}_c , with $|c| = j$, assume without loss of generality that $c = \{1, 2, \dots, j\}$, and let $c^- = \{2, 3, \dots, j\}$. Then

$$\begin{aligned} (13) \quad |\leftarrow^{\mathbf{a}_c} H| &= |\leftarrow^{\mathbf{a}_1} (\leftarrow^{\mathbf{a}_{c^-}} H)| \geq a_1 |\pi_{\{2, \dots, n\}}(\leftarrow^{\mathbf{a}_{c^-}} H)| + |\leftarrow^{\mathbf{a}_{c^-}} H| \quad (\text{by Sublemma 2.2}) \\ &= a_1 |\leftarrow^{\mathbf{a}_{c^-}} \pi_{\{2, \dots, n\}}(H)| + |\leftarrow^{\mathbf{a}_{c^-}} H| \quad (\text{by R1}). \end{aligned}$$

The lemma for \mathbf{a}_c follows by using the induction hypothesis on the resulting $(j - 1)$ -dimensional sweeps through \mathbf{a}_{c^-} in (13). For $\leftarrow^{\mathbf{a}_{c^-}} H$, the induction hypothesis gives rise to the terms $[\prod_{i \in c'} a_i] \pi_{c'}(H)$, $c' \subseteq c^- = \{2, \dots, j\}$, or $c' \subseteq \{1, \dots, j\}$, $1 \notin c'$. For $\leftarrow^{\mathbf{a}_{c^-}} \pi_{\{2, \dots, n\}}(H)$, the underlying Euclidean space is $R_2 \times \dots \times R_n$, so application of the

induction hypothesis to $a_1 | \leftarrow^{a-c} \pi_{\{2, \dots, n\}}(H) |$ gives rise to the terms

$$a_1 \left[\prod_{i \in c'} a_i \right] | \pi_{\{2, \dots, n\}-c'}(\pi_{\{2, \dots, n\}}(H)) |, \quad c' \subseteq c^- = \{2, \dots, j\}.$$

And by using the relation R2 these terms can be simplified to $[\prod_{i \in c'} a_i] | \pi_{c'}(H) |$, $c' \subseteq \{1, \dots, j\}$, $1 \in c'$. \square

6. Optimizing the shape of a cluster. To find an optimal box-array cluster, $\bar{N}'_{B+L\bar{M}^{-1}}$ should be minimized over all boxes $B \subseteq [0, 1/M_1] \times \dots \times [0, 1/M_n]$ of a given volume $(1/b) \prod_{i=1}^n 1/M_i$. Let y_1, \dots, y_n represent the dimensions of such a box. Suppose that an estimate of the probability density $P_{\mathcal{F}}(S_c(\mathbf{a}_c))$ of the occurrence of different types of queries is available. This may be the case, for example, in a well-established application area in which there are few users, each of whom has a well-defined set of retrieval needs. In this case, finding the optimal values of y_1, \dots, y_n reduces to a well-known constrained optimization problem, which can be solved by standard iterative descent algorithms. In particular,

$$(14) \quad \bar{N}'_{B+L\bar{M}^{-1}}[S_c(\mathbf{a}_c)] = \prod_{i \in c} M_i \frac{\prod_{i \in c} (y_i + a_i)}{\prod_{i \in c} (1 - a_i)},$$

so that $\bar{N}'_{B+L\bar{M}^{-1}}$ becomes a positive polynomial in $\mathbf{y} = (y_1, \dots, y_n)$, say $p(\mathbf{y})$, and the problem is to minimize this polynomial subject to the constraints $\mathbf{0} \leq (y_1, \dots, y_n) \leq (1/M_1, \dots, 1/M_n)$, and $\prod_{i=1}^n y_i = (1/b) \prod_{i=1}^n 1/M_i$. More details on finding this minimum by iterative descent algorithms appear in [8].

In the event that estimating $P_{\mathcal{F}}(S_c(\mathbf{a}_c))$ is impractical, one may attempt to estimate $p(\mathbf{y})$ directly, by computing (14) for each incoming query and accumulating the results. If this too proves impractical, a very simple procedure known as stochastic approximation [18] provides a heuristic for estimating the optimal values of y_1, \dots, y_n directly. Very briefly, the procedure begins with an initial guess at the optimal values of y_1, \dots, y_n , and as each incoming query is observed, it adjusts this guess based on the observed query, and on the number of queries observed so far. More details on this will also be found in [8].

We conclude this section by deriving analytically the optimal shape of a box-array cluster under some independence assumptions on the probability of queries. The results show the combined effects of query probabilities and the widths of the specified ranges on the optimal shape of a box-array cluster.

Suppose that a range of the i th attribute appears in a query with probability p_i , independently of the other attributes, and further that ranges of widths between 0 and ϵ_i are equally likely to be specified for the i th attribute. In that case,

$$P_{\mathcal{F}}(S_c(\mathbf{a}_c)) = \prod_{i \in c} p_i \prod_{i \in \bar{c}} (1 - p_i) \prod_{i \in c} \frac{1}{\epsilon_i} = \prod_{i=1}^n (1 - p_i) \prod_{i \in c} \frac{p_i}{(1 - p_i)\epsilon_i},$$

and

$$(15) \quad \begin{aligned} \bar{N}'_{B+L\bar{M}^{-1}}[S_c(\mathbf{a}_c)] &= \prod_{i \in c} M_i \frac{\prod_{i \in c} (y_i + a_i)}{\prod_{i \in c} (1 - a_i)} \\ &\approx \prod_{i \in c} M_i (y_i + (1 + y_i)a_i) \quad (\text{when the widths } a_i, i \in c, \text{ are small}). \end{aligned}$$

In order to find $\bar{N}'_{B+L\bar{M}^{-1}}$, we can first integrate $\bar{N}'_{B+L\bar{M}^{-1}}[S_c(\mathbf{a}_c)]$ with respect to

$dP_{\mathcal{G}}[S_c(\mathbf{a}_c)]$ over all queries specifying ranges for attributes in c , obtaining

$$(16) \quad \int_{\mathbf{0}_c \leq \mathbf{a}_c \leq \boldsymbol{\varepsilon}_c} \prod_{i=1}^n (1-p_i) \prod_{i \in c} \left(\frac{p_i M_i}{(1-p_i) \varepsilon_i} \right) (y_i + (1+y_i) a_i) d\mathbf{a}_c \quad (\text{where } \boldsymbol{\varepsilon}_c = (\varepsilon_{i_1}, \dots, \varepsilon_{i_n}))$$

$$= \prod_{i=1}^n (1-p_i) \prod_{i \in c} \left(\frac{p_i M_i}{(1-p_i)} \right) \left(y_i + \frac{(1+y_i) \varepsilon_i}{2} \right).$$

$\tilde{N}'_{B+L\bar{M}^{-1}}$ is then the sum of (16) over all combinations of attributes

$$\tilde{N}_{B+L\bar{M}^{-1}} = \prod_{i=1}^n (1-p_i) \sum_{c \subseteq \{1, \dots, n\}} \prod_{i \in c} \left(\frac{p_i M_i}{(1-p_i)} \right) \left(y_i + \frac{(1+y_i) \varepsilon_i}{2} \right),$$

and we need to minimize this, subject to the constraints $y_i \leq 1/M_i$, $1 \leq i \leq n$, and $\prod_{i=1}^n y_i = 1/b \prod_{i=1}^n 1/M_i$. A derivation similar to that of Aho and Ullman [1] can then be used to show that if we ignore the inequality constraints the optimal dimensions of a representative box are

$$y_i^* = \left[\frac{1}{b} \prod_{k=1}^n \frac{p_k (1 + \varepsilon_k/2)}{(1-p_k) + p_k M_k \varepsilon_k/2} \right]^{1/n} \frac{(1-p_i) + p_i M_i \varepsilon_i/2}{p_i M_i (1 + \varepsilon_i/2)}, \quad 1 \leq i \leq n.$$

To see how these dimensions vary with the parameters in question, let $\bar{a}_i = \varepsilon_i/2$ be the average specified width of the i th attribute for $1 \leq i \leq n$ and let $q_i = (1-p_i)/p_i M_i$, $1 \leq i \leq n$. Then

$$(17) \quad \frac{y_i^*}{y_j^*} = \left(\frac{q_i + \bar{a}_i}{1 + \bar{a}_i} \right) / \left(\frac{q_j + \bar{a}_j}{1 + \bar{a}_j} \right).$$

Suppose first that the average range of values specified for each attribute is the same. From (17) it is then easy to see that y_i^* increases with increasing $q_i = (1-p_i)/p_i M_i$. Thus, the larger is the value of p_i , the probability of specifying the i th attribute, the smaller is the corresponding optimal dimension of a representative box.

The relationship between the optimal dimensions of a representative box and the average specified range of each attribute is somewhat more complicated. To illustrate, let us fix the values of p_i , $1 \leq i \leq n$, at some constant value p , and those of M_i , $1 \leq i \leq n$, at some constant value M , and let $q = (1-p)/pM$. Then

$$\frac{y_i^*}{y_j^*} = \left(\frac{q + \bar{a}_i}{1 + \bar{a}_i} \right) / \left(\frac{q + \bar{a}_j}{1 + \bar{a}_j} \right),$$

and this relative magnitude depends on the value of q . For small q ($q < 1$) the relationship between y_i and \bar{a}_i is positive (y_i increasing with increasing \bar{a}_i), whereas for large q ($q > 1$) this relationship is negative (y_i decreasing with increasing \bar{a}_i). To explain this phenomenon, notice that q is less than one if and only if p is greater than $1/(M+1)$. So for large probabilities of the *occurrence* of attributes in a query, the relationship between y_i and \bar{a}_i is positive, but for large probabilities of the *nonoccurrence* of attributes in queries this relationship is negative. This becomes clear if we consider the two extreme cases where one of these probabilities is very large.

If p is close to one, the only queries that have any significant probability are queries specifying ranges for all the attributes. Consider, for simplicity, one similarity class of such queries, $S_{(1, \dots, n)}(a_1, \dots, a_n)$. The fraction of queries here that intersect $B + L\bar{M}^{-1}$ is

$$(18) \quad \prod_{i=1}^n \frac{M(y_i + a_i)}{1 - a_i}.$$

It is then easy to justify that subject to the constraint $\prod_{i=1}^n y_i = 1/bM^n$ the minimizing values of y_1, \dots, y_n in (18) are proportional to the values of a_1, \dots, a_n respectively. In other words, to minimize the number of queries intersecting a box of a given volume, the shape of this box must “match” the shape of each query region.

If, on the other hand, $1 - p$ is close to one, then the only queries with significant probability are queries specifying a single range of values for an attribute. Considering for simplicity only queries specifying ranges of given widths, say a_1, \dots, a_n for the 1st, \dots , n th attributes respectively, $\bar{N}'_{B+L^{-1}}$ becomes

$$(1 - p)^{n-1} \sum_{i=1}^n \frac{pM(y_i + a_i)}{1 - a_i}.$$

It is again easy to verify that subject to the constraint $\prod_{i=1}^n y_i = 1/bM^n$, the minimizing values of y_1, \dots, y_n here are proportional to $(1 - a_1), \dots, (1 - a_n)$ respectively.

7. Conclusion. Let y_1^*, \dots, y_n^* be the optimal dimensions of a representative box in a box-array cluster. Then $[0, 1]^n$ can be partitioned into optimal box-array clusters, if and only if $[0, 1/M_1] \times \dots \times [0, 1/M_n]$ can be partitioned into boxes of dimensions y_1^*, \dots, y_n^* . If this is possible, then any partition of $[0, 1]^n$ into optimal box-arrays defines an optimal addressing function with respect to \bar{N}' . Of course, such an exact fit is quite rare, and in general the values y_1^*, \dots, y_n^* would have to be adjusted somewhat to achieve this (i.e., to exactly divide $1/M_1, \dots, 1/M_n$ respectively). Following is one heuristic procedure for this adjustment. Let y_1, \dots, y_n denote the approximating dimensions and let $b_i = (1/M_i)/y_i, 1 \leq i \leq n$ (these must be positive integers). Initially let $b_i = \lceil (1/M_i)/y_i^* \rceil$. Note that the total number of buckets $\prod_{i=1}^n b_i$ may now be considerably larger than the prescribed total b . So it may be possible to more closely approximate the prescribed number of buckets by decreasing one of the b_i 's. Decreasing the largest b_i leads to the least change in $\prod_{i=1}^n b_i$ and we may try that first. Thus, we can successively choose the largest b_i and decrease it by 1 as long as such a change keeps $\prod_{i=1}^n b_i$ above the prescribed number b . The approximating dimensions may then be found by letting $y_i = (1/M_i)/b_i, 1 \leq i \leq n$. (See also Rothnie and Lozano [19].) (The number of buckets is not allowed to go below the prescribed number in order to minimize bucket overflow.) While such an adjustment changes the shape and size of a cluster from its optimal shape, and prescribed size, the changes were found computationally to be quite tolerable. (A detailed examination of such roundoff errors appears in [7] which deals with partial-match queries, but which is also sufficient for the present problem since the optimization problem considered there is identical to the present one.)

Perhaps the simplest partition of $[0, 1]^n$ into box-array clusters is obtained by using periodic clusters. It yields the following addressing function which is a simple combination of order-preserving and randomizing functions:

$$h(\mathbf{x}) = \bar{h} \left(\left[\frac{x_1}{y_1^*} \right] \bmod 1/(M_1 y_1^*), \dots, \left[\frac{x_n}{y_n^*} \right] \bmod 1/(M_n y_n^*) \right) \quad (\bar{h} \text{ one-to-one}).$$

But the clusters need not be periodic for optimality with respect to \bar{N}'_H , and many other optimal designs with respect to the approximate cost, \bar{N}' , are possible as well. To complete the suboptimal design process described in § 4, it is now necessary to choose that partition of $[0, 1]^n$ into optimal box-array clusters that minimizes the actual cost function \bar{N} . The choice is an “alternating” box-array function defined as follows.

DEFINITION. An *alternating piecewise constant addressing function* of half-period $1/M$ is a function $h: [0, 1] \rightarrow \{0, 1, \dots, b - 1\}$, which is constant over intervals

$[k/Mb, (k + 1)/Mb)$, $0 \leq k < Mb$, and is defined by

$$h\left(\left[\frac{k}{Mb}, \frac{k+1}{Mb}\right)\right) = \begin{cases} k \bmod b, & \left\lfloor \frac{k}{b} \right\rfloor \text{ even,} \\ (b-1) - k \bmod b, & \left\lfloor \frac{k}{b} \right\rfloor \text{ odd.} \end{cases}$$

Fig. 5 illustrates this definition.

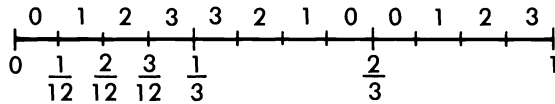


FIG. 5. An alternating piecewise constant addressing function. $M = 3, b = 4$.

An alternating box-array addressing function of half-period vector $(1/M_1, \dots, 1/M_n)$ is a function

$$h = \bar{h}(h_1, \dots, h_n): [0, 1]^n \rightarrow \{0, \dots, b_1 \cdots b_n - 1\}$$

(\bar{h} one-to-one) in which for each $i, 1 \leq i \leq n, h_i$ is an alternating piecewise constant function of half-period $1/M_i$, from $[0, 1)$ onto $\{0, \dots, b_i - 1\}$.

LEMMA 4. Consider the class of all uniformly scattering box-array addressing functions, h , with respect to (M_1, \dots, M_n) , such that $h = \bar{h}(h_1, \dots, h_n): [0, 1]^n \rightarrow \{0, \dots, b_1 \cdots b_n - 1\}$, with \bar{h} one-to-one, and $h_i: [0, 1) \rightarrow \{0, \dots, b_i - 1\}$, for fixed $b_i, 1 \leq i \leq n$. Among these, the alternating box-array function minimizes the number of clusters intersecting any region query.

Proof. Without loss of generality let $c = \{1, \dots, n\}$. Let $[v, v + a]$ be a given query region. Any box-array addressing function with the given specification divides each box of the underlying partition into $b_1 \cdots b_n$ identical boxes belonging to different clusters. And the query region $[v, v + a]$ intersects a fixed number of these subclusters in each box of the underlying partition, independently of the particular function used. Now let $m[v, v + a]$ be the maximum number of subclusters that intersect $[v, v + a]$ in any one of the boxes of the underlying partition, and consider a function h with the given specifications. The number of clusters of h that intersect $[v, v + a]$ is certainly no smaller than $m[v, v + a]$. But it is easy to see that the number of clusters of the corresponding alternating box-array function intersecting $[v, v + a]$ is exactly $m[v, v + a]$. \square

Let us now turn to the choice of the integers M_1, \dots, M_n . Because the proposed functions are balanced in every box of the underlying partition, if the density of records were uniform across each of these boxes, an approximately equal number of records would be mapped into each bucket for each box. Hence, irrespective of interbucket density variations, an approximately equal number of records would be mapped into each bucket. If possible, then, the partition should be fine enough that within its parts the records are approximately uniformly distributed. But while considerations of uniformity tend to favor finer underlying partitions, for efficiency with respect to region queries coarser partitions are more desirable. Thus, the more important an attribute is in forming queries, the coarser should be the initial partition on that attribute.

Since the underlying partition is the Cartesian product of partitions on each attribute space, the question is how fine a component partition to choose for each attribute. We can distinguish between several types of attributes in deciding this. There are "continuous" attributes such as age and weight for which range queries are clearly

meaningful and can be expected to be prevalent. Such attributes can often be partitioned into a number of equal intervals over which the records are approximately uniformly distributed. Of course, these intervals should be somewhat larger than the maximum width of a range query on the corresponding attribute for which the addressing function is to be useful. Discrete attributes, on the other hand, often have either very few possible values (for example, male and female for the attribute sex), or very many possible values (for example, the set of possible last names). Also, the ordering of the values of such attributes is quite often somewhat arbitrary. Because range queries are then quite unlikely for these attributes, there is somewhat more flexibility in choosing the corresponding initial attribute partition. It would make sense, for example, to use an initial partition of the attribute sex into male and female, so that the clusters become "unbiased" to sex. Since there would usually be widely different numbers of men and women represented in a database, such an initial partition would tend to equalize the number of people represented in each bucket. By contrast, if the attribute sex is not initially partitioned it is possible for its b_i value in the final multiple-key hashing procedure to be 2. In that case half of the buckets will contain records of males only, while the other half will contain records of females only, and this could lead to considerable nonuniformity in the number of records in each bucket. An attribute like last name, on the other hand, should not be partitioned at all initially, but the final component function on it should be completely randomizing, not order-preserving. If range queries are unlikely on last-name, there is little lost in retrieval efficiency by choosing a completely randomizing function for last name, but much is gained in uniformity with respect to addresses.

The design of a suitable addressing function in a given situation, then, begins by a subjective decision on the characteristic of the underlying partition, based on the foregoing guidelines. It then proceeds by an objective evaluation of an optimal box-array cluster, and ends with a heuristic algorithm for adjusting the shape of optimal clusters to achieve an exact fit of clusters in the record space.

Acknowledgments. This work stems from a doctoral dissertation completed at the Computer Science Division, University of California, Berkeley. It is a privilege to be able to thank numerous members of the Berkeley faculty and student body whose help and encouragement made this an enjoyable experience. In particular, I would like to thank Lotfi A. Zadeh and Richard M. Karp for their advice, support, criticism and encouragement.

REFERENCES

- [1] A. V. AHO AND J. D. ULLMAN, *Optimal partial-match retrieval when fields are independently specified*, ACM TODS, 4 (1979), pp. 168-179.
- [2] J. L. BENTLEY, *Multi-dimensional binary search trees used for associative searching*, Comm. ACM, 18 (1975), pp. 509-517.
- [3] ———, *Decomposable searching problems*, Inform. Process. Lett., 8 (1979), pp. 133-136.
- [4] J. L. BENTLEY AND J. H. FRIEDMAN, *Data structures for range searching*, Comput. Surveys, 11 (1979), pp. 397-409.
- [5] J. L. BENTLEY AND H. A. MAURER, *Efficient worst-case data structures for range searching*, Acta Inform., 13 (1980), pp. 155-168.
- [6] J. L. BENTLEY AND M. I. SHAMOS, *A problem in multivariate statistics: algorithm, data-structure, and applications*, Proceedings of 15th Allerton Conference on Communications, Controls and Computing, Sept., 1977, pp. 193-201.
- [7] A. BOLOUR, *Optimality properties of multiple key hashing functions*, J. Assoc. Comput. Mach., 26 (1979), pp. 196-210.

- [8] A. BOLOUR, *The Geometry of Efficient Retrieval Algorithms*, Ph.D. Dissertation, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, June, 1977.
- [9] R. A. FINKEL AND J. L. BENTLEY, *Quad trees, a data structure for retrieval on composite keys*, Acta Inform. 4 (1974), pp. 1-9.
- [10] G. D. KNOTT, *Hashing functions*, Comput. J., 18 (1975), pp. 265-278.
- [11] D. E. KNUTH, *The Art of Computer Programming Vol. 3, Sorting and Searching*, Addison-Wesley, Reading, MA., 1972.
- [12] D. T. LEE AND C. K. WONG, *Worst case analysis for region and partial-region searches in multi-dimensional binary search trees and quad trees*, Acta Inform., 9 (1978), pp. 23-29.
- [13] ———, *Quintary trees: a file structure for multidimensional database systems*, ACM TODS, 5 (1980), pp. 339-353.
- [14] J. H. LIOU, *Multi-dimensional directory for retrieval on secondary keys*, Technical Memorandum M-503, Electronics Research Laboratory, University of California, February 26, 1975.
- [15] G. LUEKER, *A data structure for orthogonal range queries*, in Proc. 19th IEEE Symposium on Foundations of Computer Science, October, 1978, pp. 28-34.
- [16] R. L. RIVEST, *Analysis of associative retrieval algorithms*, Rapport de Recherche No. 54, Laboratoire de Recherche en Informatique et Automatique, Rocquencourt, France, February, 1974.
- [17] ———, *Partial match retrieval algorithms*, this Journal, 5 (1976), pp. 19-50.
- [18] J. B. ROTHNIE, *The Design of Generalized Data Management Systems*, Ph.D. dissertation, Department of Civil Engineering, M.I.T., Cambridge, MA, 1972.
- [19] J. B. ROTHNIE AND T. LOZANO, *Attribute based file organization in a paged memory environment*, Comm. ACM, 17 (1974), pp. 63-69.
- [20] D. J. SAKRISON, *Stochastic approximation. A recursive method for solving regression problems*, in Advances in Communications Systems, vol. 2, Academic Press, New York, 1966.
- [21] J. B. SAXE AND J. L. BENTLEY, *Transforming static data structures to dynamic structures*, extended abstract, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.
- [22] D. E. WILLARD, *New data structures for orthogonal range queries*, Technical Report TR-22-78, Aiken Computational Lab., Harvard University, Cambridge, MA, 1978.
- [23] ———, *Balanced forests of k - d^* trees as dynamic data structures*, Technical Report TR-23-78, Aiken Computational Lab., Harvard University, Cambridge, MA, 1978.
- [24] S. B. YAO, *Approximating block accesses in database organization*, Comm. ACM, 20 (1977), pp. 260-261.

LIMITATIONS ON SEPARATING NONDETERMINISTIC COMPLEXITY CLASSES*

CHARLES W. RACKOFF† AND JOEL I. SEIFERAS‡

Abstract. If the time bounds defining two nondeterministic complexity classes are too close for separation by the two known techniques, then they are almost too close for separation by *any* relativizable technique. Proof of an analogous result for space would be a major breakthrough, implying $\text{NSPACE}(\log n) = \text{DSPACE}(\log n)$.

Key words. complexity hierarchies, computational complexity, time complexity, nondeterministic computation, relativized computation, query machines

The question of whether $\text{NTIME}(T_2) - \text{NTIME}(T_1)$ is nonempty is considered in [5]. If $\log T_2(n) \neq O(T_1(n))$,¹ then straightforward diagonalization is possible via deterministic simulation, assuming T_2 is sufficiently "honest". If $T_1(n+1) = o(T_2(n))$ ¹ (or, alternatively, if both $T_1(n+1) = O(T_2(n))$ and $T_1(n) = o(T_2(n))$), then "translational diagonalization" is possible, again assuming T_2 is sufficiently honest. If neither condition holds, then the question is open; we know of no applicable third technique. We show here that in some sense there *is* no significant third technique.

The sense in which there is no significant third technique is similar to the sense in which no "ordinary" diagonalization can prove $P \neq NP$ [1]. Both straightforward diagonalization and translational diagonalization work even for *relativized* computation. We show that no third technique yielding significantly new results can work for relativized computation.

A *nondeterministic query machine*, our model of relativized computation, is a nondeterministic multitape Turing machine [2] one of whose worktapes, the "query tape", is used to submit queries to an oracle for some language (set of finite strings of characters from some finite alphabet). When such a machine enters its distinguished "query" state, the next state is either the distinguished "yes" state or the distinguished "no" state, depending on whether the character string written on the nonblank portion of the query tape does or does not belong to the oracle's language.

An input string x belongs to *the language accepted by* a nondeterministic query machine \mathcal{M} if at least one of \mathcal{M} 's computations on input x halts in \mathcal{M} 's distinguished "accept" state. \mathcal{M} accepts its language *within time* T , where $T(n) \geq n$ is a nondecreasing function from and to the set \mathbf{N} of nonnegative integers, if each accepted string x is accepted within $T(|x|)$ steps in at least one computation, where $|x|$ denotes the length of x . $\text{NTIME}^A(T)$ is the class of all languages (over all finite alphabets) accepted within time T by nondeterministic query machines with oracle set A .

THEOREM. *If $\log T_2(n) = O(T_1(n))$ (barely ruling out straightforward diagonalization) and $T_2(n) = o(T_1(n+1))$ (ruling out translational diagonalization), then there is an oracle set A such that $\text{NTIME}^A(T_2) - \text{NTIME}^A(T_1)$ is empty.*

Remarks. (i) The second condition is only *close* to the negation of the condition for translational diagonalization.

* Received by the editors July 10, 1980, and in revised form December 15, 1980.

† Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A7. Supported in part by the National Research Council of Canada.

‡ Department of Computer Science, University of Rochester, Rochester, New York 14627. Supported in part by the National Science Foundation under grant MCS 79-05006.

¹ "O" means "at most some constant times", and "o" means "less than any fraction of (with finitely many exceptions)".

(ii) It does not immediately follow that $\text{NTIME}^A(cT_2) \subseteq \text{NTIME}^A(T_1)$ holds for every c . Whether the well known unrelativized result $\text{NTIME}(cT_2) \subseteq \text{NTIME}(T_2)$ (“linear speed-up”) carries over to relativized computation depends on the details of how inquiries can be submitted to the oracle. (To speed up submission of inquiries seems to require the ability to write several inquiry characters per step.) If T_2 satisfies the hypotheses, however, then so does some T'_2 with $T_2(n) = o(T'_2(n))$. This does give a fixed oracle set A such that for every c , $\text{NTIME}^A(cT_2) \subseteq \text{NTIME}^A(T'_2) \subseteq \text{NTIME}^A(T_1)$.

(iii) If we limit ourselves to oracle sets over just $\{0, 1\}$, then we get only the weaker conclusion that $\text{NTIME}^A(T_2) \subseteq \text{NTIME}^A(cT_1)$ for some c (possibly greater than 1). By slightly strengthening the first hypothesis, however, the following corollary does restore the stronger conclusion.

COROLLARY. *If $\log T_2(n) = o(T_1(n))$ and $T_2(n) = o(T_1(n+1))$, then there is an oracle set A over the alphabet $\{0, 1\}$ such that $\text{NTIME}^A(T_2) - \text{NTIME}^A(T_1)$ is empty.*

Example. For some oracle set A over $\{0, 1\}$, $\text{NTIME}^A(2^{2^n}) = \text{NTIME}^A(2^{2^{n+1}}/\log^* n)$. The unrelativized version of this question was raised explicitly in [5]. (The value of $\log^* n$ is the number of times we must iterate the base-2 logarithm, starting with n , to get down to 1.)

Proof of corollary. The hypotheses are also satisfied by T_2 and some $T'_1 = o(T_1)$. By the theorem, there is some oracle set A' for which $\text{NTIME}^{A'}(T_2) \subseteq \text{NTIME}^{A'}(T'_1)$. If we encode A' over $\{0, 1\}$ in a straightforward manner, then we get an oracle set A over $\{0, 1\}$ such that $\text{NTIME}^A(T_2) \subseteq \text{NTIME}^{A'}(T_2) \subseteq \text{NTIME}^{A'}(T'_1) \subseteq \text{NTIME}^A(O(T'_1)) \subseteq \text{NTIME}^A(T_1)$. \square

Proof of theorem. Since our conditions are not affected by multiplicative factors, the result is insensitive to the details of our particular notion of relativized Turing machine computation. For our particular notion, we can actually weaken the second condition to just $n = o(T_1(n+1) - T_2(n))$. Below, we use this hypothesis in the form $n = o(t(n))$ for $t(n) = \lfloor (T_1(n) - T_2(n-1))/2 \rfloor$.

So that we can consider alphabets of all finite cardinalities, we assume each alphabet is included in the set \mathbf{N} of nonnegative integers, and we encode \mathbf{N}^* (finite strings of nonnegative integers) over a single finite alphabet. A suitable encoding $h: \mathbf{N}^* \rightarrow \{0, 1\}^*$ is given by $h(n_1 \cdots n_l) = 0^{n_1+1} 1 \cdots 10^{n_l+1} 11$.

We select A over some large alphabet Σ (containing both 0 and 1) to “encode $\text{NTIME}^A(T_2)$ in an $\text{NTIME}^A(T_1)$ -decodable manner”. To do this, we simultaneously define an auxiliary “certificate function” $f: \mathbf{N} \times \mathbf{N}^* \rightarrow \Sigma^*$. Membership of $h(i)h(x)f(i, x)$ in A will certify that NQM_i^A (the i th nondeterministic query machine, with oracle set A) can accept $x \in \mathbf{N}^*$ within $T_2(|x|)$ steps. Our complete specifications for A and f are the following:

$$(1) \quad |f(i, x)| = T_1(|x|) - t(|x|),$$

$$(2) \quad A = \{h(i)h(x)f(i, x) \mid \text{NQM}_i^A \text{ accepts } x \in \mathbf{N}^*$$

$$\text{within } T_2(|x|) \text{ steps, and } |h(i)h(x)| \leq t(|x|)\}.$$

Assuming these specifications have been met, consider any language L , say over the finite alphabet $\Gamma \subseteq \mathbf{N}$, accepted within time $T_2(n)$ by NQM_i^A for some i . (If NQM_i^A does not accept its language L within time T_2 , then we do not have to show that $L \in \text{NTIME}^A(T_1)$. As a bonus, however, the argument will work on the subset $L' \subseteq L$ of strings x which NQM_i^A can accept within $T(|x|)$ steps.) Since $n = o(t(n))$, $|h(i)h(x)| \leq t(|x|)$ will hold for all but finitely many $x \in \Gamma^*$. Hence, a finite variant of L will be acceptable within time $T_1(n)$ by writing down $h(i)h(x)$ (at most $t(|x|)$ steps) followed by

a guess at $f(i, x)(T_1(|x|) - t(|x|))$ steps for a correct guess), and asking the oracle whether the result belongs to A . Thus, the proof will be complete if we can find A and f satisfying (1) and (2).

The value of $f(i, x)$ matters only for $(i, x) \in \cup_n D_n$, where $D_n = \{(i, x) \mid |x| = n, \text{ and } |h(i)h(x)| \leq t(n)\}$. In stage n , we extend f to D_n and put into A an appropriate selection of strings $h(i)h(x)f(i, x)$ with $(i, x) \in D_n$. Here is how we do it:

While $f(i, x)$ is not yet defined for some $(i, x) \in D_n$ for which $\text{NQM}_i^{A\text{-so-far}}$ accepts x within $T_2(n)$ steps in some computation, “protect” every string queried in a chosen one of these computations, choose $f(i, x)$ from $\Sigma^{T_1(n)-t(n)} - \{y \mid |h(i)h(x)y| \text{ is protected}\}$, and put $h(i)h(x)f(i, x)$ into A . On completion of this while-loop, the value of $f(i, x)$ will not matter for remaining $(i, x) \in D_n$.

No string protected in an earlier stage is longer than $T_2(n - 1) < T_1(n) - t(n)$, so no earlier stage disqualifies any string $h(i)h(x)y$ above. From the hypothesis $\log T_2(n) = O(T_1(n))$ and the fact $t(n) \leq T_1(n)/2$, it follows that $|\Sigma|^{T_1(n)-t(n)} > 2^{t(n)} T_2(n)$ if Σ is large enough. (This does *not* require that n be large.) Thus, all of stage n protects fewer than $|\Sigma|^{T_1(n)-t(n)}$ strings, and it is always possible to choose $f(i, x)$ as stipulated. This guarantees that (1) is satisfied, and the persistence of the while-loop and the protection it provides insure that (2) is also satisfied. \square

Finally, let us note that to prove an analogous theorem for space, if it holds, would require a major breakthrough. To obtain the analogous assertion, one replaces “TIME” by “SPACE”, and the condition $\log T_2(n) = O(T_1(n))$ by $T_2(n)^{1/2} = O(T_1(n))$ to rule out straightforward diagonalization via deterministic simulation [3]. We show below, however, that the result would imply $\text{NSPACE}(\log n) \neq \text{DSPACE}(\log n)$; although the latter is likely, it is one of the most notoriously elusive conjectures in theoretical computer science [4]. (Our space bounds do include the space used on the query tape, and our proof of the lemma below does depend on that convention.)

Consider space bounds S_1 and S_2 which satisfy the hypotheses $S_2(n)^{1/2} = O(S_1(n))$ and $S_2(n) = o(S_1(n + 1))$ and which also satisfy $\log n \leq S_1(n) = o(S_2(n))$ and are “space constructible” [2] (an appropriate notion of “sufficiently honest”). For example, take $S_1(n) = 2^{2^n}$ and $S_2(n) = S_1(n)^{1.5}$; then even $S_2(n)^{1/2} = o(S_1(n))$ holds. If the analogous NSPACE version of our theorem held, then we could find an oracle set A such that $\text{NSPACE}^A(S_1) = \text{NSPACE}^A(S_2)$. If S'_1 and S'_2 are intermediate space constructible bounds, in the sense $S_1 = o(S'_1) = o(S'_2) = o(S_2)$, then $\text{NSPACE}^A(O(S'_1)) = \text{NSPACE}^A(O(S'_2))$. On the other hand, $\text{DSPACE}^A(O(S'_1)) \neq \text{DSPACE}^A(O(S'_2))$, by straightforward diagonalization. Therefore, $\text{NSPACE}^A(O(S)) \neq \text{DSPACE}^A(O(S))$ for some space constructible bound S (either S'_1 or S'_2). Hence, $\text{NSPACE}^A(O(\log n)) \neq \text{DSPACE}^A(O(\log n))$, by a padding argument [3]. The lemma below shows that this finally implies the very strong *unrelativized* conclusion $\text{NSPACE}(\log n) \neq \text{DSPACE}(\log n)$.

LEMMA. *If $\text{NSPACE}(\log n) = \text{DSPACE}(\log n)$, then $\text{NSPACE}^A(O(\log n)) = \text{DSPACE}^A(O(\log n))$ for every oracle set A .*

Proof. Assume $\text{NSPACE}(\log n) = \text{DSPACE}(\log n)$, and consider any language L belonging to $\text{NSPACE}^A(c \log n)$, say via nondeterministic query machine \mathcal{M} . Design \mathcal{M}' to behave on input $x \# y$, where $\#$ is a new delimiter symbol and y is a string of 0's and 1's, like \mathcal{M} on input x . Instead of querying the oracle, however, \mathcal{M}' should treat y as the characteristic sequence of those members of the oracle set up to length $c \log |x|$. If y is not the right length or if \mathcal{M} would be led to use more than space $c \log |x|$, then \mathcal{M}' should not accept. Because our space bounds do include the space used on the query tape, \mathcal{M}' will not run into a query too long to look up in y (i.e., longer than $c \log |x|$). Since \mathcal{M}' requires only space $O(\log |x \# y|)$, the language L' it accepts belongs to

$\text{NSPACE}(\log n)$, and hence to $\text{DSPACE}(\log n)$; let \mathcal{M}'' deterministically accept L' within space $O(\log n)$, without queries. Finally, design deterministic query machine \mathcal{M}''' to behave on input x like \mathcal{M}'' on input $x \# y$, where y really is the characteristic sequence of those members of the oracle set up to length $c \log |x|$. The space required by \mathcal{M}''' will be $O(\log |x \# y|) = O(\log |x|)$, so L belongs to $\text{DSPACE}^A(O(\log n))$. \square

REFERENCES

- [1] T. BAKER, J. GILL AND R. SOLOVAY, *Relativizations of the $P = ?NP$ question*, this Journal, 4 (1975), pp. 431–442.
- [2] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
- [3] W. J. SAVITCH, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.
- [4] ———, *Nondeterministic $\log n$ space*, Proc. 8th Annual Princeton Conference on Information Sciences and Systems, Dept. of Electrical Eng., Princeton Univ., Princeton, NJ, 1974, pp. 21–23.
- [5] J. I. SEIFERAS, M. J. FISCHER AND A. R. MEYER, *Separating nondeterministic time complexity classes*, J. Assoc. Comput. Mach., 25 (1978), pp. 146–167.

COVERING GRAPHS BY SIMPLE CIRCUITS*

ALON ITAI,[†] RICHARD J. LIPTON,[†] CHRISTOS H. PAPADIMITRIOU[‡] AND M. RODEH[§]

Abstract. We show that any biconnected graph with n nodes and m edges can be covered by simple circuits whose total length is at most $\min(3m, m + 6n)$. Our proof suggests an efficient algorithm for finding such a cover.

Key words. graph algorithms, Eulerian subgraphs, edge connectivity

1. Introduction. A family C_1, \dots, C_m of simple circuits of an undirected multigraph $G = (V, E)$ covers G , provided each edge of G is in one of the circuits. The size of such a family is then the sum of the lengths of the circuits C_1, \dots, C_m . We are interested here in the question of finding covers of minimum size. Clearly, we can restrict our attention to 2-edge-connected multigraphs; that is, if a graph has a bridge, then it has no cover at all.

This problem bears a superficial similarity to the Chinese postman problem, in which one seeks to find the minimum number of edges that have to be added to G so as to result in an Eulerian multigraph. The difference is best exhibited by the famous Petersen graph (Fig. 1.a). There is an Eulerian supergraph of this graph with 20 edges, and this is best possible (Fig. 1.b). However, its minimum size cover is 21 (Fig. 1.c).

This problem of minimum cover size was first considered in [IR] where its application to irrigation systems was described. It was shown in [IR] that there is always a cover of size $|E| + 2|V| \log |V|$ and that this cover can be found in average time $O(|V|^2)$. Here we will show that every 2-connected multigraph has a cover of size

$$\min \{3|E| - 6, |E| + 6 \cdot |V| - 7\},$$

thus improving the previous results for sparse graphs. This cover can be found in $O(|V|^2)$ time.

Our construction relies heavily on that used by Jaeger [Ja] for showing that every 2-edge connected graph has a nowhere-zero flow modulo 8. Jaeger's paper does not contain full proofs and algorithms, and hence many of our results are only motivated by his. Matthews [M] also deals with a related problem; however, he misquotes Jaeger and is also subsumed by Jaeger.

In § 2, we show that in order to find a small size cover for a dense multigraph it suffices to find one for an efficiently extracted sparse one. We also give a general technique whereby, given a spanning tree T , one can find a cover of all the edges except perhaps for certain edges of T . In the next section, we show that, if the multigraph is 3-edge connected, then its edges can be covered by three Eulerian subgraphs. Then, in the next-to-last section, we extend this result to 2-edge connected multigraphs, which yields the bound sought. Finally we show that our cover can be found in $O(|V|^2)$ time.

* Received by the editors August 21, 1979, and in final revised form January 7, 1981. This research was supported in part by the National Science Foundation under grants MCS77-12517, MCS77-01192, and a Miller Fellowship.

[†] Department of Computer Science, University of California, Berkeley, California 94720.

[‡] Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, and Department of Computer Science, National Technical University of Athens, Greece.

[§] IBM, Israel Scientific Center, Technion City, Haifa 32000, Israel.

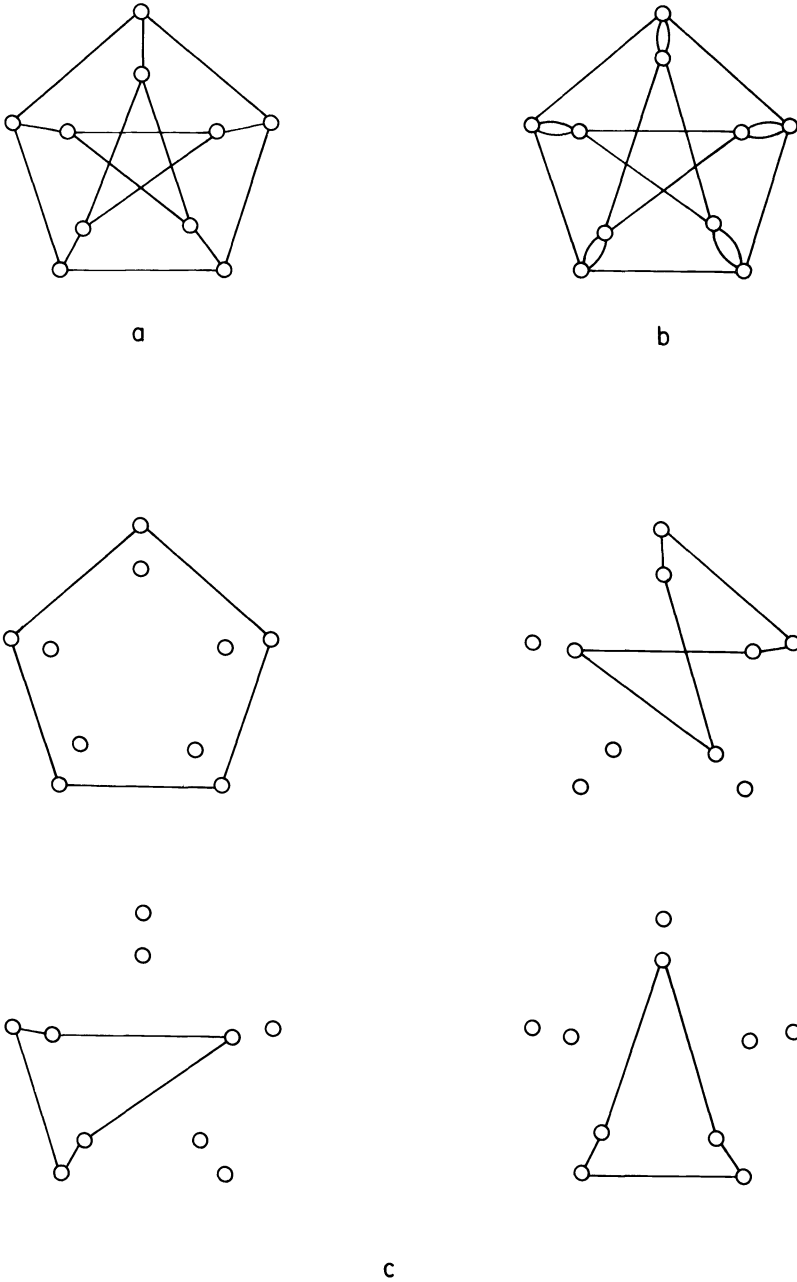


FIG. 1

2. Reduction to sparse graphs. If a graph is sparse (i.e., $|E| = O(|V|)$), then it seems reasonable to expect that there exists a cover of size $O(|E|)$. Therefore, following [IR], we will reduce the general problem to that of sparse multigraphs. As usual, an *Eulerian subgraph* of G is a subgraph of G consisting of edge-disjoint circuits (notice it need not be connected).

LEMMA 1. Let $T = (V, E_T)$ be a spanning tree of $G = (V, E)$. Then there exists an Eulerian subgraph $H_0 = (V, E_{H_0})$ of G with $E_{H_0} \supseteq E - E_T$.

Proof. E_{H_0} is constructed by successively deleting edges. Initially $E_{H_0} = E$. We then perform a depth-first search (DFS) on T . Each tree edge is traversed twice, once in the forward direction and once backwards. If when a tree edge (u, v) is traversed backwards from v to u the degree of v is odd in the current E_{H_0} , then delete the edge (u, v) from E_{H_0} . On termination, all vertices are of even degree in E_{H_0} , and hence $H_0 = (V, E_{H_0})$ is an Eulerian subgraph of G . \square

Note that the construction of this lemma can be done in $O(|E|)$ time whether T is a DFS tree or not.

In order to find a cover of G , one has to cover the edges of $E - E_{H_0} \subseteq E_T$. However, T is not a 2-edge connected multigraph and our method is not immediately applicable to it. We therefore augment T into such a graph.

Suppose that T is a DFS tree. We call an edge (u, v) of $E - E_T$ a *lowest frond* if u is the ancestor of all vertices w for which $(v, w) \in E - E_T$. Let us define the graph $H = (V, E_H)$ where E_H is T and all the lowest fronds. H is then 2-connected, has at most $2|V| - 2$ edges, and contains all the uncovered edges of H_0 as required. We summarize this as follows:

COROLLARY 1. *Suppose that one can find in time $t(|V|, |E|)$ a cover of size $s(|V|, |E|) + |E|$ for any 2-connected multigraph $G = (V, E)$. Then we can find a cover of size $|E| + s(|V|, 2|V| - 2)$ in time $t(|V|, 2|V| - 2) + O(|E|)$.*

3. Covering a 3-edge connected graph. The following lemma could have followed directly by applying Edmonds' matroid partitioning theorem [Ed 1] to the co-tree matroid of G . Our proof, however, suggests a more efficient algorithm.

LEMMA 2. *Let $G = (V, E)$ be a 3-edge connected multigraph. Then G contains three spanning trees $T_i = (V, E_{T_i})$ ($i = 1, 2, 3$) such that $E_{T_1} \cap E_{T_2} \cap E_{T_3} = \emptyset$.*

Proof. Let $D = (V, A)$ be the directed multigraph derived from G by replacing each edge by two arcs, one for each direction:

$$A = \{(u, v) \mid [u, v] \text{ is an edge of } G\}.$$

Let v be a vertex of V . By Menger's theorem on G , there exist three edge-disjoint paths from v to any triple of nodes of V . Hence, by the theorem of Edmonds [Ed 2], there exists three arc-disjoint directed trees B_1, B_2, B_3 rooted at v . Let T_1, T_2, T_3 be the underlying undirected trees. Each edge of G corresponds to two arcs of D , and hence it can appear in at most two of the undirected trees. Thus, $T_1 \cap T_2 \cap T_3 = \emptyset$. \square

From Lemma 1, each of these trees $T_i = (V, E_{T_i})$ of Lemma 2 induces a cover C_i of $(V, E - E_{T_i})$. Therefore, $C_1 \cup C_2 \cup C_3$ is a cover of G .

THEOREM 1. *Let G be a 3-edge connected multigraph. Then G can be covered by three Eulerian subgraphs.*

Using Tarjan's algorithm [Ta], the tree can be found in $O(|V| \cdot |E|)$ time. For dense graphs, Shiloach's algorithm [Sh] finds the trees faster (in time $O(|V|^2)$), but this is immaterial here since by Corollary 1 we need consider only sparse graphs (i.e., $|E| = O(|V|)$).

4. Covering a 2-edge connected graph. Theorem 1 can be strengthened to yield the next theorem.

THEOREM 2. *Let $G = (V, E)$ be a 2-edge connected multigraph. Then, G can be covered by three Eulerian subgraphs.*

Proof. We proceed by induction on the number of nodes of G . If $|V| = 1$, then the result is obvious since G consists of loops only. Now consider a multigraph $G = (V, E)$ with $|V| > 1$. By Theorem 1, we can assume that G is not 3-edge connected. Thus there exists a pair (x_1, x_2) and (y_1, y_2) of edges that disconnect G into two

components G_1 and G_2 (Fig. 2). Now create two new multigraphs G'_1 and G'_2 by deleting these edges and replacing them by two new edges (x_1, y_1) and (x_2, y_2) . Since

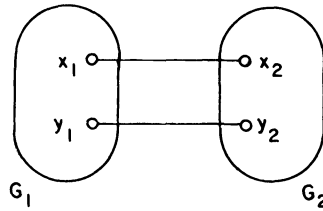


FIG. 2

G'_1 and G'_2 both have fewer than $|V|$ vertices, by our induction hypothesis they can be covered by three Eulerian subgraphs each. Let these Eulerian subgraphs have edge sets E_1, E_2, E_3 (for G'_1) and F_1, F_2, F_3 (for G'_2). By renaming, we may assume that

$$\begin{aligned} (x_1, y_1) &\in E_1, \dots, E_i, \\ (x_1, y_1) &\notin E_j, \quad j > i, \\ (x_2, y_2) &\in F_1, \dots, F_{i+k}, \\ (x_2, y_2) &\notin F_j, \quad j > i+k. \end{aligned}$$

Without loss of generality, assume $k \geq 0$. There are two cases following from this.

Case 1. ($k = 0$). The following S_1, S_2, S_3 is a cover for G :

$$S_j = \begin{cases} E_j \cup F_j - \{(x_1, y_1), (x_2, y_2)\} \cup \{(x_1, x_2), (y_1, y_2)\} & \text{if } (x_1, y_1) \in E_j, \\ E_j \cup F_j & \text{otherwise.} \end{cases}$$

Case 2. ($k > 0$). We find a new cover F'_1, F'_2, F'_3 of G'_2 by replacing F_{i+1}, \dots, F_{i+k} by

$$F_i \oplus F_{i+1}, \dots, F_i \oplus F_{i+k}.$$

Then the new covers E_1, E_2, E_3 and F'_1, F'_2, F'_3 satisfy the conditions of Case 1. \square

5. Time bounds. The cover of Theorem 2 can be found in $O(|E|^2)$ time by first finding separating pairs of bridges repeatedly until the graph is decomposed into 3-edge connected graphs (time $O(|V| \cdot |E|)$), then finding three spanning trees for each component (time $O(|V|^2)$ by Shiloach's algorithm [Sh]), then get the cover by Lemma 2 and finally combine the partial solutions together. This last may possibly require rearranging the Eulerian subgraphs as in case 2 of Theorem 2 (time $O(|E|)$). If we first use the reduction to sparse graphs by Corollary 1, then the entire algorithm runs in time $O(|V|^2)$.

6. Conclusions. We show that any 2-edge connected multigraph can be covered by three Eulerian subgraphs. If three Eulerian circuits are required, then each may contain at most $|E| - 2$ edges; therefore, any graph with $|E|$ edges can be covered with a set of circuits of total length $3|E| - 6$.

If we apply the reduction of § 2, then the graph $H_0 = (V, E_{H_0})$ has at most $|E| - 1$ edges (otherwise we are done); whereas $H = (V, E_H)$ has at most $2|V| - 2$. Therefore the total number of edges in the three Eulerian subgraphs of H and H_0 is $|E| + 6|V| - 7$.

Several problems remain open. There is no known graph which requires covers of size significantly larger than E . Thus, one may expect that the multiplicative constants

in our bound

$$\min \{|E| + 6 \cdot |V| - 7, 3|E| - 6\}$$

can be improved. It seems that the additive constants can be improved quite easily.

Finding the three spanning trees requires time $O(|V|^2)$, and this dominates the time bound. The reduction of § 4 also requires $O(|V|^2)$ time. However, an alternative based on Jaeger's original construction and the partition of the graph into tri-connected components [HT] would require $O(|E|)$ time.

Finally, nothing is known about the complexity of minimizing the size of the cover of G . We conjecture that it is NP-complete.

REFERENCES

- [Ed 1] J. EDMONDS, *Minimum partition of a matroid into independent subsets*, J. Res. NBS, 69b (1965), pp. 67–72.
- [Ed 2] ———, *Edge disjoint branchings*, in *Combinatorial Algorithms*, R. Rustin, ed., Algorithmics Press, New York, 1972, pp. 91–96.
- [HT] J. E. HOPCROFT AND R. E. TARJAN, *Dividing a graph into triconnected components*, this Journal, 2 (1973), pp. 135–158.
- [IR] A. ITAI AND M. RODEH, *Covering a graph by circuits*, Proc. ICALP Conf., Udine, 1978.
- [Ja] F. JAEGER, *On nowhere-zero flow in multigraphs*, Proc. Fifth Brit. Combinatorial Conference, 1975, pp. 373–378.
- [Sh] Y. SHILOACH, *Edge-disjoint branching in directed multigraphs*, Inform. Proc. Letters, 8 (1979), pp. 24–27.
- [Ta] R. E. TARJAN, *A good algorithm for edge-disjoint branchings*, Inform. Proc. Letters, 3 (1975), pp. 51–53.
- [M] K. R. MATTHEWS, *On the Eulericity of a graph*, J. Graph Theory, 2 (1978), pp. 143–148.

POWER OF NATURAL SEMIJOINS*

PHILIP A. BERNSTEIN† AND NATHAN GOODMAN‡

Abstract. A semijoin is a relational operator that is used to reduce the cost of processing queries in the SDD-1 distributed database system, the RAP database machine, and similar systems. Semijoin is used in these systems as part of a query pre-processing phase; its function is to “reduce” the database by delimiting those portions of the database that contain data relevant to the query. For some queries, there exist sequences of semijoins that “fully reduce” the database; those sequences delimit the exact portions of the database needed to answer the query in the sense that if any less data were delimited then the query would produce a different answer. Such sequences are called *full reducers*.

This paper characterizes the queries for which full reducers exist and presents an efficient algorithm for constructing full reducers where they do exist.

This paper extends the results of Bernstein and Chiu [J. Assoc. Comput. Mach., 28 (1981), pp. 25–40] by considering a more powerful semijoin operator. We consider “natural” semijoins instead of the “single attribute” semijoins of Bernstein and Chiu. A novel feature of our treatment is an extensive use of the “tableau methodology” of Aho, Sagiv and Ullman [SIAM J. Comput., 8 (1979), pp. 218–246] to prove the nonexistence of full reducers for a broad class of queries.

Key words. semijoin, database theory, query processing, tree database schema, acyclic database schema

1. Introduction. A major feature of relational database systems is their support of high-level query languages based on the relational calculus or algebra of [Codd]. These languages permit users to express queries in a concise and logical manner, without regard for execution efficiency. Translating these queries into efficient programs is the responsibility of the *query processing algorithm* of the database system. Many query processing algorithms have been described for relational systems implemented on “conventional” computing hardware (e.g., [ASU], [CM], [Gottlieb], [Pecherer], [Rothnie], [SACLP], [SC], [WY], [Yao]). In this paper we are concerned with aspects of query processing that arise in certain nonconventional database architectures, namely distributed database systems such as SDD-1 [BGWRR], [RBFQ] and “associative memory” database systems such as RAP [OSS].

The query processing algorithms of SDD-1 and RAP make extensive use of a relational operator called a *semijoin*.¹ A semijoin is “half of a join”²: the semijoin of relation R_1 by relation R_2 is defined to be the join of R_1 and R_2 projected back onto the attributes of R_1 . In other words, the semijoin retrieves all tuples of R_1 that join with any tuple of R_2 . While semijoins are not powerful enough to solve arbitrary relational queries, they can often reduce the cost of such solutions in SDD-1, RAP and similar systems.

Example 1.1. Suppose relations EMP (employee, specialty, department) and DPT (department, location, manager) are stored at different sites of a distributed database (see Fig. 1.1). Consider the query: List the employee, department and manager of all employees whose specialty is “sneakers” and whose department is located on the “3rd floor”. To answer the query, EMP and DPT must be joined.

* Received by the editors December 11, 1979, and in revised form October 29, 1980. This work was supported in part by the National Science Foundation under grants MCS-77-05314 and MCS-79-07762, by the IBM Corporation under an IBM Graduate Fellowship, and by the Advanced Research Projects Agency of the Department of Defense under contract N00039-77-C-0074, ARPA order 3175-6. The views and conclusions contained in this document should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

† Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138.

¹ Semijoin is called “implicit join” in [OSS].

² Relational terminology is defined in §§ 2 and 3.

a) Initial state

EMP (employee, specialty, department)	DPT (department, location, manager)
smith , sneakers , shoes	shoes , 3 , scholl
jones , sneakers , sporting gds	sporting gds, 2 , wilson
brown , pants , mens	mens , 3 , cardin
black , skirts , womens	womens , 2 , dior

b) Apply local operations: Restrict EMP to 'sneakers' experts; restrict DPT to '3rd floor' departments'

EMP (employee, specialty, department)	DPT (department, location, manager)
smith , sneakers , shoes	shoes , 3 , scholl
jones , sneakers , sporting gds	mens , 3 , cardin

c) Use semijoin to further reduce EMP: Eliminate tuples s.t. department \notin {shoes, mens}

EMP (employee, specialty, department)
smith , sneakers , shoes

d) Use semijoins to further reduce DPT: Eliminate tuples s.t. department \notin {shoes}

DPT (department, location, manager)
shoes , 3 , scholl

FIG. 1.1. Reducing a database state.

Intersite communication is a major cost of query processing in SDD-1 [BGWRR]; to reduce this cost, the size of each relation must be reduced before the join is attempted.

Two tactics are available for reducing the size of relations. One is *local processing*; e.g., EMP should be restricted to “sneakers” experts and DPT restricted to “3rd floor” departments before the join is attempted. See Fig. 1.1b. To reduce the relations further, semijoins may be used. By transmitting the set of “3rd floor” departments to EMP it is possible to compute “EMP semijoin DPT” which retrieves all “sneakers” experts who work for “3rd floor” departments; see Fig. 1.1c. If we subsequently transmit the set of department values from EMP to DPT, we can compute “DPT semijoin EMP” which retrieves all “3rd floor” departments that employ “sneakers” experts; see Fig. 1.1d. These semijoins are *cost-effective* if the amount of data eliminated from the database exceeds the quantity of data transmitted to compute the semijoins.

Semijoins are used in a similar fashion in RAP. RAP is a hardware database machine that includes restriction and semijoin instructions, but no join instruction. To compute a join, the relations involved must be transmitted to a conventional computer. Semijoins are used as in Example 1.1 to reduce the amount of data transmitted. Hardware semijoin instructions are also provided by the CAFS [Babb]³ and CASSM [SE]⁴ database machines.

³ Semijoin is called “join using bit array” in [Babb].

⁴ Semijoin is called “match” in [SE].

To use semijoins most effectively in these systems, it is important to understand which sequences of semijoins are most effective in reducing the size of relations. A sequence of semijoins *fully reduces* relation R for query q if it maximally reduces the size of R ; i.e., it eliminates from R all data not needed to answer q . For example, the semijoins in Example 1.1 fully reduce EMP and DPT for the given query. A sequence of semijoins is a *full reducer* for q if it fully reduces every relation in the database, for all states of the database. Not all relational queries have full reducers.

The principal issue in semijoin theory is to characterize those queries for which full reducers exist, and to generate algorithmically the full reducer when it exists. This issue was first addressed in [BC]. That paper studies the class of *equijoin queries* and *single attribute semijoins*. (Equijoin queries are essentially the ones studied by [ESW], [BGWRR], [HY], [SACLP], [WY]; single attribute semijoins are limited to using a single column of one relation to reduce another relation.) [BC] partitions the equijoin queries into subclasses called *tree queries* and *cyclic queries*, and proves that an equijoin query has a full reducer composed of single attribute semijoins iff it is a tree query. In addition, a linear time algorithm is presented that tests whether an equijoin query is a tree query and constructs a full reducer for it if it is a tree query.

In this paper we study the class of *natural join* (NJ) *queries* and general, *natural semijoins*. An NJ query computes the natural join of all relations in the database; a natural semijoin is half a natural join. On the surface the class of queries is trivially small; indeed for a given database there is exactly one NJ query! However, in Appendix A we show that every equijoin query can be translated into an isomorphic NJ query by renaming attributes in the database. Thus we lose no generality (relative to equijoin queries) by limiting our attention to NJ queries.

The paper is organized as follows. Section 2 defines our terminology. In §§ 3 and 4 we prove that an NJ query has a full reducer composed of natural semijoins iff it is a member of an expanded class of tree queries. Section 5 presents an efficient membership algorithm for the expanded class of tree queries. Appendix C extends our results to queries with “target lists”.

Other work on semijoin theory appears in [BC], [BG], [BG2], [Ch], [YO], [Yo1].

2. Terminology.

2.1. Relations and databases. Our terminology follows [ASU], [MMS]. A *universe* U is a finite set of *attributes* $\{A_1, \dots, A_m\}$. A *relation schema* R_i is a subset of U , and a *database schema* D is a set of relation schemas. Associated with each $A \in U$ is an infinite *domain*, $\text{dom}(A)$. The *domain* of relation schema R_i is $\text{dom}(R_i) = \times_{k=1}^l \text{dom}(A_{ik})$, where $R_i = \{A_{i1}, \dots, A_{il}\}$. A *relation state* for R_i is a finite subset of $\text{dom}(R_i)$; this state can be visualized as a table of data whose columns are labeled by $\{A_{i1}, \dots, A_{il}\}$. A *database state* for D is an assignment of relation states to its relation schemas.

Notationally, we use R_1, \dots, R_n to denote relation schemas and R_1, \dots, R_n (with possible superscript) to denote states for these schemas. Elements of a relation state are called *tuples*; r_i (with possible superscript) denotes a tuple of R_i . We use $D = \{R_1, \dots, R_n\}$ to denote a database schema and D (with possible superscript) to represent a state for D . $R_i(D)$ denotes the relation state assigned to R_i by D . Also, we assume that $U = \cup_{i=1}^n R_i$.

We define a partial order over database states for a given schema. Let D and D' be states for D . We define $D \leq D'$ if $R_i(D) \subseteq R_i(D')$ for $i = 1, \dots, n$.

2.2. Queries. Three relational operators are used in this paper. Let $X \subseteq R_i$. The *projection* of relation state R_i onto X , denoted $R_i[X]$, is the relation state obtained

by removing columns of R_i corresponding to attributes not in X and then eliminating duplicate tuples. Let R_i and R_j be states for \mathbf{R}_i and \mathbf{R}_j respectively. The *natural join* of R_i and R_j , denoted $R_i \bowtie R_j$, yields $\{r \in \text{dom}(\mathbf{R}_i \cup \mathbf{R}_j) \mid r[\mathbf{R}_i] \in R_i \wedge r[\mathbf{R}_j] \in R_j\}$, which is a relation state for $\mathbf{R}_i \cup \mathbf{R}_j$. The *natural semijoin* of R_i by R_j , denoted $R_i \ltimes R_j$, equals $(R_i \bowtie R_j)[\mathbf{R}_i]$. Equivalently, $R_i \ltimes R_j = \{r_i \in R_i \mid (\exists r_j \in R_j)(r_i[\mathbf{R}_i \cap \mathbf{R}_j] = r_j[\mathbf{R}_i \cap \mathbf{R}_j])\}$.

A *relational expression* is an expression over these operators with relation schemas (not states) as operands. A relational expression is interpreted by assigning a relation state to each operand. We generally perform this assignment by *applying* an expression to a database state; i.e., if \mathbf{E} is an expression whose operands are drawn from \mathbf{D} , we write $\mathbf{E}(D)$ to mean “interpret \mathbf{E} by assigning the state $\mathbf{R}_i(D)$ to each operand \mathbf{R}_i .” Expressions \mathbf{E}_1 and \mathbf{E}_2 are *equivalent*, written $\mathbf{E}_1 \equiv \mathbf{E}_2$, if for all states D , $\mathbf{E}_1(D) = \mathbf{E}_2(D)$.

The queries we study are called natural join (NJ) queries. The *NJ query* over database schema \mathbf{D} is the expression $\mathbf{R}_1 \bowtie \mathbf{R}_2 \bowtie \dots \bowtie \mathbf{R}_n$. (This expression is unambiguous because natural join is associative and commutative.)

We can represent a database schema and its NJ query by an undirected graph called a *qual graph*. A *qual graph* \mathbf{QG} over \mathbf{D} contains one node per relation schema in \mathbf{D} . Let $A \in \mathbf{U}$ and let $\text{Class}(A) = \{\mathbf{R}_i \in \mathbf{D} \mid A \in \mathbf{R}_i\}$. \mathbf{QG} is *A-connected* if the subgraph whose node set equals $\text{Class}(A)$ is connected; \mathbf{QG} is *U-connected* if it is *A-connected* for every $A \in \mathbf{U}$. \mathbf{QG} *represents* \mathbf{D} and its NJ query if it is *U-connected*. The relationship between *U-connected* qual graphs and NJ queries is explored in Appendix B. Generally, a single schema and NJ query is represented by many qual graphs.

An NJ query is a *tree query* if some qual graph that represents it is a tree. All other NJ queries are *cyclic queries*.

2.3. Tableaux. Tableaux are a formalism for relational expressions developed in [ABU], [ASU], [ASU1], [MMS]. We use a restricted version of this formalism. A *tableau* is a set of *rows* each of which contains one *component* per attribute in the database. If t is a row, and $\mathbf{U} = \{A_1, \dots, A_m\}$, the components of t are denoted $t[A_1], \dots, t[A_m]$. Each component takes values of three types: *distinguished variables*, *nondistinguished variables* and the constant *blank*. In addition, each row is *tagged* by a relation schema, denoted $\text{tag}(t)$.

Example 2.1. Let $\mathbf{D} = \{\mathbf{R}_1, \mathbf{R}_2, \mathbf{R}_3\}$, where $\mathbf{R}_1 = \{A_1, A_2\}$, $\mathbf{R}_2 = \{A_1, A_2, A_3\}$, and $\mathbf{R}_3 = \{A_1, A_4\}$. The following is a tableau:

$$T = \begin{array}{c|cccc|c} & A_1 & A_2 & A_3 & A_4 & (\text{tag}) \\ \hline & a1 & a2 & & & \mathbf{R}_1 \\ & a1 & a2 & x3 & & \mathbf{R}_2 \\ & a1 & & & x4 & \mathbf{R}_3 \end{array}$$

We will see that T represents $(\mathbf{R}_1 \bowtie \mathbf{R}_2 \bowtie \mathbf{R}_3)[A_1, A_2]$.

Additional constraints are placed on the form of tableaux.

1. Any variable (distinguished or nondistinguished) may appear in at most one column.
2. Each column may contain at most one *distinguished* variable. Thus, we may identify distinguished variables with columns; we shall use a_k to denote the distinguished variable for column A_k . (a_k may occur several times within column A_k , but may appear nowhere else in the tableau.)
3. Let t be any row. If $\text{tag}(t) = \mathbf{R}_i$ then $t[A_k] \neq \text{blank}$ iff $A_k \in \mathbf{R}_i$.
4. All tableaux (over a fixed universe) share the same distinguished variables.

Given \underline{D} of Example 2.1

a)

A ₁	A ₂	A ₃	A ₄	(tag)
a1	a2			<u>R₁</u>

A ₁	A ₂	A ₃	A ₄	(tag)
a1	a2	a3		<u>R₂</u>

A ₁	A ₂	A ₃	A ₄	(tag)
a1			a4	<u>R₃</u>

b)

A ₁	A ₂	A ₃	A ₄	(tag)
a1	a2			<u>R₁</u>
a1	a2	a3		<u>R₂</u>
a1			a4	<u>R₃</u>

c) Let $X = (x_1, \dots, x_4)$

$$\begin{aligned} \text{Tab}((\underline{R}_1 \sqcup \underline{R}_2 \sqcup \underline{R}_3)[A_1, A_2]) &= (\text{Tab}(\underline{R}_1 \sqcup \underline{R}_2 \sqcup \underline{R}_3)) [A_1, A_2] \\ &= \Pi(\text{Tab}(\underline{R}_1 \sqcup \underline{R}_2 \sqcup \underline{R}_3), \{A_1, A_2\}, X) \\ &= \begin{array}{|cccc|c} A_1 & A_2 & A_3 & A_4 & (\text{tag}) \\ \hline a1 & a2 & & & \underline{R}_1 \\ a1 & a2 & x3 & & \underline{R}_2 \\ a1 & & & x4 & \underline{R}_3 \end{array} \end{aligned}$$

FIG. 2.1. Tableau construction rules.

However, the nondistinguished variables of different tableaux are always distinct. Nondistinguished variables are denoted x_1, x_2, \dots with possible superscript.

The tableau for relational expression \mathbf{E} is denoted $\text{Tab}(\mathbf{E})$ and is defined as follows (see Fig. 2.1):

1. $\text{Tab}(\mathbf{R}_i) = \{t\}$, where $\text{tag}(t) = \mathbf{R}_i$, and

$$t[A_k] = \begin{cases} a_k & \text{if } A_k \in \mathbf{R}_i, \\ \text{blank} & \text{otherwise.} \end{cases}$$

2. $\text{Tab}(\mathbf{E}_1 \sqcup \mathbf{E}_2) = \text{Tab}(\mathbf{E}_1) \cup \text{Tab}(\mathbf{E}_2)$.

3. Let T be a tableau, let $\mathbf{U}' \subseteq \mathbf{U}$, and let $X' = \{x'_1, \dots, x'_m\}$ be a set of nondistinguished variables that *do not appear* in T . Then

$$\Pi(T, \mathbf{U}', X')$$

$$= \left\{ t' \mid (\exists t \in T) \left(\text{tag}(t') = \text{tag}(t) \wedge t'[A_k] = \begin{cases} x'_k & \text{if } t[A_k] = a_k \text{ and } A_k \notin \mathbf{U}' \\ t[A_k] & \text{otherwise} \end{cases} \right) \right\}.$$

In words, Π transforms the distinguished variable a_k into the nondistinguished variable x'_k , provided $A_k \notin \mathbf{U}'$. Generally, X' is implicit and we abbreviate $\Pi(T, \mathbf{U}', X')$ by $T[\mathbf{U}']$. Let \mathbf{E}_1 be a relational expression. $\text{Tab}(\mathbf{E}_1[\mathbf{U}'])$ is defined to be $(\text{Tab}(\mathbf{E}_1))[\mathbf{U}']$.

4. $\text{Tab}(\mathbf{E}_1 < \mathbf{E}_2)$ is defined by rules 1–3, since a semijoin equals a join and a projection.

Let T and T' be tableaux. A *containment mapping* from T to T' is a function θ from rows of T to rows of T' that:

(a) preserves tags—i.e., for all $t \in T$, $\text{tag}(t) = \text{tag}(\theta(t))$;

(b) preserves distinguished variables—i.e., for all $t \in T$, if $t[A_k] = a_k$ then $\theta(t)[A_k] = a_k$;

(c) preserves element equality—i.e., $\forall t_1, t_2 \in T$, if $t_1[A_k] = t_2[A_k]$ then $\theta(t_1)[A_k] = \theta(t_2)[A_k]$.

It is proved in [ASU] that if \mathbf{E}_1 and \mathbf{E}_2 are equivalent relational expressions, then containment mappings exist from $\text{Tab}(\mathbf{E}_1)$ to $\text{Tab}(\mathbf{E}_2)$ and vice versa.

2.4. Full reducers. We interpret sequences of semijoins as mappings from database states to database states. Let $\mathbf{SJ} = \langle \text{sj}_1, \dots, \text{sj}_l \rangle$ be a sequence of semijoins whose operands are drawn from \mathbf{D} . Each semijoin, e.g., $\mathbf{R}_i < \mathbf{R}_j$, maps D into D' , where $\mathbf{R}_k(D') = \mathbf{R}_k(D)$ for $k \neq i$, and $\mathbf{R}_i(D') = \mathbf{R}_i(D) < \mathbf{R}_j(D)$. \mathbf{SJ} maps D into $\text{sj}_l(\text{sj}_{l-1}(\dots(\text{sj}_1(D))\dots))$.

Let q be the NJ query over \mathbf{D} , and let D be any state of \mathbf{D} . The *full reduction* of D is D' , where $\mathbf{R}_i(D') = (q(D))[\mathbf{R}_i]$ for $i = 1, \dots, n$. D' satisfies two properties: (i) $q(D') = q(D)$; and (ii) D' is the minimum state under the \cong partial order satisfying (i).

A *full reducer* for q is a sequence of semijoins \mathbf{SJ} such that for all states D , $\mathbf{SJ}(D)$ equals the full reduction of D . Full reducers do not exist for all NJ queries. The principal goal of this paper is to characterize the NJ queries for which full reducers exist. We shall prove that an NJ query has a full reducer iff it is a tree query.

3. Full reducers for tree queries. If q is a tree query, a full reducer for q can be constructed using the techniques of [BC]. Let \mathbf{TQG} be a tree qual graph that represents q . Root \mathbf{TQG} at any node and let \mathbf{UP} (resp. \mathbf{DN}) be a directed tree obtained by directing every edge upward (resp. downward). Each directed edge may be interpreted as a semijoin; e.g., the edge $\langle \mathbf{R}_j, \mathbf{R}_i \rangle$ represents $\mathbf{R}_i < \mathbf{R}_j$. Each directed tree is interpreted as a semijoin program consisting of the semijoins represented by its edges, ordered by any topological sort. Intuitively, \mathbf{UP} depicts a leaf-to-root execution of semijoins, while \mathbf{DN} represents a root-to-leaf execution.

THEOREM 1. *Let q be a tree query and define \mathbf{UP} and \mathbf{DN} as above. $\mathbf{UP} \cdot \mathbf{DN}$ is a full reducer for q .*

Proof. [BC, Lemma 4] proves this result assuming $|\mathbf{R}_i \cap \mathbf{R}_j| = 1$ for all pairs of adjacent relations in \mathbf{UP} and \mathbf{DN} . The proof of the lemma is valid for the general case. \square

4. Cyclic queries do not have full reducers. In this section we prove that if a query has a full reducer, then it must be a tree query. We transform this into a problem involving the equivalence of relational expressions and employ tableaux to solve the transformed problem. The proof has five main steps. Let q be the NJ query over \mathbf{D} , and assume that \mathbf{SJ} is a full reducer for q .

1. We prove that a relational expression \mathbf{E} must exist such that (i) $\mathbf{E} \equiv q[\mathbf{R}_i]$ for $1 \leq i \leq n$, and (ii) all operators in \mathbf{E} are semijoins.

2. We examine the parse tree of \mathbf{E} and use tableaux to designate certain leaves of the parse tree as *essential* for specific relation schemas. Also, we prove that the tree must contain at least one essential leaf for each $\mathbf{R}_i \in \mathbf{D}$.

3. We label paths in the parse tree and prove that if \mathbf{leaf}_i is essential for \mathbf{R}_i and \mathbf{leaf}_j is essential for \mathbf{R}_j , then the label of the path between them is $\mathbf{R}_i \cap \mathbf{R}_j$.

4. We transform the parse tree into a qual graph in a way that preserves treeness and preserves the labeled paths identified in step 3. The qual graph we obtain is \mathbf{U} -connected and so represents q .

5. The conclusion is that if q has a full reducer, then there exists a tree qual graph that represents q . In other words, if q has a full reducer, q must be a tree query.

4.1. Existence of semijoin expressions. A *semijoin expression* is a relational expression whose operators are semijoins. (Notice that a semijoin expression maps database states into relation states whereas a semijoin sequence maps database states into database states.)

LEMMA 4.1. *Let \mathbf{SJ} be any sequence of semijoins and let $\mathbf{R}_i \in \mathbf{D}$. There exists a semijoin expression \mathbf{E} such that, for all states D , $\mathbf{E}(D) = \mathbf{R}_i(\mathbf{SJ}(D))$.*

Proof. Let $\mathbf{SJ} = \langle \mathbf{sj}_1, \dots, \mathbf{sj}_l \rangle$. The proof is by induction on l .

Basis step. $l = 1$. Let \mathbf{sj}_1 be $\mathbf{R}_j <] \mathbf{R}_k$. For $i \neq j$, $\mathbf{R}_i(\mathbf{SJ}(D)) = \mathbf{R}_i(D)$, and so \mathbf{E} is the null expression; for $i = j$, \mathbf{E} is $\mathbf{R}_j <] \mathbf{R}_k$.

Induction step. Let $\mathbf{SJ}' = \langle \mathbf{sj}_1, \dots, \mathbf{sj}_{l-1} \rangle$ and for $j = 1, \dots, n - 1$ let \mathbf{E}_j be a semijoin expression such that, for all D , $\mathbf{E}_j(D) = \mathbf{R}_j(\mathbf{SJ}'(D))$; these expressions exist by the induction hypothesis. Let \mathbf{sj}_l be $\mathbf{R}_j <] \mathbf{R}_k$. For $i \neq j$, $\mathbf{R}_i(\mathbf{SJ}(D)) = \mathbf{R}_i(\mathbf{SJ}'(D))$, and is $\mathbf{E} = \mathbf{E}_i$; for $i = j$, $\mathbf{E} = \mathbf{E}_j <] \mathbf{E}_k$. \square

LEMMA 4.2. *If \mathbf{SJ} is a full reducer for q , then for all $\mathbf{R}_i \in \mathbf{D}$, there exists a semijoin expression \mathbf{E} equivalent to $q[\mathbf{R}_i]$.*

Proof. By the definition of a full reducer, $\mathbf{R}_i(\mathbf{SJ}(D)) = (q(D))[\mathbf{R}_i]$ for all states D , while by Lemma 4.1 a semijoin expression \mathbf{E} exists such that $\mathbf{E}(D) = \mathbf{R}_i(\mathbf{SJ}(D))$ for all states D . \square

4.2. Essential leaves. The parse tree of a semijoin expression \mathbf{E} is denoted $\text{Parse}(\mathbf{E})$. $\text{Parse}(\mathbf{E})$ is a binary tree whose leaves represent relation schemas and whose interior nodes represent semijoins; see Fig. 4.1.

The following notation will facilitate our discussion. In the context of a specific parse tree, \mathbf{ro} denotes the root of the tree; \mathbf{no} is an arbitrary node of the tree; \mathbf{pa} is \mathbf{no} 's parent, assuming \mathbf{no} is not the root; and \mathbf{lc} , \mathbf{rc} denote \mathbf{no} 's left and right children, assuming \mathbf{no} is not a leaf. Also we use $\mathbf{E}(\mathbf{no})$ to denote the expression whose parse tree is rooted by \mathbf{no} ; $\text{ATTR}(\mathbf{no}) = \mathbf{R}_i$, where the leftmost descendant leaf of \mathbf{no} represents \mathbf{R}_i . We use $\text{Tab}(\mathbf{no})$ as an abbreviation for $\text{Tab}(\mathbf{E}(\mathbf{no}))$.

We observe that $\mathbf{E}(\mathbf{no}) = \mathbf{E}(\mathbf{lc}) <] \mathbf{E}(\mathbf{rc}) = (\mathbf{E}(\mathbf{lc}) [] \mathbf{E}(\mathbf{rc}))[\text{ATTR}(\mathbf{no})]$ for all non-leaf nodes. Therefore $\text{Tab}(\mathbf{no}) = (\text{Tab}(\mathbf{lc}) \cup \text{Tab}(\mathbf{rc})) [\text{ATTR}(\mathbf{no})]$ for all nonleaf nodes. It is important to recall that this notation is a shorthand for $\text{Tab}(\mathbf{no}) = \Pi(\text{Tab}(\mathbf{lc}) \cup \text{Tab}(\mathbf{rc}), \text{ATTR}(\mathbf{no}), X(\mathbf{no}))$, where $X(\mathbf{no})$ is a set of nondistinguished variables not present in $\text{Tab}(\mathbf{lc}) \cup \text{Tab}(\mathbf{rc})$; see § 2.3.

By Lemma 4.2, there exists a semijoin expression \mathbf{E} equivalent to $q[\mathbf{R}_1]$, and by [ASU] there exists a containment mapping θ from $\text{Tab}(q[\mathbf{R}_1])$ to $\text{Tab}(\mathbf{E})$. For each node of $\text{Parse}(\mathbf{E})$ we define

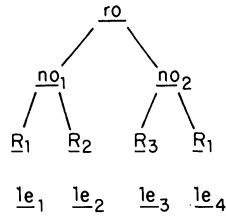
$$\text{Tab}'(\mathbf{no}) = \begin{cases} \theta(\text{Tab}(q[\mathbf{R}_1])) & \text{if } \mathbf{no} = \mathbf{ro}, \\ \{t \in \text{Tab}(\mathbf{no}) \mid \Pi(t, \text{ATTR}(\mathbf{pa}), X(\mathbf{pa})) \in \text{Tab}'(\mathbf{pa})\} & \text{otherwise.} \end{cases}$$

$\text{Tab}'(\mathbf{no})$ is called the *essential subtableau* of \mathbf{no} . See Fig. 4.2.

Essential subtableaux play a major role in our analysis and a few words motivating their definition are in order. $\text{Tab}'(\mathbf{ro})$ contains the rows of $\text{Tab}(\mathbf{E})$ needed for θ to exist. Proceeding down the parse tree, $\text{Tab}'(\mathbf{no})$ contains the rows of $\text{Tab}(\mathbf{no})$ that are

- Define \underline{D} as in Example 2.1
- Let $\underline{E}_{SJ} = (\underline{R}_1 < \underline{1R}_2) < \underline{1}(\underline{R}_3 < \underline{1R}_1)$

a) $\text{Parse}(\underline{E}_{SJ}) =$



- b) $\underline{E}(\underline{ro}) = \underline{E}_{SJ}$
 $\underline{E}(\underline{no}_1) = \underline{R}_1 < \underline{1R}_2$
 $\underline{E}(\underline{no}_2) = \underline{R}_3 < \underline{1R}_1$
 $\underline{E}(\underline{le}_1) = \underline{R}_1$
 etc.

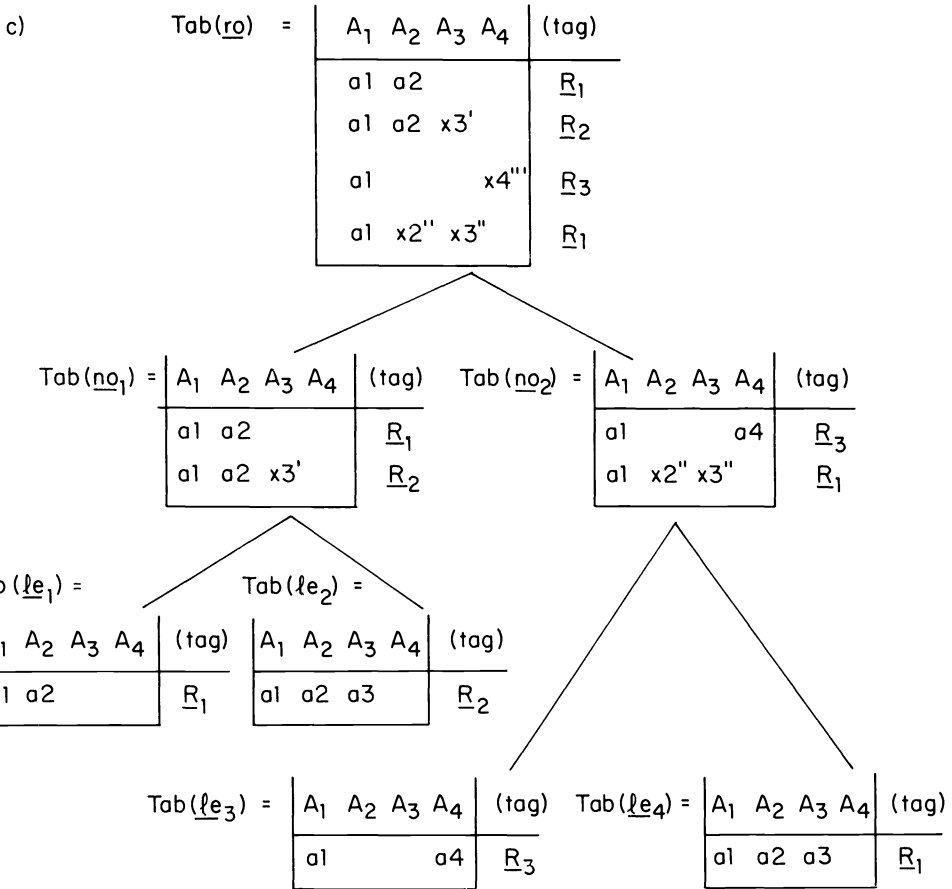


FIG. 4.1. Parse trees for semijoin expressions.

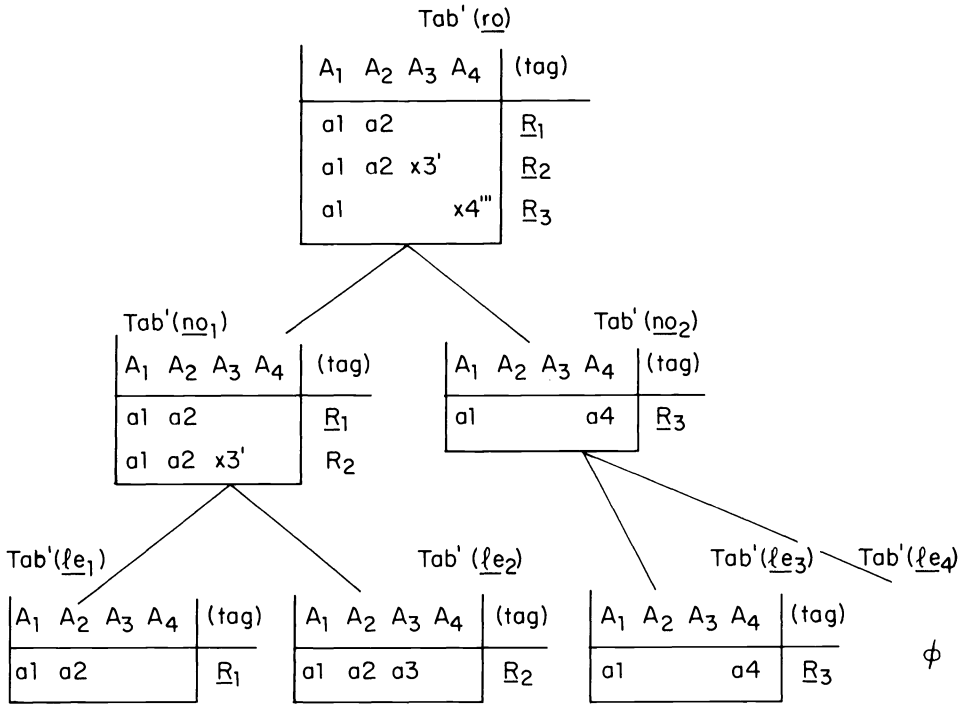


FIG. 4.2. Essential subtableaux for Fig. 4.1.

“sources” for rows of Tab'(**ro**). I.e., $t \in \text{Tab}'(\mathbf{no})$ iff t has an image in Tab'(**ro**) given by the sequence of projections “between” **no** and **ro**. (This concept is formalized in § 4.3.) For example, row $\langle a1, a2, a3, \rangle$ of Tab'(**le₂**) (see Fig. 4.2) has the image $\langle a1, a2, x3' \rangle = \Pi(\langle a1, a2, a3 \rangle, \{A_1, A_2\}, X(\mathbf{no}_1))$ in Tab'(**no₁**), and $\langle a1, a2, x3' \rangle$ has itself as an image in Tab'(**ro**).

A leaf of Parse(**E**) is *essential* if its essential subtableau is nonempty. A leaf is *essential for R_i* if **R_i** is the tag in its essential subtableau. For example, in Fig. 4.2, **le₁** is essential for **R₁**, while **le₄** is not essential for any relation schema. The following lemmas prove that Parse (**E**) contains at least one essential leaf for each **R_i** ∈ **D**.

LEMMA 4.3. *Let E be any semijoin expression equivalent to $q[\mathbf{R}_1]$, and let no be any nonleaf node of Parse(E). Then $\text{Tab}'(\mathbf{no}) = (\text{Tab}'(\mathbf{lc}) \cup \text{Tab}'(\mathbf{rc}))[\text{ATTR}(\mathbf{lc})]$.*

Proof. $\text{Tab}'(\mathbf{lc}) = \{t \in \text{Tab}(\mathbf{lc}) \mid t[\text{ATTR}(\mathbf{no})] \in \text{Tab}'(\mathbf{no})\}$, and $\text{Tab}'(\mathbf{rc}) = \{t \in \text{Tab}(\mathbf{rc}) \mid t[\text{ATTR}(\mathbf{no})] \in \text{Tab}'(\mathbf{no})\}$, by definition of Tab'. Therefore

$$\begin{aligned}
 & (\text{Tab}'(\mathbf{lc}) \cup \text{Tab}'(\mathbf{rc}))[\text{ATTR}(\mathbf{lc})] \\
 &= \{t' \mid (\exists t \in \text{Tab}(\mathbf{lc}) \cup \text{Tab}(\mathbf{rc}))(t' = t[\text{ATTR}(\mathbf{lc})] \wedge t[\text{ATTR}(\mathbf{no})] \in \text{Tab}'(\mathbf{no}))\} \\
 &= \{t' \mid (\exists t \in \text{Tab}(\mathbf{lc}) \cup \text{Tab}(\mathbf{rc}))(t' = t[\text{ATTR}(\mathbf{lc})] \wedge t' \in \text{Tab}'(\mathbf{no}))\}, \text{ since} \\
 & \quad \text{ATTR}(\mathbf{lc}) = \text{ATTR}(\mathbf{no}) \text{ because } \mathbf{no} \text{ and } \mathbf{lc} \text{ have the same leftmost descendant} \\
 &= \{t' \mid (\exists t \in \text{Tab}(\mathbf{lc}) \cup \text{Tab}(\mathbf{rc}))(t' = t[\text{ATTR}(\mathbf{lc})])\} \\
 & \quad \cap \{t' \mid (\exists t \in \text{Tab}(\mathbf{lc}) \cup \text{Tab}(\mathbf{rc}))(t' = t[\text{ATTR}(\mathbf{lc})] \in \text{Tab}'(\mathbf{no}))\} \\
 &= \text{Tab}'(\mathbf{no}) \cap \text{Tab}'(\mathbf{no}), \text{ since } \text{Tab}'(\mathbf{no}) = (\text{Tab}'(\mathbf{lc}) \cup \text{Tab}'(\mathbf{rc}))[\text{ATTR}(\mathbf{lc})] \\
 &= \text{Tab}'(\mathbf{no}), \text{ since } \text{Tab}'(\mathbf{no}) \subseteq \text{Tab}(\mathbf{no}), \text{ by definition of Tab'}. \quad \square
 \end{aligned}$$

LEMMA 4.4. *Let \mathbf{E} be any semijoin expression equivalent to $q[\mathbf{R}_1]$, and let \mathbf{ro} be the root of $\text{Parse}(\mathbf{E})$. Then $\text{Tab}'(\mathbf{ro})$ contains a row with tag \mathbf{R}_i for $i = 1, \dots, n$.*

Proof. By the definition of a NJ query, $q = \mathbf{R}_1[] \cdots []\mathbf{R}_n$, and $\text{Tab}(q) = \bigcup_{i=1}^n \text{Tab}(\mathbf{R}_i)$. $\text{Tab}(q)$ contains rows with tags $\mathbf{R}_1, \dots, \mathbf{R}_n$. $\text{Tab}(q[\mathbf{R}_1]) = (\text{Tab}(q))[\mathbf{R}_1]$ also contains rows with these tags, since the tableau projection operator defined in § 2.3 never deletes or changes a tag. By the definition of essential subtableaux, $\text{Tab}'(\mathbf{ro}) = \theta(\text{Tab}(q[\mathbf{R}_1]))$ for some containment mapping θ . $\text{Tab}'(\mathbf{ro})$ must also contain rows with tags $\mathbf{R}_1, \dots, \mathbf{R}_n$ since containment mappings preserve tags. \square

LEMMA 4.5. *Let \mathbf{E} be any semijoin expression equivalent to $q[\mathbf{R}_1]$. Then $\text{Parse}(\mathbf{E})$ contains at least one essential leaf for each $\mathbf{R}_i \in \mathbf{D}$.*

Proof. By Lemma 4.4, $\text{Tab}'(\mathbf{ro})$ contains a row with tag \mathbf{R}_i for each $\mathbf{R}_i \in \mathbf{D}$. By Lemma 4.3, if $\text{Tab}'(\mathbf{ro})$ contains a row with tag \mathbf{R}_i , then either $\text{Tab}'(\mathbf{lc})$ or $\text{Tab}'(\mathbf{rc})$ must contain such a row. Applying this argument inductively, some leaf \mathbf{le} must have such a row in $\text{Tab}'(\mathbf{le})$. Therefore \mathbf{le} is essential for \mathbf{R}_i . \square

4.3. Labeled essential paths. Let \mathbf{E} be any semijoin expression. For any pair of nodes $\{\mathbf{no}, \mathbf{no}'\}$ in $\text{Parse}(\mathbf{E})$, let P be the path between the nodes, and define $\text{label}(\{\mathbf{no}, \mathbf{no}'\}) = \bigcap_{\text{all nodes } \mathbf{no}'' \text{ on } P} \text{ATTR}(\mathbf{no}'')$. We shall prove that if \mathbf{le}_i and \mathbf{le}_j are essential leaves for \mathbf{R}_i and \mathbf{R}_j respectively, then $\text{label}(\{\mathbf{le}_i, \mathbf{le}_j\}) = \mathbf{R}_i \cap \mathbf{R}_j$.

4.3.1. The ancestor mapping. The ancestor mapping relates the tableau for any node \mathbf{no} to the tableaux of its ancestors in the parse tree. The ancestor mapping is defined recursively as follows:

1. If $\mathbf{an} = \mathbf{no}$, then $\text{Anc}_{\mathbf{no}, \mathbf{an}}$ is the identity function.
2. If \mathbf{an} is the parent of \mathbf{no} , $\text{Anc}_{\mathbf{no}, \mathbf{an}}$ maps tableau T into $\Pi(T, \text{ATTR}(\mathbf{an}), X(\mathbf{an})) = T[\text{ATTR}(\mathbf{an})]$.
3. If \mathbf{an} is any other ancestor of \mathbf{no} , and $p = \langle \mathbf{no}, \mathbf{n}_1, \dots, \mathbf{n}_i, \mathbf{an} \rangle$ is the path between \mathbf{no} and \mathbf{an} , then

$$\text{Anc}_{\mathbf{no}, \mathbf{an}} = \text{Anc}_{\mathbf{no}, \mathbf{n}_1} \cdot \text{Anc}_{\mathbf{n}_1, \mathbf{n}_2} \cdots \text{Anc}_{\mathbf{n}_i, \mathbf{an}}.$$

It is obvious that $\text{Anc}_{\mathbf{no}, \mathbf{an}}(\text{Tab}(\mathbf{no})) \subseteq \text{Tab}(\mathbf{an})$ for all nodes \mathbf{no} and ancestors \mathbf{an} . Lemma 4.6 proves that this property holds for essential subtableaux.

LEMMA 4.6. *Let \mathbf{E} be a semijoin expression equivalent to $q[\mathbf{R}_1]$, let \mathbf{no} be any node of $\text{Parse}(\mathbf{E})$ and let \mathbf{an} be any ancestor of \mathbf{no} . Then $\text{Anc}_{\mathbf{no}, \mathbf{an}}(\text{Tab}'(\mathbf{no})) \subseteq \text{Tab}'(\mathbf{an})$.*

Proof. The proof is by induction on the distance (i.e., number of edges) between \mathbf{no} and \mathbf{an} .

Basis steps. If distance = 0, then $\mathbf{no} = \mathbf{an}$, $\text{Anc}_{\mathbf{no}, \mathbf{an}}$ is the identity function and the result is immediate.

If distance = 1, then \mathbf{an} is the parent of \mathbf{no} . Let \mathbf{lc} and \mathbf{rc} denote the left and right children of \mathbf{an} , respectively. Notice that $\mathbf{no} \in \{\mathbf{rc}, \mathbf{lc}\}$. By Lemma 4.3

$$\begin{aligned} \text{Tab}'(\mathbf{an}) &= (\text{Tab}'(\mathbf{lc}) \cup \text{Tab}'(\mathbf{rc}))[\text{ATTR}(\mathbf{lc})] \\ &= (\text{Tab}'(\mathbf{lc}) \cup \text{Tab}'(\mathbf{rc}))[\text{ATTR}(\mathbf{an})] \text{ since every node has the same attributes as its left child} \\ &= \text{Tab}'(\mathbf{lc})[\text{ATTR}(\mathbf{an})] \cup \text{Tab}'(\mathbf{rc})[\text{ATTR}(\mathbf{an})] \\ &= \text{Anc}_{\mathbf{lc}, \mathbf{an}}(\text{Tab}'(\mathbf{lc})) \cup \text{Anc}_{\mathbf{rc}, \mathbf{an}}(\text{Tab}'(\mathbf{rc})). \end{aligned}$$

Thus the result holds for distance = 1, whether $\mathbf{no} = \mathbf{lc}$ or $\mathbf{no} = \mathbf{rc}$.

Induction step. Let **pa** be the parent of **no**. By the definition of Anc , $\text{Anc}_{\text{no},\text{an}} = \text{Anc}_{\text{no},\text{pa}} \cdot \text{Anc}_{\text{pa},\text{an}}$. Therefore

$$\begin{aligned} \text{Anc}_{\text{no},\text{an}}(\text{Tab}'(\mathbf{no})) &= \text{Anc}_{\text{pa},\text{an}}(\text{Anc}_{\text{no},\text{pa}}(\text{Tab}'(\mathbf{no}))) \\ &\subseteq \text{Anc}_{\text{pa},\text{an}}(\text{Tab}'(\mathbf{pa})), \text{ (since } \text{Tab}'(\mathbf{pa}) \subseteq \text{Anc}_{\text{no},\text{pa}}(\text{Tab}'(\mathbf{no})) \text{)} \\ &\quad \text{by the basis step)} \\ &\subseteq \text{Tab}'(\mathbf{an}), \text{ (by the induction hypothesis). } \quad \square \end{aligned}$$

The next two lemmas use the ancestor mapping to establish a relationship between labeled paths and distinguished variables in tableaux.

LEMMA 4.7. *Let t be a row of a tableau, and let $t' = t[\mathbf{U}']$ for some $\mathbf{U}' \subseteq \mathbf{U}$. Then $t'[A_k] = a_k$ iff $t[A_k] = a_k$ and $A_k \in \mathbf{U}'$.*

Proof. The “if” part is immediate from the definition of tableau projection. There are two cases for the converse.

Case 1. $t[A_k] \neq a_k$. $t'[A_k] \neq a_k$ in this case, because tableau projection never maps a nondistinguished variable or blank into a distinguished variable.

Case 2. $A_k \notin \mathbf{U}'$. The same conclusion holds in this case since if $t[A_k] = a_k$ and $A_k \notin \mathbf{U}'$, the projection operator will map $t[A_k]$ into some nondistinguished variable x'_k . \square

LEMMA 4.8. *Let \mathbf{E} be any semijoin expression, let \mathbf{le} be a leaf of $\text{Parse}(\mathbf{E})$, let $\text{Tab}(\mathbf{le}) = \{t\}$, and let \mathbf{an} be any ancestor of \mathbf{le} . Then for all attributes A_k , $\text{Anc}_{\mathbf{le},\text{an}}(t)[A_k] = a_k$ iff $A_k \in \text{label}(\{\mathbf{le}, \mathbf{an}\})$.*

Proof. Let $t' = \text{Anc}_{\mathbf{le},\text{an}}(t)$ and let $P = \langle \mathbf{le}, \mathbf{n}_1, \dots, \mathbf{n}_i, \mathbf{an} \rangle$ be the path from \mathbf{le} to \mathbf{an} . By the definition of $\text{Anc}_{\mathbf{le},\text{an}}$,

$$t' = ((\dots ((t[\text{ATTR}(\mathbf{n}_1))][\text{ATTR}(\mathbf{n}_2))] \dots)[\text{ATTR}(\mathbf{n}_i))][\text{ATTR}(\mathbf{an})].$$

Applying Lemma 4.7 iteratively yields that $t'[A_k] = a_k$ iff $t[A_k] = a_k$ and $A_k \in \bigcap_{i=1}^i \text{ATTR}(\mathbf{n}_i) \cap \text{ATTR}(\mathbf{an})$. Moreover, $t[A_k] = a_k$ iff $A_k \in \mathbf{R}_i$, where $\mathbf{R}_i = \text{tag}(t)$; and since every leaf is its own leftmost descendant, $\text{ATTR}(\mathbf{le}) = \mathbf{R}_i$. Thus $t'[A_k] = a_k$ iff $A_k \in \text{ATTR}(\mathbf{le}) \cap \bigcap_{i=1}^i \text{ATTR}(\mathbf{n}_i) \cap \text{ATTR}(\mathbf{an}) = \text{label}(\{\mathbf{le}, \mathbf{an}\})$. \square

4.3.2. The form of $\text{Tab}'(\mathbf{ro})$. Let \mathbf{E} be a semijoin expression equivalent to $q[\mathbf{R}_1]$, and let \mathbf{ro} be the root of $\text{Parse}(\mathbf{E})$. $\text{Tab}'(\mathbf{ro})$ has a very simple structure which we can exploit. In particular, we shall prove that each column contains at most one variable, either distinguished or nondistinguished. Moreover, the variable in the A_k column is distinguished iff $A_k \in \mathbf{R}_1$.

LEMMA 4.9. *Let $X = \{x_1, \dots, x_m\}$ be a set of nondistinguished variables, where $m = |\mathbf{U}|$. $\text{Tab}(q[\mathbf{R}_1]) = \{t_1, \dots, t_n\}$, where for $i = 1, \dots, n$*

(a) $\text{tag}(t_i) = \mathbf{R}_i$; and

$$(b) \quad t_i[A_k] = \begin{cases} a_k & \text{if } A_k \in \mathbf{R}_i \cap \mathbf{R}_1, \\ x_k & \text{if } A_k \in \mathbf{R}_i - \mathbf{R}_1, \\ \text{blank} & \text{if } A_k \notin \mathbf{R}_i. \end{cases}$$

Proof. $\text{Tab}(q[\mathbf{R}_1]) = (\text{Tab}(q))[\mathbf{R}_1] = (\bigcup_{i=1}^n \text{Tab}(\mathbf{R}_i))[\mathbf{R}_1] = (\{t'_1, \dots, t'_n\})[\mathbf{R}_1]$, where for $i = 1, \dots, n$

(a) $\text{tag}(t'_i) = \mathbf{R}_i$, and

$$(b) \quad t'_i[A_k] = \begin{cases} a_k & \text{if } A_k \in \mathbf{R}_i, \\ \text{blank} & \text{if } A_k \notin \mathbf{R}_i. \end{cases}$$

The result follows by application of tableau projection to $\text{Tab}(q)$. \square

LEMMA 4.10. *Let \mathbf{E} be a semijoin expression equivalent to $q[\mathbf{R}_1]$. For all $t \in \text{Tab}(\mathbf{E})$, if $t[A_k] = a_k$ then $A_k \in \mathbf{R}_1$.*

Proof. Since $\mathbf{E} \equiv q[\mathbf{R}_1]$, there exists a containment mapping θ from $\text{Tab}(\mathbf{E})$ to $\text{Tab}(q[\mathbf{R}_1])$. θ must preserve distinguished variables, so if $t[A_k] = a_k$, then $(\theta(t))[A_k] = a_k$. And for all $t' \in \text{Tab}(q[\mathbf{R}_1])$, if $t'[A_k] = a_k$ then $A_k \in \mathbf{R}_1$, by Lemma 4.9. \square

LEMMA 4.11. *Let \mathbf{E} be a semijoin expression equivalent to $q[\mathbf{R}_1]$ and let \mathbf{ro} be the root of $\text{Parse}(\mathbf{E})$. Let $X = \{x_1, \dots, x_m\}$ be a set of nondistinguished variables. Then*

- (i) *for all $A_k \in \mathbf{R}_1$ and for all $t \in \text{Tab}'(\mathbf{ro})$, $t[A_k] = a_k$ or is blank;*
- (ii) *for all $A_k \notin \mathbf{R}_1$ and for all $t \in \text{Tab}'(\mathbf{ro})$, $t[A_k] = x_k$ or is blank.*

Proof. Let θ be a containment mapping such that $\theta(\text{Tab}(q[\mathbf{R}_1])) = \text{Tab}'(\mathbf{ro})$; θ exists by definition of essential subtableaux. By Lemma 4.9 each column of $\text{Tab}(q[\mathbf{R}_1])$ contains at most one variable. Since θ must preserve equality of elements and can never map blanks into variables, each column of $\text{Tab}'(\mathbf{ro})$ also contains at most one variable.

(i) If $A_k \in \mathbf{R}_1$, the variable in the A_k column of $\text{Tab}(q[\mathbf{R}_1])$ is a_k . Since θ must preserve distinguished variables, the variable in the A_k column of $\text{Tab}'(\mathbf{ro})$ is also a_k .

(ii) If $A_k \notin \mathbf{R}_1$, $t'[A_k] \neq a_k$ for any $t' \in \text{Tab}'(\mathbf{ro})$, by Lemma 4.10. So in this case the variable in the A_k column of $\text{Tab}'(\mathbf{ro})$ must be x_k . \square

4.3.3. Projectors. Let \mathbf{E} be a semijoin expression equivalent to $q[\mathbf{R}_1]$ and let \mathbf{le} be an essential leaf for \mathbf{R}_i in $\text{Parse}(\mathbf{E})$. Let $\text{Tab}(\mathbf{le}) = \{t\}$ and $t' = \text{Anc}_{\mathbf{le}, \mathbf{ro}}(t)$. Since \mathbf{le} is essential, Lemma 4.6 proves that $t' \in \text{Tab}'(\mathbf{ro})$. Choose any $A_k \notin \mathbf{R}_1$. By Lemma 4.11, $t'[A_k] \neq a_k$, and so by Lemma 4.8, $A_k \notin \text{label}(\{\mathbf{le}, \mathbf{ro}\})$. In other words, there exists a node \mathbf{no} on the path from \mathbf{le} to \mathbf{ro} such that $A_k \notin \text{ATTR}(\mathbf{no})$. The first such node (i.e., the one closest to \mathbf{le}) is called the A_k -projector of \mathbf{le} . Intuitively, this is the node that “projects a_k out of t ”. This intuition is formalized by the following lemma.

LEMMA 4.12. *Let \mathbf{E} be a semijoin program equivalent to $q[\mathbf{R}_1]$. Let \mathbf{le} be an essential leaf for \mathbf{R}_i in $\text{Parse}(\mathbf{E})$, let $A_k \notin \mathbf{R}_1$, and let $p = \langle \mathbf{le}, \mathbf{n}_1, \mathbf{n}_2, \dots, \mathbf{n}_b, \mathbf{ro} \rangle$ be the path from \mathbf{le} to \mathbf{ro} . Also let $\text{Tab}(\mathbf{le}) = \{t\}$, and let \mathbf{no} be the A_k -projector for \mathbf{le} . Then*

- (i) *if \mathbf{n}_i is any node before \mathbf{no} in p , $\text{Anc}_{\mathbf{le}, \mathbf{n}_i}(t)[A_k] = a_k$; while*
- (ii) *if \mathbf{n}_i is \mathbf{no} or any node after \mathbf{no} in p , $\text{Anc}_{\mathbf{le}, \mathbf{n}_i}(t)[A_k] = x_k$ for some nondistinguished variable x_k .*

Proof. (i) $A_k \in \text{ATTR}(\mathbf{le})$, since \mathbf{le} represents \mathbf{R}_i . Also $A_k \in \text{ATTR}(\mathbf{n}_i)$ for every \mathbf{n}_i before \mathbf{no} since \mathbf{no} is the first node in p that does not contain A_k . Thus $A_k \in \text{label}(\{\mathbf{le}, \mathbf{n}_i\})$ for all \mathbf{n}_i before \mathbf{no} , and the result follows by Lemma 4.8.

(ii) By the definition of an A_k -projector, $A_k \notin \text{ATTR}(\mathbf{no})$ and so $\text{Anc}_{\mathbf{le}, \mathbf{no}}(t)[A_k] \neq a_k$ by Lemma 4.8. Since projection operators never map variables into blanks, $\text{Anc}_{\mathbf{le}, \mathbf{no}}(t)[A_k] = x_k$ for some nondistinguished variable x_k . Subsequent projections cannot change a nondistinguished variable, so the result follows. \square

LEMMA 4.13. *Define \mathbf{E} , \mathbf{le} , and \mathbf{no} as in Lemma 4.12, and let $A_k \in \mathbf{R}_i - \mathbf{R}_1$. Then:*

(i) *The nondistinguished variable x_k assigned to $\text{Anc}_{\mathbf{le}, \mathbf{no}}(t)[A_k]$ is distinct from all nondistinguished variables in the tableaux of \mathbf{no} 's children.*

(ii) *\mathbf{le} is in the right subtree of \mathbf{no} .*

Proof. (i) follows from the definition of tableau projection. (ii) holds because $\text{ATTR}(\mathbf{le}) = \text{ATTR}(\mathbf{no})$ for all nodes. \square

The final lemma of this subsection considers a pair of leaves, \mathbf{le}_i and \mathbf{le}_j , which are essential for \mathbf{R}_i and \mathbf{R}_j , respectively. Let $A_k \in \mathbf{R}_i \cap \mathbf{R}_j - \mathbf{R}_1$. It is evident that \mathbf{le}_i and \mathbf{le}_j must both have A_k -projectors. The critical fact proved in the next lemma is that these A_k -projectors must be identical.

LEMMA 4.14. *Let E be a semijoin expression equivalent to $q[\mathbf{R}_1]$, and let \mathbf{le}_i and \mathbf{le}_j be essential leaves in $\text{Parse}(\mathbf{E})$ for \mathbf{R}_i and \mathbf{R}_j , respectively. Then for all $A_k \in \mathbf{R}_i \cap \mathbf{R}_j - \mathbf{R}_1$ the A_k -projector for \mathbf{le}_i is also the A_k -projector for \mathbf{le}_j .*

Proof. Let $\text{Tab}(\mathbf{le}_i) = \{t_i\}$ and let $t'_i = \text{Anc}_{\mathbf{le}_i, \mathbf{ro}}(t_i)$. Select any $A_k \in \mathbf{R}_i \cap \mathbf{R}_j - \mathbf{R}_1$. By Lemma 4.12(ii), $t'_i[A_k] = x_{ik}$ for some nondistinguished variable x_{ik} . Define t_j , t'_j and x_{jk} analogously. By Lemma 4.6, t'_i and t'_j are elements of $\text{Tab}'(\mathbf{ro})$, while by Lemma 4.11 $\text{Tab}'(\mathbf{ro})$ contains at most one variable per column. It follows that $x_{ik} = x_{jk}$. Now let \mathbf{no}_i and \mathbf{no}_j be the A_k -projectors for \mathbf{le}_i and \mathbf{le}_j , respectively. We shall prove that if $\mathbf{no}_i \neq \mathbf{no}_j$, then $x_{ik} \neq x_{jk}$, contradicting the argument above. There are two cases:

1. \mathbf{no}_i is an ancestor of \mathbf{no}_j (or vice versa); or
2. \mathbf{no}_i and \mathbf{no}_j are incomparable nodes.

In the first case, let \mathbf{ch}_j be the child of \mathbf{no}_i that lies on the path between \mathbf{no}_i and \mathbf{no}_j . By Lemma 4.12(ii), $\text{Anc}_{\mathbf{le}_j, \mathbf{ch}_j}(t_j)[A_k] = x_{jk}$ and so x_{jk} is a nondistinguished variable that is present in a child of \mathbf{no}_i . Consequently, by Lemma 4.13(i), $\text{Anc}_{\mathbf{le}_i, \mathbf{no}_i}(t_i)[A_k] \neq x_{jk}$. But by Lemma 4.12(ii) $\text{Anc}_{\mathbf{le}_i, \mathbf{no}_i}(t_i)[A_k] = t'_i[A_k]$, while $t'_i[A_k] = x_{ik}$ by assumption. The conclusion is $x_{ik} \neq x_{jk}$ as claimed.

In the second case, $\text{Tab}(\mathbf{no}_i)$ and $\text{Tab}(\mathbf{no}_j)$ are distinct tableaux and we are *required* to assume they share no nondistinguished variables (see § 2.3). Moreover by Lemma 4.12(ii) $\text{Anc}_{\mathbf{le}_i, \mathbf{no}_i}(t_i)[A_k] = t'_i[A_k] = x_{ik}$ and similarly $\text{Anc}_{\mathbf{le}_j, \mathbf{no}_j}(t_j)[A_k] = x_{jk}$. Thus, x_{ik} and x_{jk} are respectively nondistinguished variables in $\text{Tab}(\mathbf{no}_i)$ and $\text{Tab}(\mathbf{no}_j)$, and $x_{ik} \neq x_{jk}$ follows in this case as well. \square

4.3.4. Labeled paths between essential leaves. We now use the ancestor mapping and the notion of A_k -projectors to prove the main result of step 3 of the proof of Theorem 2.

LEMMA 4.15. *Let E be a semijoin expression equivalent to $q[\mathbf{R}_1]$, and let \mathbf{le}_i and \mathbf{le}_j be essential leaves in $\text{Parse}(\mathbf{E})$ for \mathbf{R}_i and \mathbf{R}_j , respectively. Then $\text{label}(\{\mathbf{le}_i, \mathbf{le}_j\}) = \mathbf{R}_i \cap \mathbf{R}_j$.*

Proof. $\text{label}(\{\mathbf{le}_i, \mathbf{le}_j\}) \subseteq \mathbf{R}_i \cap \mathbf{R}_j$ is obvious from the definition. To prove inclusion in the opposite direction there are two cases.

Case 1. $A_k \in \mathbf{R}_i \cap \mathbf{R}_j - \mathbf{R}_1$. Let \mathbf{no} be the A_k -projector of \mathbf{le}_j . In addition, by Lemma 4.13(ii), \mathbf{le}_i and \mathbf{le}_j are both descendants of \mathbf{rc} , the right child of \mathbf{no} . By the definition of an A_k -projector, $A_k \in \text{ATTR}(\mathbf{no}')$ for every \mathbf{no}' between \mathbf{le}_i and \mathbf{rc} , and \mathbf{le}_j and \mathbf{rc} . Thus $A_k \in \text{label}(\{\mathbf{le}_i, \mathbf{rc}\})$ and $A_k \in \text{label}(\{\mathbf{le}_j, \mathbf{rc}\})$. Finally, since the path from \mathbf{le}_i to \mathbf{le}_j is a subset of the (possibly repeated) path from \mathbf{le}_i to \mathbf{rc} to \mathbf{le}_j , $\text{label}(\{\mathbf{le}_i, \mathbf{le}_j\}) \supseteq \text{label}(\{\mathbf{le}_i, \mathbf{rc}\}) \cap \text{label}(\{\mathbf{le}_j, \mathbf{rc}\}) \supseteq \{A_k\}$.

Case 2. $A_k \in \mathbf{R}_i \cap \mathbf{R}_j \cap \mathbf{R}_1$. Let $\text{Tab}(\mathbf{le}_i) = \{t_i\}$ and let $t'_i = \text{Anc}_{\mathbf{le}_i, \mathbf{ro}}(t_i)$. By Lemma 4.6 $t'_i \in \text{Tab}'(\mathbf{ro})$. Since $A_k \in \mathbf{R}_i$, $t_i[A_k]$ is nonblank, and since projections never map variables into blanks, $t'_i[A_k]$ is also nonblank. Since $A_k \in \mathbf{R}_1$, Lemma 4.11 proves that $t'_i[A_k] = a_k$, and so by Lemma 4.8 $A_k \in \text{label}(\{\mathbf{le}_i, \mathbf{ro}\})$. A symmetric argument shows that $A_k \in \text{label}(\{\mathbf{le}_j, \mathbf{ro}\})$. Thus $\text{label}(\{\mathbf{le}_i, \mathbf{le}_j\}) \supseteq \{A_k\}$ follows by the same argument as in Case 1. \square

4.4. Obtaining a tree qual graph. Let E be a semijoin expression equivalent to $q[\mathbf{R}_1]$. Sections 4.2 and 4.3 have established two important similarities between the essential leaves of $\text{Parse}(\mathbf{E})$ and a tree qual graph \mathbf{TQG} that represents q . Section 4.2 has shown that $\text{Parse}(\mathbf{E})$ contains at least one essential leaf per relation schema; i.e., the essential leaves of $\text{Parse}(\mathbf{E})$ are a superset of the nodes of \mathbf{TQG} . Section 4.3 has shown that the path between each pair of essential leaves is labeled with the intersection of the leaves' attributes; this property is similar to the property of \mathbf{U} -connectivity that holds in \mathbf{TQG} (see § 2.2). In addition, $\text{Parse}(\mathbf{E})$ is a tree.

Parse(**E**) differs from **TQG** in one important respect: it contains too many nodes. All nodes of Parse(**E**) that are not essential leaves have no place in **TQG**. Also, Parse(**E**) may contain *multiple* essential leaves per relation schema whereas **TQG** contains exactly *one* node per relation schema. In this section we prove that the “extra” nodes of Parse(**E**) can be eliminated without destroying the “**U**-connectivity” of the graph and without destroying its treeness.

We transform Parse(**E**) iteratively, eliminating one node and one edge at each step. We begin by associating each node of Parse(**E**) with the relation schema represented by its leftmost descendant; i.e., **no** is called a *node* for **R_i** if its leftmost descendant represents the expression **R_i**. Also, we use the term *essential node* in place of *essential leaf*. T_1, \dots, T_L denote the sequence of graphs obtained by our transformation, with $T_1 = \text{Parse}(\mathbf{E})$. Finally, for $l = 1, \dots, L$, and for every **no_i** and **no_j** in T_l ,

$$\text{label}_l(\{\mathbf{no}_i, \mathbf{no}_j\}) = \bigcup_{\substack{\text{all paths } P \\ \text{between } \mathbf{no}_i \text{ and} \\ \mathbf{no}_j \text{ in } T_l}} \bigcap_{\substack{\text{all nodes} \\ \text{on path } P}} \text{ATTR}(\mathbf{no}).$$

(We shall prove that each T_l is a tree and so the “union” in the above definition can be dropped.)

The transformation from T_l to T_{l+1} occurs via rule Tr: Let **no** be any essential node for any **R_i**, $1 \leq i \leq n$, and let **no'** be any other node for **R_i**, essential or not. The transformation has two steps.

1. Eliminate the *first edge* on the path from **no'** to **no**, thereby disconnecting the graph.
2. Reconnect the graph by merging **no** and **no'**; to be precise, replace every edge {**no'**, **no''**} by {**no**, **no''**}, and then remove **no'** from the graph. **no** remains in the graph, and remains an *essential node* for **R_i**.

Tr is illustrated in Fig. 4.3. Tr^+ denotes the successive application of Tr until no further transformations are possible. Intuitively, Tr^+ is *correct* if $\text{Tr}^+(\text{Parse}(\mathbf{E}))$ is a tree qual graph that represents q . We characterize correctness by four properties:

- (i) *termination*—the sequence T_1, \dots, T_L must be finite.
- (ii) *uniqueness of nodes*— T_L must contain exactly one node for each relation schema in **D**.
- (iii) *treeness*— T_L must be a tree.
- (iv) For all nodes **no_i** and **no_j** in T_L , $\text{label}_L(\{\mathbf{no}_i, \mathbf{no}_j\}) = \mathbf{R}_i \cap \mathbf{R}_j$, where **no_i** is a node for **R_i** and **no_j** is a node for **R_j**.

LEMMA 4.16. *If properties (i)–(iv) hold, T_L is a tree qual graph that represents q .*

Proof. By (ii), T_L has the correct structure for a qual graph. By (iii), T_L is a tree. By (iv) T_L represents q , since this property implies that for all $A_k \in \mathbf{U}$, $\text{Class}(A_k)$ forms a connected subgraph of T_L ; i.e., T_L is **U**-connected. \square

The remaining task is to prove that $\text{Tr}^+(\text{Parse}(\mathbf{E}))$ satisfies properties (i)–(iv).

LEMMA 4.17. *Let **E** be a semijoin expression equivalent to $q[\mathbf{R}_1]$. Then $\text{Tr}^+(\text{Parse}(\mathbf{E}))$ satisfies properties (i)–(iii).*

Proof. (i) Termination is ensured since T_{l+1} is strictly smaller than T_l for $l = 1, \dots, L - 1$.

(ii) T_L contains at least one node per relation schema since Tr *preserves essential nodes*—i.e., if T_l contains an essential node for **R_i** then so does $\text{Tr}(T_l)$ —and Parse(**E**) contains at least one essential node per schema by Lemma 4.5. T_L contains no more than one node per relation schema, else T_L could be further transformed by Tr, contradicting the definition of Tr^+ .

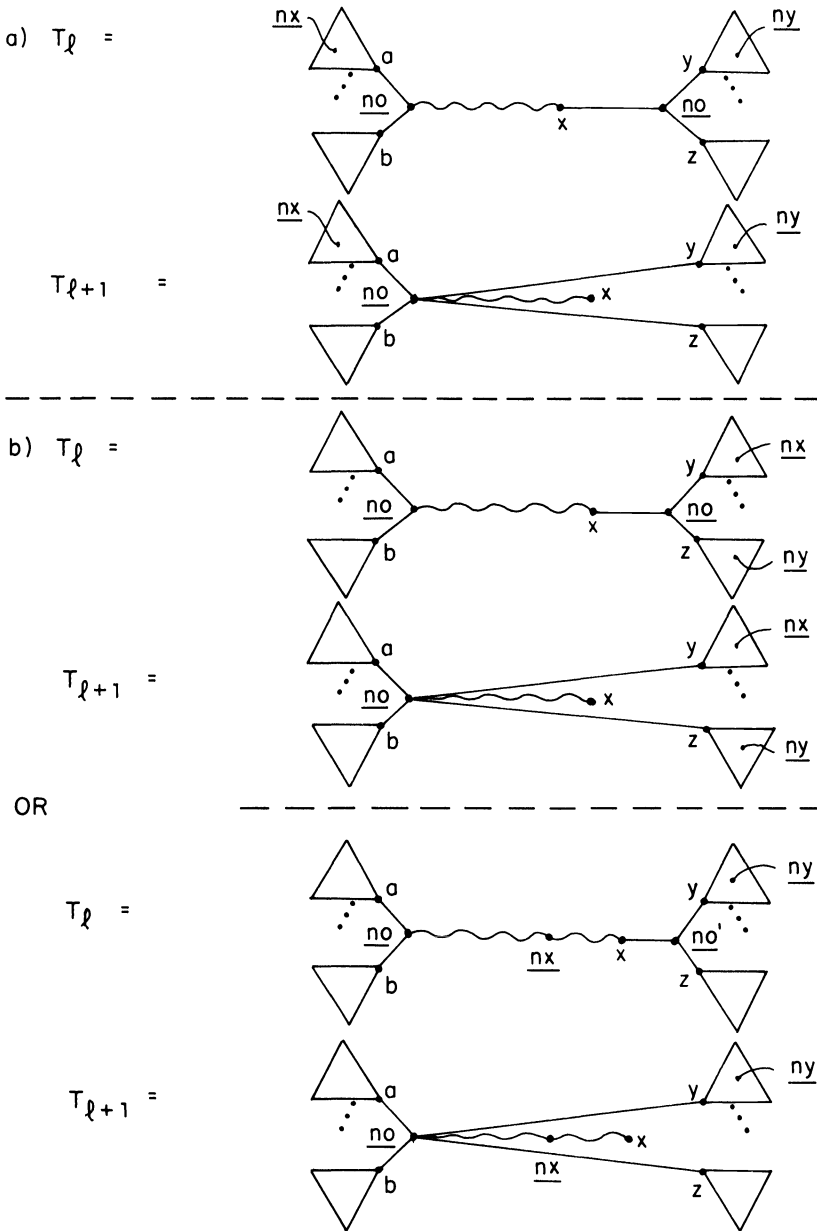


FIG. 4.3. Cases for Lemma 4.18.

(iii) Tr preserves treeness, since it eliminates one edge and one node from its operand while preserving the connectivity of the remaining nodes. \square

LEMMA 4.18. Let \mathbf{E} be a semijoin expression equivalent to $q[\mathbf{R}_1]$. Then $\text{Tr}^+(\text{Parse}(\mathbf{E}))$ satisfies property (iv).

Proof. We shall prove that Tr preserves essential labels; i.e., if $\text{label}(\{\mathbf{no}_i, \mathbf{no}_j\}) = \text{ATTR}(\mathbf{no}_i) \cap \text{ATTR}(\mathbf{no}_j)$ for every pair of essential nodes in T_b , then the same property holds in T_{l+1} . Since the property holds in $\text{Parse}(\mathbf{E})$ by Lemma 4.15, this suffices to prove the lemma.

Define \mathbf{no} and \mathbf{no}' as in the definition of Tr and let \mathbf{nx} and \mathbf{ny} be distinct essential nodes of T_l , with $\mathbf{nx} \neq \mathbf{no}'$ and $\mathbf{ny} \neq \mathbf{no}'$. Thus \mathbf{nx} and \mathbf{ny} are essential nodes of T_{l+1} as well. We consider three cases.

Case 1. The path from \mathbf{nx} to \mathbf{ny} in T_l does not include \mathbf{no}' . In this case, the transformation from T_l to T_{l+1} does not affect the path between \mathbf{nx} and \mathbf{ny} , hence does not affect their label.

Case 2. The path from \mathbf{nx} to \mathbf{ny} in T_l include \mathbf{no}' and \mathbf{no} (see Fig. 4.3a). In this case, the transformation shortens the path from \mathbf{nx} to \mathbf{ny} by “splicing out” the portion from \mathbf{no} to \mathbf{no}' . I.e., if the path in T_l is $\langle \mathbf{nx}, \mathbf{n}_1, \dots, \mathbf{n}_k, \mathbf{no}, \dots, \mathbf{no}', \mathbf{n}_{k+1}, \dots, \mathbf{ny} \rangle$, the path in T_{l+1} is $\langle \mathbf{nx}, \mathbf{n}_1, \dots, \mathbf{n}_k, \mathbf{no}, \mathbf{n}_{k+1}, \dots, \mathbf{ny} \rangle$. Thus $\text{label}_{l+1}(\{\mathbf{nx}, \mathbf{ny}\}) \supseteq \text{label}_l(\{\mathbf{nx}, \mathbf{ny}\})$. Since $\text{label}_l(\{\mathbf{nx}, \mathbf{ny}\}) = \text{ATTR}(\mathbf{nx}) \cap \text{ATTR}(\mathbf{ny})$ by assumption, and $\text{label}_{l+1}(\{\mathbf{nx}, \mathbf{ny}\}) \subseteq \text{ATTR}(\mathbf{nx}) \cap \text{ATTR}(\mathbf{ny})$ by definition of label, $\text{label}_{l+1}(\{\mathbf{nx}, \mathbf{ny}\}) = \text{ATTR}(\mathbf{nx}) \cap \text{ATTR}(\mathbf{ny})$ as desired.

Case 3. The path from \mathbf{nx} to \mathbf{ny} in T_l includes \mathbf{no}' but not \mathbf{no} (see Fig. 4.3b). Since $\text{label}_l(\{\mathbf{nx}, \mathbf{ny}\}) = \text{ATTR}(\mathbf{nx}) \cap \text{ATTR}(\mathbf{ny})$ by assumption, and since \mathbf{no}' is on the path between these nodes, $\text{ATTR}(\mathbf{no}') \supseteq \text{ATTR}(\mathbf{nx}) \cap \text{ATTR}(\mathbf{ny})$. The transformed path has the form $\langle \mathbf{nx}, \dots, \mathbf{no}, \dots, \mathbf{ny} \rangle$. By definition of Tr , \mathbf{no} is an essential node, and so Cases 1 and 2 prove that $\text{label}_{l+1}(\{\mathbf{nx}, \mathbf{no}\}) = \text{label}_l(\{\mathbf{nx}, \mathbf{no}\}) = \text{ATTR}(\mathbf{nx}) \cap \text{ATTR}(\mathbf{no})$; similarly $\text{label}_{l+1}(\{\mathbf{ny}, \mathbf{no}\}) = \text{ATTR}(\mathbf{ny}) \cap \text{ATTR}(\mathbf{no})$. Thus $\text{label}_{l+1}(\{\mathbf{nx}, \mathbf{ny}\}) = \text{label}_{l+1}(\{\mathbf{nx}, \mathbf{no}\}) \cap \text{label}_{l+1}(\{\mathbf{ny}, \mathbf{no}\}) = \text{ATTR}(\mathbf{nx}) \cap \text{ATTR}(\mathbf{ny}) \cap \text{ATTR}(\mathbf{no})$. However, $\text{ATTR}(\mathbf{no}) = \text{ATTR}(\mathbf{no}')$ and we have already shown that $\text{ATTR}(\mathbf{no}') \supseteq \text{ATTR}(\mathbf{nx}) \cap \text{ATTR}(\mathbf{ny})$. Consequently, $\text{label}_{l+1}(\{\mathbf{nx}, \mathbf{ny}\}) = \text{ATTR}(\mathbf{nx}) \cap \text{ATTR}(\mathbf{ny})$ as claimed. \square

4.5. Conclusion of proof. The proof began by assuming that q has a full reducer SJ . Given this assumption § 4.1 proved that a semijoin expression \mathbf{E} equivalent to $q[\mathbf{R}_1]$ must exist. Section 4.2 proved that $\text{Parse}(\mathbf{E})$ must contain at least one essential leaf for each $\mathbf{R}_i \in \mathbf{D}$, and § 4.3 proved that these essential leaves are “U-connected”; if \mathbf{no}_i and \mathbf{no}_j are essential for \mathbf{R}_i and \mathbf{R}_j , respectively, then $\text{label}(\{\mathbf{no}_i, \mathbf{no}_j\}) = \mathbf{R}_i \cap \mathbf{R}_j$. Finally § 4.4 proved that $\text{Parse}(\mathbf{E})$ can be transformed into a tree qual graph that represents q . Thus we have proved the following theorem.

THEOREM 2. *If q has a full reducer, then q is a tree query.*

5. Tree query membership algorithm. Theorems 1 and 2 prove that an NJ query has a full reducer iff it is a tree query. For this result to be useful we need an efficient algorithm that tests whether an NJ query is a tree query, and constructs a full reducer for the query if it is a tree query. An algorithm that solves this problem for single attribute semijoins is presented in [BC]. Algorithms for the natural semijoin case appear in [BG1], [YO], [YO1]. In this section we present a refined version of the algorithm described in [BG1].

Let q be the NJ query over \mathbf{D} . For each $A \in \mathbf{U}$, let $\text{Weight}(A) = |\text{Class}(A)| - 1 =$ the number of edges in a spanning tree for $\text{Class}(A)$. Also let $\text{Weight}(q) = \sum_{A \in \mathbf{U}} \text{Weight}(A)$. Let \mathbf{QG} be any qual graph over \mathbf{D} . For each edge $\{\mathbf{R}_i, \mathbf{R}_j\} \in \mathbf{QG}$, $\text{Weight}(\mathbf{QG}, \{\mathbf{R}_i, \mathbf{R}_j\}) = |\mathbf{R}_i \cap \mathbf{R}_j|$; $\text{Weight}(\mathbf{QG}) = \sum_{E \in \mathbf{QG}} \text{Weight}(\mathbf{QG}, E)$.

LEMMA 5.1. *Let \mathbf{TQG} be a tree qual graph over \mathbf{D} . Then*

- (i) $\text{Weight}(\mathbf{TQG}) \leq \text{Weight}(q)$; and
- (ii) $\text{Weight}(\mathbf{TQG}) = \text{Weight}(q)$ iff \mathbf{TQG} represents q .

Proof. For each $A \in \mathbf{U}$, let $E(A) = \{\{\mathbf{R}_i, \mathbf{R}_j\} \in \mathbf{TQG} \mid A \in \mathbf{R}_i \cap \mathbf{R}_j\}$, let $G(A)$ be the subgraph of \mathbf{TQG} whose nodeset is $\text{Class}(A)$ and whose edgeset is $E(A)$, and let $\text{Weight}(\mathbf{TQG}, A) = |E(A)|$. Observe that $\text{Weight}(\mathbf{TQG}) = \sum_{A \in \mathbf{U}} \text{Weight}(\mathbf{TQG}, A)$.

(i) For each $A \in \mathbf{U}$, $|E(A)| \leq |\text{Class}(A)| - 1$, since $G(A)$ is acyclic; i.e., $\text{Weight}(\mathbf{TQG}, A) \leq \text{Weight}(A)$. $\text{Weight}(\mathbf{TQG}) \leq \text{Weight}(q)$ follows, since $\text{Weight}(\mathbf{TQG}) = \sum_{A \in \mathbf{U}} \text{Weight}(\mathbf{TQG}, A) \leq \sum_{A \in \mathbf{U}} \text{Weight}(A) = \text{Weight}(q)$.

(ii) If \mathbf{TQG} represents q , it is \mathbf{U} -connected, by definition. This means that $G(A)$ is a connected graph for all $A \in \mathbf{U}$, and for $G(A)$ to remain acyclic, $|E(A)| = |\text{Class}(A)| - 1$; i.e., $\text{Weight}(\mathbf{TQG}, A) = \text{Weight}(A)$. Thus $\text{Weight}(\mathbf{TQG}) = \sum_{A \in \mathbf{U}} \text{Weight}(\mathbf{TQG}, A) = \sum_{A \in \mathbf{U}} \text{Weight}(A) = \text{Weight}(q)$.

If \mathbf{TQG} does not represent q , it is not \mathbf{U} -connected and, for some $A' \in \mathbf{U}$, $G(A')$ is not a connected graph. For $G(A')$ to remain acyclic, $|E(A')| < |\text{Class}(A')| - 1$ is required, i.e., $\text{Weight}(\mathbf{TQG}, A') < \text{Weight}(A')$. Therefore $\text{Weight}(\mathbf{TQG}) = [\sum_{A \in \mathbf{U} - \{A'\}} \text{Weight}(\mathbf{TQG}, A) + \text{Weight}(\mathbf{TQG}, A')] < [\sum_{A \in \mathbf{U} - \{A'\}} \text{Weight}(A) + \text{Weight}(A')] = \text{Weight}(q)$. I.e., $\text{Weight}(\mathbf{TQG}) < \text{Weight}(q)$ as claimed. \square

Lemma 5.1 suggests the following tree query membership algorithm.

ALGORITHM TQ.

Input: data schema \mathbf{D} .

Output: tree qual graph \mathbf{TQG} that represents the NJ query q over \mathbf{D} , if one exists; else FALSE.

1. Let \mathbf{QG}^+ be a complete graph over \mathbf{D} —i.e., the edgeset of \mathbf{QG}^+ is $\{\{\mathbf{R}_i, \mathbf{R}_j\} \mid \mathbf{R}_i, \mathbf{R}_j \in \mathbf{D} \wedge i \neq j\}$.
 2. Let \mathbf{TQG} be a maximal weight spanning tree of \mathbf{QG}^+ .
 3. If $\text{Weight}(\mathbf{TQG}) = \text{Weight}(q)$, then output \mathbf{TQG} , else output FALSE.
- end.

The correctness of Algorithm TQ follows immediately from Lemma 5.1. The algorithm can be implemented with $O(m \cdot n^2)$ time complexity, where $m = |\mathbf{U}|$ and $n = |\mathbf{D}|$. Each $\mathbf{R}_i \in \mathbf{D}$ is represented by a bit vector AT_i of length m such that $AT_i[k] = 1$ iff $A_k \in \mathbf{R}_i$. This representation can be constructed in $O(m \cdot n^2)$ time. The weight of all edges in \mathbf{QG}^+ can also be constructed in $O(m \cdot n^2)$ time. The weight of all edges in \mathbf{QG}^+ can also be computed in $O(m \cdot n^2)$ time by the following loop:

for $1 \leq i < j \leq n$ do

$$\begin{aligned} \text{Weight}(\mathbf{QG}^+, \{\mathbf{R}_i, \mathbf{R}_j\}) &= \text{the number of 1-bits in } AT_i \wedge AT_j \\ &= \sum_{k=1}^m AT_i[k] \wedge AT_j[k]. \end{aligned}$$

end.

These weights are used as input to Prim's maximal spanning tree algorithm [Prim], which has $O(n^2)$ complexity. $\text{Weight}(\mathbf{TQG})$ can be calculated in linear time and $\text{Weight}(q)$ can be determined in $O(m \cdot n)$ time. Overall, then, the complexity of the implementation is $O(m \cdot n^2)$, as claimed.

6. Conclusions and open problems. We have proven that an NJ query has a full reducer composed of natural semijoins iff it is tree query, and we have presented an efficient tree query membership algorithm. Thus, the significance of tree queries established by [BC] for single attribute semijoins has been extended to the general case.

The following open problems remain.

1. Let \mathbf{D} be a database schema whose NJ query q is cyclic. We have proven that for all semijoin sequences \mathbf{SJ} , there exists a state D^{bad} , such that \mathbf{SJ} does not fully reduce D^{bad} . The result in [BC] is tighter; they prove there exists a state D^{bad} , such that for all \mathbf{SJ} , \mathbf{SJ} does not fully reduce D^{bad} . The tighter result for the general case remains open.⁵

⁵ Note added in proof. The tighter result has recently been proved in [BFMMUY], [GS].

2. Another research direction is to consider more general queries, e.g., arbitrary expressions over project and natural join. Since equivalence and minimization are NP-complete problems for this class of queries [ASU], [CM], we expect full reducer questions to be difficult as well.

3. It is also interesting to study queries which permit “inequality joins”. In [BG2], we consider queries over $<$, \cong and $=$; we characterize the subclass that have full reducers composed of single attribute semijoins and present an efficient membership algorithm for this subclass. Interestingly, some queries that have full reducers are *not* tree queries. In [YO1], a tree query membership algorithm is presented for queries over $<$, \cong and $=$ with “multiattribute” semijoins. An open problem is to characterize the inequality join queries that have full reducers composed of multiattribute semijoins.

4. We have only studied the *existence* of full reducers and have not concerned ourselves with their *efficiency*. In a system context, the latter problem is critically important. Heuristic algorithms that construct efficient sequences of semijoins for arbitrary NJ queries are reported in [BGWRR], [HY]. Algorithms that construct optimal sequences for special types of tree queries are described by [CH], [GD], [HY]. The optimization problem for arbitrary queries remains open.

To date, semijoins have been recommended for distributed database systems and database machines. We conjecture that semijoins and the concept of “reduction” can also be valuable tactics in conventional database implementations. We see this issue as a step toward an integrated theory of relational query processing.

Appendix A. Mapping equijoin queries into natural join queries. Syntactically, an *equijoin query* consists of a *qualification* which is a conjunction of clauses of the form $\mathbf{R}_i \cdot A = \mathbf{R}_j \cdot A'$, where $A \in \mathbf{R}_i$ and $A' \in \mathbf{R}_j$. Semantically, an equijoin query with qualification \mathbf{Q} maps database state D into $\{\langle r_1, \dots, r_n \rangle \in \mathbf{R}_1(D) \times \dots \times \mathbf{R}_n(D) \mid \mathbf{Q}(\langle r_1, \dots, r_n \rangle)$ is true under the substitution $\mathbf{R}_i \cdot A = r_i[A]$, etc.}. The *natural qualification* over scheme \mathbf{D} is $\bigwedge_{\mathbf{R}_i, \mathbf{R}_j \in \mathbf{D}} \bigwedge_{A \in \mathbf{R}_i \cap \mathbf{R}_j} \mathbf{R}_i \cdot A = \mathbf{R}_j \cdot A$. The query with this qualification is called the *natural query* over \mathbf{D} .

LEMMA A. Let q be the natural query over \mathbf{D} and q' the NJ query over \mathbf{D} . For all states D , $q(D) \cong q'(D)$ (\cong denotes isomorphism).

Sketch of proof. Let f map tuples $\langle r_1, \dots, r_n \rangle$ into elements of $\text{dom}(\mathbf{U})$, where $f(\langle r_1, \dots, r_n \rangle) = r$ such that $r[\mathbf{R}_i] = r_i$ for $i = 1, \dots, n$. f is well defined for all $\langle r_1, \dots, r_n \rangle \in q(D)$ and is the desired isomorphism. \square

We can map any qualification into a natural qualification by renaming attributes in the qualification and the database schema. There are three steps. Let \mathbf{Q} be a qualification over \mathbf{D} .

1. Rename attributes \mathbf{Q} and \mathbf{D} so that all relation schemas are disjoint.
2. Close \mathbf{Q} under transitivity of equality.
3. While \mathbf{Q} contains a clause $\mathbf{R}_i \cdot A = \mathbf{R}_j \cdot A'$ where $A \neq A'$:
 - 3.1 replace A' by A in \mathbf{R}_j ;
 - 3.2 replace $\mathbf{R}_j \cdot A'$ by $\mathbf{R}_j \cdot A$ wherever it appears in \mathbf{Q} ;
 - 3.3 if $i = j$, restrict the state of \mathbf{R}_i by the clause.

It is easy to prove that the renamed qualification is natural over the renamed database schema, and the renamed qualification is isomorphic to the original one up to attribute names [BG]. Since natural queries are isomorphic to NJ queries by Lemma A, our main result follows.

PROPOSITION A. For every equijoin query there exists an isomorphic NJ query (over a possibly different database schema).

Appendix B. Natural join queries and qual graphs. The relationship between NJ queries and qual graphs is developed fully in [BG]. This relationship is mediated by a subclass of the equijoin queries called *subnatural queries*. A subnatural query is one whose qualification has the form $\bigwedge_{\mathbf{R}_i, \mathbf{R}_j \in \mathbf{D}} \bigwedge_{A \in X \subseteq \mathbf{R}_i \cap \mathbf{R}_j} \mathbf{R}_i \cdot A = \mathbf{R}_j \cdot A$. To represent subnatural queries, qual graphs are extended to include edge labels, with $\text{label}(\{\mathbf{R}_i, \mathbf{R}_j\}) \subseteq \mathbf{R}_i \cap \mathbf{R}_j$. A labeled qual graph **QG** represents the query with qualification

$$\bigwedge_{\substack{\text{all edges} \\ \{\mathbf{R}_i, \mathbf{R}_j\} \text{ in } \mathbf{QG}}} \bigwedge_{A \in \text{label}(\{\mathbf{R}_i, \mathbf{R}_j\})} \mathbf{R}_i \cdot A = \mathbf{R}_j \cdot A.$$

A labeled qual graph is *A-connected* if for all $\mathbf{R}_i, \mathbf{R}_j \in \text{Class}(A)$, the graph contains a path P from \mathbf{R}_i to \mathbf{R}_j such that $A \in \bigcap_{\text{all edges } E \text{ in } P} \text{label}(E)$. A labeled qual graph is *U-connected* if it is *A-connected* for all $A \in \mathbf{U}$. Labeled *U-connectivity* implies unlabeled *U-connectivity* since $\text{label}(\{\mathbf{R}_i, \mathbf{R}_j\}) \subseteq \mathbf{R}_i \cap \mathbf{R}_j$ for all edges. Conversely, every *U-connected* unlabeled qual graph can be transformed into a *U-connected* labeled qual graph by assigning $\text{label}(\{\mathbf{R}_i, \mathbf{R}_j\}) = \mathbf{R}_i \cap \mathbf{R}_j$ for all edges $\{\mathbf{R}_i, \mathbf{R}_j\}$.

In § 2 we said that an unlabeled qual graph represents an NJ query iff the graph is *U-connected*. In this appendix we justify the definition by proving that a labeled qual graph represents a natural query iff the labeled graph is *U-connected*. It follows that an unlabeled qual graph represents an NJ query iff the same qual graph with maximal labels represents a natural query; and by Lemma A, the two queries are isomorphic.

LEMMA B. *Let q be the query represented by **QG**. $q \Rightarrow (\mathbf{R}_i \cdot A = \mathbf{R}_j \cdot A)$ iff **QG** contains a path P from \mathbf{R}_i to \mathbf{R}_j such that $A \in \bigcap_{\text{all edges } E \text{ in } P} \text{label}(E)$.*

Sketch of proof. $q \Rightarrow (\mathbf{R}_i \cdot A = \mathbf{R}_j \cdot A)$ by *transitivity of equality* iff P exists, and transitivity of equality is a sound and complete inference rule for equijoin queries. \square .

PROPOSITION B. ***QG** represents a natural query iff it is *U-connected*.*

Sketch of proof. Let \mathbf{QG}^+ be the \cup/\cap transitive closure of **QG**, i.e., \mathbf{QG}^+ is a complete graph over the nodeset of **QG**, and for all $\mathbf{R}_i, \mathbf{R}_j \in \mathbf{D}$; the label of $\{\mathbf{R}_i, \mathbf{R}_j\}$ is

$$\text{label}^+(\{\mathbf{R}_i, \mathbf{R}_j\}) = \bigcup_{\substack{\text{all paths, } p \\ \text{in } \mathbf{QG}^+ \text{ from } \\ \mathbf{R}_i \text{ to } \mathbf{R}_j}} \bigcap_{\substack{\text{all edges} \\ E \text{ in } p}} \text{label}(E).$$

By Lemma B, \mathbf{QG}^+ represents the same query q as **QG**, and $q \Rightarrow (\mathbf{R}_i \cdot A = \mathbf{R}_j \cdot A)$ iff $A \in \text{label}^+(\{\mathbf{R}_i, \mathbf{R}_j\})$. By definition, q is a natural query iff $q \Rightarrow (\mathbf{R}_i \cdot A = \mathbf{R}_j \cdot A)$ for all $A \in \mathbf{U}$ and all $\mathbf{R}_i, \mathbf{R}_j \in \text{Class}(A)$, and so q is natural iff $A \in \text{label}^+(\{\mathbf{R}_i, \mathbf{R}_j\})$ for all $A \in \mathbf{U}$ and all $\mathbf{R}_i, \mathbf{R}_j \in \text{Class}(A)$. The latter property holds iff **QG** is *U-connected*. \square

Appendix C. Queries with target lists. *Target-lists* are a mechanism for specifying a query followed by a projection. Target lists or some comparable facility are supported in virtually every relational query language. In this appendix we show that target lists do not change our main results by showing that a query with a target list has a full reducer iff the query (without the target list) is a tree query.

Let q be the NJ query over \mathbf{D} and let $\mathbf{t} \subseteq \mathbf{U}$, $\mathbf{t} \neq \{\}$. Let D be any database state. A *full reduction* of D with respect to \mathbf{t} is D' such that $(\mathbf{R}_i(D'))[\mathbf{R}_i \cap \mathbf{t}] = (q(D))[\mathbf{R}_i \cap \mathbf{t}]$ for $i = 1, \dots, n$. A semijoin program **SJ** is a *full reducer* for $q[\mathbf{t}]$ if for all states D , **SJ**(D) is a full reduction of D with respect to \mathbf{t} .

THEOREM C. *$q[\mathbf{t}]$ has a full reducer iff q is a tree query.*

Sketch of proof. If q is a tree query it has a full reducer by Theorem 1, and any full reducer for q is obviously a full reducer for $q[\mathbf{t}]$.

To prove the converse let \mathbf{R}_i be any relation schema such that $\mathbf{R}_i \cap \mathbf{t} \neq Q$. Let us augment \mathbf{D} by adding the relation schema $\mathbf{R}'_i = \mathbf{R}_i \cap \mathbf{t}$, let \mathbf{D}' be the resulting database schema, and let q' be the NJ query over \mathbf{D}' .

Let $\mathbf{S}\mathbf{J}$ be any full reducer for $q[\mathbf{t}]$, and let $\mathbf{S}\mathbf{J}' = (\mathbf{R}_i < \mathbf{R}'_i) \cdot \mathbf{S}\mathbf{J} \cdot (\mathbf{R}'_i < \mathbf{R}_i)$. Since $\mathbf{S}\mathbf{J}$ fully reduces $\mathbf{R}_i[\mathbf{R}'_i]$ for all states D of \mathbf{D} , $\mathbf{S}\mathbf{J}'$ fully reduces \mathbf{R}'_i for all states D' of \mathbf{D}' . That is, for all states D' of \mathbf{D}' , $\mathbf{R}'_i(\mathbf{S}\mathbf{J}'(D')) = (q'(D'))[\mathbf{R}'_i]$. Also, by Lemma 4.1, there exists a semijoin expression \mathbf{E} such that for all states D' , $\mathbf{E}(D') = \mathbf{R}'_i(\mathbf{S}\mathbf{J}'(D'))$. It follows that $\mathbf{E} \equiv q'[\mathbf{R}'_i]$. But if such an \mathbf{E} exists, §§ 4.2–4.4 prove that q' is a tree query. The theorem follows since if q' is a tree query, so is q . \square

Acknowledgments. We thank the referees and the editor (J. D. Ullman) for improving the presentation of this paper. We are indebted to Renate D'Arcangelo for expert preparation of the manuscript.

REFERENCES

- [ABU] A. V. AHO, C. BEERI AND J. D. ULLMAN, *Theory of joins in relational databases*, ACM Trans. Database Syst., 4 (1979), pp. 297–314.
- [ASU] A. V. AHO, Y. SAGIV AND J. D. ULLMAN, *Equivalence of relational expressions*, this Journal, 8 (1979), pp. 218–246.
- [ASU1] ———, *Efficient optimization of a class of relational expressions*, ACM Trans. Database Syst., 4 (1979), pp. 435–454.
- [Babb] E. BABB, *Implementing a relational database by means of specialized hardware*, ACM Trans. Database Syst., 4 (1979), pp. 1–29.
- [BC] P. A. BERNSTEIN AND D. W. CHIU, *Using semi-joins to solve relational queries*, J. Assoc. Comput. Mach., 28 (1981), pp. 25–40.
- [BFMMUY] C. BEERI, R. FAGIN, D. MAIER, A. MENDELZON, J. ULLMAN AND M. YANNAKAKIS, *Properties of acyclic database schemes*, Proc. 13th ACM Symposium on Theory of Computing, May 1981, pp. 355–362.
- [BG] P. A. BERNSTEIN AND N. GOODMAN, *The theory of semi-joins*, Tech. Rep. CCA-79-27, Computer Corp. of America, 1979.
- [BG1] ———, *Full reducers for relational queries using multi-attribute semi-joins*, Proc. Comp. Network Symp., IEEE Comp. Society, 1979.
- [BG2] ———, *Power of inequality semijoins*, Information Systems, to appear.
- [BGWRR] P. A. BERNSTEIN, N. GOODMAN, E. WONG, C. L. REVE AND J. B. ROTHNIE, JR., *Query Processing in SDD-1*, ACM Trans. Database Syst., to appear.
- [CH] D. W. CHIU AND Y. C. HO, *A methodology for interpreting tree queries into optimal semi-join expressions*, Proc. ACM-SIGMOD Conference, (1980), pp. 169–178.
- [Codd] E. F. CODD, *A relational model of data for large shared data banks*, Comm. ACM, (1970), pp. 377–287.
- [CM] A. K. CHANDRA AND P. M. MERLIN, *Optimal implementation of conjunctive queries in relational databases*, Proc. 9th ACM Symposium on Theory of Computation, 1976, pp. 47–61.
- [ESW] R. EPSTEIN, M. STONEBRAKER AND E. WONG, *Distributed query processing in a relational database system*, Proc. ACM SIGMOD Conference, 1978, pp. 169–180.
- [Gotlieb] L. GOTLIEB, *Computing joins of relations*, Proc. ACM SIGMOD Conference, 1975, pp. 55–63.
- [GD] M. G. GOUDA AND U. DAYAL, *Optimal semijoin schedules for query processing in local distributed database systems*, Proc. ACM-SIGMOD Conf., April 1981, pp. 164–175.
- [GS] N. GOODMAN AND O. SHMUELI, *Database state reductions for tree and cyclic schemas*, ACM Trans. Database Syst., to appear.
- [HY] A. R. HEVNER AND S. B. YAO, *Query processing in distributed database systems*, IEEE Trans. Software Engng., SE-5 (1979), pp. 177–187.
- [MMS] D. MAIER, A. O. MENDELZON AND Y. SAGIV, *Testing implications of data dependencies*, ACM Trans. Database Syst., 4 (1979), pp. 455–469.
- [OSS] E. A. OZKARAHAN, S. A. SCHUSTER AND K. C. SEVCIK, *Performance evaluation of a relational associative processor*, ACM Trans. Database Syst., 2 (1977), pp. 175–196.
- [Percherer] R. M. PERCHERER, *Efficient evaluation of expressions in relational algebra*, Proc. ACM Pacific Conference, 1975, pp. 44–49.

- [Prim] R. C. PRIM, *Shortest connection networks and some generalizations*, Bell System Tech. J., (1957), pp. 1389–1401.
- [RBFGB] J. B. ROTHNIE, JR., P. A. BERNSTEIN, S. A. FOX, N. GOODMAN, M. M. HAMMER, T. A. LANDERS, C. L. REEVE, D. W. SHIPMAN AND E. WONG, *SDD-1: A system for distributed databases*, ACM Trans. Database Syst., 5 (1980), pp. 1–17.
- [Rothnie] J. B. ROTHNIE, JR., *Evaluation inter-entry retrieval expressions in a relational database management system*, Proc. AFIPS NCC, 44 (1975), pp. 417–423.
- [SACLP] P. G. SELINGER, M. M. ASTRAHAN, D. D. CHAMBERLIN, R. A. LORIE AND T. G. PRICE, *Access path selection in a relational database management system*, Proc. ACM-SIGMOD Conference, (1979), pp. 23–34.
- [SC] J. M. SMITH AND P. Y. T. CHANG, *Optimizing the performance of a relational algebra database interface*, Comm. ACM, 18 (1975), pp. 568–579.
- [SE] S. Y. W. SU AND A. EMAM, *CASDAL: CASSM's Data language*, ACM Trans. Database Syst., 3 (1978), pp. 57–91.
- [WY] E. WONG AND K. YOUSSEFI, *Decomposition—A strategy for query processing*, ACM Trans. Database Syst., 1 (1976), pp. 223–241.
- [Yao] S. B. YAO, *Optimization of query evaluation algorithms*, Ibid., 4 (1979), pp. 133–155.
- [YO] C. T. YU AND M. Z. OZSOYOGLU, *An algorithm for tree-query membership of a distilled query*, Proc. Compsac 79, IEEE Comp. Society, 1979, pp. 306–312.
- [YO1] ———. *On determining tree query membership of a distributed query*, Tech. Rep. TR80-1, Dept. of Computing Science, U. of Alberta, 1980.

MINIMIZING THE NUMBER OF EVALUATION PASSES FOR ATTRIBUTE GRAMMARS*

KARI-JOUKO RÄIHÄ† AND ESKO UKKONEN‡

Abstract. The problem of constructing multi-pass evaluators for attribute grammars is studied. We show that the construction algorithm used heretofore can in the worst case produce evaluators which perform $2n - 1$ passes over the parse tree, where n is the minimum number of passes required. Furthermore, the problem of constructing an optimal evaluation order is shown to be NP-complete. We then develop a new characterization for attribute grammars evaluable in passes. It can be directly applied as an efficient membership test. Finally, the characterization is used for deriving a polynomial time construction algorithm for a large subclass of pass-oriented attribute grammars. The subclass is argued to be of practical importance.

Key words. attribute grammars, multi-pass evaluators, NP-completeness, approximation algorithms

1. Introduction. Attribute grammars were introduced by Knuth [10] for describing the semantics of context-free languages. Just as it is possible to automatically generate parsers for context-free grammars, methods have been devised which can be used to generate so-called semantic evaluators for attribute grammars. A review of various evaluation techniques and of the corresponding construction algorithms is given in [3].

In pass-oriented evaluators attributes are evaluated during one or more depth-first traversals (called *passes*) of the parse tree. Such evaluators have attracted particular attention because of the natural way in which the evaluation passes model the compilation passes in a conventional hand-written compiler. One-pass evaluators have been studied in [11]. In this paper we concentrate on the construction of efficient multi-pass evaluators for grammars which cannot be evaluated in a single pass. Although it has been shown that in principle all translations can be described using a one-pass grammar [10], there are simple examples of features which are more conveniently described using multi-pass grammars [4].

The declarative nature of attribute grammars is partly due to the fact that the evaluation order of attributes is not explicitly given in the grammar. Thus it is the task of the construction algorithm to assign the attributes to evaluation passes. The first such algorithm was proposed by Bochmann [2]. He allows only left-to-right evaluation passes. This restriction yields a simple construction algorithm which is easily seen to produce evaluators performing a minimum number of passes.

Optimizing compilers also contain algorithms which work backwards from the end of the program to its start, i.e., algorithms which are most conveniently described using attributes which require a right-to-left evaluation pass (see, e.g., [6]). As a generalization of Bochmann's evaluators, Jazayeri [5] suggested that right-to-left passes should alternate with left-to-right ones. Evaluators that may perform both left-to-right and right-to-left passes are called alternating semantic evaluators. Clearly, the class of grammars suitable for such evaluators (called ASE grammars) strictly contains the grammars accepted by Bochmann's method.

It was soon realized [9] that the approach of [5] is wasteful, since some of the passes can be merged with neighboring ones. Based on this observation, a new form

* Received by the editors December 19, 1979, and in final form November 20, 1980.

† Department of Computer Science, University of Helsinki, Helsinki, Finland. The work of this author was supported by the Academy of Finland.

‡ Department of Computer Science, University of Helsinki, Helsinki, Finland. Now at University of California, Computer Science Division, Berkeley, California 94720.

of the construction algorithm was developed in [14] and [15]. This algorithm is given in Fig. 1. We assume that the reader is familiar with attribute grammars as defined in [10]. Let A denote the set of attributes in the grammar. The algorithm computes the number of passes, denoted by m_{ASE} . Furthermore, for each pass p from 1 to m_{ASE} it defines the following entities:

$$d_p = \begin{cases} L, & \text{indicating a left-to-right pass,} \\ R, & \text{indicating a right-to-left pass,} \end{cases}$$

A_p = the set of attributes evaluated during the p th pass.

The string $d_1 d_2 \cdots d_{m_{ASE}}$ is called an *evaluation order* for the attribute grammar.

```

algorithm ASECONSTRUCTION;
begin
   $p := 0; B := A;$ 
  repeat
     $p := p + 1;$ 
     $A_L := \text{NEXTSET}(B, L);$ 
     $A_R := \text{NEXTSET}(B, R);$ 
    if  $A_R \subset A_L$  then  $(d_p, A_p) := (L, A_L)$  else
    if  $A_L \subset A_R$  then  $(d_p, A_p) := (R, A_R)$  else
       $(d_p, A_p) := \text{SELECT}(p, A_L, A_R)$  fi fi;
     $B := B - A_p$ 
  until  $B = \emptyset \vee A_L = A_R = \emptyset;$ 
  if  $B = \emptyset$ 
    then  $m_{ASE} := p$ 
    else the grammar cannot be evaluated using the ASE technique
  fi
end of ASECONSTRUCTION;

```

FIG. 1

The main algorithm in Fig. 1 uses the subalgorithm $\text{NEXTSET}(B, d)$ for computing for the set of remaining attributes B the subset which can be evaluated during the next pass in case the direction chosen is that indicated by d . NEXTSET is based on repeatedly inspecting the semantic rules and deleting from the attribute set those attributes which present conflicts with the evaluation direction under consideration. Similar versions of the NEXTSET algorithm are given in [13]–[15] and will not be repeated here.

More interesting is the way the direction for the next pass, d_p , is chosen. If one of A_R and A_L is contained in the other, the direction for the larger set is chosen; this choice is clearly optimal. If the sets are incommensurate, the decision procedure SELECT is used. In this case both [14] and [15] proposed to let the evaluator alternate its direction as originally suggested in [5]. This means that SELECT is specified as in Fig. 2.

```

algorithm SELECT( $p, A_L, A_R$ );
begin
  if  $p = 1$  then return  $(L, A_L)$  else
  if  $d_{p-1} = R$  then return  $(L, A_L)$  else return  $(R, A_R)$  fi fi;
end of SELECT;

```

FIG. 2

Although Bochmann’s algorithm always produces left-to-right evaluators which require a minimum number of passes, the possibility of choice introduced by right-to-left passes has the effect that m_{ASE} is not necessarily minimal. Let m_{opt} denote the minimum number of passes required when the attributes are assigned to passes in the best possible manner. In § 2 we give a grammar for which $m_{ASE} = 2m_{opt} - 1$. This situation is the worst possible: we will also show that for any grammar $m_{ASE} \leq 2m_{opt} - 1$, when SELECT is specified as above.

It would be desirable to be able to specify SELECT so that no more than m_{opt} passes are ever produced. However, in § 3 we will show that this problem is NP-complete. If we wish to achieve optimality in the general case, we have to abandon efficiency.

In order to develop a polynomial approximation algorithm we first derive a new characterization for multi-pass attribute grammars in § 4. This characterization is then used in § 5 as the basis of a new construction algorithm which, although nonoptimal in the general case, produces optimal evaluators for a larger class of grammars than the algorithm of Fig. 1.

2. Nonoptimality of the selection algorithm. We say that a construction algorithm is *optimal* if it always produces evaluators which use only m_{opt} passes. This definition, though but one of several possibilities, is natural: in traditional implementations of multi-pass evaluators (e.g. [16]) a decrease in the number of passes results in savings in the overhead caused by traversing the tree. Even in approaches where the number of passes is not of equal importance [7], the optimality of the construction algorithm in the above sense decreases the lifetime of attributes, which is good for storage management [8], [13].

Nonterminal	Inherited attributes	Synthesized attributes
Z	—	—
X_1	a_1, b_1, c_1, d_1	p_1, q_1, r_1
$X_i, i = 2, \dots, n - 1$	a_i, b_i, c_i	p_i, q_i, r_i
Productions with semantic rules		
1° $Z \rightarrow X_1^L X_1^R$	$a_1(X_1^L) \leftarrow \text{constant}$ $b_1(X_1^L) \leftarrow \text{constant}$ $c_1(X_1^L) \leftarrow p_1(X_1^R)$ $d_1(X_1^L) \leftarrow \text{constant}$	$a_1(X_1^R) \leftarrow q_1(X_1^L)$ $b_1(X_1^R) \leftarrow r_1(X_1^L)$ $c_1(X_1^R) \leftarrow \text{constant}$ $d_1(X_1^R) \leftarrow p_1(X_1^L)$
2° $X_i \rightarrow X_{i+1}^L X_{i+1}^R,$ $i = 1, \dots, n - 3$	$p_i(X_i) \leftarrow b_i(X_i) + c_i(X_i)$ $q_i(X_i) \leftarrow a_i(X_i) + p_{i+1}(X_{i+1}^L)$ $r_i(X_i) \leftarrow p_{i+1}(X_{i+1}^L)$ $a_{i+1}(X_{i+1}^L) \leftarrow \text{constant}$ $b_{i+1}(X_{i+1}^L) \leftarrow \text{constant}$ $c_{i+1}(X_{i+1}^L) \leftarrow p_{i+1}(X_{i+1}^R)$ $a_{i+1}(X_{i+1}^R) \leftarrow q_{i+1}(X_{i+1}^L)$ $b_{i+1}(X_{i+1}^R) \leftarrow r_{i+1}(X_{i+1}^L)$ $c_{i+1}(X_{i+1}^R) \leftarrow \text{constant}$	
3° $X_{n-2} \rightarrow X_{n-1}^L X_{n-1}^R$	like 2°, with the following exception: $b_{n-1}(X_{n-1}^R) \leftarrow \text{constant}$	
4° $X_{n-1} \rightarrow x$	$p_{n-1}(X_{n-1}) \leftarrow b_{n-1}(X_{n-1}) + c_{n-1}(X_{n-1})$ $q_{n-1}(X_{n-1}) \leftarrow a_{n-1}(X_{n-1})$ $r_{n-1}(X_{n-1}) \leftarrow \text{constant}$	

FIG. 3

We demonstrate the behavior of the construction algorithm using an example grammar given in Fig. 3. Here x denotes an arbitrary terminal symbol. The superscripts L and R are used to distinguish two occurrences of the same nonterminal in a production.

This grammar generates exactly one attributed parse tree, fragments of which are shown in Fig. 4. Inherited attributes are shown on the left and synthesized attributes on the right of each nonterminal. Attributes with no entering dependency arcs are constants.

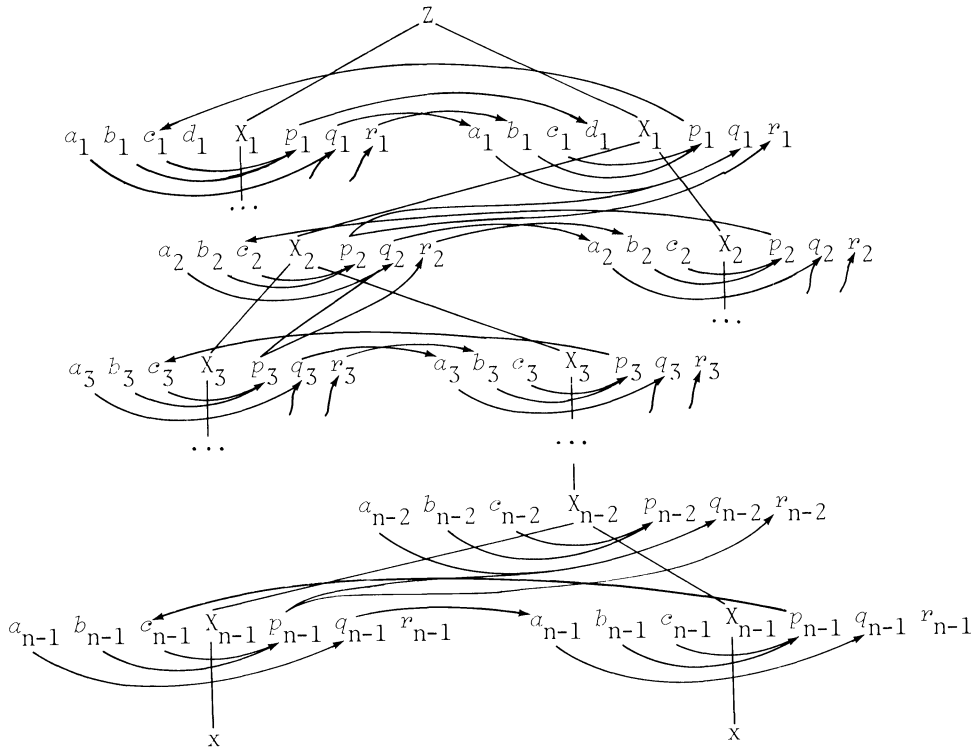


FIG. 4

The evaluation order produced by ASECONSTRUCTION is given in Fig. 5.

By inspecting the dependencies in Fig. 4 we note that none of the attributes a_i, q_i ($i = 1, \dots, n-1$) contributes in the evaluation of the other attributes. Consequently, also the evaluation order in Fig. 6 could be used.

Thus we see that for the grammar in Fig. 3, $m_{ASE} \cong 2m_{opt} - 1$.

pass	direction d_p	attributes to be evaluated
1	L	$a_{n-1}, b_{n-1}, q_{n-1}, r_{n-1}$
2	R	$c_{n-1}, p_{n-1}, r_{n-2}$
3	L	$a_{n-2}, b_{n-2}, q_{n-2}$
4	R	$c_{n-2}, p_{n-2}, r_{n-3}$
5	L	$a_{n-3}, b_{n-3}, q_{n-3}$
\vdots	\vdots	\vdots
$2n-3$	L	a_1, b_1, q_1
$2n-2$	R	c_1, p_1
$2n-1$	L	d_1

FIG. 5.

pass	direction d_p	attributes to be evaluated
1	R	$b_{n-1}, c_{n-1}, p_{n-1}, r_{n-1}, r_{n-2}$
2	R	$b_{n-2}, c_{n-2}, p_{n-2}, r_{n-3}$
3	R	$b_{n-3}, c_{n-3}, p_{n-3}, r_{n-4}$
\vdots	\vdots	\vdots
$n-1$	R	b_1, c_1, p_1
n	L	$a_i, q_i (i = 1, \dots, n-1), d_1$

FIG. 6

The basic reason for the nonoptimality of ASECONSTRUCTION is that we do not know which direction to choose when A_L and A_R are incommensurate. The SELECT algorithm follows the heuristics that the direction of the next pass is opposite to the direction of the preceding pass. This decision has the consequence (proved in Lemma 1) that when an attribute becomes ready for evaluation, its evaluation is not delayed for more than one pass. Thus SELECT tries to take new attributes into the evaluation process as soon as possible hoping that this would make other attributes evaluable.

Let $ASE(p) = \cup_{i=1}^p A_i$ when $p \leq m_{ASE}$, and $ASE(p) = A$ for $p > m_{ASE}$. We have

LEMMA 1. Let A_L and A_R be the sets computed for pass p in ASECONSTRUCTION. Then $A_L \cup A_R \subset ASE(p+1)$.

Proof. Suppose that we choose A_L as A_p (the other case is symmetric). Thus $A_L \subset ASE(p) \subset ASE(p+1)$. Let A'_L and A'_R be the sets computed for determining the direction of pass $p+1$. Clearly, $A_R - A_L \subset A'_R$. If $A'_R \subset A'_L$ then $A_{p+1} = A'_L$, otherwise $A_{p+1} = A'_R$. In either case, $A_R = (A_R \cap A_L) \cup (A_R - A_L) \subset A_L \cup A'_R \subset A_p \cup A_{p+1} \subset ASE(p+1)$. \square

We can use Lemma 1 to show that the grammar in Fig. 3 represents the worst case for ASECONSTRUCTION.

THEOREM 1. For any attribute grammar, $m_{ASE} \leq 2m_{opt} - 1$.

Proof. If $m_{opt} = 1$, we clearly have $m_{ASE} = 1$, and the result is immediate. Suppose then that $m_{opt} > 1$. Let $OPT(p)$ denote the set of attributes that have been evaluated after the p th pass when the attributes are assigned to passes in an optimal manner, i.e., when only m_{opt} passes are required. We will show that

$$(*) \quad OPT(p) \subset ASE(2p) \quad \text{for } p = 1, 2, \dots, m_{opt} - 1.$$

1) Let A_L and A_R be the sets computed for the first pass in ASECONSTRUCTION. Clearly, $OPT(1) \subset A_L$ or $OPT(1) \subset A_R$. By Lemma 1 we have $OPT(1) \subset ASE(2)$.

2) Suppose that (*) holds for $p = 1, 2, \dots, k-1$. If $OPT(k) \subset ASE(2(k-1))$, (*) follows for $p = k$. Otherwise let A_L and A_R be the sets computed for pass $2k-1$, and let $S = OPT(k) - ASE(2(k-1))$. Since $OPT(k-1) \subset ASE(2(k-1))$, S is a subset of the attributes in the k th pass in the optimal evaluation order. Hence S must be evaluable in a single pass. Thus $S \subset A_L$ or $S \subset A_R$, yielding $OPT(k) \subset ASE(2(k-1)) \cup S \subset ASE(2(k-1)) \cup (A_L \cup A_R)$. By Lemma 1, $ASE(2(k-1)) \cup (A_L \cup A_R) \subset ASE(2k)$, which proves (*).

Consider finally the case $k = m_{opt}$. Let again $S = OPT(m_{opt}) - ASE(2(m_{opt}-1))$ and A_L and A_R be the sets computed for pass $2m_{opt}-1$. Since $OPT(m_{opt}) = A$, we have $ASE(2(m_{opt}-1)) \cup S = A$. Hence $A_L \subset S$ and $A_R \subset S$. On the other hand, $S \subset A_L$ or $S \subset A_R$, since S is evaluable in a single pass. Therefore $A_L \subset A_R$ or $A_R \subset A_L$, and only one pass is required for enlarging $ASE(2(m_{opt}-1))$ into A . Thus $m_{ASE} \leq 2m_{opt} - 1$. \square

A simple solution which yields an optimal evaluator is to delay the choice of direction if A_L and A_R are incommensurate, and to see how many passes each choice would produce. This means that SELECT would call ASECONSTRUCTION recursively for suitably reduced grammars. However, since each pass can contain only one attribute, such a recursive approach would clearly result in an exponential algorithm. On the other hand, the result of the next section indicates that in the general case there is not much hope for anything better.

3. Intractability of the construction of optimal evaluators. If we add nondeterminism to the SELECT algorithm so that it every time guesses the right choice, we obviously get an algorithm in class NP for constructing an optimal evaluator. To show that the problem of constructing an optimal evaluator is actually NP-complete, we transform the shortest common supersequence problem over binary alphabet into the optimal evaluator construction problem.

Given a string S over an alphabet Σ , a *supersequence* S' of S is any string $S' = w_0s_1w_1s_2w_2 \cdots s_mw_m$ over Σ such that $S = s_1s_2 \cdots s_m$ and each $w_i \in \Sigma^*$; we also say that S is a *subsequence* of S' . A common supersequence of a set of strings $\mathcal{S} = \{S^1, \cdots, S^n\}$ is a string S over Σ such that S is a supersequence of each S^i . The shortest common supersequence problem is defined as follows: given an alphabet Σ , a finite set \mathcal{S} of strings from Σ^* , and a positive integer k , is there a common supersequence of \mathcal{S} of length $\leq k$? This problem was shown to be NP-complete by Maier [12], provided that the size of the alphabet Σ is ≥ 5 . The result has been sharpened to any alphabet with at least two elements in [18]. This latter result will be used to prove Theorem 2.

Nonterminal	Inherited attributes	Synthesized attributes
Z	—	—
$Y_i, i = 1, \cdots, n$	—	—
$X_i, i = 1, \cdots, n$	$a_j^i, j = 1, \cdots, m_i$	$b_j^i, j = 1, \cdots, m_i$

Productions with semantic rules

```

Z → Y1Y2⋯Yn (no semantic rules)
Yi → X1LX1R,
    i = 1, ⋯, n
    if mi > 0 then
        if s1i = L then {a1i(X1L) ← constant; a1i(X1R) ← b1i(X1L)}
            else {a1i(X1L) ← b1i(X1R); a1i(X1R) ← constant}
        fi fi ∪
        for j = 2, ⋯, mi do
            if sji = L then
                {aji(XjR) ← bji(XjL)} ∪
                if sj-1i = L then {aji(XjL) ← bj-1i(XjR)}
                    else {aji(XjL) ← bj-1i(XjL)}
                fi
            else co sji = R oc
                {aji(XjL) ← bji(XjR)} ∪
                if sj-1i = L then {aji(XjR) ← bj-1i(XjR)}
                    else {aji(XjR) ← bj-1i(XjL)}
                fi
            fi
        od
Xi → x,
    i = 1, ⋯, n
    bji(Xi) ← aji(Xi), j = 1, ⋯, mi
    
```

FIG. 7

THEOREM 2. *The problem of constructing an alternating semantic evaluator which uses a minimum number of passes is NP-complete.*

Proof. We will use reduction from the shortest common supersequence problem over alphabet $\Sigma = \{L, R\}$. Suppose we are given sequences $\mathcal{S} = \{S^1, \dots, S^n\}$ in Σ^* and an integer k . For each S^i , let $S^i = s_1^i s_2^i \dots s_{m_i}^i$, where $m_i \geq 0$. We construct for \mathcal{S} an attribute grammar G given in Fig. 7.

The grammar generates exactly one attributed parse tree. As an example, suppose that $S^3 = LRLL$. The dependencies induced by the grammar for the subtree with root Y_3 are shown in Fig. 8.

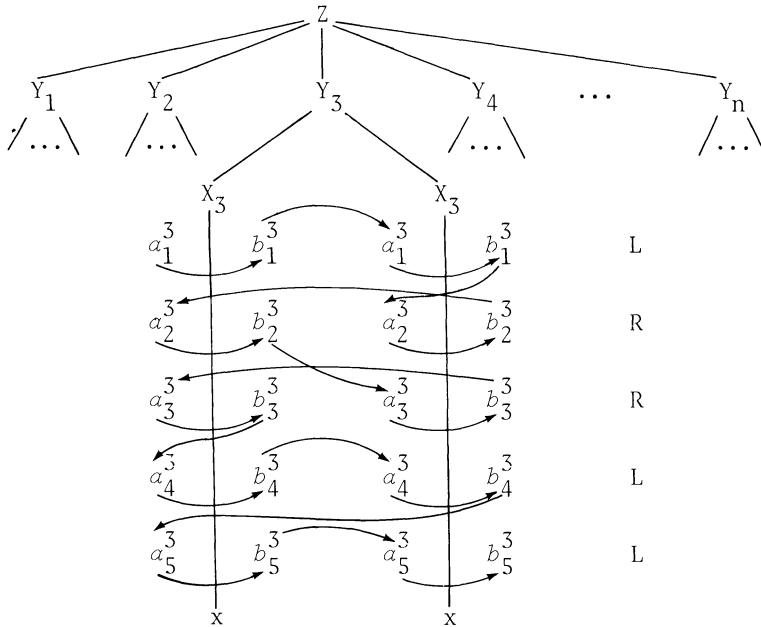


FIG. 8

We will first prove

LEMMA 2. $S = s_1 s_2 \dots s_m$ is an evaluation order for G if and only if S is a common supersequence of \mathcal{S} .

Proof. 1) *If.* Suppose that S is a common supersequence of \mathcal{S} . We show that the attributes in the subtree with root Y_i can be evaluated using S . Since the attributes in different Y -subtrees do not interfere with each other, this proves that all the attributes in G can be evaluated using S .

Let j_0, \dots, j_{m_i} be a sequence of indices such that

- (1) $s_{j_1} s_{j_2} \dots s_{j_{m_i}} = S^i$,
- (2) $s_k \neq s_r^i$ for all $j_{r-1} < k < j_r$, $r = 1, \dots, m_i$ (define $j_0 = 0$).

The existence of such a set of indices is guaranteed because S is a supersequence of S^i . These indices can be used to prove:

LEMMA 3. For each $r = 1, \dots, m_i$, exactly the attributes $\{a_r^i, b_r^i\}$ are evaluated during the j_r th pass, when S is used as the evaluation order.

Proof. 1) $r = 1$. Suppose that $s_1^i = L$ (the other case where $s_1^i = R$ is analogous). By the construction, the attributed parse tree contains the dependency chain $a_1^i(X_1^L) \rightarrow b_1^i(X_1^L) \rightarrow a_1^i(X_1^R)$. In a pass-oriented evaluator all the instances of an attribute must be evaluated during the same pass; in particular, this holds for $a_1^i(X_1^L)$ and $a_1^i(X_1^R)$.

Because of the above dependency chain, the direction of this pass has to be left-to-right, and b_1^i must be evaluable during the same pass. On the other hand, on a left-to-right pass nothing prevents the evaluation of a_1^i and b_1^i . By the definition of j_1 , the first left-to-right pass is the j_1 st one. We conclude that a_1^i and b_1^i will be evaluated on the j_1 st pass.

Furthermore, all the remaining attributes in the Y_i -subtree depend (directly or indirectly) on $b_1^i(X_i^R)$. Thus none of them can be evaluated during the passes which precede the j_1 st pass. If $s_2^i = L$, the dependency is caused by $a_2^i(X_i^L) \leftarrow b_1^i(X_i^R)$. Since the direction of the j_1 st pass is left-to-right, a_2^i cannot be evaluated during this pass. If $s_2^i = R$, the dependency is caused by $a_2^i(X_i^R) \leftarrow b_1^i(X_i^R) \leftarrow a_1^i(X_i^R)$. In this case a_2^i can never be evaluated on the same pass as b_1^i and a_1^i , no matter what the direction of the pass is. Thus we conclude that the set of attributes evaluated during the j_1 st pass in the Y_i -subtree is exactly $\{a_1^i, b_1^i\}$.

2) Suppose the lemma holds for $r = 1, \dots, k$; we must show that it holds also for $r = k + 1$. From the point of view of the j_{k+1} st evaluation pass, all the attributes evaluated during previous passes can be regarded as constants. By the induction hypothesis, in the Y_i -subtree this concerns exactly the attributes a_r^i and b_r^i , $r = 1, \dots, k$. As a consequence, the situation for the attributes a_{k+1}^i and b_{k+1}^i is just like that for a_1^i and b_1^i in case 1). By carrying out the same steps as in case 1) we obtain the result. \square

Proof of Lemma 2 continued. By Lemma 3, all the attributes in the Y_i -subtree are evaluated during passes j_1, j_2, \dots, j_m , proving the if-part.

2) *Only if.* Suppose that S is not a common supersequence of \mathcal{S} . Then there must exist at least one $S^i \in \mathcal{S}$ such that S^i is not a subsequence of S . It is an immediate consequence of Lemma 3 that the attributes in the Y_i -subtree cannot be evaluated using S as the evaluation order. Thus S is not an evaluation order for G . \square

COROLLARY 1. *G has an evaluation order of length $\leq k$ if and only if \mathcal{S} has a common supersequence of length $\leq k$.*

It is obvious that the construction of the attribute grammar G can be carried out in a number of steps which is polynomial in the size of \mathcal{S} . The theorem then follows from Corollary 1 and [18].

We proceed by developing an approximation algorithm which can be performed in polynomial time and which produces optimal evaluators for a large class of grammars. To do this, we first isolate from the attribute grammar the essential information required for assigning attributes to passes.

4. Characterization of ASE grammars. A *dependency graph* $D = (V, E)$ for an attribute grammar G has as vertices the set of attributes, i.e., $V = A$. The arcs (a, b) denote the dependencies, i.e., $(a, b) \in E$ if a is used in some semantic rule which defines b . Moreover, arcs are labeled by the symbols L, R, ANY and NO . The label of an arc (a, b) is denoted by $\delta(a, b)$ and it will be defined below so that it indicates the following: if the attributes a and b are to be evaluated during the same pass, the direction of the pass must be left-to-right ($\delta(a, b) = L$), right-to-left ($\delta(a, b) = R$), or either one ($\delta(a, b) = ANY$); the label $\delta(a, b) = NO$ indicates that a and b must be assigned to separate passes.

The types of dependencies which cause each of these labels to be attached to an arc are described in Fig. 9. Here i denotes an inherited attribute and s a synthesized attribute.

We assume that the attribute grammar G is in canonical form [5]; i.e., attributes that are defined within a production are not used as arguments within the same production. It is well known that all attribute grammars can be transformed into

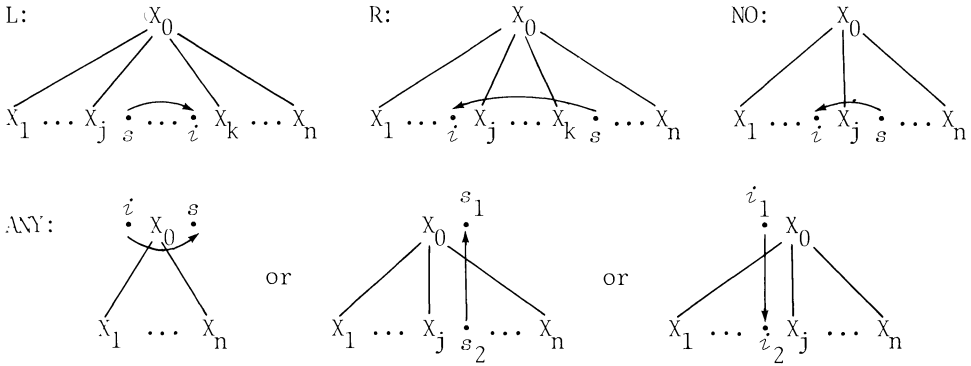


FIG. 9

canonical form by simple textual substitution [5]. In a canonical grammar all attribute dependencies are of one of the forms in Fig. 9. Thus we have a rule for finding for each dependency the label induced by it.

Different semantic rules may induce different labels to be attached to an arc. However, we shall attach only one label to each arc. The following principle is used for finding the label δ induced by a set of labels $\Delta \subset \{L, R, ANY, NO\}$:

Rule 1. $\delta(\Delta) = \text{if } NO \in \Delta \vee (L \in \Delta \wedge R \in \Delta) \text{ then } NO \text{ else}$
 if $L \in \Delta$ then L else
 if $R \in \Delta$ then R else ANY fi fi fi

For instance, for the grammar in Fig. 3 we get the labeled graph of Fig. 10.

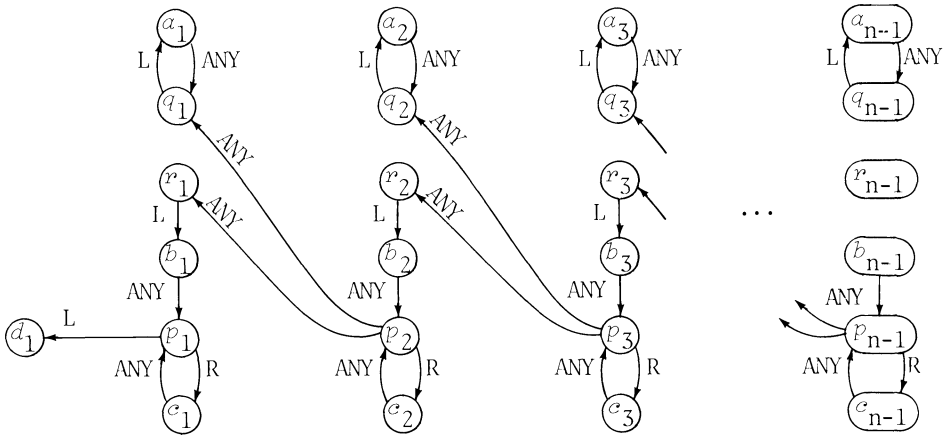


FIG. 10

Let the graphs $D_i = (V_i, E_i)$, $1 \leq i \leq r$, be the *strongly connected components* of a dependency graph $D = (V, E)$. That is, a and b are in the equivalence class V_i if and only if there is a directed path (of length zero or more) from a to b and from b to a in D . Moreover, each $E_i = \{(a, b) \in E \mid a, b \in V_i\}$. For each D_i we define the label of D_i , denoted by $\delta(D_i)$, as the label obtained by applying Rule 1 to the set $\Delta_i = \{\delta(a, b) \mid (a, b) \in E_i\}$. Note that if $\Delta_i = \emptyset$, Rule 1 yields the label ANY. Each D_i is also called a *strongly connected $\delta(D_i)$ -component* of D . The *compressed dependency graph*

C for a dependency graph $D = (V, E)$ is defined as $C = (V', E')$, where $V' = \{V_i | (V_i, E_i) \text{ is a strongly connected component of } D\}$, and $(V_1, V_2) \in E'$ if there are vertices a, b in V such that $a \in V_1, b \in V_2, V_1 \neq V_2$, and $(a, b) \in E$. The labels of the strongly connected components and those of their connecting arcs are carried over as the labels of the vertices and arcs of the compressed dependency graph. For instance, for the dependency graph in Fig. 10 we get the compressed dependency graph of Fig. 11.

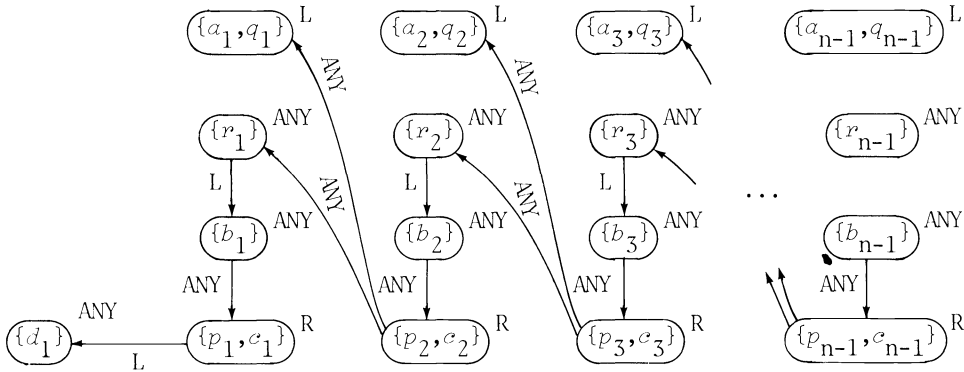


FIG. 11

LEMMA 4. *In alternating semantic evaluators, all the attributes V_i of a strongly connected component $D_i = (V_i, E_i)$ of a dependency graph D must be evaluated during the same pass.*

Proof. Suppose that $a \in V_i$ and $b \in V_i$, and b is evaluated on a later pass than a . Since D_i is strongly connected, there is a path from b to a in D_i , and thus in D . But this means that some instance of a depends on some instance of b . Because in a pass-oriented evaluator all the instances of an attribute must be evaluated during the same pass, b cannot be evaluated on a later pass than a , a contradiction. \square

LEMMA 5. *Let $D_i = (V_i, E_i)$ be a strongly connected component of a dependency graph D such that $\delta(D_i) \neq \text{NO}$ and in the compressed dependency graph there are no arcs entering V_i . Then the attributes in V_i can be evaluated in a single pass of an alternating semantic evaluator.*

Proof. We will show that the attributes in V_i can be evaluated during a pass in the direction indicated by $\delta(D_i)$. By assumption, $\delta(D_i) \neq \text{NO}$; if $\delta(D_i) = \text{ANY}$, either direction can be chosen. Consider then a left-to-right pass, i.e., $\delta(D_i) = L$ or $\delta(D_i) = \text{ANY}$ (a right-to-left pass can be handled analogously). To prove the lemma, it is sufficient to show that for any attribute $a \in V_i$ and for any instance of an associated nonterminal X , all the argument values needed for evaluating $a(X)$ are available by the time of evaluation.

Case 1. a is inherited. Since $\delta(D_i) \neq R$ and $\delta(D_i) \neq \text{NO}$, Rule 1 implies that there are no R -arcs or NO -arcs in E_i . By Fig. 10, all the dependencies directly affecting a are of the two forms in Fig. 12.

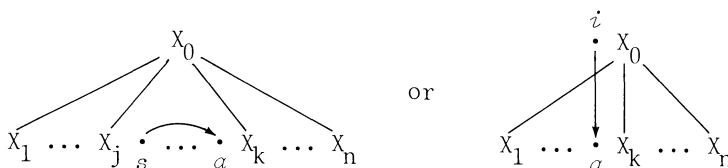


FIG. 12

Since there are no arcs entering V_i in the compressed dependency graph, we have for each such argument s or i that $s \in V_i$ and $i \in V_i$. By the evaluation procedure (and by the induction principle), we know that both $s(X_j)$ and $i(X_0)$ have been evaluated before the evaluation of $a(X_k)$ during the pass under consideration, since the direction of the pass is left-to-right.

Case 2. a is synthesized. By Fig. 10, all the dependencies directly affecting a are of the two forms in Fig. 13.

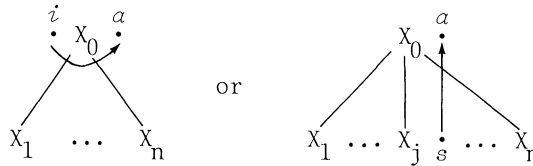


FIG. 13

Again, the evaluation procedure and the induction principle tell us that $i(X_0)$ and $s(X_j)$ have been evaluated before the evaluation of $a(X_0)$ during the pass under consideration, even regardless of the direction of the pass. \square

THEOREM 3. *An attribute grammar G can be evaluated using an alternating semantic evaluator if and only if its dependency graph D does not contain strongly connected NO-components.*

Proof. 1) *If.* Let $C = (V', E')$ be the compressed dependency graph for $D = (V, E)$. Obviously, C cannot contain any cycles. Furthermore, since V' is finite there must be a node V_i in V' with in-degree zero. Thus the attributes in V_i depend on none of the other attributes. By Lemma 5, the attributes in V_i can be evaluated during a single pass with direction $\delta(V_i)$. If $\delta(V_i) = \text{ANY}$, either one of L and R can be chosen; the assumption guarantees that $\delta(V_i) \neq \text{NO}$. Since the attributes in V_i can be regarded as constants from the point of view of the remaining passes, the node V_i and all arcs leaving it can be deleted from C . The above process can be repeatedly applied to reduced graphs, until all the nodes in V' are chosen in some order $V_{i_1} V_{i_2} \dots V_{i_r}$. The evaluation order for G is given by $\delta(V_{i_1})\delta(V_{i_2}) \dots \delta(V_{i_r})$, where each ANY is arbitrarily replaced by L or R .

2) *Only if.* Suppose that D contains a strongly connected NO-component $D_i = (V_i, E_i)$. Since $\delta(D_i) = \text{NO}$, Rule 1 indicates that either $\delta(a, b) = \text{NO}$ for some $(a, b) \in E_i$, or there are arcs $(a, b) \in E_i$ and $(a', b') \in E_i$ such that $\delta(a, b) = L$ and $\delta(a', b') = R$. In the former case a and b cannot be evaluated during the same pass, contradicting Lemma 4. In the latter case a and b can only be evaluated during the same pass if the direction of the pass is left-to-right. Similarly, a' and b' require a right-to-left pass. Thus the direction of the pass can never be chosen so that all of a, b, a' and b' could be evaluated during the same pass, again contradicting Lemma 4. \square

Similarly we can prove that an attribute grammar G can be evaluated using a multi-pass left-to-right evaluator if and only if its dependency graph D does not contain strongly connected NO- or R -components.

Our modified construction algorithm is based on Theorem 3. We first construct the compressed dependency graph to check that the grammar belongs to the ASE class. This membership test is efficient. Let π denote the number of attribute dependencies in the attribute grammar G , i.e., π is the sum of the number of attributes on the right-hand sides of semantic rules. The construction of the dependency graph $D = (A, E)$ is of time complexity $\mathcal{O}(\text{MAX}(|A|, \pi))$, and finding the strongly connected components

has time complexity $\mathcal{O}(\text{MAX}(|A|, |E|))$ (cf. [19]). Since $|E| \leq \pi$, the complexity of the membership test is thus $\mathcal{O}(\text{MAX}(|A|, \pi))$. For grammars not in the ASE class this is much better than using ASECONSTRUCTION, which may form a large part of the evaluation order before finding out that the job cannot be completed.

If the label of each arc in the dependency graph is augmented with a list of production numbers inducing the arc (and the label), we can for non-ASE grammars print a complete legend of the semantic rules which cause the ASE conflicts. This is useful for transforming the grammar into ASE form.

For the grammars which pass the membership test we form the evaluation order using the algorithm of the next section.

5. A new algorithm for constructing the evaluation order. For a compressed dependency graph $C = (V', E')$ we recursively define the sets $L(k)$ and $R(k)$, $k \geq 0$, as follows.

$$L(k) = \left\{ V_i \mid V_i \in V' - \bigcup_{j=0}^{k-1} R(j) \text{ and } \delta(V_i) \in \{L, \text{ANY}\} \text{ and } (V_h, V_i) \in E' \right.$$

$$\left. \text{implies } (V_h \in L(k) \text{ and } \delta(V_h, V_i) \in \{L, \text{ANY}\}) \text{ or } V_h \in \bigcup_{j=0}^{k-1} R(j) \right\}.$$

The definition of $R(k)$ is obtained by interchanging the roles of L and R . Intuitively, it is easy to see that $L(k)$ is the set of blocks of attributes which can be evaluated during a left-to-right pass immediately after k preceding right-to-left passes. Thus the sets A_L and A_R computed in ASECONSTRUCTION correspond to $L(0)$ and $R(0)$.

Our purpose is to modify ASECONSTRUCTION by adding to it tests which guarantee the optimality of the choice of evaluation direction in certain situations. Therefore we say that $L(0)$ is *complete* if $L(k) \subset L(0)$ for all $k > 0$, that is, the set of attributes evaluable during a left-to-right pass will not grow regardless of the number

```

algorithm ASECONSTRUCTION';
begin
   $p := 0; V'' := V'; E'' := E'; \text{maxk} := |A|;$ 
  repeat
     $p := p + 1;$ 
    compute  $L(i)$  and  $R(i)$  for the graph  $(V'', E'')$  for  $i = 0, \dots, \text{maxk};$ 
    if  $R(0) \subset L(0)$  then  $(d_p, A_p) := \text{CHOOSE}(L, L(0))$  else
    if  $L(0) \subset R(0)$  then  $(d_p, A_p) := \text{CHOOSE}(R, R(0))$  else
    if  $L(0)$  is complete then  $(d_p, A_p) := \text{CHOOSE}(L, L(0))$  else
    if  $R(0)$  is complete then  $(d_p, A_p) := \text{CHOOSE}(R, R(0))$  else
       $k_L := \text{MIN} \{k \mid L(k) \not\subset L(0)\};$ 
       $k_R := \text{MIN} \{k \mid R(k) \not\subset R(0)\};$ 
       $(d_p, A_p) := \text{SELECT}(p, L(0), R(0), k_L, L(k_L), k_R, R(k_R))$ 
    fi fi fi fi;
     $V'' := V'' \setminus (\text{if } d_p = L \text{ then } L(0) \text{ else } R(0));$ 
     $E'' := E'' \cap V'' \times V'';$ 
     $\text{maxk} := \text{maxk} - |A_p|$ 
  until  $\text{maxk} = 0;$ 
   $m_{\text{ASE}} := p$ 
end of ASECONSTRUCTION';

algorithm CHOOSE  $(d, \mathcal{V});$ 
begin
  return  $(d, \bigcup_{V_i \in \mathcal{V}} V_i)$ 
end of CHOOSE;

```

FIG. 14

of right-to-left passes. The completeness of $R(0)$ is defined similarly. If either $L(0)$ or $R(0)$ is complete, there is no reason for delaying the choice of the corresponding direction: doing some passes in the opposite direction will never help. This observation gives rise to the modified construction algorithm given in Fig. 14. It takes as input the compressed dependency graph $C = (V', E')$ for an attribute grammar G which is known to satisfy the ASE membership test. The algorithm computes the number of passes m_{ASE} , the evaluation order $d_1 d_2 \cdots d_{m_{ASE}}$, and the sets of attributes A_p evaluated during the p th pass. We have anticipated further refinements by adding some parameters to the call of SELECT.

By the preceding discussion, our modification of the construction algorithm is an improvement over the version in Fig. 1 in that our algorithm is able to guarantee the optimality of the evaluation order more often, i.e., for grammars which do not require the call of SELECT. However, as soon as SELECT is called even once there is not much that can be said about the relation of the algorithms in Figs. 1 and 14. It is easy to construct artificial examples where each performs better than the other. The upper bound given in Theorem 1 is easily seen to hold also for the modified algorithm (with SELECT specified as in Fig. 2).

Initial investigations [17] show that for real attribute grammars the number of right-to-left passes is small. This leads us to conjecture that even our slight modification of the original algorithm is in practice sufficient for preventing the algorithm from getting into dead-end sidetracks. For instance, for the grammar in Fig. 3 our algorithm is able to produce the optimal evaluation order, as shown in Fig. 15.

pass	$L(0)$	complete?	$R(0)$	complete?	d_p	A_p
1	$\{\{a_{n-1}, q_{n-1}\}, \{r_{n-1}\}, \{b_{n-1}\}\}$	no	$\{\{r_{n-1}\}, \{b_{n-1}\}, \{p_{n-1}, c_{n-1}\}, \{r_{n-2}\}\}$	yes	R	$\{r_{n-1}, b_{n-1}, p_{n-1}, c_{n-1}, r_{n-2}\}$
2	$\{\{a_{n-1}, q_{n-1}\}, \{a_{n-2}, q_{n-2}\}, \{b_{n-2}\}\}$	no	$\{\{b_{n-2}\}, \{p_{n-2}, c_{n-2}\}, \{r_{n-3}\}\}$	yes	R	$\{b_{n-2}, p_{n-2}, c_{n-2}, r_{n-3}\}$
\vdots						
$n-1$	$\{\{a_{n-1}, q_{n-1}\}, \{a_{n-2}, q_{n-2}\}, \dots, \{a_1, q_1\}, \{b_1\}\}$	no	$\{\{b_1\}, \{p_1, c_1\}\}$	yes	R	$\{b_1, p_1, c_1\}$
n	$\{\{a_{n-1}, q_{n-1}\}, \{a_{n-2}, q_{n-2}\}, \dots, \{a_1, q_1\}, \{d_1\}\}$	\emptyset			L	$\bigcup_{i=1}^{n-1} \{a_i, q_i\} \cup \{d_1\}$

FIG. 15

In fact, the observations in [17] lead us to question the practicality of the present heuristics in SELECT which favors changing the previously chosen evaluation direction. It might be better to stick to one direction until this choice has increased the size of the opposite pass. This principle, augmented with a technique for breaking ties, is contained in a modification of SELECT given in Fig. 16.

It is fairly easy to implement the computation of an $L(i)$ or $R(i)$ set using depth-first search in time $\mathcal{O}(\text{MAX}(|V''|, |E''|))$ (see, e.g., [1]). Thus we immediately get the time bound $\mathcal{O}(\text{MAX}(|A|^3, |A|^2 \cdot |E|))$ for ASECONSTRUCTION'. Recalling the $\mathcal{O}(\text{MAX}(|A|, \pi))$ complexity of the membership test, we conclude that our modified construction algorithm has time complexity $\mathcal{O}(\text{MAX}(|A|^3, |A|^2 \cdot |E|, \pi))$. This bound could be improved by computing the $L(i)$ and $R(i)$ sets more intelligently, i.e., by

avoiding their recomputation each time in the inner loop. Since $|E| \leq |A|^2$, this would bring the complexity of the modified algorithm down to $\mathcal{O}(\text{MAX}(|A|^3, \pi))$, which was obtained for the original algorithm in [13].

```

algorithm SELECT' ( $p, L(0), R(0), k_L, L(k_L), k_R, R(k_R)$ );
begin
  if  $k_R < k_L$  then return (CHOOSE ( $L, L(0)$ )) else
  if  $k_L < k_R$  then return (CHOOSE ( $R, R(0)$ )) else
  if  $|L(k_L) - L(0)| > |R(k_R) - R(0)|$  then return (CHOOSE ( $L, L(0)$ )) else
  if  $|R(k_R) - R(0)| > |L(k_L) - L(0)|$  then return (CHOOSE ( $R, R(0)$ )) else
  if  $p = 1$  then return (CHOOSE ( $L, L(0)$ )) else
  if  $d_{p-1} = R$  then return (CHOOSE ( $L, L(0)$ )) else return (CHOOSE ( $R, R(0)$ ))
  fi fi fi fi fi fi
end of SELECT';

```

FIG. 16

Our experience with real attribute grammars is that the construction algorithm is very fast. Even the largest grammars for languages like Pascal or Simula can be processed within a minute. If the optimality of the evaluation order is not guaranteed, i.e., if ASECONSTRUCTION' has to call SELECT', it might therefore be worth while to perform the algorithm several times with some variations. For instance, the version of SELECT' in Fig. 16 favors the direction left-to-right for the first pass, but the opposite choice could produce an altogether different evaluation order.

It has been observed [13] that if attributes are assigned to passes starting from the end rather than from the beginning, the resulting evaluation order tends to get shorter. Our algorithm can be immediately used to produce such evaluation orders. Let $C = (V', E')$ be the compressed dependency graph for an attribute grammar. Applying ASECONSTRUCTION' to the inverted graph (V', E'') where $E'' = \{(V_i, V_j) | (V_j, V_i) \in E'\}$ yields a sequence $d_1 d_2 \cdots d_{m_{ASE}}$, the inverse of which is an evaluation order for G . Here, too, we can try both choices for the initial pass.

If several evaluation orders are produced as suggested above, the shortest is chosen. If there are several possibilities of equal length, we can choose the one which is best for memory management [13]. That is, let $\text{first}(a)$ be the number of the pass which is used for evaluating attribute a , i.e., $a \in A_{\text{first}(a)}$, and let $\text{last}(a)$ be the number of the pass where a is used for the last time, i.e., $\text{last}(a) = \text{MAX}\{p | (a, b) \in E \text{ and } \text{first}(b) = p\}$. The *lifetime* of attribute a is defined as $\text{lifetime}(a) = \text{last}(a) - \text{first}(a)$. From several evaluation orders of equal length we choose the one which minimizes $\sum_{a \in A} \text{lifetime}(a)$.

6. Conclusions. We have demonstrated that the algorithm for constructing alternating semantic evaluators produces nonoptimal evaluators. If the minimal evaluator uses n passes, the evaluation order produced by the construction algorithm can contain $2n - 1$ passes. Furthermore, the problem of constructing optimal evaluation orders was shown to be NP-complete.

A dependency graph was introduced as a convenient data structure for dealing with pass-oriented attribute grammars. It was used in deriving a new characterization for the class of grammars which can be evaluated with alternating semantic evaluators. The characterization provides a direct and efficient membership test for ASE grammars. Furthermore, the dependency graph can easily be augmented so that the membership test can produce complete diagnostics of ASE conflicts.

Finally, we used the dependency graph to develop a new construction algorithm for alternating semantic evaluators. Our algorithm can guarantee the optimality of its

result for a class of grammars which strictly contains the class for which the algorithm used so far can guarantee optimality. Moreover, in cases where optimality is not guaranteed our algorithm uses several heuristics which are based on the properties of the attribute grammar. Thus we expect it to behave better than the original algorithm which simply alternates the evaluation direction independently of the grammar.

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. V. BOCHMANN, *Semantic evaluation from left to right*, Comm. ACM, 19 (1976), pp. 55–62.
- [3] R. GIEGERICH AND R. WILHELM, *Attribute evaluation*, Le Point sur la Compilation, Cours de la Commission des Communautés Européennes, M. Amirchahy and D. Neel, eds., IRIA, Le Chesnay, 1978, pp. 337–365.
- [4] ———, *Counter-one-pass features in one-pass compilation: a formalization using attribute grammars*, Inform. Process. Lett., 7 (1978), pp. 279–284.
- [5] M. JAZAYERI, *On attribute grammars and the semantic specification of programming languages*, Report 1159, Jennings Comp. Center, Case Western Reserve Univ., Cleveland, OH, 1974.
- [6] ———, *Live variable analysis, attribute grammars, and program optimization*, manuscript, Dept. of Computer Science, University of North Carolina, Chapel Hill, NC, 1975.
- [7] M. JAZAYERI AND D. POZEFSKY, *Algorithms for efficient evaluation of multi-pass attribute grammars without a parse tree*, Report TR 77-001, Dept. of Computer Science, University of North Carolina, Chapel Hill, NC, 1977, revised 1979.
- [8] ———, *Space-efficient storage management in an attribute grammar evaluator*, Report TR 79-007, Dept. of Computer Science, University of North Carolina, Chapel Hill, NC, 1979.
- [9] M. JAZAYERI AND K. G. WALTER, *Alternating semantic evaluator*, Proc. ACM 1975 Annual Conference, 1975, pp. 230–234.
- [10] D. E. KNUTH, *Semantics of context-free languages*, Math. Systems Theory, 2 (1968), pp. 127–145.
- [11] P. M. LEWIS, D. J. ROSENKRANTZ AND R. E. STEARNS, *Attributed translations*, J. Comput. System Sci., 9 (1974), pp. 279–307.
- [12] D. MAIER, *The complexity of some problems on subsequences and supersequences*, J. Assoc. Comput. Mach., 25 (1978), pp. 322–336.
- [13] D. POZEFSKY, *Building efficient pass-oriented attribute grammar evaluators*, Report TR 79-006, Dept. of Computer Science, University of North Carolina, Chapel Hill, NC, 1979.
- [14] D. POZEFSKY AND M. JAZAYERI, *A family of pass-oriented attribute grammar evaluators*, Proc. ACM 1978 Annual Conference, 1978, pp. 261–270.
- [15] K.-J. RÄIHÄ, *On attribute grammars and their use in a compiler writing system*, Report A-1977-4, Dept. of Computer Science, University of Helsinki, 1977.
- [16] K.-J. RÄIHÄ, M. SAARINEN, E. SOISALON-SOININEN AND M. TIENARI, *The compiler writing system HLP (Helsinki Language Processor)*, Report A-1978-2, Dept. of Computer Science, University of Helsinki, 1978.
- [17] K.-J. RÄIHÄ AND E. UKKONEN, *On the optimal assignment of attributes to passes in multi-pass attribute evaluators*, Automata, Languages and Programming, Noordwijkerhout, The Netherlands, J. W. de Bakker and J. van Leeuwen, eds., Lecture Notes in Computer Science 85, Springer-Verlag, Berlin-Heidelberg-New York 1980, pp. 500–511.
- [18] ———, *The shortest common supersequence problem over binary alphabet is NP-complete*, Theoret. Comput. Sci., 16 (1981), to appear.
- [19] R. E. TARJAN, *Depth-first search and linear graph algorithms*, this Journal, 1 (1972), pp. 146–160.

COMPLETENESS, APPROXIMATION AND DENSITY*

KER-I KO[†] AND DANIEL MOORE[‡]

Abstract. Polynomial time approximations to exponential time computable problems (EXP) are considered from the point of view of structure. Infinitely often speedable and almost everywhere complex problems are studied using the notions of polynomial time productivity and immunity. In particular, the existence of a polynomial time immune set which is not polynomial time approximable at all but which is polynomial time t -complete in EXP is proven. The relationship between completeness and approximability is also studied. It is shown that being polynomial time m -complete in EXP does not provide any control of the probability of erroneous approximations.

Key words. complexity, polynomial time, exponential time, approximations

1. Introduction. Many commonly encountered important problems appear to be intractable in the sense that no polynomial time algorithms are known for solving them (see, for example, [7]). Heuristic algorithms have been hoped to yield approximate solutions in polynomial time. Worst case analysis has shown that many practically useful heuristic algorithms are not polynomial time algorithms in the worst case. Furthermore, for many important combinatorial problems, it has been shown that no polynomial time algorithm can solve the corresponding worst case approximation problem unless $P = NP$, which is widely believed to be false [6], [16]. However, many practically useful heuristic algorithms are, on the average, polynomial time bounded. For instance, Karp [10] has given a theoretical framework for average case analysis and has shown that in this framework many intuitively sound algorithms are indeed "good" algorithms.

This paper involves the average case analysis of polynomial time approximation from an abstract point of view. L. Berman and Hartmanis [3], [9], while examining the structure of complete sets, showed that a complete set for a deterministic class must have an infinite "easy subset" and that no sparse sets can be complete. A similar result for NP-complete and PSPACE-complete sets is observed by P. Berman [4] and Meyer and Patterson [14]. This paper reports a further study along this line of the structure of exponential time computable sets. We show that, despite the fact that many-one completeness guarantees an infinitely often speedup, truth-table complete sets may be not approximable at all. We also consider the question of the size of the "easy subsets" of complete sets and are able to answer an open question asked by L. Berman. Our result reveals that completeness is not a useful tool in classifying intractable problems according to their approximability.

Let Σ be a finite set of symbols and Σ^* be all finite strings of symbols over Σ . A problem is a language contained in Σ^* . Let $|x|$ denote the length of $x \in \Sigma^*$. Consider the following simplified view of approximation.

For each problem S which is not polynomial time computable, A is a polynomial time approximation algorithm for S if:

- (1) There exists a polynomial p such that, for any $x \in \Sigma^*$, $A(x)$ halts in $p(|x|)$ or fewer steps and $A(x) = 1$ (accepts x), 0 (rejects x), or "?" (does not know the answer).
- (2) For every x , $A(x) = 1 \Rightarrow x \in S$, and $A(x) = 0 \Rightarrow x \notin S$.

* Received by the editors August 16, 1979 and in final revised form on January 26, 1981. This research is based in part on the first author's doctoral dissertation prepared at the Department of Computer and Information Science, The Ohio State University, under the supervision of the second author.

[†] Department of Computer Science, University of Houston, Houston, Texas 77004.

[‡] 5E-116, Bell Laboratories, 600 Mountain Ave., Murray Hill, New Jersey 07974.

The set $\{x \in \Sigma^*: A(x) \neq ?\}$ is called the “definite part” of A . Obviously, the size of the definite part of an algorithm A determines how good it is.

We first consider the infinitely often (i.o.) speedability and almost everywhere (a.e.) complexity questions; that is, questions which ask whether there is a polynomial time approximation algorithm A for S such that the definite part is infinite or not. In § 3 we first define the p -productive sets as the polynomial time analogue of the productive sets and show that there are no exponential time computable p -productive sets. As a consequence, this notion is not useful in studying the i.o. speedability of exponential time computable problems (EXP). Then we define the p -immune sets as the polynomial time analogue of the immune sets. It has been observed by L. Berman [3] that polynomial time many-one complete sets in EXP are not polynomial time immune. We show that some polynomial time immune set is polynomial time truth-table complete in EXP and hence can be used as evidence that, in EXP, polynomial time many-one complete sets and truth-table complete sets do not coincide.

In § 4, the density of a set is defined assuming a uniform distribution over any initial segment of integers. Based on this definition of density we show that there exist some polynomial time m -complete problems whose “definite part” is always of density 0. Thus there are no polynomial time approximation algorithms which can solve even a small fraction of the inputs. Moreover, the “definite part” of polynomial time m -complete sets can have density ranging from 0 to 1. Hence the polynomial time m -completeness of a set does not provide any information about its approximability. Therefore, a different approach needs to be taken in order to more successfully classify the problems according to their approximability.

Finally, in § 5, we give a summary of the currently known results and open questions on the approximability of complete sets.

2. Preliminaries. Standard definitions of deterministic, nondeterministic and oracle Turing machines (TM) are used here (see, for example, [1], [15]). The amount of time used by a machine M (deterministic, nondeterministic or oracle) on input x is the number of steps in the shortest accepting computation if x is accepted; and the number of steps in the longest computation if x is not accepted. A Turing machine M runs in time T for some function T if, for all $n > 0$, M uses less than $T(n)$ steps on every input x of length n . Some interesting time bounded classes of problems are seen in the following.

$$\text{DTIME}(T)(\text{NTIME}(T)) = \{L \subseteq \Sigma^*: L \text{ is accepted by some} \\ \text{(non)deterministic TM which runs in time } T\}.$$

Let \mathcal{C} be a class of functions.

$$\text{DTIME}(\mathcal{C}) = \bigcup_{T \in \mathcal{C}} \text{DTIME}(T),$$

$$\text{NTIME}(\mathcal{C}) = \bigcup_{T \in \mathcal{C}} \text{NTIME}(T),$$

$$P = \text{DTIME}(\text{Poly}) = \bigcup_{i=0}^{\infty} \text{DTIME}(n^i),$$

$$\text{NP} = \text{NTIME}(\text{Poly}) = \bigcup_{i=0}^{\infty} \text{NTIME}(n^i),$$

$$\text{EXP} = \text{DTIME}(2^{\text{linear}}) = \bigcup_{i=0}^{\infty} \text{DTIME}(2^{ni}).$$

The polynomial time reducibilities are those in [11], namely,

- \cong_T^p : polynomial time Turing reducibility,
- \cong_u^p : polynomial time truth-table reducibility,
- \cong_m^p : polynomial time many-one reducibility.

We say that a set L is \cong_T^p - (\cong_u^p -, \cong_m^p -) *hard* in \mathcal{C} , for some class \mathcal{C} of problems, if $L' \cong_T^p L$ ($L' \cong_u^p L$, $L' \cong_m^p L$) for all $L' \in \mathcal{C}$. We say that L is \cong_T^p - (\cong_u^p -, \cong_m^p -) *complete* in \mathcal{C} if L is \cong_T^p - (\cong_u^p -, \cong_m^p -) *hard* in \mathcal{C} and $L \in \mathcal{C}$.

A set L is said to be *p-sparse* if there is a polynomial p such that, for all $n \geq 1$, the cardinality of the set $\{x \in L : |x| \leq n\}$ is less than or equal to $p(n)$. A set L is said to be *almost polynomial time computable* ($L \in \text{APT}$) if there is a polynomial time approximation algorithm A for L such that its “uncommitted part” is *p-sparse* [14].

We know that a finite set is always polynomial time computable because we can determine the membership of an input in a constant number of steps by using a table look-up technique. Therefore, for any polynomial time approximation algorithm A for a set $S \notin P$, the “uncommitted part” of A is infinite. Actually, Lynch [13] has shown that, for every $S \notin P$, there exists a fixed uncommitted part.

Let χ_S denote the characteristic function of S .

LEMMA 1 [13]. *If S is a recursive set with $S \notin P$, then there exists an infinite recursive set $X \subseteq S$ such that*

$$(\forall p, \text{ a polynomial})(\forall M, \text{ a DTM}) [M \text{ computes } \chi_S \Rightarrow T_M(x) \cong p(|x|) \text{ a.e. on } X],$$

where $T_M(x)$ is the number of steps M takes to compute $\chi_S(x)$, and a.e. is the abbreviation of “for all but finitely many.”

The set X is called a *polynomial complexity core* of S [13]. It is clear that, for any polynomial time approximation algorithm A for X , the intersection of X and the definite part of A is finite. The size of a polynomial complexity core of S thus gives a lower bound on the uncommitted part of any polynomial time approximation algorithm A for S .

3. P-productive sets and P-immune sets. In recursive function theory, the concept of productivity has been used to study the speedability of nonrecursive sets. In particular, Blum and Marques [5] have shown that a recursively enumerable set is “effectively speedable” if and only if it is “subcreative.” We define the notion of polynomial time productivity and show that there are, unfortunately, no such sets in EXP.

We assume that a certain enumeration of polynomial time bounded Turing machines is available, satisfying two properties.

1. The i th Turing machine M_i (with program encoded in a string \tilde{M}_i) has time clock $p_i = \lambda x[x^i + i]$ attached. On inputs of length x , M_i halts in $p_i(x)$ or fewer steps.
2. This enumeration is efficient; i.e., there exist two polynomial time computable functions ϕ_1 and ϕ_2 and two polynomials q_1 and q_2 such that

$$\begin{aligned} \phi_1(i) &= \tilde{M}_i, & \phi_2(\tilde{M}_i) &= i, \\ |i| &\leq q_1(|\tilde{M}_i|), & |\tilde{M}_i| &\leq q_2(|i|) \quad \text{for all } i = 1, 2, \dots \end{aligned}$$

Property 1 has been used in [2]. Property 2 is also essential for [2, Lemma 1]. It means that, when $\langle i, x \rangle$ is an input, we can simulate M_i on x using not more than polynomially bounded time.

Let L_i be the language recognized by M_i .

DEFINITION 1. A recursive set A is said to be p -productive if there is a polynomial time computable function f (called the p -productive function for A) such that $f(i) \in A - L_i$ whenever $L_i \subseteq A$.

THEOREM 1. There exists a p -productive set in $\text{DTIME}(2^{2^{\text{linear}}})$.

Proof. Let $A = \{\tilde{M}_i \# w : M_i \text{ accepts } \tilde{M}_i \# w \text{ in } \leq 2^{2^{|\tilde{M}_i \# w|}} \text{ steps}\}$. We first observe that $A \in \text{DTIME}(2^{2^{\text{linear}}})$ because for any input of the form $\tilde{M}_i \# w$, we can efficiently simulate M_i on $\tilde{M}_i \# w$ for $2^{2^{|\tilde{M}_i \# w|}}$ steps using only $O(|\tilde{M}_i \# w|) \cdot 2^{2^{|\tilde{M}_i \# w|}}$ steps. Thus, \bar{A} is also in $\text{DTIME}(2^{2^{\text{linear}}})$. We claim that \bar{A} is p -productive.

First define $f(i) = \tilde{M}_i \# 0^m$, where m is a number greater than 0 such that $f(i)$ is padded with zeros so that its length becomes $3 \cdot q_1(|\tilde{M}_i|) + 3$. It is clear that f is polynomial time computable by Property 2. Now, if $L_i \subseteq \bar{A}$ we show that $f(i) \in \bar{A} - L_i$.

1. Suppose, by way of contradiction, $f(i) \in A$. Then, from the definition of A , M_i accepts $f(i) = \tilde{M}_i \# 0^m$. Thus, $f(i) \in L_i \subseteq \bar{A}$, a contradiction. Thus $f(i) \notin A$.

2. Suppose, by way of contradiction, $f(i) \in L_i$. Then M_i accepts $f(i)$ in $\leq p_i(|f(i)|)$ steps. It is easily seen that $p_i(|f(i)|) \leq 2^{2^{|f(i)|}}$, since $|f(i)| = 3 \cdot q_1(|\tilde{M}_i|) + 3$. Thus, by definition of A , $f(i) \in A$. This is a contradiction, and so we have shown that $f(i) \notin L_i$. \square

THEOREM 2. There is no p -productive set in $\text{DTIME}(2^{\text{poly}})$.

Proof. Assume, by way of contradiction, that A is p -productive and can be computed by a deterministic Turing machine (DTM) M_A with time complexity $2^{p_A(n)}$ for some polynomial p_A . Let f be the p -productive function for A with time complexity bounded by p_f , a polynomial.

Now, for each integer i , construct a DTM M'_i in the following manner.

For given input x , M'_i simulates M_A on x for $|x|^i + i$ steps and accepts x only if M_A accepts x in $|x|^i + i$ steps.

It is clear that the machine M'_i is of polynomially bounded time complexity.

Since the construction of M'_i is uniform in i , there is a recursive function σ such that $M'_i = M_{\sigma(i)}$, where $\{M_j\}$ is the standard enumeration of polynomial time DTMs described in Property 1. In addition, σ is polynomial time computable because the map $\lambda i[\tilde{M}'_i]$ can be easily computed in polynomial time by attaching a ‘‘clock machine’’ to M_A . So, we may assume the existence of a polynomial q such that $q(|i|) \geq |\sigma(i)|$ and $|\sigma(i)| \geq |i|$.

It is clear from the description of M'_i that $L_{\sigma(i)} \subseteq A$. So, $f(\sigma(i)) \in A - L_{\sigma(i)}$.

Consider i so large that

$$2^{i^{i-1}} \geq p_A \circ p_f \circ q(|i|).$$

Then $f(\sigma(i)) \in A$ implies that

$$M_A \text{ accepts } f(\sigma(i)) \text{ in } \leq 2^{p_A(|f(\sigma(i))|)} \text{ steps}$$

and

$$\begin{aligned} P_A(|f(\sigma(i))|) &\leq P_A \circ p_f(|\sigma(i)|) \\ &\leq P_A \circ p_f \circ q(|i|) \leq 2^{i^{i-1}} \leq i. \end{aligned}$$

Thus, M_A accepts $f(\sigma(i))$ in $\leq 2^i$ steps, or $\leq |f(\sigma(i))|^i$ steps. So, $f(\sigma(i))$ is accepted by $M_{\sigma(i)}$ and is in $L_{\sigma(i)}$. This is a contradiction and we have shown that such an A does not exist. \square

Thus the concept of p -productivity is not very useful in studying the structure of sets in EXP. On the other hand, p -productivity is not the only tool we have. It is well known that a recursively enumerable set is m -complete if and only if its complement is productive. Also, Gill and Morris [8] showed that subcreativity is equivalent to

“S-completeness.” So, we may well consider the relationship between completeness and speedability. Actually, L. Berman [3] has shown that if S is any set which is \leq_m^p -complete for a deterministic time class $T(n)$, then S has effective i.o. speedup from $T(n^r)$ to polynomial time for some $r > 0$.

What about other sets which are not \leq_m^p -complete? What about \leq_u^p -complete sets? These questions are considered next. First we define p -immune sets as the analogue of the immune sets at the polynomial time level.

DEFINITION 2. An infinite set A is said to be p -immune if, for every subset $B \subseteq A, B \in P \Rightarrow B$ is finite.

We first show that there exists a set S in EXP such that both the set S and its complement are p -immune. In other words, Σ^* , the set of all input strings, is a polynomial complexity core of S .

LEMMA 2 [9]. *There exists a set A_0 in EXP such that*

- (i) A_0 is not p -sparse, and
- (ii) if $A_0 \leq_m^p B$ via ρ , then ρ is a one-to-one mapping a.e.

THEOREM 3. *Both the set A_0 in Lemma 2 and its complement \bar{A}_0 are p -immune.*

Proof. First note that A_0 must be infinite and co-infinite because otherwise A_0 would be in P and thus \leq_m^p -reducible to any set, in contradiction to (ii) of Lemma 2.

Assume, by way of contradiction, that A_0 has an infinite subset C which is in P . Now, let B be a \leq_m^p -complete set in EXP and $A_0 \leq_m^p B$ via ρ . Then the function ρ' defined by

$$\rho'(x) = \begin{cases} \rho(x) & \text{if } x \notin C, \\ x_0 & \text{if } x \in C, \end{cases}$$

where x_0 is a fixed number in B , \leq_m^p -reduces A_0 to B but is not one-to-one a.e. This contradicts (ii) of Lemma 2, and hence such an infinite set C does not exist. In other words, A_0 is p -immune.

A similar proof shows that \bar{A}_0 is p -immune. \square

We can also construct a p -immune set using a variation of Post’s simple set construction.

Assume that all numbers are written in binary form as strings in $\{0, 1\}^*$.

THEOREM 4. *There exists a p -immune set which is \leq_u^p -complete in EXP.*

Proof. First construct a set S in EXP such that \bar{S} is p -immune and, for any n ,

$$|S \cap \{x \in \Sigma^* : x \leq 2^n\}| \leq \log_2 n + 1,$$

by the following algorithm M_S :

```

Obtain input  $x$ ;
do  $i = 0$  to  $\log_2 |x|$ ;
  do  $j = 2^{2^i}$  to  $x$ ;
    simulate  $M_i$  on  $j$  for  $2^{|j|}$  steps;
    if  $j$  is accepted and  $j = x$ 
      then accept  $x$  and halt;
    else if  $j$  is accepted and  $j < x$ 
      then go to OUT;
  end;
OUT: end;
  Reject  $x$  and halt.
    
```

We now check the following assertions ((1)–(4)):

(1) $S \in \text{EXP}$.

If $|x| = n$, then $x < 2^{n+1}$, and so the execution time of M_S on x is $\leq c_1 + (\log_2 n + 1) \cdot x \cdot (c_2 \cdot 2^{2^n})$, for some constants c_1 and c_2 , which is $= O(2^{4n})$.

(2) \bar{S} is infinite.

For any $x \in S$, from the algorithm M_S we know that there exists exactly one i such that

(a) $i \leq \log_2 x$;

(b) the computation of M_i on x is simulated by M_S on x and halts in $\leq 2^{|x|}$ steps;

(c) $(\forall j, 2^{2^i} \leq j < x) [M_i \text{ does not accept } j \text{ in } \leq 2^{|j|} \text{ steps}]$.

The reason is that if M_i accepts some j , $2^{2^i} \leq j < x$, in $\leq 2^{|j|}$ steps then the execution flows to OUT, and M_i is no longer to be simulated.

Let $\phi: S \rightarrow N$ be defined by $\phi(x) =$ the i described above. We claim that ϕ is one-to-one.

Assume, by way of contradiction, that $x < y$ and $\phi(x) = \phi(y) = i$. Then, by (a), (b) and the fact that $\phi(x) = i$, we have $i \leq \log_2 |x|$ and M_i accepts x in $\leq 2^{|x|}$ steps. However, by (c) and $\phi(y) = i$, we have that M_i does not accept x in $\leq 2^{|x|}$ steps. This is a contradiction. So, ϕ is one-to-one.

But the fact that ϕ is one-to-one means that

$$|S \cap \{x : |x| \leq n\}| \leq \log_2 n.$$

Hence \bar{S} is infinite.

(3) \bar{S} has no infinite subset in P .

By way of contradiction, assume that $L_k \subseteq \bar{S}$ and L_k is infinite.

Since $2^n > p_k(n)$ a.e., we know that the set $\{x \in \Sigma^* : |x| > 2^k \text{ and } M_k \text{ accepts } x \text{ in } \leq 2^{|x|} \text{ steps}\}$ is infinite. Let x_k be the smallest element in this set. Then $x_k \in L_k \subseteq \bar{S}$.

Consider the workings of algorithm M_S applied to input x_k . Since M_S will reject x_k , M_S will go through the simulation of M_i on j for every $i = 0, 1, \dots, \log_2 |x_k|$ and some $j = 2^{2^i}, 2^{2^i} + 1, \dots, x_k$. So, M_S will simulate M_k on j 's, $2^{2^k} \leq j \leq x_k$. But x_k is the smallest number in $\{2^{2^k}, \dots, x_k\}$ which M_k will accept in $\leq 2^{|x|}$ steps. So, M_S will eventually simulate M_k on x_k for $2^{|x_k|}$ steps and accepts x_k . This contradicts the fact that $x_k \notin S$. So, we have shown that \bar{S} has no infinite subset in P .

(4) $|S \cap \{x \in \Sigma^* : x \leq 2^n\}| \leq \log_2 n + 1$.

This has been shown in (2).

Now we construct finite sets $S_n = \{y_n, y_n + 1, \dots, y_n + k_n - 1\}$ where $k_n = \lceil \log_2 n \rceil$ and $y_n = nk_n - 2^{k_n} - k_n + 1$, for $n = 1, 2, \dots$.

First observe the following facts ((5) and (6)).

(5) The function $\lambda n[y_n]$ is polynomial time computable because $\lambda n[k_n]$ can be computed by binary searching.

(6) For all but finitely many n , $S_n \cap \bar{S} \neq \emptyset$.

Since $|S \cap \{x \in \Sigma^* : x \leq 2^n\}| \leq \log_2 n + 1$, we need only to check that, for almost all n , $y_n + k_n - 1 < 2^{n/2}$. (So, $|S \cap S_n| \leq |S \cap \{x \in \Sigma^* : x < 2^{n/2}\}| < \log_2 n \leq k_n$, and hence $|S \cap S_n| \geq 1$). But $y_n + k_n - 1 = nk_n - 2^{k_n} \leq n^2 \leq 2^{n/2}$ if n is large enough, and so $S_n \cap \bar{S} \neq \emptyset$ for almost all n .

Now assume that the set A is \leq_m^p -complete in EXP. Let $S^* = S \cup (\cup \{S_n : n \in A\})$. We claim that \bar{S}^* is p -immune and \leq_{IT}^p -complete in EXP ((7)–(9)):

(7) $S^* \in \text{EXP}$.

The following straightforward algorithm for S^* works in exponential time. Given input x , first perform a binary search over k such that $y_k \leq x < y_{k+1}$ (From (5), this can be done in $p(|x|)$ steps for some polynomial p .) Then test whether $k \in A$. If yes,

then accept x ; otherwise, accept x if and only if x is in S . (Both A and S are in EXP.)

(8) S^* is \cong_{it}^p -complete in EXP (and so is \bar{S}^*).

We need only show that $A \cong_{it}^p S^*$.

Note that, for n large enough, $\bar{S} \cap S_n \neq \emptyset$, so, $(\forall n) [n \in A \Leftrightarrow S_n \subseteq S^*]$ where $\forall n$ is the abbreviation of "for all but finitely many n ." Thus, $A \cong_{it}^p S^*$ via the following tt -function $\langle f, g \rangle$:

$$f(n) = \langle y_n, y_n + 1, \dots, y_n + k_n - 1 \rangle;$$

$$g(\langle n, m_1, m_2, \dots, m_{k_n} \rangle) = \text{true if and only if}$$

$$m_1 = m_2 = \dots = m_{k_n} = \text{true, for large enough } n;$$

f and g are trivial on small n .

(9) \bar{S}^* is p -immune.

Since \bar{A} is infinite and $(\forall n) [S_n \cap \bar{S} \neq \emptyset]$, \bar{S}^* must be infinite. Now if $B \in P$ is a subset of \bar{S}^* , then $B \subseteq \bar{S}$ and B must be finite. \square

Combining Theorem 4 and L. Berman's result in [3], we conclude that, in EXP, \cong_m^p -complete sets are always i.o. speedable, but some \cong_{it}^p -complete sets are not i.o. speedable. In [8], effective speedable sets are shown to be equivalent to S -complete sets and \cong_S is incomparable with \cong_{it} . An interesting open question is whether there exists a polynomial time reducibility \cong_S^p as an analogue of \cong_S which satisfies similar properties.

4. Density and complexity. We have just seen that the size of the cores of sets in EXP could be as large as the set Σ^* of all inputs, but a \cong_m^p -complete set must have an infinite "easy" part. How large can this "easy" part be? Is there an upper bound for it? We now discuss these questions.

DEFINITION 3 [12]. The density of a set $S \subseteq N$ is less than (greater than) r , for $0 \leq r \leq 1$, if

$$\text{dens}_n(S) \stackrel{\text{def}}{=} \frac{|S \cap \{0, 1, \dots, n\}|}{n + 1} < (>) r \quad \text{a.e.}$$

This definition basically assumes a uniform distribution over any initial segment of integers. We may compare it with Karp's [10] more practical definition.

DEFINITION 4 [10]. Let S_n be a probability distribution over $x \in \{0, 1\}^n$. We say that a set $X \subseteq \{0, 1\}^*$ is empty a.e. if

$$\sum_{n=0}^{\infty} \Pr \{X \cap \{0, 1\}^n\}_{S_n} < \infty$$

and $X = Y$ a.e. if $(X - Y) \cup (Y - X) = \emptyset$ a.e.

Under the uniform distribution, the notion $X = \emptyset$ a.e. is stronger than the notion $\text{dens}_n(X) = 0$ a.e., and p -sparseness is even stronger in the sense that each of the following conditions on X implies its successor but no two of them are equivalent:

- (i) X is p -sparse.
- (ii) $X = \emptyset$ a.e. under uniform distribution.
- (iii) $(\forall \epsilon > 0) [\text{dens}_n(X) < \epsilon \text{ a.e.}]$.

We now use this definition to study the size of the polynomial complexity cores of sets in EXP.

THEOREM 5. If A is \cong_m^p -complete in EXP, then A is not almost polynomial time computable; i.e.,

$$[B \subseteq A, C \subseteq \bar{A}] \Rightarrow \overline{B \cup C} \text{ is not } p\text{-sparse.}$$

Proof. This is a simple corollary of Lemma 2 (due to Hartmanis and L. Berman [9]). Actually, it is proved in [9] that Lemma 2 implies that \cong_m^p -complete sets in EXP cannot be p -sparse. It can be easily strengthened to show this theorem. \square

The above theorem says that the uncommitted part of any polynomial time algorithm for a \leq_m^p -complete set in EXP must be greater than any p -sparse set. Can it be of density 0? Can it be of density 1? We give the positive answers in the next two theorems.

THEOREM 6. *There exists a set A which is \leq_m^p -complete in EXP such that*

$$(\exists B, C \in P)[B \subseteq A, C \subseteq \bar{A}, \text{ and } \lim_{n \rightarrow \infty} \text{dens}_n(B \cup C) = 1].$$

Proof. Let X be a \leq_m^p -complete set in EXP. Let $A = \{x \in \Sigma^* : (\exists y \in X)y^2 = x\}$. Then, obviously, A is \leq_m^p -complete in EXP. To see that A satisfies the above condition, just let $B = \emptyset$, $C = \{x \in \Sigma^* : (\exists y \in \Sigma^*)y^2 \neq x\}$ and observe that $\text{dens}_n\{x \in \Sigma^* : (\exists y \in \Sigma^*)y^2 = x\} < \epsilon$ a.e. for all $\epsilon > 0$. \square

THEOREM 7. *There exists a set A which is \leq_m^p -complete in EXP such that, for any sets B, C in P ,*

$$[B \subseteq A \ \& \ C \subseteq \bar{A}] \Rightarrow \lim_{n \rightarrow \infty} \text{dens}_n(B \cup C) = 0.$$

Proof. First note that, from the proof of Theorem 4, there exists a \leq_m^p -complete set S in EXP such that \bar{S} is p -immune and S is p -sparse. Thus, for any $B, C \in P$, if $B \subseteq S, C \subseteq \bar{S}$, then $B \cup C$ is p -sparse, and hence $\text{dens}_n(B \cup C) < \epsilon$ a.e. for all $\epsilon > 0$; i.e., $\lim_{n \rightarrow \infty} \text{dens}_n(B \cup C) = 0$.

All we need to do is modify S to get a \leq_m^p -complete set A in EXP such that density of A is small and \bar{A} is hard to approximate. This can be done easily: let $Y = \{x : x \text{ is a perfect square}\}$, and $f : N \rightarrow \bar{Y}$ be a one-to-one, onto, increasing function between N and \bar{Y} . (Then, $f(n)$ is the n th smallest element in \bar{Y} .) Both f and f^{-1} are polynomial time computable. Now, for a \leq_m^p -complete set X in EXP, let $A = \{x \in Y : (\exists y \in X)y^2 = x\} \cup f(S)$.

It is easily observed that A is \leq_m^p -complete in EXP and that $\lim_{n \rightarrow \infty} \text{dens}_n(A) = 0$. So, all we have left is to show that if $C \in P$ and $C \subseteq \bar{A}$, then $\lim_{n \rightarrow \infty} \text{dens}_n(C) = 0$.

Assume that $C \in P$ and $C \subseteq \bar{A}$. Then $D = C \cap \bar{Y}$ is in P . Now, $f^{-1}(D)$ is also in P and $f^{-1}(D) \subseteq \bar{S}$. So, $f^{-1}(D)$ is finite, and hence D is finite. Therefore, $\lim_{n \rightarrow \infty} \text{dens}_n(C) \leq \lim_{n \rightarrow \infty} \text{dens}_n(D) + \lim_{n \rightarrow \infty} \text{dens}_n(Y) = 0$. \square

The next theorem, which states that the polynomial complexity core of a \leq_m^p -complete set in EXP can be of any size, solves an open question posed by L. Berman [3].

THEOREM 8. *For any $r \in Q, 0 < r < 1$, there exists a set A which is \leq_m^p -complete in EXP such that*

- (i) $(\forall B, C \in P)[B \subseteq A \ \& \ C \subseteq \bar{A}] \Rightarrow \limsup_{n \rightarrow \infty} \text{dens}_n(B \cup C) \leq r$;
- (ii) $(\exists B, C \in P)[B \subseteq A, C \subseteq \bar{A} \ \& \ \liminf_{n \rightarrow \infty} \text{dens}_n(B \cup C) \geq r]$.

Proof. Intuitively, if $r = a/b$, we consider the intervals of b consecutive integers. For each such interval, choose a numbers in \bar{A} , and let the other $b - a$ numbers be either in A or in \bar{A} so that A is complex enough. A formal proof follows.

Let A_1 be the set found in Theorem 7. That is, A_1 is \leq_m^p -complete in EXP and if $B, C \in P, B \subseteq A_1$ and $C \subseteq \bar{A}_1$, then $\lim_{n \rightarrow \infty} \text{dens}_n(B \cup C) = 0$.

We define $A = \{bx + y : a \leq y < b, x \in A_1\}$. Then it is easy to see that A is \leq_m^p -complete in EXP ($A_1 \leq_m^p A$ via $\lambda x[bx + a]$). We check the conditions (i) and (ii) in the following.

- (i) From the definition of A , we have

$$|A \cap \{0, 1, \dots, bn - 1\}| = (b - a) \cdot |A_1 \cap \{0, \dots, n - 1\}|.$$

Therefore, for n large enough, we have

$$\text{dens}_n(A) \leq \frac{b-a}{b} \text{dens}_{n/b}(A_1) + \frac{b}{n+1},$$

and hence,

$$\lim_{n \rightarrow \infty} \text{dens}_n(A) = 0.$$

Now, if $C \in P$ and $C \subseteq \bar{A}$, let $D = C \cap \{bx + y : 0 \leq y < a\}$ and $E = C - D$. Then $\limsup_{n \rightarrow \infty} \text{dens}_n(D) \leq \limsup_{n \rightarrow \infty} \text{dens}_n(\{bx + y : 0 \leq y < a\}) \leq r$.

The only thing left to show is that $\lim_{n \rightarrow \infty} \text{dens}_n(E) = 0$. For each y , $a \leq y \leq b-1$, let $E_y = \{x \in E : x \equiv y \pmod{b}\}$ and $f_y(x) = \lfloor (x-y)/b \rfloor$. Then f_y is a polynomial time computable one-to-one mapping on E_y and $f_y(E_y) \subseteq \bar{A}_1$. Since $C \in P$, we know that $E_y \in P$, and that $f_y(E_y) \subseteq \bar{A}_1$ is in P . So,

$$\text{dens}_{bn}(E_y) = (1/b) \cdot \text{dens}_n(f_y(E_y))$$

and

$$\lim_{n \rightarrow \infty} \text{dens}_n(E_y) = 0,$$

since $f_y(E_y) \subseteq \bar{A}_1$ is in P .

Thus, $E = \bigcup_{y=a}^{b-1} E_y$ implies that $\lim_{n \rightarrow \infty} \text{dens}_n(E) = 0$. So,

$$\limsup_{n \rightarrow \infty} \text{dens}_n(C) \leq \limsup_{n \rightarrow \infty} \text{dens}_n(D) + \limsup_{n \rightarrow \infty} \text{dens}_n(E) \leq r.$$

Hence, condition (i) is satisfied by A .

(ii) Let $B = \emptyset$ and $C = \{bx + y : 0 \leq y \leq a\}$. Then B and C satisfy condition (ii). \square

These three theorems tell us that the knowledge that a set is \leq_m^p -complete in EXP does not give us any information about its approximability. The next theorem, which can be proved using a similar technique, shows that, for some \leq_m^p -complete sets in EXP, the situation may be even worse.

THEOREM 9. *There exists a \leq_m^p -complete set in EXP such that for any $\epsilon > 0$,*

- (i) $(\forall B, C \in P)[B \subseteq A, C \subseteq \bar{A} \Rightarrow \liminf_{n \rightarrow \infty} \text{dens}_n(B \cup C) = 0]$
- (ii) $(\exists B, C \in P)[B \subseteq A, C \subseteq \bar{A} \ \& \ \limsup_{n \rightarrow \infty} \text{dens}_n(B \cup C) = 1].$

5. Summary and open questions. We have studied polynomial time approximation to sets in EXP. In summary, we review the following five statements.

1. A set in EXP may be so hard that its polynomial complexity core is exactly Σ^* .
2. A \leq_n^p -complete set in EXP may be so hard that it is p -immune.
3. A \leq_m^p -complete set in EXP cannot be p -immune.
4. The polynomial complexity core of a \leq_m^p -complete set in EXP may be of any size.
5. The polynomial complexity core of a \leq_m^p -complete set in EXP is larger than any p -sparse set. [9]

What about the sets in NP? Can we show similar results? Recently, P. Berman [4] and Meyer and Paterson [14] solved question 5 for the class of NP-complete sets; namely, the following problems do not have p -sparse complexity cores unless they are in P:

NP-complete problems, PSPACE-complete problems, the primality problem and the graph isomorphism problem.

The other four questions remain open. We believe that any new results concerning these questions will lead to better understanding of the structure of the intractable problems.

REFERENCES

- [1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974, Chapter 10.
- [2] T. BAKER, J. GILL AND R. SOLOVAY, *Relativizations of the $P = ?NP$ question*, this Journal, 4 (1975), pp. 431–442.
- [3] 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.
- [4] P. BERMAN, *Relationship between density and deterministic complexity of NP-complete languages*, Fifth International Coll. Auto. Lang. and Programming, Lecture Notes in Computer Science, 62, Springer, New York, 1978, pp. 63–71.
- [5] M. BLUM AND I. MARQUES, *On complexity properties of recursively enumerable sets*, J. Symb. Log., 38 (1973), pp. 579–593.
- [6] M. R. GAREY AND D. S. JOHNSON, *Approximation algorithms for combinatorial problems* (annotated bibliography), Algorithms and Complexity, J. F. Traub, ed., Academic Press, New York, 1976.
- [7] ———, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.
- [8] J. GILL AND P. H. MORRIS, *On subcreative sets and S-reducibility*, J. Symb. Log., 39 (1974), pp. 669–677.
- [9] J. HARTMANIS AND L. BERMAN, *On isomorphisms and density of NP and other complete sets*, this Journal, 6 (1977), pp. 305–322.
- [10] R. M. KARP, *The probabilistic analysis of some combinatorial search algorithms*, Algorithms and Complexity, J. F. Traub, ed., Academic Press, New York, 1976, pp. 1–19.
- [11] R. LADNER, N. LYNCH AND A. L. SELMAN, *Comparison of polynomial time reducibilities*, Proc. Sixth Symposium on Theory of Computing, 1974, pp. 110–120.
- [12] N. LYNCH, *Approximations to the halting problem*, J. Comput. Syst. Science, 9 (1974), pp. 143–150.
- [13] ———, *On reducibility to complex or sparse sets*, J. Assoc. Comput. Mach., 22 (1975), pp. 341–345.
- [14] A. R. MEYER AND M. S. PATERSON, *With what frequency are apparently intractable problems difficult?* MIT/LCS/TM-126, MIT, Cambridge, MA, 1979.
- [15] H. ROGERS, JR., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967.
- [16] S. SAHNI AND T. GONZALEZ, *P-complete approximation problems*, J. Assoc. Comput. Mach., 23 (1976), pp. 555–565.

AN EFFICIENT CYCLE VECTOR SPACE ALGORITHM FOR LISTING ALL CYCLES OF A PLANAR GRAPH*

MACIEJ M. SYSŁO†

Abstract. All known cycle vector space algorithms for listing cycles of a graph are inefficient, and in the worst case they compute all vectors of the cycle space. This is a very significant drawback of the cycle space approach. In this paper, a cycle vector space algorithm for enumerating all cycles of a planar graph, which produces only cycles of a graph and requires $O(n)$ space and $O(n + nc)$ time (where n and c denote the number of vertices and cycles of a graph, resp.), is presented. Thus we show that in the class of planar graphs, the cycle space approach can be as efficient as the backtrack algorithms.

Key words. all-cycle algorithm, cycle basis, cycle graph, cycle space method, planar graph

1. Introduction. A simple graph $G = \langle V(G), E(G) \rangle$ is a collection of vertices $V(G)$ and edges $E(G)$. Let $n(G)$ and $m(G)$ denote the number of vertices and the number of edges of a graph G , resp. An edge is denoted by $e = uv$. Simple graphs do not contain loops and multiple edges, and only such graphs are considered in this paper.

A sequence of distinct vertices (v_0, v_1, \dots, v_l) , $l \geq 0$, such that $v_i v_{i+1} \in E(G)$, $i = 0, 1, 2, \dots, l-1$, is called a simple path from v_0 to v_l . A closed simple path, that is if $v_0 = v_l$, $l \geq 2$, is called a cycle. A path is nontrivial if $l \geq 1$. In what follows, we shall use sometimes the same symbol to denote a cycle and the set of its edges.

It is well known that the empty set, the set of all cycles and edge-disjoint unions of cycles of G , is a vector space called the cycle space of G , over $GF(2)$, the field of integers modulo 2 with the vector addition of elements defined as the ring sum of sets of edges. A cycle basis of G is defined as a basis for the cycle space of G which consists entirely of cycles. The dimension of the cycle space of a graph G is equal to $\mu(G) = m(G) - n(G) + p(G)$, where $p(G)$ denotes the number of connected components of G .

The cycles of a graph are minimal elements of the cycle space of G in the sense that no cycle can properly contain any nonzero element of the space. The set of cycles of a graph is a subset of this space and is generally proper. The cycle space of a graph G consists of $2^{\mu(G)}$ vectors but there are only four reduced graphs with $c(G) = 2^{\mu(G)} - 1$, where $c(G)$ denotes the number of cycles of G . A simple graph is said to be reduced, if G has no vertex of degree 0 or 1 and, for every vertex of degree 2, the two vertices adjacent to it are themselves adjacent. Mateti and Deo proved in [5] (see also [6]) that only the four reduced graphs K_3 , K_4 , $K_4 - x$ and $K_{3,3}$ have all vectors as cycles, where K_p and $K_{p,q}$ denote the complete graph on p vertices and the complete bipartite graph on $p + q$ vertices, resp.

Mateti and Deo also presented several classes of graphs for which the ratio of the number of cycles to the number of all vectors in the cycle space goes asymptotically to zero as the number of vertices increases, i.e., $\lim_{n \rightarrow \infty} (c(G)/2^{\mu(G)}) = 0$. For instance, for a wheel graph W_n we have $\mu(W_n) = n - 1$ and $c(W_n) = (n - 1)(n - 2) + 1$.

There are several problems which depend on finding a certain cycle (see [1] and [9]), a subset of cycles [4] and all cycles of a graph [5], which can be solved by using the cycle space methods. A cycle space method applied to a graph G starts from a cycle basis of G and produces the desired solution as an element or a subset of elements of

* Received by the editors April 6, 1979, and in revised form December 16, 1980. A preliminary version of this paper was presented at the International Colloquium on Algebraic Methods in Graph Theory, August 23-31, 1978, Szeged, Hungary and appeared in the Proceedings of the Colloquium (North-Holland, 1980).

† Institute of Computer Science, University of Wrocław, Pl. Grunwaldzki 2/4, 50-384 Wrocław, Poland.

the cycle space of G . For instance, to obtain all cycles of a graph G , one can start from a cycle basis of G and compute all the $2^{\mu(G)} - \mu(G) - 1$ vectors. Since in general not every vector represents a cycle, it is necessary to test whether the vector generated corresponds to a cycle. An attempt to generate only a subset of all vectors and yet enumerate all cycles of a graph has been made in several papers, but in the worst case, all known cycle vector space algorithms are inefficient and compute all the $2^{\mu(G)}$ vectors [5]. It is very significant drawback of the cycle space approach.

The main purpose of this paper is to present a cycle space algorithm for listing all cycles of a planar graph which requires $O(n)$ space and $O(n + nc)$ time. Thus we show that, in the class of planar graphs, the cycle space approach and backtrack algorithms are of the same efficiency (for some backtrack algorithms see [5] and [6]).

2. The cycle graph approach. Let $\mathcal{C} = \{C_i\}_I$ be an arbitrary cycle basis of a graph G . The intersection graph $B(G, \mathcal{C})$ of \mathcal{C} over the set of edges $E(G)$ is called a *cycle graph of G with respect to \mathcal{C}* and defined $V(B(G, \mathcal{C})) = \mathcal{C}$, with C_i and C_j adjacent whenever $i \neq j$ and $C_i \cap C_j \cap E(G) \neq \emptyset$ [5]. To distinguish between elements of graphs and their cycle graphs we will refer to *vertices* of the graphs and *nodes* of the cycle graphs.

Let G be a planar graph, and without loss of generality assume that G is 2-connected. For a given embedding of G in the plane, the set of the interior faces is a cycle basis of G and such a cycle basis of the plane graph G is called a *plane cycle basis* of G . The cycle graph of G with respect to the plane cycle basis of G is denoted by G^c . Notice, that in the class of 2-connected planar graphs, one can consider the notion of the cycle graph as a generalization of that of the dual graph. If G^* denotes the geometric dual graph of G then it is easy to see that G^c is isomorphic to the simple graph of $G^* - v_{\text{ex}}$, where v_{ex} denotes the vertex of G corresponding to the exterior face of G and a *simple graph of a multigraph H* is defined as the maximal spanning subgraph of H , which is a simple graph. Therefore G^c is planar, and in the sequel we assume that for a plane graph G , G^c is also plane (i.e., it has been obtained from G^* by removing vertex v_{ex} and replacing each set of parallel edges by one edge) and a subgraph F of G^c is also plane (i.e., it has been obtained from G^c by removing those nodes and edges which do not belong to F).

A graph F is said to be a *cycle graph* if there exist a graph G and its cycle basis \mathcal{C} , such that F and $B(G, \mathcal{C})$ are isomorphic. Some partial characterizations of cycle graphs are presented in [8].

Let G be a graph, $\mathcal{C} = \{C_i\}_I$ be one of its cycle bases, and $\bigoplus_J C_i$ denote the ring-sum of cycles $\{C_i\}_J$, where $J \subseteq I$. It is evident that if C is a cycle of G and $C = \bigoplus_{I'} C_i$ then for any partition $I'_1 \cup I'_2$ of I' into two nonempty subsets, $\bigoplus_{I'_1} C_i$ intersects $\bigoplus_{I'_2} C_i$. Therefore, the subgraph of $B(G, \mathcal{C})$ induced by $\{C_i\}_{I'}$ is connected, and in the listing of cycles of G only those subsets of vertices of $B(G, \mathcal{C})$ which induce connected subgraphs of $B(G, \mathcal{C})$ are of interest. However, in general this correspondence between cycles of G and connected induced subgraphs of $B(G, \mathcal{C})$ is not necessarily onto.

The algorithm proposed by Mateti and Deo [5] for enumerating all cycles of a graph utilizing a cycle graph depends on generating connected induced subgraphs of a cycle graph and then testing whether the corresponding elements of the cycle space are cycles.

In general, this approach is also inefficient. Let us consider a modified wheel graph U_n embedded in the plane as is shown in Fig. 2.1. We shall prove that the ratio of the number of cycles of U_n to the number of all connected induced subgraphs of U_n^c tends

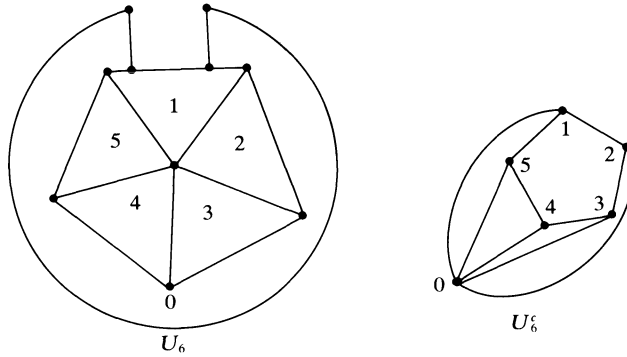


FIG. 2.1

to zero. Let $c_0(U_n)$ and $c_1(U_n)$ denote the number of cycles of U_n which are contained in W_n and the number of cycles of U_n which are the ring-sums of cycle 0 and of some other cycles contained in W_n , respectively. Notice that $0 \oplus \bigoplus_{i \in I} i$ is a cycle of U_n if and only if $I = N$, where $N = \{1, 2, \dots, n-1\}$ or I does not contain cycle 1 and $\bigoplus_{i \in I} i$ is a cycle, since $0 \oplus 1 = C_{\text{ex}} \cup \bigoplus_{N-\{1\}} i$, where C_{ex} denotes the exterior cycle of U_n , i.e., $0 \oplus 1$ is the union of edge-disjoint cycles. Therefore, we have $c(U_n) = c_0(U_n) + c_1(U_n)$, where $c_0(U_n) = (n-1)(n-2) + 1$ and $c_1(U_n) = (n-1)(n-2)/2 + 2$. On the other hand, let $s(U_n^c)$ denote the number of connected induced subgraphs of U_n^c . We have $s(U_n^c) = s_0(U_n) + 2^{n-1}$, where $s_0(U_n)$ is the number of connected induced subgraphs of U_n^c which do not contain node 0 and 2^{n-1} is the number of those which contain node 0. Thus, $\lim_{n \rightarrow \infty} (c(U_n)/s(U_n^c)) = 0$.

Notice that if a graph U_n is embedded in the plane, as shown in Fig. 2.2, then $c(U_n^c) = s(U_n^c)$.

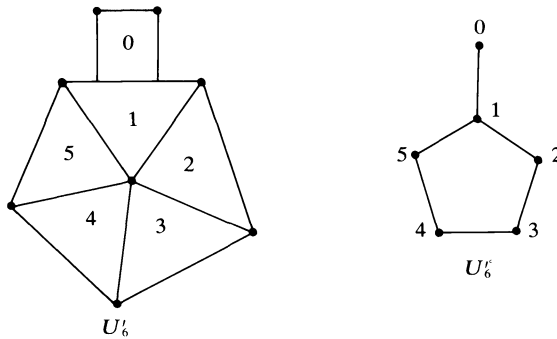


FIG. 2.2

Figure 2.3 shows a graph for which $c(H) < s(B(H, \mathcal{C}))$ for every cycle basis \mathcal{C} of H , for details see [11] and [12]. Thus, a graph can have no cycle basis for which there exists a one-to-one correspondence between the cycles of the graph and connected induced subgraphs of the cycle graph. Hence, the following question arises.

Problem [5]. Let G be a graph. What cycle basis \mathcal{C} of G should be chosen in order for $B(G, \mathcal{C})$ to have a minimum number of connected induced subgraphs?

Mateti and Deo suggested that maybe $B(G, \mathcal{C})$ has a minimum number of connected induced subgraphs for a cycle basis \mathcal{C} for which the sum of the lengths of

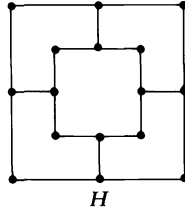


FIG. 2.3

the basic cycles is a minimum or the number of edges in $B(G, \mathcal{C})$ is a minimum. Paper [10] contains some counterexamples to these conjectures.

The success of the cycle space algorithm proposed by Mateti and Deo depends on the efficiency of an algorithm for listing the subsets of nodes of a cycle graph which induce connected subgraphs and on the method for finding a “good” cycle basis of a graph.

In view of the above results, two approaches to improving the original algorithm are possible. First, for a given graph G , we can apply the original algorithm to a cycle graph of G with respect to a cycle basis which minimizes the number of connected induced subgraphs of the cycle graph. Since nothing is known about such a cycle basis or about the method for finding it, at present, we are not able to develop such an approach. Secondly, one can try to find a suitable cycle basis and modify the algorithm to obtain the method which prunes unnecessary computations of connected induced subgraphs which do not correspond to cycles.

In the next section, using the second approach to planar graphs, we develop a cycle space algorithm for listing all cycles of a planar graph which has the same asymptotic space and time requirements as the most efficient backtrack algorithms have.

3. An efficient cycle space algorithm for listing all cycles of a planar graph. In the sequel only plane graphs are considered. Moreover, if G is a plane graph then we take G^c as $G^* - v_{ex}$. Therefore G^c is also plane, and every subgraph F of $G(G^c)$ is considered as a plane graph; that is, F can be obtained from $G(G^c)$ by removing edges and vertices (nodes) from $G(G^c)$, which do not belong to F . Thus, interior and exterior vertices, nodes and edges are well-defined notions.

We now prove a lemma which then will be used as the main tool in pruning unnecessary computations.

LEMMA 3.1. *Let G be a 2-connected plane graph and $\mathcal{C}_p = \{C_i\}_I$ be its plane cycle basis. If $\bigoplus_J C_i$ is a cycle of G then J contains all the interior nodes, of the subgraph of G^c , which is induced by J .*

Proof. Let $J \subset I = V(G^c)$ and let K denote the set of those the interior nodes of the subgraph of G^c induced by J which do not belong to J . Suppose that $K \neq \emptyset$. If C_{ex} denotes the exterior cycle of G , then

$$C_{ex} = \bigoplus_I C_i = \bigoplus_{I-J-K} C_i \oplus \bigoplus_J C_i \oplus \bigoplus_K C_i,$$

and hence,

$$\bigoplus_J C_i = C_{ex} \oplus \bigoplus_{I-J-K} C_i \oplus \bigoplus_K C_i.$$

Let us notice that C_i and C_j have no common edge for $i \in I \cup \{ex\} - J - K$ and $j \in K$, that is $\bigoplus_J C_i$ is not a cycle. This proves the lemma. \square

A connected induced subgraph of G^c which contains all its interior nodes is called a *full subgraph* of G^c . It is not hard to see that every full subgraph F of G^c can be obtained from G^c in the following way: The nodes of $V(G^c) - V(F)$ can be ordered as v_1, v_2, \dots, v_l , such that F_i is a full subgraph of G^c induced by $V(G^c) - \{v_1, v_2, \dots, v_i\}$, where $i = 1, 2, \dots, l$ and then $F_l = F$.

Figure 3.1 shows that there exists a graph G such that not every full subgraph of G^c which can be obtained from G^c in this way corresponds to a cycle of G . Namely, F can be obtained from G^c by removing node 1 but it corresponds to $1 \oplus 2 \oplus 3 \oplus 4 \cup 1$.

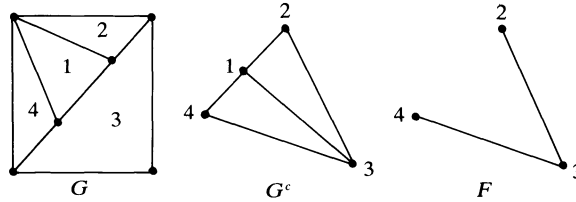


FIG. 3.1

A subgraph of G^c which corresponds to a cycle of G is called a *feasible subgraph* of G^c . Evidently, every feasible subgraph of G^c is connected and full. Let F be a feasible subgraph of G^c ; then $v \in V(F)$ is a *feasible node* of F if the subgraph of G^c induced by $V(F) - \{v\}$ is also feasible. The following lemma characterizes feasible nodes.

LEMMA 3.2. *Let F be a feasible subgraph of G^c . Then $v \in V(F)$ is feasible if and only if v is an exterior node of F and the cycle C_v of G corresponding to v has exactly one nontrivial (i.e., different from a vertex) common path with the cycle $C_F = \bigoplus_{v \in V(F)} C_v$.*

Proof. By Lemma 3.1, if $V(F) - \{v\}$ corresponds to a cycle in G then v is an exterior node of F . The subgraph of F induced by $V(F) - \{v\}$ corresponds in G to $C_v \oplus C_F$ which is a cycle in G if and only if C_v and C_F have exactly one nontrivial common path. \square

Therefore, v is not a feasible vertex of a feasible subgraph F of G^c if and only if C_v has more than one common path with C_F , or C_v has no common path with C_F . In the former case, C_v cuts C_F into at least two edge-disjoint cycles, therefore v disconnects F , and in the latter, v can be an interior as well as an exterior node of F . It is interesting that once the exterior nodes of G^c are labeled as feasible or infeasible, all the feasible subgraphs of G^c can be generated without referring to the graph G . This follows from the following lemma.

LEMMA 3.3. *Let v be a feasible node of a feasible subgraph F of G^c and let F' be the subgraph induced by $V(F) - \{v\}$. Then $u \in V(F')$ is a feasible node of F' if and only if*

- (1) u is not a cutnode of F' ; and
- (2) either u is a feasible node of F or u is adjacent to v in F .

Proof. Let $u \in V(F') = V(F) - \{v\}$, where v is a feasible node of a feasible subgraph F of G^c . If u is feasible in F , i.e., if C_u has exactly one nontrivial common path with C_F then it is clear that u is feasible in F' provided u has not become a cutnode after removing v from F . If u is not feasible in F and is adjacent to v in F , i.e., if C_u and C_v have at least one common edge, then u also becomes a feasible node of F' provided u is not a cutnode of F' . Conversely, if u is a feasible node of F' then evidently u is not a cutnode of F' and C_u has exactly one nontrivial common path with $\bigoplus_{w \in V(F')} C_w = \bigoplus_{w \in V(F)} C_w \oplus C_v$, where C_v has exactly one nontrivial common path with $\bigoplus_{w \in V(F)} C_w$. Therefore, either u is feasible or u is adjacent to v in F . \square

It is not hard to see that for every feasible subgraph F of G^c the nodes of $W = V(G^c) - V(F)$ can be ordered as v_1, v_2, \dots, v_l such that every F_i is a full subgraph of G^c and v_i is a feasible node of F_{i-1} , where $F_0 = G^c$, $F_l = F$, and F_i is the subgraph of G^c induced by $V(G^c) - \{v_1, v_2, \dots, v_i\}$ for $i = 1, 2, \dots, l$.

Thus, we have succeeded in finding a method which for a 2-connected plane graph G determines those induced subgraphs of G^c which correspond one-to-one with the cycles of G . In what follows, we shall show how to implement this method efficiently.

A feasible subgraph F of G^c can be uniquely represented by using the sequence of its exterior face nodes listed in a clockwise order and additionally labeled as feasible or infeasible; this will be called a *code* of F and denoted $d(F)$. Notice that, for $v \in d(F)$, v is a cutnode of F if and only if v appears in $d(F)$ at least twice.

Figure 3.2(a) shows the cycle graph U_6^c of the planar graph U_6 , embedded as is shown in Fig. 2.1, and its code in which the feasible nodes are underlined. Figures 3.2(b) and (c) show two feasible subgraphs of U_6^c and their codes obtained from U_6^c by removing feasible nodes 0 and 1, resp.

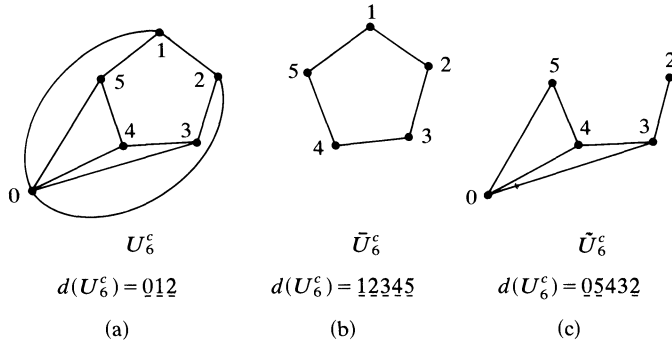


FIG. 3.2

We now show how all feasible subgraphs of G^c can be generated.

Let \mathcal{G} denote the family of all feasible subgraphs of G^c and $\mathcal{G}(U, W)$ denote the family of all feasible subgraphs of G^c which contain all nodes of U and do not contain the nodes of W and possibly some other vertices of $V(G^c) - U - W$. Let (v_1, v_2, \dots, v_p) be the sequence of all feasible nodes of G^c listed in a clockwise order and let $W_j = (v_1, v_2, \dots, v_j)$, where $0 \leq j \leq p$ and we assume $W_0 = \emptyset$. One can easily show that $\mathcal{G}(W_g, \{v_{g+1}\}) \cap \mathcal{G}(W_h, \{v_{h+1}\}) = \emptyset$ ($g, h = 0, 1, \dots, p, g < h$), where we assume $\mathcal{G}(W_p, \{v_{p+1}\}) = \{G^c\}$, since every subgraph of the former family does not contain node v_{g+1} , which belongs to every subgraph of the latter one. It is also evident that $\bigcup_{j=0}^p \mathcal{G}(W_j, \{v_{j+1}\}) = \mathcal{G}$, since for every feasible subgraph F of G^c either F is isomorphic to G^c or there exists k ($1 \leq k \leq p$) such that $v_j \in V(F)$ for $j = 1, 2, \dots, k-1$, and $v_k \notin V(F)$ therefore $F \in \mathcal{G}(W_{k-1}, \{v_k\})$. This process of partitioning \mathcal{G} into nonempty and disjoint subfamilies can again be applied to each of the subfamilies. For instance, to partition $\mathcal{G}(W_{k-1}, \{v_k\})$, $1 \leq k \leq p$, let us consider the feasible subgraph G_k^c obtained from G^c by removing node v_k and all edges incident with v_k . Let $(v_1, v_2, \dots, v_{k-1}, v'_k, v'_{k+1}, \dots, v'_{p_k})$ be the sequence of exterior nodes of G_k^c listed in clockwise order such that v'_k, \dots, v'_{p_k} are the feasible nodes of G_k^c and lie on the exterior cycle of G_k^c between v_{k-1} and v_1 . It is easy to verify that if $u_1, u_2, \dots, u_l, l \geq 0$, are infeasible nodes of G^c lying between v_{k-1} and v_k , then either $v'_k = u_l$ if $l > 0$ and u_l is not a cutnode in G_k^c , or v'_k is one of the nodes which follow u_l in the code of G_k^c . Let $W_j^k = \{v_1, v_2, \dots, v_{k-1}, v'_k, \dots, v'_j\}$, where $k-1 \leq j \leq p$, and assume $v'_{k-1} = v_{k-1}$.

One can easily verify that $\mathcal{G}(W_g^k, \{v'_{g+1}\}) \cap \mathcal{G}(W_h^k, \{v'_{h+1}\}) = \emptyset$ ($g, h = k - 1, \dots, p_k, g < h$), where we assume $\mathcal{G}(W_{p_k}^k, \{v'_{p_k+1}\}) = \{G^c_k\}$, and

$$\bigcup_{j=k-1}^{p_k} \mathcal{G}(W_j^k, \{v'_{j+1}\}) = \mathcal{G}(W_{k-1}, \{v_k\}).$$

Therefore we have the following lemma.

LEMMA 3.4. *Let (v_1, v_2, \dots, v_q) be the code of a feasible subgraph F of G^c , and let \mathcal{F}_k denote the family of all feasible subgraphs of F which contain the nodes $\{v_1, v_2, \dots, v_k\}$, where $1 \leq k < q$. Let $(v_{k_1}, v_{k_2}, \dots, v_{k_l})$, where $k < k_1 < k_2 < \dots < k_l \leq q$, denote the sequence of all feasible nodes of F listed in a clockwise order and lying on the exterior cycle of F between v_k and v_1 . Then, \mathcal{F}_k can be partitioned into the subfamilies $\mathcal{F}_k(W'_{j-1}, \{v_{k_j}\})$, where $W'_{j-1} = \{v_1, v_2, \dots, v_k, v_{k+1}, \dots, v_{k_{j-1}}\}$ and $j = 1, 2, \dots, l$.*

It is easy to see that the generation of all feasible subgraphs of G^c by partitioning as it is shown above corresponds to a breadth-first search of \mathcal{G} .

We are now ready to present a cycle space algorithm for listing all cycles of a planar graph G , which utilizes a cycle graph with respect to a plane cycle basis of G and is based on the above consideration.

ALGORITHM. Let G be a simple graph with n vertices and m edges given in the list form $\{A(v) | v \in V(G)\}$. Without loss of generality we may assume that G is 2-connected and $m \leq 3n - 6$. If G is not 2-connected then, using a depth-first search, it can be divided into biconnected components in time $O(n)$.

- Step 1. Test whether G is planar. If G is planar then embed it in the plane and change the initial list representation of G into that in which for every vertex v , the edges incident to v are listed in $A(v)$ in clockwise order according to the plane embedding of G . Such a representation of a planar graph G is called a *plane list representation* of G .
- Step 2. Number the faces of G and give each edge e of G two labels, which are the numbers of the faces containing e .
- Step 3. Using the cycle numbers and the edge labels obtained in step 2, construct the plane list representation $\{B(v) | v \in V(G^c)\}$ of the plane cycle graph G^c of G .
- Step 4. Find the code $d(G^c)$ of G^c and label the elements of $d(G^c)$ as feasible or infeasible.
- Step 5. Starting from $d(G^c)$ and using only the plane list representation $\{B(v)\}$ of G^c generate the codes of all feasible subgraphs of G^c .

Step 5 is the core of the algorithm. Notice that in fact the generation of cycles of G is performed on G^c , which indicates only the mutual relations between the basic cycles of G . The main advantage of such an approach is that once the graph G^c is constructed and its code determined, the basic cycles of G are represented by single elements—the nodes in G^c and all other cycles of G can be generated in unique form as the ring-sums of the basic cycles of G , without referring to G . Evidently, every cycle of G can be also produced in the usual form as a sequence of edges of G . The former form however may be more convenient in some applications than the latter.

We now discuss an implementation of steps 1–5 which gives rise to an algorithm with the same asymptotic time and space requirements as for the backtracking algorithm presented in [7].

A graph G can be tested for planarity by using a modified version of the algorithm of Hopcroft and Tarjan [3], which if G is planar also constructs in $O(n)$ time a plane representation of G , so that a plane list representation of G can easily be obtained.

Let r be a fixed vertex of G belonging to the exterior cycle of G . In step 2, first, we give each exterior cycle edge of G label 0 and then, starting from vertex r we remove one interior face cycle of G at a time, and give all its edges the first unused label k . Simultaneously, the plane list representation $\{B(v)|v \in V(G^c)\}$ of G^c can be constructed. For details see [13], where an Algol procedure for constructing a dual graph and a cycle graph of a plane graph and their plane list representations, is presented. It is not hard to implement steps 2 and 3 in time bounded by $O(n)$.

The code $d(G^c)$ and labels of its elements can be derived from the elements of the exterior cycle of G by applying Lemma 3.2 and using the plane list representations of G and G^c . This step also requires $O(n)$ time.

Thus, steps 1–4 can be implemented in $O(n)$ time and it is also easy to see that these steps require $O(n)$ storage space.

As has already been noted, the method of listing all feasible subgraphs of G^c , which results from Lemma 3.4, corresponds to a breadth-first search of \mathcal{G} , the family of all feasible subgraphs of G^c . Since such an approach requires very much computer space, we present an implementation of step 5, which uses depth-first search of \mathcal{G} . The complete implementation of step 5 is presented below in an Algol-like notation. Procedure CYCLE GENERATION uses a plane list representation of G^c and starting with the code of G^c obtained in step 4 and stored in CYCLE, produces all feasible subgraphs of G^c , from which the cycles of G can easily be obtained. Array CYCLE is a double-linked array containing nodes of the exterior cycle of a current feasible subgraph of G^c , see Fig. 3.3 as an illustration. Array OCC contains the information about the nodes of G^c in a current feasible subgraph F of G^c : if a node v does not belong to the exterior cycle of F then $OCC[v] = 0$, otherwise $OCC[v] = 1$, if v is feasible in F ; $OCC[v] = -1$, if v is a cutnode of F ; and $OCC[v] = -2$ if v is infeasible and is not a cutnode of F . COUNT is equal to the number of nodes in a current feasible subgraph.

procedure CYCLE GENERATION;

comment Given a plane list representation and the code of G^c , procedure CYCLE GENERATION produces all feasible subgraphs of G^c from which all cycles of G can be obtained;

integer I, K ;

begin

procedure DELETE (IS, IT);

integer IS, IT;

comment Assume that F is a current feasible subgraph of G^c and $d(F) = (v_1, v_2, \dots, v_p)$. IS and IT are integers such that there exist l and k , where $1 \leq l, k \leq p$, and $CYCLE[IS] = v_l$ and $CYCLE[IT] = v_k$, respectively. Procedure DELETE generates the subfamilies $\mathcal{F}(W_{j-1}, \{v_j\})$ of the family \mathcal{F} of all feasible subgraphs of F , where $W_{j-1} = \{v_k, v_{k+1}, \dots, v_l, v_{l+1}, \dots, v_{j-1}\}$ ($j = l, l+1, \dots, k-1$) and v_j is a feasible node of F . All indices of array CYCLE are taken mod $p+1$, where p is the current length of array CYCLE;

begin

integer I, J, IS1, IT1, SW;

for I := IS, CYCLE[J] **while** I \neq IT **do**

begin

if OCC [CYCLE [I]] = 1

then begin

 A: delete feasible node CYCLE [I] from current feasible subgraph and modify arrays CYCLE and OCC;

```

B:  output the cycle of  $G$  which corresponds to current feasible subgraph of
 $G^c$ ;
COUNT := COUNT - 1;
if COUNT > 2
then begin
  if SW = 0
  then begin
    comment SW = 0 when every feasible node of the code of a
    current feasible subgraph can be removed, i.e., when no back-
    tracking step has been performed. In this case, IS1 is such that
    OCC[CYCLE[IS1]] = 1 and we set IT1 = IS1;
    IS1 := S;
    C:  while OCC [CYCLE [IS1]] ≠ 1 do IS1 := CYCLE [IS1 + 1];
    IT1 := IS1
  end SW = 0
  else begin
    comment When SW ≠ 0, then IS1 points either to the node
    which precedes node CYCLE [I] in current code provided it was
    not a feasible node, or to node CYCLE [I];
    IS1 := if OCC [CYCLE [CYCLE [I + 2]]] ≠ 1 then CYCLE
    [I + 2]
    if IS1 = IT then go to BACK;
    IT1 := IT
  end SW = 1;
  DELETE (IS1, IT1)
end COUNT > 2
else begin
  comment If COUNT = 2 then backtracking follows;
  SW := 1;
  go to BACK
end COUNT = 2
end OCC [CYCLE [I]] = 1;
J := I + 1
end I;
BACK: COUNT := COUNT + 1;
D:  add to current feasible subgraph the node which was deleted most recently
and modify arrays CYCLE and OCC;
end DELETE;
K := S; COUNT :=  $m - n + 1$ ; SW := 0;
comment S is given, such that CYCLE [S] is a node belonging to the exterior cycle
of  $G^c$ ;
for I := CYCLE [K] while OCC [I] ≠ 1 do K := CYCLE [K + 1];
comment CYCLE [K] is a feasible node of  $G^c$ ;
DELETE (K, K)
end CYCLE GENERATION;

```

In procedure DELETE for a current feasible subgraph F of G^c , the labeled code of a subgraph of G^c obtained from F by deleting a feasible, exterior node of F or by adding the most recently deleted node can be found easily by using the plane list representation of G^c to determine the elements of array CYCLE and applying Lemma

3.3 to determine values of array OCC. Details are left to the reader, who should not have any difficulty with the implementation of procedure steps *A* and *D* in time $O(n)$.

Step *B* outputs a current cycle of G . As already pointed out, a cycle which is to be printed out is of the form $C = \bigoplus_{i \in I-I'} C_i$, where $\{C_i\}_{I'}$ is the set of basic cycles which correspond to the nodes that do not belong to current feasible subgraph of G^c . Therefore, we may simply print out the elements of $I-I'$. On the other hand, we can also obtain C as a sequence of the graph vertices. To this end we should keep track of the cycle which corresponds to current feasible subgraph of G^c (this can be incorporated in steps *A* and *D* and implemented, also, in $O(n)$ time) and then use the edge sets of the basic cycles which can be produced in step 3 of the Algorithm. In both cases, step *B* can easily be implemented in $O(n)$ time.

Thus, after a feasible node v of current feasible subgraph is found, a new cycle which corresponds to the subgraph with v deleted can be generated in $O(n)$ time. Since the number of calls on DELETE is c , where c is the number of cycles of G , the total time required by procedure CYCLE GENERATION is $O(n + cn)$.

It can easily be checked that the total storage required by the procedure is $O(n)$.

Therefore the algorithm can be implemented in $O(n + cn)$ time and it requires $O(n)$ storage.

Thus we have presented the method, which for planar graphs is asymptotically as efficient as the backtrack algorithms; see [7].

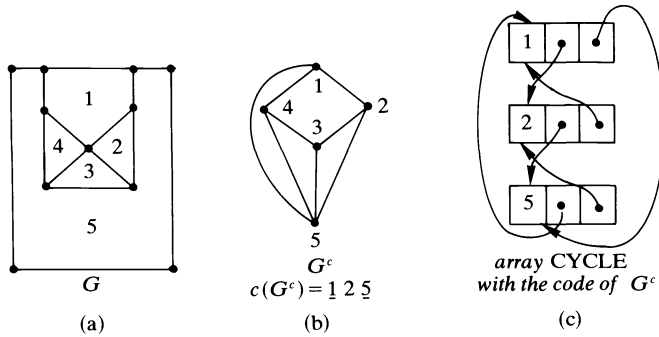


FIG. 3.3

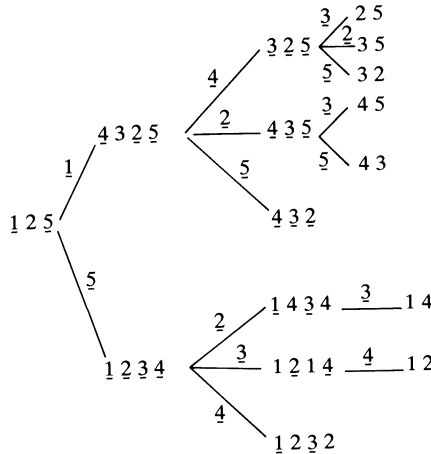


FIG. 3.5. The tree of codes generated by the algorithm for the graph shown in Fig. 3.3. Numbers on arrows indicate nodes which are deleted. The codes are generated from left to right and from top down.

TABLE 3.4

List of codes produced by the algorithm for the graph of Fig. 3.3. Numbers with signs in the column "node operation" indicate nodes which are either deleted in statement A (sign -) or added in statement D (sign +).

Codes of feasible subgraphs	Actual parameters of DELETE (IS, IT)			node operation	COUNT	SW
	IS	IT	I			
1 2 5	1	1 = 4	1	1-	5	0
4 3 2 5	1	1 = 5	1	4-	4	0
3 2 5	1	1 = 3	1	3-	3	0
2 5				3+	2	1
3 2 5			2	2-	3	1
3 5				2+	2	1
3 2 5			3	5-	3	1
3 2				5+	2	1
3 2 5				4+	3	1
4 3 2 5			3	2-	4	1
4 3 5	2	1 = 4	2	3-	3	1
4 5				3+	2	1
4 3 5			3	5-	3	1
4 3				5+	2	1
4 3 5				2+	3	1
4 3 2 5			4	5-	4	1
4 3 2	4 = 1	1 = 4		5+	3	1
4 3 2 5				1+	4	1
1 2 5			3	5-	5	1
1 2 3 4	2	1 = 5	2	2-	4	1
1 4 3 4	3	1 = 5	3	3-	3	1
1 4				3+	2	1
1 4 3 4				2+	3	1
1 2 3 4			3	3-	4	1
1 2 1 4	3	1 = 5	4	4-	3	1
1 2				4+	2	1
1 2 1 4				3+	3	1
1 2 3 4			4	4-	4	1
1 2 3 2	4	1 = 5		4+	3	1
1 2 3 4				5+	4	1
1 2 5					5	1

Figures 3.3(a) and (b) show a graph and its cycle graph, which we shall use to illustrate the algorithm. Array CYCLE with the code of G^c is shown in Fig. 3.3(c). Codes of all feasible subgraphs of G^c , produced by the algorithm, together with some additional quantities used, are presented in Table 3.4 and Fig. 3.5.

4. Application and extension. Since every tree is planar, steps 4 and 5 of the algorithm presented in the previous section, applied to a tree, can be used for listing all subtrees of the tree. In this case, the algorithm can be simplified, since for a tree a node v is feasible if and only if it is a pendant node; all other nodes are cutnodes.

One might ask whether the method proposed in this paper can be extended to the case of an arbitrary graph. We conjecture that it is possible. It seems that first a complete characterization of cycle graphs should be found. Only some partial characterizations are known; however, the following lemma and theorem allow us to conjecture that there exists a characterization of cycle graphs in terms of their planar subgraphs.

LEMMA 4.1. *A triangle-free 2-connected graph G is a cycle graph if and only if G is a planar graph.*

THEOREM 4.1. *If a graph F is a cycle graph, then every triangle-free induced subgraph of F is planar.*

Some other partial characterizations of cycle graphs and fundamental cycle graphs (i.e., cycle graphs with respect to fundamental cycle sets) have been presented in [8].

Acknowledgment. The author is very grateful to the referee for many substantial corrections which improved the paper.

REFERENCES

- [1] E. T. DIXON AND S. E. GOODMAN, *An algorithm for the longest cycle problem*, Networks, 6 (1976), pp. 139–149.
- [2] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
- [3] J. HOPCROFT AND R. E. TARJAN, *Efficient planarity testing*, J. Assoc. Comput. Mech., 21 (1974), pp. 549–568.
- [4] E. HUBICKA AND M. M. SYSŁO, *Minimal bases of cycles of a graph*, Recent Advances in Graph Theory, Proc. Second Czechoslovak Symposium on Graph Theory, M. Fiedler, ed., Academia, Praha, 1975, pp. 283–293.
- [5] P. MATETI AND N. DEO, *On algorithms for enumerating all circuits of a graph*, UICDCD-R-73-585 (revised), Dept. of Computer Science, Univ. of Illinois, Urbana, IL., 1973; this Journal, 5 (1975), pp. 90–101.
- [6] S. B. MAURER, *Cycle-complete graphs*, in Proc. 5th British Combinatorial Conference, 1975, Congressus Numerantium XV, Utilitas Mathematica, Winnipeg, pp. 447–453.
- [7] R. C. READ AND R. E. TARJAN, *Bounds on backtrack algorithms for listing cycles, paths and spanning trees*, Networks, 5 (1975), pp. 237–252.
- [8a] M. M. SYSŁO, *On characterizations of cycle graphs*, Colloque CNRS, Problèmes Combinatoires et Théorie des Graphes, Orsay, 1976, Paris, 1978, pp. 395–398.
- [8b] ———, *On characterizations of cycle graphs and on other families of intersection graphs*, Rep. Nr. N-40, Inst. of Computer Science, University of Wrocław, Poland, June, 1978.
- [9] ———, *On cycle bases of a graph*, Networks, 9 (1979), pp. 123–132.
- [10] ———, *On some problems related to fundamental cycle sets of a graph: a research note*, Stefan Banach International Mathematical Center, Warsaw, 1981, to appear.
- [11] ———, *On two cycle space methods for listing cycles of a graph*, Rostock. Math. Kolloq., 11 (1979), pp. 115–122.
- [12] ———, *On the structure of cycles of a graph*, Bull. Acad. Polon. Sci., Ser. Sci. Math., 27 (1979), pp. 241–246.
- [13] ———, *An algorithm for constructing a dual graph and a cycle graph of a planar graph*, Rep. Nr. N-95, Inst. of Computer Science, University of Wrocław, Poland, September, 1980.

DISTRIBUTED PROCESSOR SCHEDULING AND USER COUNTERMEASURES*

AMNON B. BARAK[†] AND PETER J. DOWNEY[‡]

Abstract. Consider an m -processor distributed system having both local node and network traffic. Because users have incomplete information about the state of the queues at remote nodes, they can be tempted to shorten the expected completion time of a job by running multiple incarnations of each task in parallel on all m processors. To assess the temptation of such a "user countermeasure," a job consisting of a sequence of n tasks to be performed sequentially is considered. Under simple assumptions about execution and network times, users can cut their expected job completion time by a factor of the square root of m . The implications of such user countermeasures on system design are discussed.

Key words. Distributed computing, system performance analysis, queuing theory

1. Introduction. Suppose a husband and wife arrive at their bank to cash a check and find equally long queues in front of the two available tellers.¹ The couple can shorten the expected time to cash their check by each joining a different line. Assuming that every customer requires service time exponentially distributed with mean τ , and that each line has n customers (excluding our couple), then the expected waiting time for one or the other spouse to arrive at a teller is [Dow80]

$$\tau \left[n - \left(\frac{1}{2} \right)^{2n} n \binom{2n}{n} \right],$$

which asymptotically is

$$\tau \left[n - \frac{\sqrt{n}}{\sqrt{\pi}} \right].$$

By contrast, if the service time of each customer were exactly τ (no uncertainty) then each spouse would "expect" to wait τn units of time for service. There would then be no point in each joining a separate queue.

From this simple example we see that *variance* in the service distribution of tasks can induce customers to join many queues, provided that both task replication and task intercommunication are easy to accomplish. We shall refer to such replication as *cloning*, for while all tasks replicated are identical, their future life courses are unknown a priori and are independent. A distributed computing system provides an example of a service system in which there are many processors with queues, incomplete information for users, ready facilities for cloning tasks (programs or processes) and communication facilities designed specifically for the efficient intercommunication of widely dispersed tasks. In this paper we suggest that such a system is prone to an Achilles heel: users aware of the advantages of cloning tasks will be tempted to use such tricks; wide use of cloning will obviously degrade the system. In the bank line example, if all customers always clone, the bank may as well fire one of the two tellers.

* Received by the editors March 17, 1978, and in final revised form January 8, 1981.

[†] Department of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel.

[‡] Department of Computer Science, University of Arizona, Tucson, Arizona 85721. The work of this author was supported by the National Science Foundation under grant MCS80-04679.

¹ We assume this bank is not sophisticated about queuing theory, and so has not roped off a single multi-server queue. Many movie theatre ticket lines encourage the behavior described.

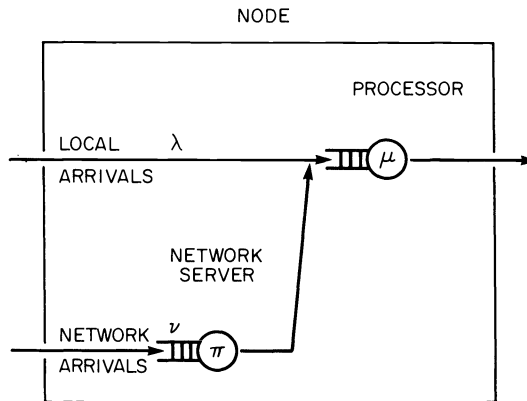


FIG. 1.1. Model of a network node.

In the present paper we will assess the payoff to a user who clones in a distributed system in which all other users “behave themselves,” and discuss implications of this result upon scheduling distributed systems.

Consider a distributed computer system having m nodes, each supported by its own processor. Each processor will serve *local* traffic—tasks entering the processor queue at the node, and *network* traffic—tasks entering the processor queue but originating from some other node. We will model network communication delays by assuming the existence of a network server along with each node. Refer to Fig. 1.1. If node i wishes to run a task on node j , the task is queued at the j th node’s network server. When the task leaves network server j it queues at processor j (along with local and other network traffic).

The network will be completely symmetric and homogeneous: all processors are assumed to be identical and every node can be reached from every other node with equal ease through the network. The model will not distinguish among different methods for internode communication; thus the network could be implemented by telecommunications, shared bus or shared memory, as long as symmetry is preserved.

Consider a potential user observing such a network. Because the user does not have complete information about future demands upon service centers in the network, he observes irregularity in the system. Local arrivals and network arrivals are irregularly spaced in time; the demands of tasks for processor time vary; different propagation times occur in the network because of differing message sizes. This irregularity leads to congestion at both the processor and network server.

Consider a user on processor i who wishes to run the *same* task T on two distinct processors j and k . In a network free of irregularities and uncertainties caused by competing arrivals, the execution time of T on both j and k would be identical. If such a system enjoyed a completely symmetric network, the propagation time to j and k from i would be identical. There would thus be no variation in the user’s response time from j or k . The user would have no inducement to replicate T . He should be content to execute T on his “home” processor i .

However, the completely deterministic situation described above is never the case. Irregularities in arrival and service patterns lead to congestion at the queues. The user has incomplete information about the *future* state of the queues at nodes i , j and k (and typically may not even know the current state of remote queues). This is much like the situation encountered by our couple in the bank line example described above. Rather than choosing one particular node’s queue for task T and sticking to

it, the user may wish to hedge by cloning task T and queuing the clones on nodes i , j , k and perhaps others. He can accept results from the first node that answers, and abandon the remaining clones.

The above considerations suggest that the irregularities in the network nodes be modeled by treating each node as a queuing system having a processor, network server and associated queues, as well as predicating a stochastic arrival process at each queue (Fig. 1.1). It is important to stress that while a user's own tasks have deterministic running times known to the user, incomplete information about the system state and contention for processors leads to the need for a stochastic model of the system as a whole.

Given a distributed system described above, suppose a user wishes to run a job consisting of a chain of n tasks in sequence (Fig. 1.2). Examples of such jobs are a lengthy UNIX pipeline [Rit74] or n iterations of a simulation or approximation step. Such a tasking structure has no inherent parallelism, and it is not obvious how to utilize multiple processors to decrease the expected finishing time of the job.

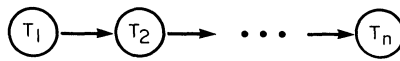


FIG. 1.2. Task chain C .

After describing a simple model of the distributed system in § 2, we will analyze in § 3 the expected finishing time of a user who methodically exploits cloning. Section 4 develops an asymptotic expansion for the expected finishing time. In § 5 we measure the speed-up to be expected from the use of cloning. We conclude that such speed-up is considerable for a large number of nodes, and hence does provoke the use of cloning as a scheduling countermeasure.

The model used to derive these results is overly simplistic; we will review its deficiencies and their effects upon the interpretation of our results.

Since ruthless cloning must be undesirable if widespread, we discuss the implications for the design of distributed systems.

2. The model. We will model each node of the distributed system as a feed-forward queuing network having two service stations and depicted in Fig. 1.1. All m nodes will be assumed statistically identical. Service at both the processor and network server will be assumed to be first-come-first-served.

We will make certain simplifying assumptions about the arrival and service distributions at each node. These assumptions are made for reasons of analytic tractability, but should allow us at least to explore the qualitative effects of cloning on the distributed system.

All tasks run on the system will be assumed statistically identical. Task execution times are assumed to be independent and identically distributed random variables having an exponential distribution with mean $1/\mu$. Similarly, task service times on the network server are independent, identically distributed exponential random variables with mean $1/\pi$. Local arrivals are governed by a Poisson process with rate λ , while network arrivals are assumed to be Poisson with rate ν . These assumptions are summarized in Fig. 1.1.

It should be noted that in our model of an individual node we are ignoring the effects of departures from the processor which feed back into the network of m nodes. Modeling such effects would not allow us to assume that network arrivals are Poisson [Kle76].

Imagine now that to such a system of m nodes, a user arrives bent upon cloning to reduce his expected job completion time. We will assume that no other users are employing cloning; this will be the situation of maximum temptation for our arriving user. The user wishes to run the task structure shown in Fig. 1.2.

The user's strategy is as follows. When he reaches the server on his "home" node, he immediately initiates task T_1 and queues $m - 1$ clones of this task on the other system nodes. Any clone which finishes sends a time-stamped notification of its finish, along with the resulting data, to every other node on the network, with instructions to run the next task in sequence (T_2). The sending clone will immediately initiate T_2 by entering this task in its processor queue. Only the notification from the first clone to finish will be acted upon by other nodes; all other clones finishing the task later will be ignored. Clones running T_1 which receive the message will immediately initiate T_2 at their node by entering this task at the end of the processor queue and terminating T_1 . If no clone is running when the message arrives, the task T_2 simply enters the end of the processor queue. Eventually, some clone becomes the first to finish T_2 , and the process of notification and queuing is repeated, with instructions to run T_3 . This strategy is continued until the first finish of T_n .

We may imagine that the code sequences for tasks T_1, \dots, T_n have been prepositioned in a local memory device at each node. This is particularly relevant, for example, when the tasks are operating system commands.

3. Analysis. Referring once again to Fig. 1.1, we can immediately simplify the description of the node. By Burke's theorem [Bur66] the departure process from the network server is Poisson with rate ν , and so the arrival process at the processor queue is Poisson with rate $\lambda + \nu$. Consider the subsystem consisting of the network server and queue. This is an M/M/1 queuing system, and the distribution of total system time (queuing plus service) is exponential with rate $\pi - \nu$ [Kle75]. Thus a network arrival perceives this subsystem as if it were an M/M/ ∞ system with service distribution exponential of rate $\pi - \nu$. In other words, an arrival does not queue but is immediately served at rate $\pi - \nu$.

Similar reasoning implies that the processor subsystem can be replaced by an M/M/ ∞ system with arrival rate $\lambda + \nu$ and service distribution exponential with rate $\mu - \lambda - \nu$.

In summary, an arrival from the network sees the node as a single M/M/ ∞ server with a service distribution which is the sum of two exponential stages of differing rates. The first (network) stage has service time which is exponentially distributed with mean

$$\sigma = \frac{1}{\pi - \nu}$$

and the second (processor) stage has service time which is exponentially distributed with mean

$$\tau = \frac{1}{\mu - \lambda - \nu}.$$

This equivalent configuration is summarized in Fig. 3.1.

From now on we refer to the node modeled as in Fig. 3.1, the *node server*. For convenience in the derivation below, we define the effective service rates for the stages

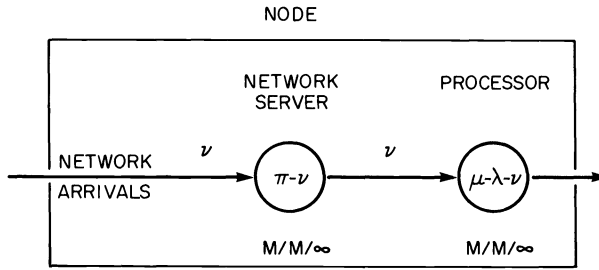


FIG. 3.1. Equivalent server.

of the node server as follows:

$$\alpha = \frac{1}{\sigma} = \pi - \nu$$

and

$$\beta = \frac{1}{\tau} = \mu - \lambda - \nu.$$

Baudet, Brent and Kung [Bau78], [Bau80], [Kun76] have analyzed what is in effect a special case of our model in which $\alpha = \beta$, in the context of modeling an asynchronous multiprocessing system such as C.mmp [Wul72]. These authors were the first to suggest the strategy of cloning as a method for reducing expected finishing time in the presence of multiple processors. The present paper can be considered a generalization of their work, reinterpreted in the context of distributed processing.

In the remainder of this section, we derive an expression for the expected time to the first finish of the job of Fig. 1.2 using the cloning strategy.

Let the random variable s_i , $1 \leq i \leq n$, denote the finishing time of the clone of T_i which finishes first; $s_0 = 0$. We shall call these epochs *useful finishes*. A typical realization under the cloning strategy is depicted in Fig. 3.2 for $m = 4$.

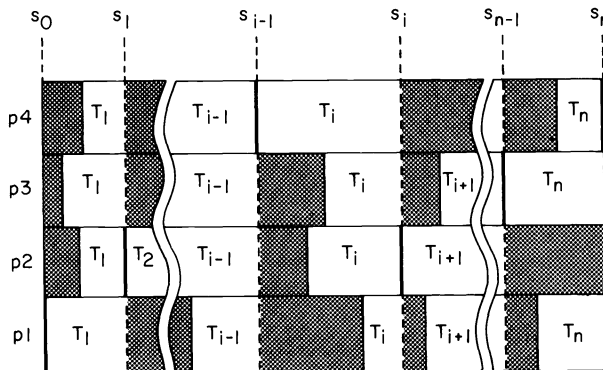


FIG 3.2. Realization of the cloning strategy. The shaded areas represent execution of U . Heavy bars are useful finishes of tasks in job C .

Whenever a useful finish occurs on crocessor, we imagine that a message is instantaneously sent to all other processors through the network. That is, the message to run the successor task is queued on the target node's network server. The service

time necessary for the message to reach the target processor is exponentially distributed with mean σ , as developed above; this is the service time of a task U run on the node server of Fig. 3.1. We will refer to it as the *network task*. Clearly each node server must process the network task U before initiating the appropriate task in chain C .

At time $s_0 = 0$, one processor (the user's "home" processor) begins execution of T_1 and all other node servers begin processing U . Since all chain tasks T_i are i.i.d., and all tasks U are i.i.d., and because of the memoryless property of the exponential distribution [Kle75], all of the "interfinish" random variables

$$t_i = s_i - s_{i-1}, \quad 1 \leq i \leq n$$

are i.i.d. Denote this random variable's expectation by t .

Our objective is to compute the expected time to the first finish of C , i.e., $E(s_n)$. Now

$$(3.1) \quad \begin{aligned} E(s_n) &= E(t_1) + \dots + E(t_n) \\ &= nt. \end{aligned}$$

Thus we need to concentrate on deriving an expression for the expectation t . It is enough to analyze the expectation $E(s_i - s_{i-1})$ for some i , $1 \leq i \leq n$. In what follows, let i be fixed. Reference to Fig. 3.2 will prove helpful.

Between the successive useful finishes s_{i-1} and s_i there are 0 or 1 or \dots or $m - 1$ finishes of network tasks U ; we call these *network finishes*. A network finish coincides with the starting time of a clone of T_i on the node server. These network finishes partition the time from s_{i-1} to s_i into 1 or 2 or \dots or m intervals. Task T_i may have its first finish at the end of interval 1 or 2 or \dots or m ; these possibilities are mutually exclusive and exhaustive.

Let us calculate the probability that the first finish of T_i occurs *at or after* the end of the j th interval. For $1 \leq j \leq m$, let

$$p_j = \Pr \{T_i \text{ first finishes at the end of some interval } k \geq j\}$$

and let

$$e_j = E(\text{length of interval } j).$$

Clearly, $p_1 = 1$. Also

$$p_2 = \frac{(m-1)\alpha}{\beta + (m-1)\alpha},$$

which is the probability that some one of the $m - 1$ tasks U of mean $1/\alpha$ finishes before the single task T_i of mean $1/\beta$. The expected length of this first interval is

$$e_1 = \frac{1}{\beta + (m-1)\alpha}.$$

Now T_i will finish at or after the end of the third interval if and only if it finishes past the first interval and one of the $m - 2$ tasks U in the second interval finishes before the *two* clones of T_i in that interval. Thus

$$p_3 = \frac{(m-1)\alpha}{\beta + (m-1)\alpha} \cdot \frac{(m-2)\alpha}{2\beta + (m-2)\alpha}$$

and

$$e_2 = \frac{1}{2\beta + (m-2)\alpha}.$$

In general for $1 \leq j \leq m$

$$(3.2) \quad p_j = \frac{(m-1)\alpha}{\beta + (m-1)\alpha} \cdots \frac{(m-j+1)\alpha}{(j-1)\beta + (m-j+1)\alpha},$$

$$e_j = \frac{1}{j\beta + (m-j)\alpha}.$$

To compute $t = E(s_i - s_{i-1})$ we can weight each e_j by the probability of its contribution to the expectation:

$$(3.3) \quad t = \sum_{j=1}^m p_j e_j.$$

Simplifying notation in (3.2) by setting

$$(3.4) \quad \delta = \frac{\beta}{\alpha} = \frac{\sigma}{\tau}$$

we obtain

$$(3.5) \quad p_j = \frac{(m-1)}{\delta + (m-1)} \cdots \frac{(m-j+1)}{(j-1)\delta + (m-j+1)},$$

$$e_j = \tau \frac{\delta}{j\delta + (m-j)}.$$

Finally, combining (3.1), (3.3) and (3.5) and simplifying yields

THEOREM 3.1. *The expected time to complete a chain C of n tasks on m processors using the cloning strategy is given by*

$$I(m, \delta) = n\tau K(m, \delta)$$

where for $\delta = \sigma/\tau$

$$(3.6) \quad K(m, \delta) = \frac{\delta}{m} \sum_{j=1}^m \frac{m!}{(m-j)!} \frac{1}{(m-(1-\delta)) \cdots (m-j(1-\delta))}.$$

From the above result we can see that the user who does not employ cloning, but runs job C on his home node, will expect a time of

$$(3.7) \quad I(1, \delta) = n\tau K(1, \delta) = n\tau = n\tau K(m, \infty) = I(m, \infty).$$

The latter equalities simply reflect the fact that huge network propagation delays limit the user to his home node only.

The first finish of T_n could occur on any node. If the user must have the results of his job reported back to his "home" node, we must allow for some notification time at the end of the run. If the user accepts results from the first node reporting back, a successful finish of T_n , then we have

COROLLARY 3.2. *The expected time to complete a chain C of n tasks using cloning on m processors and to notify the user at his home node is bounded by*

$$(3.8) \quad n\tau K(m, \delta) + \sigma = n\tau \left[K(m, \delta) + \frac{\delta}{n} \right].$$

In a distributed system with memory shared among all m processors, the "delay" term in (3.8) would not be needed. This is the model studied in [Bau78], [Bau80].

In the next section we will examine the behavior of (3.6) for large values of m.

4. Asymptotic approximation. To describe the behavior of $I(m, \delta)$ for fixed δ but a large number of processors m , we resort to an asymptotic approximation for the sum in (3.6). Following this development we will interpret our findings.

The sum $K(m, \delta)$ in (3.6) is a generalization of a sum treated by Knuth [Knu73, § 1.2.11.3, eq. (25)] using a different technique. The approach here will be to express the sum in terms of an integral, which then can be approximated by the method of Laplace [Olv74].

We first define $\varepsilon = 1 - \delta$ and rewrite (3.6) as

$$(4.1) \quad K(m, \delta) = \delta \frac{m!}{m} \sum_{j=1}^m \frac{1}{(m-j)!} \cdot \frac{1}{(m-\varepsilon) \cdots (m-j\varepsilon)}.$$

Observe that

$$(4.2) \quad \begin{aligned} & \frac{1}{(j-1)!} \int_0^1 (1-x)^{j-1} x^{m/\varepsilon-j-1} dx \\ &= \frac{1}{(j-1)!} \cdot \frac{\Gamma(j)\Gamma(m/\varepsilon-j)}{\Gamma(m/\varepsilon)} \text{ is a beta function,} \\ &= \frac{1}{(m/\varepsilon-1) \cdots (m/\varepsilon-j)} \\ &= \frac{\varepsilon^j}{(m-\varepsilon) \cdots (m-j\varepsilon)}. \end{aligned}$$

Substituting this into (4.1) gives

$$(4.3) \quad K(m, \delta) = \delta(m-1)! \sum_{j=1}^m \frac{\varepsilon^{-j}}{(m-j)!} \frac{1}{(j-1)!} \int_0^1 (1-x)^{j-1} x^{m/\varepsilon-j-1} dx.$$

The change of variable $k = j - 1$ gives

$$(4.4) \quad \begin{aligned} K(m, \delta) &= \delta \sum_{k=0}^{m-1} \binom{m-1}{k} \varepsilon^{-k-1} \int_0^1 (1-x)^k x^{m/\varepsilon-k-2} dx \\ &= \frac{\delta}{\varepsilon} \int_0^1 \sum_{k=0}^{m-1} \binom{m-1}{k} [(1-x)/\varepsilon x]^k x^{m/\varepsilon-2} dx \\ &= \frac{\delta}{\varepsilon} \int_0^1 \left[1 + \frac{1-x}{\varepsilon x} \right]^{m-1} x^{m/\varepsilon-2} dx \\ &= \frac{\delta}{\varepsilon} \int_0^1 \left[\frac{1}{\varepsilon} x^{1/\varepsilon-1} (1-\delta x) \right]^{m-1} x^{1/\varepsilon-2} dx. \end{aligned}$$

Now make the change of variable $u = x^{1/\varepsilon-1} = x^{\delta/\varepsilon}$. Then (4.4) becomes

$$(4.5) \quad K(m, \delta) = \int_0^1 [(1/\varepsilon)(u - \delta u^{1/\delta})]^{m-1} du.$$

The method of Laplace [Olv74] suggests that we express the integrand as $\exp(f(u))$ and expand $f(u)$ about the point where it achieves its maximum within the interval of integration. It is readily verified that this maximum point is $u = 1$. Next we change variables to bring this maximum point to the origin by setting $t = 1 - u$.

This yields

$$(4.6) \quad K(m, \delta) = \int_0^1 e^{-mp(t)} q(t) dt,$$

where

$$p(t) = -\ln \frac{1}{\epsilon} [(1-t) - \delta(1-t)^{1/\delta}],$$

$$q(t) = \epsilon [(1-t) - \delta(1-t)^{1/\delta}]^{-1}.$$

Since only the contribution near the peak value of the integrand is important, we approximate the integrand by using a Taylor expansion of $p(t)$ and $q(t)$ about $t=0$ and replace the upper limit of integration by ∞ . This yields the approximation

$$(4.7) \quad K(m, \delta) \sim \int_0^\infty e^{-mt^2/2\delta} dt.$$

This can be directly integrated to yield the first term of the asymptotic approximation

$$K(m, \delta) \sim \left(\frac{\pi\delta}{2m}\right)^{1/2}.$$

A more careful treatment of (4.6) allows us to compute higher order terms in the asymptotic expansion [Olv74]. The result is:

THEOREM 4.1. For large m and fixed δ

$$(4.8) \quad K(m, \delta) = \left(\frac{\pi\delta}{2m}\right)^{1/2} + \frac{(1-2\delta)}{3m} + O(m^{-3/2}).$$

The special case found in Knuth [Knu73, §1.2.11.3, eq. (25)] is identical to $mK(m, 1)$.

5. Interpretation. The foregoing sections were concerned with an expression for the expected time to complete a chain of n tasks on m nodes. However, we are usually interested in questions of relative merit: how much faster can C be completed by using cloning on m nodes than on a single processor?

Define the *speed-up factor* S achievable by cloning to be the time needed to complete job C on one node divided by the time needed if cloning on m nodes is used.

From (3.7) and (3.8) we have

$$(5.1) \quad S \cong \frac{n\tau}{n\tau[K(m, \delta) + \delta/n]} = \frac{1}{K(m, \delta) + \delta/n}.$$

For large m this gives, from (4.8), a lower bound of

$$(5.2) \quad S \cong \frac{n}{\delta} \left(1 - \frac{n}{\delta} \left(\frac{\pi\delta}{2m}\right)^{1/2} + O\left(\frac{1}{m}\right) \right).$$

Since much recent interest has centered on distributed systems having a very large number of identical microprocessors, these results indicate that a motivation to clone exists in such systems, provided network delays are not severe. Since one of the design objectives of such distributed systems is to make network delays as "invisible" as possible to encourage parallel processing, such a system will be particularly vulnerable to cloning.

One conclusion from (5.2) is that the ratio n/δ controls the amount of speed-up achievable by cloning. Network delay and number of tasks thus have directly inverse effects on speed-up, as long as the number of nodes is large relative to the number of tasks. As remarked, the direction of present and future research on distributed systems will be to *decrease* δ and *increase* m .

As more and more users employ cloning, the system degenerates from a distributed facility to m stochastically identical copies of the same uniprocessor system. What is needed is a method for discouraging such behavior.

In [Cof68], Coffman and Kleinrock evaluated scheduling methods for timesharing systems in terms of their vulnerability to countermeasures which a user might employ in an attempt to defeat the intent of the scheduling method. From this point of view, a scheduling policy is thought of as designed to encourage user behavior perceived as desirable for the normal running of the system. In timesharing systems, scheduling methods (such as generalized foreground-background [Cof68]) are designed to encourage short jobs. To be practical, such scheduling methods must also be simple, efficient and use only readily available information to discriminate among tasks.

In the context of distributed processing, most scheduling algorithms which have been suggested tend to encourage the use of multiple nodes where possible in completing a job. After all, such systems are presumably designed to facilitate the execution of jobs having highly parallel tasking structure. It would require an impractical amount of computation for a scheduler to discriminate, for example, between jobs consisting of n tasks in parallel and those consisting of n tasks in sequence. Furthermore, it is virtually impossible to tell a clone from a legitimately different task if the user is subtle enough.

We have remarked in §1 that cloning is attractive in the face of incomplete information about the future status of the system. It is unlikely that even complete instantaneous information about queue lengths at each node, even if it could be provided, would discourage a user bent on decreasing his expected finishing time.

The above observations indicate that cloning cannot be prevented in a distributed system. Keeping such behavior under control will have to be done by economic sanctions—charging a user for time spent on every processor.

With such a costing scheme, a user who clones will pay for every clone spawned to finish a task. The rate at which he pays to finish a job will be the average number of clones created per task. If a user has a job consisting of m tasks in parallel, and uses m processors to finish quickly, he pays the same cost as a user with m tasks in sequence who does not clone. Such legitimate uses of the distributed system are not penalized. The costing scheme allows cloning to be used, but at a premium rate.

Model limitations. The present model has limited applicability to real distributed systems, because of the many simplifying assumptions used. We enumerate here the directions for further improvement of the model.

The most severe restriction occurs in modeling the node server as a sequence of two exponential servers. More general service time distributions might be handled by decomposing them as a feedforward network of exponential stages [Kle76] and generalizing the derivation of Theorem 3.1 and the approximation of §4. Any service time distribution which has nonzero variance will be susceptible to cloning, but the degree to which the user is tempted to employ it will depend on the distribution.

The present model fails to take into account feedback effects in the network of m nodes. In particular, the arrival process at the network server should be generalized from the Poisson. Another feedback effect to be taken into account is the extra traffic fed back to the processor queue by the cloning of tasks.

The present analysis assumes only one user will clone. Clearly the effect upon the system of several cloning users needs to be examined.

In real systems, not all tasks make identical demands upon the processor. Real systems are never completely symmetrical and homogeneous, especially with regard to local traffic rates. Progress needs to be made in assessing the effect of such nonuniformities upon cloning.

6. Conclusion. We have suggested in this paper that a type of user behavior which is observed where there are multiple identical servers will occur in distributed systems, given their future direction. The behavior, called here *cloning*, is encouraged by large numbers of processors and small network delays. For a simple model of a symmetric distributed system, we have been able to estimate the reduction in expected finishing time of a "sequential" task structure obtained by cloning. Since it is difficult in distributed systems to identify clones from legitimate tasks, we have proposed economic controls for limiting the overuse of such cloning. The distributed system model used here is admittedly a poor approximation to reality; we have proposed directions in which improvements on these results could be made.

Acknowledgments. We are grateful to Richard Brent, Ravi Krishnamurthy and the referees for helpful comments. Mourad Ismail pointed out the role of beta functions in the derivation of § 4.

REFERENCES

- [Bau78] G. M. BAUDET, *The design and analysis of algorithms for asynchronous multiprocessors*, Tech. Rept. CMU-CS-78-116, Department of Computer Science, Carnegie-Mellon University, Pittsburgh PA, 1978.
- [Bau80] G. M. BAUDET, R. BRENT AND H. T. KUNG, *Parallel execution of a sequence of tasks on an asynchronous multiprocessor*, Australian Comput. J., 12 (1980), pp. 105-112.
- [Bur66] P. J. BURKE, *The output of a queuing system*, Oper. Res., 4 (1966), pp. 699-704.
- [Cof68] E. G. COFFMAN, JR. AND L. KLEINROCK, *Computer scheduling methods and their countermeasures*. Proc. AFIPS SJCC 1968, pp. 11-21.
- [Dow80] P. J. DOWNEY, *They also serve who only stand and wait*, Rept. TR 80-23, Department of Computer Science, University of Arizona, Tucson, 1980.
- [Kle75] L. KLEINROCK, *Queuing Systems, Vol. I: Theory*, John Wiley, New York, 1975.
- [Kle76] ———, *Queuing Systems, Vol. II: Computer Applications*, John Wiley, New York, 1976.
- [Kun73] D. E. KNUTH, *The Art of Computer Programming, Vol. I: Fundamental Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1973.
- [Kun76] H. T. KUNG, *Synchronized and asynchronous parallel algorithms for multiprocessors*, Algorithms And Complexity: New Directions and Recent Results, J. F. Traub, ed., Academic Press, New York, 1976.
- [Olv74] F. W. J. OLVER, *Asymptotics and Special Functions*, Academic Press, New York, 1974.
- [Rit74] D. M. RITCHIE AND K. THOMPSON, *The UNIX time-sharing system*, Comm. ACM, 17 (1974), pp. 365-375.
- [Wul72] W. A. WULF AND C. G. BELL, *C.mmp—a multi-mini-processor*, Proc. AFIPS FJCC, 41, Pt. II (1972), pp. 765-777.