SIAM J. Comput.
Vol. 2, No. 1, March 1973

THE ENUMERATION OF MAXIMAL CLIQUES OF LARGE GRAPHS*

E. A. AKKOYUNLU†

Abstract. An algorithm for enumerating maximal cliques (complete subgraphs) is proposed. The aim is to deal with the difficulties caused by the size of the problem.

Key words. Algorithms, clique, enumeration, graph.

1. Introduction. The problem considered here is best formulated in terms of an undirected graph G (Fig. 1). If S_0 is the set of nodes and E the set of edges, the goal is to identify completely connected subgraphs (or cliques) which are maximal.

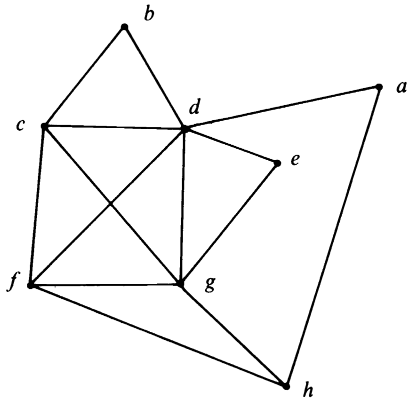


FIG. 1

More generally, S_0 is a set of elements, and there is defined over pairs in S_0 a symmetric, nontransitive binary relation R. (E is a set of pairs which are in relation R.) A subset S of S_0 is a *maximal subset* (ms) if

(i) every pair in S is in relation R, *and*

(ii) S is not a proper subset of any set with property (i).

The ms are of interest in many contexts: graph theory (the coloring problem), switching theory (state minimization [1]), operations research (scheduling [2], [3]), information systems, etc. There are several known algorithms [1], [4], [5], [6] for enumerating these sets. As a rule, however, these algorithms cannot handle large problems [3] efficiently. The difficulty arises because the terms generated include duplications, or even submaximal sets. To avoid this, a list of all the terms has to be kept and repeatedly scanned. In the worst cases, the list cannot be accommodated in core.

This paper proposes a new algorithm which is especially efficient in dealing with large problems, and has none of the shortcomings mentioned above. The set generated corresponds precisely (and without duplications) to the set desired.

* Received by the editors March 4, 1972, and in revised form November 20, 1972.

† Department of Computer Science, State University of New York at Stony Brook, Stony Brook, New York 11790. This research was supported in part by the National Science Foundation under Grants GK-1241 and GJ-243.

	a	b	c	d	e	f	g
b							
c		x					
d	x	x	x				
e				x			
f			x	x			
g			x	x	x	x	
h	x					x	x

FIG. 2

There is thus no need to refer to previously generated terms, and no need to maintain a list in core. Even for the largest problems memory size is not a critical limitation.

Let $\mathcal{M}$ denote the set of ms. Certain subsets of $\mathcal{M}$ are also important. In the state minimization problem where the relation R is mutual compatibility, what is wanted is *minimal cover*, i.e., the smallest number of ms in which each element occurs at least once. In scheduling, on the other hand, R is mutual exclusion, and one is typically interested in finding a *minimal pair-cover*, i.e., the smallest subset of $\mathcal{M}$ such that every pair in relation R occurs together in some term. This paper is mainly concerned with an algorithm for enumerating the set $\mathcal{M}$.

The information contained in the graph G is normally represented in the form of the table in Fig. 2. The somewhat redundant representation of Fig. 3 will be used instead, for it displays certain features more saliently. In particular, each element x divides the remaining elements into two sets: the set C_x of elements which are in relation R to x, and the set D_x of elements which are not. This is made explicit in Fig. 3.

2. Preliminaries. In Fig. 3, consider the row associated with c. For every ms, exactly one of the following statements must be true:

1. The ms includes $\{c\}$.
2. The ms includes at least one of $\{a, e, h\}$.

D_x	x	C_x
b c e f g	a	d h
a e f g h	b	c d
a e h	c	b d f g
h	d	a b c e f g
a b c f h	e	d g
a b e	f	c d g h
a b	g	c d e f h
b b d e	h	a f g

FIG. 3

To verify: if both statements were true, the ms would include one of the disallowed pairs, *ac*, *ce*, *ch*. If neither were true, the addition of c to this term could not possibly violate a constraint, so that the term could not have been maximal.

Consider next row b. An ms which includes b cannot include any of a, e, or h since $\{a, e, h\} \subset D_b$. This, together with the above, means that every ms which includes b must also include c. To generalize, we state:

If $D_x \subseteq D_y$, then x occurs in every ms where y occurs.

Further notation is introduced at this point in order to facilitate the manipulation of subsets of $\mathscr{M}$ according to certain characteristics. For $S \subseteq S_0$, let $L(S)$ denote "the set of all ms which include *at least one* element of S." Similarly, let $E(S)$ denote "the set of all ms which include *every* element of S." Thus,

$$L(S) = \{M | M \in \mathscr{M}, S \cap M \neq \emptyset\},$$

$$E(S) = \{M | M \in \mathscr{M}, S \subseteq M\}.$$

The following relations are direct consequences of these definitions:

(1) $$L(\{x\}) = E(\{x\}),$$

(2) $$E(S_1) \cap E(S_2) = E(S_1 \cup S_2),$$

(3) $$L(S_1) \cap L(S_2) = L(S_1) \quad \text{if } S_1 \subseteq S_2,$$

(4) $$L(\emptyset) = \emptyset.$$

Also, since any ms which includes x can only include elements in C_x, we can write

$$E(\{x\}) \cap L(S) = \begin{cases} E(\{x\}) & \text{if } x \in S, \\ E(\{x\}) \cap L(S \cap C_x) & \text{otherwise.} \end{cases}$$

More generally,

(5) $$E(\{x\}) \cap \left(\bigcap_i L(S_i) \right) = E(\{x\}) \cap \left(\bigcap_{i : x \notin S_i} L(S_i \cap C_x) \right).$$

3. Enumeration through disjoint subproblems.

In the previous example, where $S_0 = \{a, b, c, d, e, f, g, h\}$, the set of all ms can be written

$$\mathscr{M} = L(S_0) = L(\{a, b, c, d, e, f, g, h\}),$$

which simply states that $\mathscr{M}$ is the set of all ms which include at least one element. As discussed above, $\mathscr{M}$ has two disjoint subsets,

(i) all ms which include $\{c\}$,

(ii) all ms which include at least one of $\{a, e, h\}$,

and this is expressed by writing,

$$\mathscr{M} = L(S_0) = E(\{c\}) \cup L(\{a, e, h\}).$$

To formalize this notion, we write

(6) $$L(S \cup \{x\}) = E(\{x\}) \cup (L(S) \cap L(D_x)).$$

Here, the right-hand side is the union of disjoint sets: all ms which include x, and all which exclude x but include some element in S.

In what follows, the notation will be simplified by writing $L(a, b, \cdots, x)$ instead of the formally correct $L(\{a, b, \cdots, x\})$, etc. Equations (1) through (6) can be used to reduce the problem of enumerating $\mathscr{M}$ into a series of smaller problems. The key step is the substitution specified by (6) which is applied repeatedly. For the problem in Fig. 3, we have

$$\begin{aligned}\mathscr{M} &= L(a, b, c, d, e, f, g, h) = E(c) \cup L(a, e, h)\\ &= E(c) \cup E(a) \cup (L(e, h) \cap L(b, c, e, f, g))\\ &= E(c) \cup E(a) \cup ((E(e) \cup L(h) \cap L(a, b, c, f, h)) \cap L(b, c, e, f, g)).\end{aligned}$$

Since $L(h) = E(h)$, equation (5) allows the reductions

$$E(e) \cap L(b, c, e, f, g) = E(e),$$

$$E(h) \cap L(a, b, c, f, h) \cap L(b, c, e, f, g) = E(h) \cap L(f, g),$$

$$\mathscr{M} = E(c) \cup E(a) \cup E(e) \cup (E(h) \cap L(f, g)).$$

Now $L(f, g) = E(f) \cup (L(g) \cap L(a, b, e))$, so that we write,

$$E(h) \cap L(f, g) = (E(h) \cap E(f)) \cup (E(h) \cap L(g) \cap L(a, b, e)),$$

$$\mathscr{M} = E(c) \cup E(a) \cup E(e) \cup (E(h, f) \cup (E(h) \cap L(g) \cap L(a, b, e))).$$

Using equation (5), we have

$$E(h) \cap L(g) \cap L(a, b, e) = E(h) \cap L(g) \cap L(a).$$

Finally, since $L(g) \cap L(a) = E(g) \cap L(a) = \varnothing$, we obtain

$$\mathscr{M} = E(c) \cup E(a) \cup E(e) \cup E(h, f).$$

The problem is thus reduced to four smaller problems whose solutions correspond to disjoint subsets of $\mathscr{M}$. Each of these can now be solved separately and the complete solution $\mathscr{M}$ obtained.

In evaluating $E(a)$, for example, only the set C_a need be considered. This corresponds to Fig. 4 whose ms set is $\{d, h\}$. The addition of a to every member of this set yields $E(a) = \{ad, ah\}$. Similarly, $E(h, f)$ is obtained by adding h and f to each term of the ms set derived from Fig. 5 which consists of variables in $C_h \cap C_f$ alone. This ms set is simply $\{g\}$ so that $E(h, f) = \{fgh\}$.

To generalize, let S be a set for which $x, y \in S$ implies $x \in C_y$. Then,

$$E(S) = \begin{cases} S & \text{if } \bigcap_{x \in S} C_x = \varnothing, \\ E(S) \cap L(\bigcap_{x \in S} C_x) & \text{otherwise.} \end{cases} \tag{7}$$

4. Enumeration algorithm. This section formalizes the approach outlined above by giving an algorithm for enumerating $\mathscr{M}$. The algorithm manipulates

D_x	x	C_x
h	d	—
d	h	—

FIG. 4

D_x	x	C_x
—	g	—

FIG. 5

symbolic expressions composed of E- and L-sets. The data base consists of the sets S_0, C_x and D_x, for $x \in S_0$. The algorithm is organized around a pushdown stack which serves to store partially formulated disjoint subproblems. Items on the stack are expressions involving sets of the form $E(S)$ and $L(S)$. A possible item on the stack could be

$$E(h) \cap L(g) \cap L(a, b, e).$$

Specifically, each item on the stack is a multiple set intersection between one or more L-sets and at most one E-set.

Starting with the expression $L(S_0)$, the algorithm consists of repeated applications of equations (6) and (7), supplemented by the reduction equations (1) through (5). Equation (6) is used to split an expression into two others which correspond to disjoint subsets of the original set; these are then pushed on the stack, and the process is repeated until an expression of the form $E(S)$ is obtained. Equation (7) is then applied once.

ALGORITHM A.

Input: S_0, $\{C_x | x \in S_0\}$, $\{D_x | x \in S_0\}$.

Output: $\mathscr{M}$.

Step 1 (Initialize). Place $L(S_0)$ on the stack.

Step 2 (Prepare to split). (a) If the stack is empty, stop.

(b) Unload the top expression from the stack and call it T. T is a multiple intersection whose form is either

$$E(S') \cap \left(\bigcap_{i \in I} L(S_i) \right)$$

or simply

$$\bigcap_{i \in I} L(S_i).$$

In the first case set $V = S'$, in the second case set $V = \varnothing$.

Step 3 (Choose the L-term to be split). (a) Select $k \in I$ so that S_k has the fewest elements possible. Also choose $x \in S_k$, and let $S = S_k - \{x\}$.

(b) If $S = \varnothing$ go to Step 5. (Equation (1) can be applied instead of equation (6).)

Step 4 (Formulate the second subproblem). (a) Compute

$$Q = \begin{cases} D_x & \text{if } V = \varnothing, \\ D_x \cap (\bigcap_{y \in V} C_y) & \text{otherwise.} \end{cases}$$

(b) If $Q = \varnothing$ go to Step 5. (Every ms which covers V also covers $\{x\}$.)

(c) Make a copy of T with $L(S) \cap L(Q)$ substituted for $L(S_k)$, and load this copy on the stack.

Step 5 (Formulate the first subproblem). (a) Let $J = \{j | j \in I, x \in S_j\}$. If $J = \varnothing$ go to Step 6. (Equation (7) is applicable.)

(b) For all $j \in J$ compute $W_j = S_j \cap C_x$. If $W_j = \varnothing$ for any $j \in J$ go to Step 2. (No ms covers both V and $\{x\}$.)

(c) Place the expression

$$E(V \cup \{x\}) \cap \left(\bigcap_{j \in J} L(W_j) \right)$$

on the stack and go to Step 2.

Step 6 (Apply equation (7).) (a) Compute

$$P = \bigcap_{y \in V \cup \{x\}} C_y.$$

(b) If $P = \varnothing$ output $V \cup \{x\}$, then go to Step 2.

(c) Load the stack with $E(V \cup \{x\}) \cap L(P)$, and go to Step 2. (End Algorithm A.)

In practice, rather than compute the set

$$\bigcap_{y \in V} C_y$$

each time, it is more convenient to store it along with the associated item on the stack.

5. Conclusion. Techniques for generating maximal subgraphs were discussed with reference to the difficulties caused by the size of the problem. An effective procedure was proposed for systematically reducing these problems into smaller ones whose ms sets are disjoint. An important feature of the enumeration algorithm is that it is organized around a stack, so that its core requirements are minimal.

Acknowledgments. I would like to thank Professor S. H. Unger of Columbia University for the long discussions which helped focus this work.

REFERENCES

[1] M. C. PAULL AND S. H. UNGER, *Minimizing the number of states in incompletely specified sequential switching functions*, IRE Trans. Electronic Computers, EC-8 (1959), pp. 356–367.

[2] E. A. AKKOYUNLU, *Allocating facilities to interdependent activities*, Proc. Fifth Annual Princeton Conference on Information Sciences and Systems, Princeton, N.J., 1971, pp. 86–87.

[3] A. D. HALL, JR. AND F. S. ACTON, *Scheduling university course examinations by computer*, Comm. ACM, 10 (1967), pp. 235–238.

[4] P. M. MARCUS, *Derivation of maximal compatibles using Boolean algebra*, IBM J. Res. Develop., 8 (1964), pp. 537–538.

[5] G. D. MULLIGAN AND D. G. CORNEIL, *Corrections to Bierstone's algorithm for generating cliques*, J. Assoc. Comput. Mach., 19 (1972), pp. 244–247.

[6] R. M. BURSTALL, *Tree searching methods with an application to a network design problem*, Machine Intelligence, 1 (1967), pp. 65–85.

SIAM J. Comput.
Vol. 2, No. 1, March 1973

THE SPECIALIZATION OF PROGRAMS BY THEOREM PROVING*

C. L. CHANG, R. C. T. LEE AND J. K. DIXON†

Abstract. Suppose a program P is written to accept a set of inputs I. If we are only interested in a nonempty subset I^* of I, we usually can simplify P to another program P^* such that P^* runs faster on I^* than P does. The problem of specialization is to find such P^*. In this paper, the program P and the input I^* will be specified by axioms. Using these axioms, we can obtain P^* from P through theorem-proving techniques.

Key words. axioms, compilers, describing formulas, halting clauses, programs, resolution, specializers, theorem proving.

1. Introduction. Very often we encounter the following question:

We have written a program P which is intended to accept a set I of input data. However, later on we are only interested in a restricted subset I^* of input data. Of course, P still can accept I^* and produce an output. However, since I^* is smaller than I, we should be able to modify program P so that the modified program can run faster on set I^*. The question now is how do we modify program P?

This problem has been considered by Dixon [1], [2] and Futamura [3]. Dixon has written a program, called Specializer, to do the modification. What the specializer does is the following: Suppose P is a program which has N input variables (arguments). Suppose that specific values are assigned to M of these arguments ($M \leqq N$). Then, P^* is a program which takes only $N - M$ arguments. The other variables have been "fixed" and P^* is a specialized version of P for specific values of the fixed arguments. The advantage of specialization is the same as compilation: P^* usually runs faster than P. P^* may also take less storage space than P.

Dixon's specializer is written in the LISP [5] language. It can only specialize LISP programs. The restriction of input is done by fixing values of some input variables. Roughly speaking, the specializer performs the following two operations: (i) deleting any instruction which can be evaluated and replacing it with a quoted value, (ii) removing of a branch of a conditional instruction if it is not needed. A "pattern specializer," also defined by Dixon [2], uses a more general approach.

In this paper, we shall use a different approach to program specialization. That is, we shall use theorem-proving techniques [6], [7], [8]. Our approach can be applied to any program representable by a flow chart. Therefore, our technique is very general. Furthermore, the restriction of input need not be done by fixing values of some input variables. It can be done more generally. For example, some input variable may either have a range of values, or satisfy some relations.

Our approach is based upon the result given by Lee and Chang [4]. In [4], a program is described by a set of logical formulas. Considering these logical formulas as axioms, we can use the resolution principle [7] to deduce logical

* Received by the editors November 16, 1971, and in revised form November 22, 1972.

† Heuristics Laboratory, Division of Computer Research and Technology, National Institutes of Health, Bethesda, Maryland 20014.

consequences from these axioms. It was shown in [4] that a halting clause can be deduced if and only if the program terminates. Based upon the deduction of a halting clause, we shall show in this paper how a specialization of a program can be obtained.

It is assumed that the reader is familiar with the resolution principle. For a comprehensive study of this inference rule, consult [6], [7], [8].

2. Describing formulas of a program. The following definitions are from [4]. Programs are represented by directed graphs.

DEFINITION. A *program* consists of an input (variable) vector $\mathbf{x} = (x_1, \cdots, x_L)$, a program (variable) vector $\mathbf{y} = (y_1, \cdots, y_M)$, an output (variable) vector $\mathbf{z} = (z_1, \cdots, z_N)$, and a finite directed graph such that the following conditions are satisfied:

(i) In the graph, there is exactly one vertex, called the *start vertex* S, which is not a terminal vertex of any arc; there is exactly one vertex, called the *halt vertex* H, which is not an initial vertex of any arc; and every vertex v is on some path from S to H.

(ii) In the directed graph, each arc a not entering H is associated with a quantifier-free formula $P_a(\mathbf{x}, \mathbf{y})$ and an assignment $\mathbf{y} \leftarrow f_a(\mathbf{x}, \mathbf{y})$; each arc a entering the halt vertex H is associated with a quantifier-free formula $P_a(\mathbf{x}, \mathbf{y})$ and an assignment $\mathbf{z} \leftarrow f_a(\mathbf{x}, \mathbf{y})$. P_a is called the *testing predicate* associated with the arc a and $P_a(\mathbf{x}, \mathbf{y})$ is called the *testing formula* associated with the arc a.

(iii) For each vertex v ($v \neq H$), let $a_1, a_2, \cdots, a_r$ be all the arcs leaving from v and $P_{a_1}, P_{a_2}, \cdots, P_{a_r}$ be the testing predicates associated with arcs $a_1, a_2, \cdots, a_r$ respectively. Then for all $\mathbf{x}$ and $\mathbf{y}$, one and only one of $P_{a_1}(\mathbf{x}, \mathbf{y}), P_{a_2}(\mathbf{x}, \mathbf{y}), \cdots, P_{a_r}(\mathbf{x}, \mathbf{y})$ is true.

Example 1. Consider a program which accepts two nonnegative integers x_1 and x_2. The only testing predicate available is the testing of whether a number is equal to zero and the only functions used are addition and subtraction. The program is as follows.

```
                 $y_1 \leftarrow x_1$
                 $y_2 \leftarrow x_2$
1: **If** $y_2 = 0$, **then** [$z \leftarrow y_1$, **halt**];
          **else** [$y_1 \leftarrow y_1 + 1$,
                $y_2 \leftarrow y_2 - 1$,
                **go to** 1].
```

where x_1 and x_2 are input variables, y_1 and y_2 are program variables and z is the output variable.

This program can be represented by the directed graph shown in Fig. 1.

Given an input $\mathbf{x}$ for a program, the execution of this program is carried out according to the following algorithm:

Step 1. The control is always assumed to be initially at S.

Step 2. Let $j = 0$, $v^0 = S$ and $\mathbf{y}^0$ be the value of the program variable.

Step 3. If $v^j = H$, the execution is terminated; otherwise, go to the next step.

Step 4. Let a_j be the arc of which v^j is the initial vertex and $P_{a_j}(\mathbf{x}, \mathbf{y})$ is true.

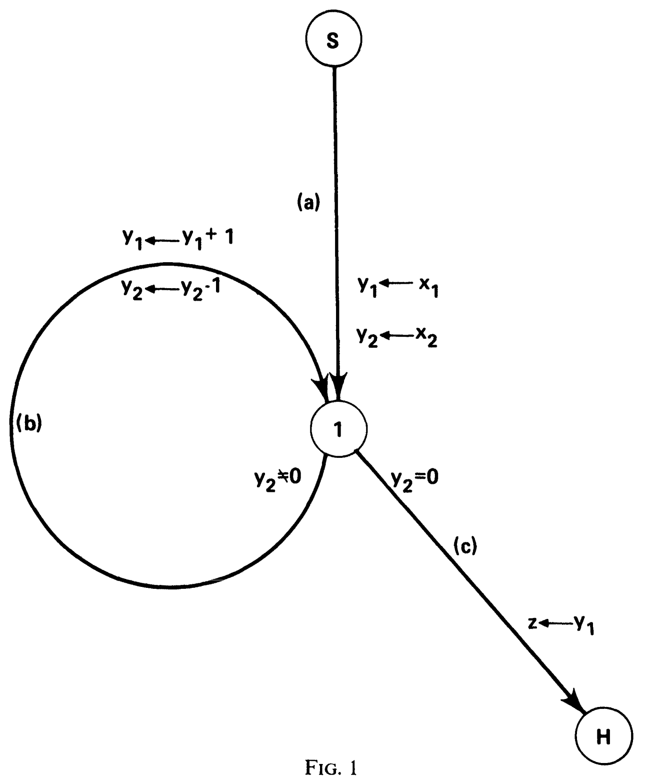


FIG. 1

Let v^{j+1} be the terminal vertex of a_j. Then the control moves along a_j to vertex v^{j+1} and one of the following assignments is executed.

(a) $\mathbf{y}^{j+1} \leftarrow f_{a_j}(\mathbf{x}, \mathbf{y}^j)$ if v^{j+1} is not H,

(b) $\mathbf{z} \leftarrow f_{a_j}(\mathbf{x}, \mathbf{y}^j)$ if v^{j+1} is H.

Step 5. Let $j = j + 1$ and go to Step 3.

The above execution is said to be *finite* if and only if for some k, $v^k = H$. In this case, we say that the program *terminates* for the input **x**.

The above rules concerning the execution of programs can be conveniently described by logical formulas. This is done by using an ingenious concept defined below.

DEFINITION. For each vertex v_i in a program, $Q_i(\mathbf{x}, \mathbf{y})$ ($Q_i(\mathbf{x}, \mathbf{z})$ if $v_i = H$) is defined as the condition that the control is passed from S to v_i with the input vector **x** and with the program vector changed to **y** (or the output program vector changed to **z** if $v_i = H$). This condition is called the *access condition* of v_i. In other words, if the input vector is **x**, then whenever the control is passed to vertex v_i, $Q_i(x, y)$ is T. The predicate Q_i is called the *access predicate* of v_i.

Since the control is always assumed to be initially at S, we may simply let $Q_S(\mathbf{x}, \mathbf{y})$ be T. Because the passing of the control to vertex H means that the program terminates, Q_H plays an important role in program analysis, as we shall see in the next section. In the sequel, we shall call Q_H the *halting predicate.*

Let a be an arc of a program whose initial vertex is v_i and whose terminal vertex is v_j. Suppose $P_a(\mathbf{x}, \mathbf{y})$ and $f_a(\mathbf{x}, \mathbf{y})$ are the testing formula and assignment associated with arc a. If the control is passed to v_i (thereby $Q_i(\mathbf{x}, \mathbf{y})$ is T) and if $P_a(\mathbf{x}, \mathbf{y})$ is T, then the control will be passed through arc a to vertex v_j, with $\mathbf{y}$ assigned $f_a(\mathbf{x}, \mathbf{y})$ (thereby $Q_j(\mathbf{x}, f_a(\mathbf{x}, \mathbf{y}))$ is T). Thus we can describe the execution by a formula defined as below.

DEFINITION. Let a be an arc in a program. Let v_i and v_j be the initial and terminal vertices of arc a respectively. Suppose $P_a(\mathbf{x}, \mathbf{y})$ and $f_a(\mathbf{x}, \mathbf{y})$ are the testing formula and assignment associated with arc a respectively. Then, we define a formula W_a associated with arc a as follows:

$$W_a : Q_i(\mathbf{x}, \mathbf{y}) \,\&\, P_a(\mathbf{x}, \mathbf{y}) \rightarrow Q_j(\mathbf{x}, f_a(\mathbf{x}, \mathbf{y})).$$

W_a is called the *describing formula* for arc a. If we use a clause C_a to represent W_a, then C_a is called the *describing clause* for arc a.

Note that if $v_i = S$, then $Q_S(\mathbf{x}, \mathbf{y}) = T$ and W_a becomes:

$$W_a : P_a(\mathbf{x}, \mathbf{y}) \rightarrow Q_j(\mathbf{x}, f_a(\mathbf{x}, \mathbf{y})).$$

Example 2. Consider the program in Example 1. The describing formulas for arcs a, b and c are:

$$W_a : Q_1(x_1, x_2, x_1, x_2),$$

$$W_b : Q_1(x_1, x_2, y_1, y_2) \,\&\, y_2 \neq 0 \rightarrow Q_1(x_1, x_2, y_1 + 1, y_2 - 1),$$

$$W_c : Q_1(x_1, x_2, y_1, y_2) \,\&\, y_2 = 0 \rightarrow Q_H(x_1, x_2, y_1).$$

DEFINITION. Let arcs $a_1, a_2, \cdots, a_R$ be all the arcs in a program P. Then $(\mathbf{y})(W_{a_1} \,\&\, W_{a_2} \,\&\, \cdots \,\&\, W_{a_R})$ is called the *describing formula* of the program P, where every W_{a_i}, $1 \leqq i \leqq R$, is the describing formula for arc a_i.

In the sequel, the describing formula of a program P will be denoted as A_P. We shall represent A_P by a set of clauses. A_P will be viewed as axioms describing the program P.

3. The specialization of programs.

DEFINITION. A clause in which every predicate that appears is a halting predicate is a *halting* clause.

Suppose a program P is written for input I. Let A_P denote the describing formulas of the program P, A_F denote axioms concerning testing predicate and assignment functions in P, and A_I denote axioms concerning the input I.

DEFINITION. For a set S of clauses, the *resolution of* S, denoted by $R(S)$, is defined as the set consisting of the members of S, together with all the resolvents of pairs of members of S. The *n-th resolution of* S, $n \geqq 0$, denoted by $R^n(S)$, is defined by $R^0(S) = S$ and $R^{n+1}(S) = R(R^n(S))$.

In [4], the following theorem is proved.

THEOREM. *Let P be a program. Let S denote the set of clauses representing $A_P \,\&\, A_F \,\&\, A_I$. Then P terminates if and only if for some $n \geqq 0$, there is a halting clause $C_H \in R^n(S)$.*

Example 3. Consider the program in Fig. 1 again. As shown in Example 2, the describing clauses for arcs a, b and c are:

(1) C_a: $Q_1(x_1, x_2, x_1, x_2)$,

(2) C_b: $\sim Q_1(x_1, x_2, y_1, y_2) \vee y_2 = 0 \vee Q_1(x_1, x_2, y_1 + 1, y_2 - 1)$,

(3) C_c: $\sim Q_1(x_1, x_2, y_1, y_2) \vee y_2 \neq 0 \vee Q_H(x_1, x_2, y_1)$.

Since the program involves a loop, we have to use an induction axiom as follows:

$$\begin{array}{l}
\quad (\exists y_2)(y_2 > 0 \,\&\, Q_1(x_1, x_2, x_1, y_2)) \\
\;\& (y_3)(y_3 > 0 \,\&\, Q_1(x_1, x_2, x_1, y_3) \rightarrow Q_1(x_1, x_2, x_1 + 1, y_3 - 1)) \\
\rightarrow Q_1(x_1, x_2, x_1 + x_2, 0).
\end{array}$$

The above axiom can be broken into three clauses, where f is a Skolem function:

$$\begin{array}{ll}
(4) & y_2 \not> 0 \vee \sim Q_1(x_1, x_2, x_1, y_2) \vee f(y_2) > 0 \vee Q_1(x_1, x_2, x_1 + x_2, 0), \\
(5) & y_2 \not> 0 \vee \sim Q_1(x_1, x_2, x_1, y_2) \vee Q_1(x_1, x_2, x_1, f(y_2)) \\
 & \qquad \vee\, Q_1(x_1, x_2, x_1 + x_2, 0), \\
(6) & y_2 \not> 0 \vee \sim Q_1(x_1, x_2, x_1, y_2) \vee \sim Q_1(x_1, x_2, x_1 + 1, f(y_2) - 1) \\
 & \qquad \vee\, Q_1(x_1, x_2, x_1 + x_2, 0).
\end{array}$$

A_I is given as:

$$(7) \qquad x_2 > 0.$$

A_F is given as:

$$\begin{array}{ll}
(8) & 0 = 0, \\
(9) & u \not> 0 \vee u \neq 0.
\end{array}$$

From (1) through (9), we can generate the following resolvents:

$$\begin{array}{lll}
(10) & (4)\,\&\,(7) & \sim Q_1(x_1, x_2, x_1, x_2) \vee f(x_2) > 0 \vee Q_1(x_1, x_2, x_1 + x_2, 0), \\
(11) & (5)\,\&\,(7) & \sim Q_1(x_1, x_2, x_1, x_2) \vee Q_1(x_1, x_2, x_1, f(x_2)) \\
 & & \quad \vee\, Q_1(x_1, x_2, x_1 + x_2, 0), \\
(12) & (6)\,\&\,(7) & \sim Q_1(x_1, x_2, x_1, x_2) \vee \sim Q_1(x_1, x_2, x_1 + 1, f(x_2) - 1) \\
 & & \quad \vee\, Q_1(x_1, x_2, x_1 + x_2, 0), \\
(13) & (1)\,\&\,(10) & f(x_2) > 0 \vee Q_1(x_1, x_2, x_1 + x_2, 0), \\
(14) & (1)\,\&\,(11) & Q_1(x_1, x_2, x_1, f(x_2)) \vee Q_1(x_1, x_2, x_1 + x_2, 0), \\
(15) & (1)\,\&\,(12) & \sim Q_1(x_1, x_2, x_1 + 1, f(x_2) - 1) \vee Q_1(x_1, x_2, x_1 + x_2, 0), \\
(16) & (2)\,\&\,(14) & f(x_2) = 0 \;\; \bar{x}\, Q_1(x_1, x_2, x_1 + 1, f(x_2) - 1) \\
 & & \quad \vee\, Q_1(x_1, x_2, x_1 + x_2, 0), \\
(17) & (9)\,\&\,(13) & f(x_2) \neq 0 \vee Q_1(x_1, x_2, x_1 + x_2, 0), \\
(18) & (16)\,\&\,(17) & Q_1(x_1, x_2, x_1 + 1, f(x_2) - 1) \vee Q_1(x_1, x_2, x_1 + x_2, 0), \\
(19) & (15)\,\&\,(18) & Q_1(x_1, x_2, x_1 + x_2, 0), \\
(20) & (3)\,\&\,(19) & 0 \neq 0 \vee Q_H(x_1, x_2, x_1 + x_2), \\
(21) & (8)\,\&\,(20) & Q_H(x_1, x_2, x_1 + x_2).
\end{array}$$

Clause (21) shows that the function of this program is to add two numbers x_1 and x_2.

Now, we consider the specialization problem. Suppose a program P is written for the input I and we are only interested in a restricted input I^*. Of course, program P still accepts I^*. However, since I^* is more restricted than I, we should take this advantage and simplify P. That is, we should find a specialized version P^* of P so that P^* can run faster on I^*. In the following, we shall describe a procedure to find such P^*.

THE SPECIALIZATION PROCEDURE.

Step 1. Suppose A_P and A_F are defined as above. Let A_I^* be axioms specifying the restricted input I^*. Let S denote the set of clauses representing A_P & A_F & A_I^*.

Step 2. Using resolution, derive a halting clause C_H from S.

Step 3. Let D be the deduction of C_H.

Step 4. Let P^* be the program obtained from P by deleting any arc whose describing clause is not used in D, and by simplifying the remaining program. P^* is a specialization of P for I^*.

Example 4.

Consider the program P shown in Fig. 2(a), where the input x is an integer and $D(x, y)$ means that y divides x. The program P is for any integer. Suppose we now know that x is less than 10. Find a specialization of P for such input. For this program P, A_P is given as follows:

(1) $C_1 = \sim D(x, 7) \vee Q_1(x, x)$,

(2) $C_2 = D(x, 7) \vee Q_H(x, a)$,

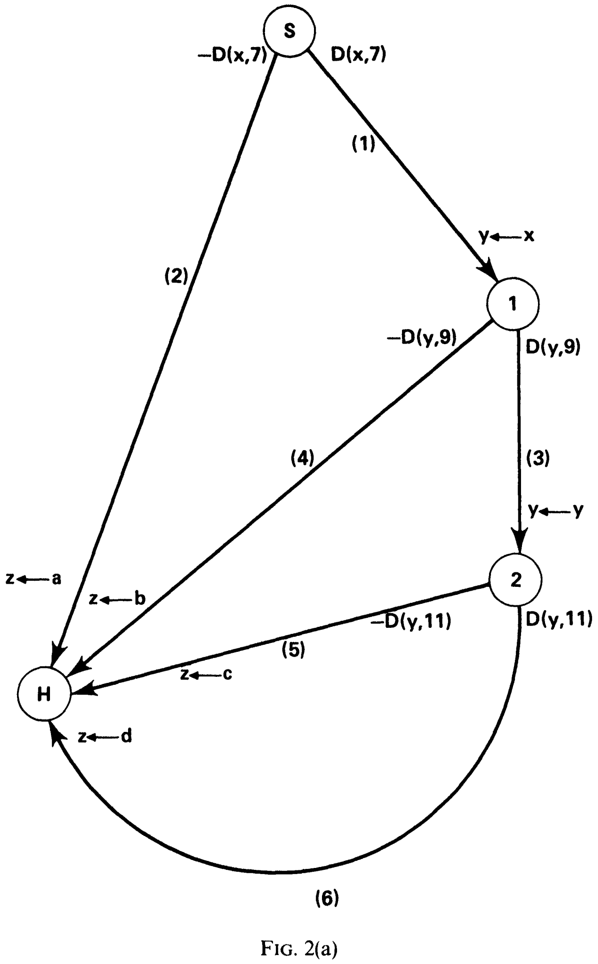


FIG. 2(a)

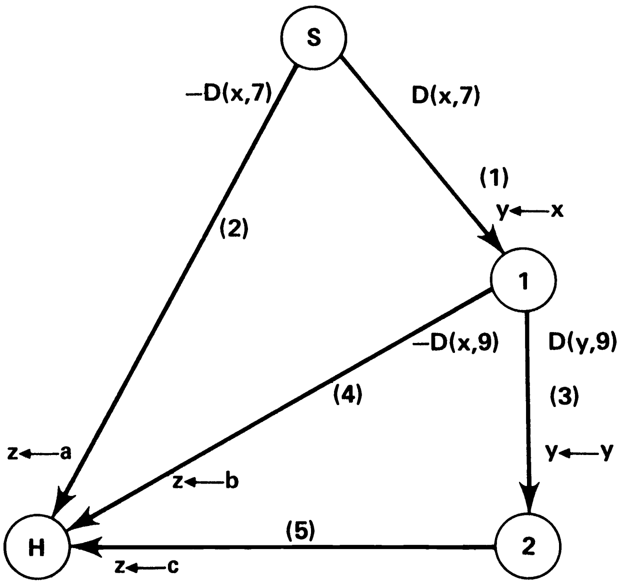


FIG. 2(b)

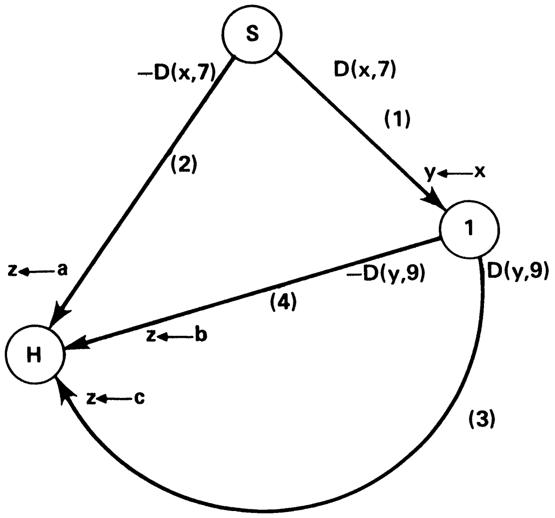


FIG. 2(c)

(3) $C_3 = \sim Q_1(x, y) \vee \sim D(y, 9) \vee Q_2(x, y)$,
(4) $C_4 = \sim Q_1(x, y) \vee D(y, 9) \vee Q_H(x, b)$,
(5) $C_5 = \sim Q_2(x, y) \vee D(y, 11) \vee Q_H(x, c)$,
(6) $C_6 = \sim Q_2(x, y) \vee \sim D(y, 11) \vee Q_H(x, d)$.

A_F is given as

(7) $x \nless 10 \vee \sim D(x, 11)$.

A_I^* is given as

(8) $x < 10$.

Clause (7) means that if $x < 10$, then 11 does not divide x. From A_P & A_F & A_I^*, we obtain the following deduction D of a halting clause:

(9)	(7) & (8)	$\sim D(x, 11)$,
(10)	(5) & (9)	$\sim Q_2(x, y) \vee Q_H(x, c)$,
(11)	(3) & (4)	$\sim Q_1(x, y) \vee Q_2(x, y) \vee Q_H(x, b)$,
(12)	(10) & (11)	$\sim Q_1(x, y) \vee Q_H(x, c) \vee Q_H(x, b)$,
(13)	(1) & (2)	$Q_1(x, x) \vee Q_H(x, a)$,
(14)	(12) & (13)	$Q_H(x, a) \vee Q_H(x, c) \vee Q_H(x, b)$.

Clause (14) is a halting clause which indicates that the output can be either a, b or c (never d). Since C_6 is not used in this deduction, arc (6) can be deleted. We can obtain P^* shown in Fig. 2(b). Clearly, P^* can be further simplified to the program shown in Fig. 2(c).

Example 5. Consider Fig. 2(a) again. Suppose this time we know that if 9 divides x, then 11 does not divide x. Let us see how we can simplify the program. The describing formulas from (1) to (6) are still the same.

A_I^* is given as

(7) $\sim D(x, 9) \vee \sim D(x, 11)$.

We have the following deductions:

(8)	(1) & (2)	$Q_1(x, x) \vee Q_H(x, a)$,
(9)	(8) & (3)	$\sim D(x, 9) \vee Q_2(x, x) \vee Q_H(x, a)$,
(10)	(9) & (5)	$D(x, 11) \vee \sim D(x, 9) \vee Q_H(x, a) \vee Q_H(x, c)$,
(11)	(10) & (7)	$\sim D(x, 9) \vee Q_H(x, a) \vee Q_H(x, c)$,
(12)	(11) & (4)	$\sim Q_1(x, x) \vee Q_H(x, a) \vee Q_H(x, b) \vee Q_H(x, c)$,
(13)	(12) & (8)	$Q_H(x, a) \vee Q_H(x, b) \vee Q_H(x, c)$.

Note that clause (6) is not used in the above deduction. Therefore arc (6) can be deleted as shown in Fig. 2(b). This program can be further simplified to that in Fig. 2(c).

The above example shows that this new approach can handle quite complicated specification of the input, which is an improvement of the old approaches [1], [2], [3].

4. Concluding remarks. The purpose of program specialization is the same as that of compilation—to improve efficiency. In addition, it allows one to write a general program so that it may be tailored to any specific user when he has a specific input specification. We have shown that techniques used in theorem proving can be applied to the specialization problem. Our approach is very general because it is not limited to any particular language and any kind of input specification can be used.

At this stage, we have to admit that our techniques are not pragmatic because the mechanical theorem-proving techniques are not efficient enough. But, as we are witnessing progress being made in this field, we do believe that the foundation for applicable results is being laid.

REFERENCES

[1] J. DIXON, *The Specializer: a method of automatically writing computer programs*, submitted for publication.

[2] ———, *An improved method for solving deductive problems on a computer by compiled axioms*, Doctoral thesis, Univ. of California, Davis, 1970; available from University Microfilm, Ann Arbor, Mich.

[3] Y. FUTAMURA, *Partial evaluation of computer programs; an approach to a compiler-compiler*, J. Electronics and Communication Engineers of Japan, Vol. 54–6, No. 8, 1971.

[4] R. C. T. LEE AND C. L. CHANG, *Program analysis and theorem proving*, submitted for publication.

[5] J. MCCARTHY, *LISP 1.5 Programmer's Manual*, MIT Press, Cambridge, Mass., 1962.

[6] J. NILSSON, *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.

[7] J. ROBINSON, *A machine-oriented logic based on the resolution principle*, J. Assoc. Comput. Mach., 12 (1965), pp. 23–41.

[8] J. R. SLAGLE, *Artificial Intelligence, the Heuristic Programming Approach*, McGraw-Hill, New York, 1971.

SIAM J. Comput.
Vol. 2, No. 1, March 1973

EULERIAN WALKS IN GRAPHS*

S. GOODMAN and S. HEDETNIEMI†

Abstract. The general problem of finding the shortest edge covering walks in an arbitrary undirected graph is investigated. An exact combinatoric expression is obtained for the length of such a walk, and the graph theoretic properties of these walks are studied. Explicit solutions are exhibited for some important classes of graphs.

Key words. Graph theory, Eulerian walks, Chinese postman's problem.

A graph G is called Eulerian if it is possible to start at an arbitrary point u of G, walk in some sequence along the lines of G and return to the starting point u having traversed every line exactly once. In 1736, Euler, after whom such graphs were named, showed that a graph G possesses such a traversal if and only if every point of G is incident with an even number of lines, i.e., every point has even degree.

Most graphs however contain at least two points of odd degree and hence are not Eulerian. In non-Eulerian graphs G, if one wanted to start at an arbitrary point u, traverse all the lines of G and return to point u, one would have to traverse some line, or lines, more than once. We shall call a walk which begins and ends with the same point and contains every line, a *closed covering walk of G*. An *Eulerian walk in a graph G* is a closed covering walk of G of minimum length.

The original formulation of this problem, due to Kwan Mei-ko [3] and known as the Chinese postman's problem, was as follows: "A mailman has to cover his assigned route before returning to the post office. The problem is to find the shortest walking distance for the mailman." More generally, the problem is of interest in many situations where one has to periodically traverse or inspect every link in a network; for example, in garbage collection, security patroling, and rail line inspection.

In [2] exact expressions were obtained for the length of an Eulerian walk in graphs having 0 or 2 odd points, in trees, and in complete graphs. Weak upper and lower bounds were also obtained for arbitrary graphs having $2n$ odd points. In this paper we shall determine the exact length of an Eulerian walk in an arbitrary graph G having $2n$ odd points. We shall characterize Eulerian walks in terms of minimum sets of connecting paths between pairs of odd points in G. A rather different development was given by Kwan Mei-ko [3] who characterized such walks in terms of an interesting cycle property which they possess. We shall also investigate some general properties of Eulerian walks and exhibit a number of specific solutions for some important classes of graphs.

1. Definitions and terminology.[1]

A graph $G = (V, E)$ consists of a finite, nonempty set V of *points* and a set E of unordered pairs (u, v) of distinct points, called *lines*. A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if $V' \subset V$ and

* Received by the editors June 6, 1972, and in revised form December 26, 1972.

† Department of Applied Mathematics and Computer Science, University of Virginia, Charlottesville, Virginia 22901.

[1] Any terms not defined here can be found in Harary [1].

$E' \subset E$; G' is a *full subgraph* of G if for every pair of points $u, v \in V'$, $(u, v) \in E$ implies $(u, v) \in E'$.

By our definition, a graph can contain at most one line (u, v) for a given pair of distinct points u, v. A *multigraph* is a graph except that it may contain several lines (u, v) for a given pair of distinct points u, v. The term "graph" will not be used to denote multigraphs.

A *walk of length* n in a graph $G = (V, E)$ from point u_1 to point u_{n+1} is a sequence of points $W = u_1, u_2, \cdots, u_n, u_{n+1}$ and line set E such that for $i = 1, 2, \cdots, n$, $(u_i, u_{i+1}) \in E$; the walk W is *open* or *closed* depending on whether $u_1 \neq u_{n+1}$ or $u_1 = u_{n+1}$, respectively. A *path* is an open walk in which all points are distinct. A *cycle* is a closed walk $u_1, u_2, \cdots, u_n, u_1$, $n \geqq 3$, in which the points $u_1, u_2, \cdots, u_n$ are all distinct. The *distance* $d(u, v)$ between two points u, v in a graph G is the length of any shortest path from u to v.

A walk W in a graph G is said to be *spanning* if it contains every point of G; W is said to be *covering* if it contains every line of G. An *Eulerian walk* in a graph G is a closed covering walk of minimum length.

A *homomorphism* of a graph G into (onto) a graph G' is a function ϕ from V into (onto) V' such that for every $u, v \in V$, $(u, v) \in E$ implies $(u\phi, v\phi) \in E'$. If, in addition, for every $u', v' \in V\phi$, $(u', v') \in E'$ implies there exist points $u, v \in V$ such that $u\phi = u'$, $v\phi = v'$ and $(u, v) \in E$, then ϕ is said to be a *full homomorphism.* The image of G under the homomorphism ϕ is the graph $G\phi = (V\phi, E\phi)$, where $V\phi = \{u\phi | u \in V\}$ and $E\phi = \{(u\phi, v\phi) | (u, v) \in E\}$. It follows from the definition that if ϕ is a homomorphism of G into G', then $G\phi$ is a subgraph of G', and if ϕ is a full homomorphism, then $G\phi$ is a full subgraph of G'. If ϕ is full and onto, then $G\phi = G'$.

Notice at this point that if G is a connected graph having q lines, then there exists a full homomorphism ϕ of the cycle of length $2q$, C_{2q} onto G, i.e., $C_{2q}\phi = G$. In general however, C_{2q} is not the smallest cycle which has a full homomorphism onto G. Let $C_k = u_1, u_2, \cdots, u_k, u_1$ be a smallest cycle which has a full homomorphism onto G, i.e., $C_k\phi = G$. Then $u_1\phi, u_2\phi, \cdots, u_k\phi, u_1\phi$ is an Eulerian walk in G. Conversely, if $v_1, v_2, \cdots, v_k, v_1$ is an Eulerian walk in G, then the homomorphism ϕ, defined by $u_i\phi = v_i$, maps the cycle $C_k = u_1, u_2, \cdots, u_k, u_1$ onto G, where C_k is the smallest cycle that maps onto G. Thus, Eulerian walks in graphs correspond 1–1 with full homomorphisms of smallest cycles C_k onto G.

A point v is a *cutpoint* of a connected graph G if its removal results in a disconnected graph. A *bridge* is such a line. A *nonseparable* graph is connected, nontrivial, and has no cutpoints. A maximal nonseparable subgraph is called a *block.* A *wheel,* W_p, is a $(p - 1)$-cycle each point of which is adjacent to a single point not on the cycle.

2. The length of an Eulerian walk: Exact solution. We define the length of an Eulerian walk in an arbitrary graph G as the *Euler length of* G and denote this by $e(G)$.

We shall now derive an exact equation for $e(G)$ for any connected graph G. If G is not connected the following theory will apply to its components.

Let G be a connected graph having q lines and $2n$ points of odd degree. Denote the set of odd points of G by $O(G) = \{u_1, u_2, \cdots, u_{2n}\}$. Define a *pair-partition of the odd points of* G to be a partition π of $O(G)$ into n pairs, say

$\pi = \{\{u_{11}, u_{12}\}, \{u_{21}, u_{22}\}, \cdots, \{u_{n1}, u_{n2}\}\}$. Given a pair-partition π, define the sum

$$d(\pi) = \sum_{i=1}^{n} d(u_{i1}, u_{i2}),$$

i.e., $d(\pi)$ is the sum of the distances $d(u_{i1}, u_{i2})$ between the pairs of points enumerated in the pair-partition π.

If we denote the set of all pair-partitions of G by $\pi(G)$, then we can define

$$m(G) = \min_{\pi \in \pi(G)} d(\pi),$$

which is clearly a finite integer for any graph G. Let $\bar{\pi}$ be a pair-partition that actually gives the minimum $d(\bar{\pi}) = m(G)$; then for each pair $\{u_{i1}, u_{i2}\}$ in $\bar{\pi}$, let P_i be any path in G from u_{i1} to u_{i2} that is of minimum length $d(u_{i1}, u_{i2})$. We shall refer to such a set of n paths $\{P_1, P_2, \cdots, P_n\}$ as an *m-set of* G. Also, we let $|P_i|$ denote the length of P_i, i.e., $|P_i| = d(u_{i1}, u_{i2})$.

THEOREM 1. *For any connected graph G having q lines and 2n points of odd degree,*

$$e(G) = q + m(G).$$

Proof. We first show that $e(G) \leqq q + m(G)$, i.e., there exists a closed covering walk in G of length $q + m(G)$.

Let $\{P_1, P_2, \cdots, P_n\}$ be an m-set of G, for some pair-partition $\pi = \{\{u_{11}, u_{12}\}, \{u_{21}, u_{22}\}, \cdots, \{u_{n1}, u_{n2}\}\}$ of G. Then

$$|P_1| + |P_2| + \cdots + |P_n| = d(u_{11}, u_{12}) + d(u_{21}, u_{22}) + \cdots + d(u_{n1}, u_{n2}) = m(G).$$

Let G' be the multigraph which is obtained by adding to G all of the lines in the m-set $\{P_1, P_2, \cdots, P_n\}$. Notice that while G contains $2n$ points of odd degree, every point in the multigraph G' has even degree. Therefore G' is an Eulerian multigraph having $q + m(G)$ lines. Thus G' has a closed covering walk W of length $q + m(G)$ which contains all the lines of G' exactly once. This walk W in G' is also a closed covering walk in G. Thus any Eulerian walk in G will necessarily contain no more lines than W, i.e., $e(G) \leqq q + m(G)$.

We next show that $e(G) \geqq q + m(G)$. Assume that there exists a cycle $C_k = u_1, u_2, \cdots, u_k, u_1$ of length $k < q + m(G)$ and a full homomorphism ϕ that maps C_k onto G, $C_k\phi = G$.

Consider the multigraph G_k which is defined by C_k and the homomorphism ϕ, i.e., $V(G_k) = V(G)$, and there are as many lines between two points v, w in G_k as there are pairs of adjacent points u_i, u_j in C_k such that $u_i\phi = v$ and $u_j\phi = w$. It follows therefore, since C_k has k lines, that G_k has k lines. The degree of a point v in G_k equals the sum of the degrees of the points u_i for which $u_i\phi = v$. Since every point u_i has even degree (i.e., degree 2) in C_k, every point v has even degree in G_k.

Let us now consider the graph $G_k - G$ obtained by deleting the q lines of G from G_k. Since G has $2n$ points of odd degree, it follows that the same $2n$ points have odd degree in $G_k - G$, and in fact these are the only points of odd degree in $G_k - G$.

Thus by the classical theorem of Euler (cf. Harary [1, p. 64]), there exist n line disjoint walks that contain all the lines of $G_k - G$. These n walks connect the $2n$

odd points of $G_k - G$ in pairs, and hence define a pair-partition π of G, for which

$$d(\pi) = \text{number of lines in } G_k - G = k - q.$$

By definition however, $d(\pi) \geqq m(G)$; thus $k - q \geqq m(G)$, i.e.,

$$k \geqq q + m(G),$$

which contradicts our assumption that $k < q + m(G)$. Thus there does not exist a cycle of length less that $q + m(G)$ which has a homomorphism onto G. Thus $e(G) \geqq q + m(G)$.

Thus since we have shown $e(G) \leqq q + m(G)$ and $e(G) \geqq q + m(G)$, we conclude that $e(G) = q + m(G)$.

COROLLARY 1a. *If a connected graph G is Eulerian, i.e., has no points of odd degree, and has q lines, then*

$$e(G) = q.$$

This is Euler's classic result.

COROLLARY 1b. *If G is a connected graph having q lines and exactly* 2 *points u, v of odd degree, then*

$$e(G) = q + d(u, v).$$

Proof. The result follows from the fact that $O(G)$ has only one possible pair-partition in this case.

These last two corollaries appear in [2].

For graphs with more than 2 odd points it becomes increasingly difficult to find m-sets because of combinatorial complications. In the remainder of this paper we shall go into some of these difficulties in detail and investigate some general properties of these m-sets.

A completely different characterization of Eulerian walks in graphs was given in 1962 by Kwan Mei-ko [3]. In order to contrast his result with ours, let us first restate Theorem 1.

COROLLARY 1c. *A closed covering walk W in a graph G is an Eulerian walk if and only if the set of lines appearing more than once in W forms an m-set of G.*

We next restate Mei-ko's theorem in the form and language of Corollary 1c.

THEOREM 2 (Mei-ko). *A closed covering walk W in a graph G is an Eulerian walk if and only if* (i) *no line of G appears more than twice in W, and* (ii) *for any cycle C of G the number of lines of C which appear twice in W does not exceed half the length of C.*

Theorem 2 and an algorithm derived from it are essentially the content of [3]. Condition (ii) of this theorem is particularly interesting. It is not obvious that the conditions in Corollary 1c and Theorem 2 are equivalent. This equivalence can be seen from the following schematic:

$$m\text{-sets} \rightleftarrows \text{Eulerian walks} \rightleftarrows \text{(i) and (ii) in Theorem 2.}$$

3. The length of an Eulerian walk: Specific solutions. A *factor* of a graph G is a spanning subgraph of G which is not trivial. A 1-*factor* is regular of degree 1. A graph with $2n$ odd points is said to have a 1-*factor on its odd points* if the subgraph induced by its set of odd points has a 1-factor.

LEMMA 3. *For any graph G with 2n odd points, $m(G) \geqq n$.*

Proof. We have established that an m-set consists of n paths connecting the $2n$ odd vertices in pairs such that the sum of their lengths is minimum. Clearly no set of n nonzero paths which begin and end on n distinct pairs of points can have a combined length less than n.

THEOREM 3. *If G has a 1-factor on its odd points, then $m(G) = n$.*

Proof. If G has a 1-factor on its odd points, it is possible to connect the $2n$ odd points in pairs by n lines, thus equaling the lower bound in Lemma 2.

COROLLARY 3a. *If G is a graph in any one of the following classes and G has 2n odd points, then $m(G) = n$:*

(i) *the complete graphs, K_{2n},*

(ii) *bridgeless cubic graphs,*

(iii) *graphs that are both s-regular and s-connected,*

(iv) *the wheels, W_p.*

Proof. All the graphs in the above classes can be shown to have a 1-factor on their odd points.

The following theorem is easily demonstrated.

THEOREM 4. (a) *For the complete bipartite graphs $G = K_{2p+1,2n}$, where p and n are any positive integers,*

$$m(G) = 2n.$$

(b) *For the complete bipartite graphs $G = K_{p,p'}$, where p and p' are odd integers with $p \geqq p'$,*

$$m(G) = p.$$

4. Properties of Eulerian walks in graphs. Theorem 1 provides an exact equation for the length of an Eulerian walk in an arbitrary connected graph G. However, it does not provide, except for trying all possibilities, an efficient algorithm for actually finding such walks. The next several results reveal some properties of Eulerian walks in graphs which are useful in finding such walks.

The next theorem will be recognized as equivalent to the first condition in Theorem 2. The proof given below is different from that described in [3].

THEOREM 5. *Let W be an Eulerian walk in a graph G. Then no line of G appears more than twice in W.*

Proof. Suppose W contains a line uv which occurs three times. Consider how this line occurs in W; there are essentially three possibilities:

(i) $W = AuvBuvCuvD$, where A, B, C, and D are sequences of points of G which may or may not contain other occurrences of the line (u, v); if W has this form then $W' = AuvCu\overleftarrow{B}vD$ would be a closed walk containing all the lines of G, having two fewer lines than W, which would contradict the minimality of W ($\overleftarrow{B}$ is the reverse of B);

(ii) $W = AuvBuvCvuD$; if W has this form then $W' = AuvCvBuD$ would be a closed walk containing all the lines of G having fewer lines than W, another contradiction;

(iii) $W = AuvBvuCuvD$; if W has this form then $W' = AuCuvBvD$ would be a closed walk containing all the lines of G, having two fewer lines than W, another contradiction.

Thus in all cases, the supposition that there exists a line which occurs more than twice in W leads to the existence of a closed walk containing all the lines of G, having fewer lines than W, which contradicts the minimality of W. Therefore no line can appear more than twice in any Eulerian walk in a graph G.

COROLLARY 5a. *Let $\{P_1, P_2, \cdots, P_n\}$ be an arbitrary m-set of a graph G. Then no line of G appears twice in $\{P_1, P_2, \cdots, P_n\}$, i.e., the set of paths in any m-set of a graph G are line-disjoint.*

Proof. An Eulerian walk in a graph G can be obtained by adding the lines in an m-set to G to form an Eulerian multigraph. If a line appeared twice in an m-set, it would appear three times in the corresponding Eulerian walk; this would contradict Theorem 5.

The next two results are designed to assist in finding m-sets of a graph G. They show that in order to find an m-set for G it suffices to find m-sets for certain smaller subgraphs of G.

THEOREM 6 (Cutpoint theorem). *Let G be a connected graph having blocks $B_1, B_2, \cdots, B_k$. Then the union of the lines in m-sets for each of the blocks B_i forms an m-set for G, and conversely, the lines in any m-set of G form m-sets for each of the blocks B_i.*

Proof. Let G have q lines and let the blocks B_i have q_i lines; $q = q_1 + q_2 + \cdots + q_k$. Let P_i be an m-set of B_i having $m(B_i) = m_i$ lines.

We first claim that $P_1 \cup P_2 \cup \cdots \cup P_k$ is an m-set of G, i.e., $m(G) = m_1 + m_2 + \cdots + m_k$ and $G \cup P_1 \cup P_2 \cup \cdots \cup P_k = G_k$ is an Eulerian multigraph. It is obvious that G_k is an Eulerian multigraph since

$$G_k = (B_1 \cup P_1) \cup (B_2 \cup P_2) \cup \cdots \cup (B_k \cup P_k)$$

and each $B_i \cup P_i$ is an Eulerian multigraph.

Suppose however that $m(G) < m_1 + m_2 + \cdots + m_k$. Let C_l be a smallest cycle which has a full homomorphism ϕ onto G, i.e., $m(G) = l - q < m_1 + m_2 + \cdots + m_k$. Then, by definition, the multigraph G_l, formed as in the proof of Theorem 1 by C_l, ϕ and G, is Eulerian. Since $G = B_1 \cup B_2 \cup \cdots \cup B_k$, it follows that

$$G_l = B_{1l} \cup B_{2l} \cup \cdots \cup B_{kl},$$

where each multigraph B_{il} is a block of G_l. We claim that each of the multigraphs B_{il} is Eulerian. Since C_l and the homomorphism ϕ define a closed covering walk in G, every time this walk passes through a cutpoint v of G and enters a particular block B_i, it must sooner or later leave the same block B_i through the same cutpoint v. Thus the degree of any cutpoint v in any block B_{il} of G_l must be even. Furthermore, since the degree of any non-cutpoint u in a block B_{il} is the same as its degree in G_l, and since every point has even degree in G_l, every non-cutpoint u has even degree in B_{il}. Thus every point of B_{il} has even degree, hence B_{il} is Eulerian.

Let the multigraphs B_{il} have q_{il} lines. Then the number of lines in G_l is

$$l = q + m(G) = q_{1l} + q_{2l} + \cdots + q_{kl}.$$

Furthermore since we are assuming that

$$m(G) = l - q < m_1 + m_2 + \cdots + m_k,$$

we have that

$$l - q = (q_{1l} - q_1) + (q_{2l} - q_2) + \cdots + (q_{kl} - q_k) < m_1 + m_2 + \cdots + m_k.$$

But this implies that for at least one j,

$$q_{jl} - q_j < m_j,$$

which contradicts the minimality of the m-set P_j of block B_j.

Thus there cannot exist a cycle C_l with a full homomorphism onto G, with $l - q < m_1 + m_2 + \cdots + m_k$. Thus the m-sets P_i form an m-set for G.

Conversely, let P denote the set of lines in an arbitrary m-set of G. We must show that the lines in P form m-sets for each of the blocks B_i of G.

Certainly, $P = \{P \cap B_1\} \cup \{P \cap B_2\} \cup \cdots \cup \{P \cap B_k\}$.

Certainly also, $B_i \cup \{P \cap B_i\}$ is an Eulerian multigraph for $1 \leqq i \leqq k$. This follows by exactly the same argument used above to show that the multigraphs B_{il} are Eulerian.

Suppose however that $\{P \cap B_i\}$ is not an m-set of B_i; let S be an m-set of B_i. Then $P' = P - \{P \cap B_i\} \cup S$ is a set of lines such that $G \cup P'$ is an Eulerian multigraph. Furthermore, since S has fewer lines than $\{P \cap B_i\}$, it follows that P' has fewer lines than P; this contradicts the minimality of P. Thus $\{P \cap B_i\}$ must be an m-set for all i, $1 \leqq i \leqq k$.

COROLLARY 6a. *Every bridge uv of a graph G appears in every m-set of G.*

COROLLARY 6b. *If T is a tree having q lines, then* $m(T) = q$.

This corollary for trees was proved somewhat exhaustively in [2]; here it is an immediate consequence of Theorem 6.

Theorem 6 asserts that in order to find an m-set for a connected graph G all one has to do is to find and put together m-sets for each of the blocks of G. The next theorem provides a technique which simplifies the problem of finding m-sets for the blocks of G.

Let G be a 2-connected graph having two points u, v such that $G - \{u, v\}$ is disconnected. Let $G_1', G_2', \cdots, G_k'$ be the connected components of $G - \{u, v\}$, and let $G_i = G_i' + \{u, v\}$ be the graph obtained by adding points u, v to G_i' together with all lines of G joining a point of G_i' and either u or v; we shall refer to $G_1, G_2, \cdots, G_k$ as the *components of G joined together at u and v.* Notice that if points u and v are adjacent in G, then one of the subgraphs G_i will consist of just the two points u, v and the line (u, v); in this case no other subgraph will contain the line (u, v) (cf. Fig. 1).

Let C_l be a cycle of length $l = e(G) = q + m(G)$; let ϕ be a full homomorphism of C_l onto G; let G_l be the multigraph defined by C_l, ϕ, and G; and let $G_{1l}, G_{2l}, \cdots, G_{kl}$ be the multigraphs which are the components of G_l joined together at u and v.

Consider whether or not an arbitrary G_{il} is an Eulerian multigraph. Since the degree of every point in G_l is even, the degree of every point except possibly u and v is even in G_{il}. If the degree of both u and v is even in G_{il}, then G_{il} is an Eulerian multigraph.

If the degree of both u and v is odd, then G_{il} has a walk which begins at point u, contains every line of G_{il} exactly once and ends at point v; let us call G_{il} in this case *nearly-Eulerian with respect to u and v.*

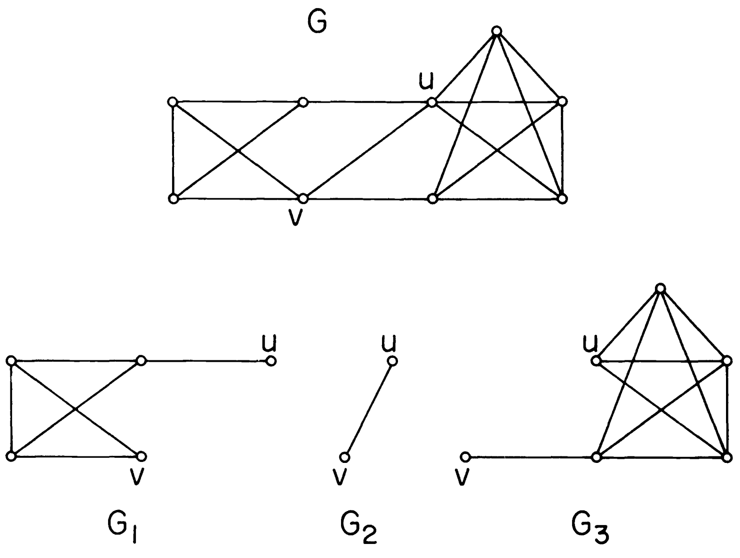


FIG. 1

The remaining case, in which one of u or v has odd degree, and the other, even degree in G_{il} is not possible, since any multigraph must have an even number of points of odd degree.

Thus every multigraph G_{il} is either Eulerian or nearly-Eulerian. Observe next that the number of nearly-Eulerian subgraphs G_{il} must be even. This follows since the degree of u (and v) in G_l is even and the degree of u (and v) in G_l equals the sum of the degrees of u (and v) in the subgraphs G_{il}. Thus there are an even number of subgraphs in which the degree of u is odd, i.e., an even number of nearly-Eulerian subgraphs with respect to u and v.

In what follows let l_i denote the fewest number of multilines needed to be added to G_i in order that the resulting multigraph is nearly-Eulerian with respect to u and v, and let $m_i = m(G_i)$.

THEOREM 7. *Let G be a 2-connected graph having q lines and a cutset of two points u and v; let $G_1, G_2, \cdots, G_k$ be the components of G joined together at u and v. Then*

$$m(G) = \min \sum_{i=1}^{k} n_i,$$

such that for $1 \leqq i \leqq k$, $n_i \in \{l_i, m_i\}$ and the number of l_i appearing in the summation $\sum n_i$ is even.

Proof. We first establish that $m(G) \leqq \min \sum n_i$. Let $n_1 + n_2 + \cdots + n_k = n$ be a minimum sum such that an even number (possibly 0) of the n_j's are l_j's. For each j, let us add a corresponding set of n_j multilines to the subgraph G_j to form the multigraph, say G_{jn}, which is either Eulerian (if $n_j = m_j$) or nearly-Eulerian with respect to u and v (if $n_j = l_j$). But since an even number of n_j's are l_j's, an even number of the subgraphs G_{jn} are nearly-Eulerian. Thus $G_n = G_{1n} \cup G_{2n} \cup \cdots \cup G_{kn}$ is an Eulerian multigraph having $q + n$ lines, i.e., $m(G) \leqq n$.

Conversely we must show that $m(G) \geqq \min \sum n_i$. Suppose $m(G) < \min \sum n_i$. Let $e(G) = l = q + m(G)$ and let C_l be a smallest cycle having a full homomorphism ϕ onto G. As in the proof of Theorem 1, let G_l be the multigraph defined by C_l, ϕ and G; also let $G_{1l}, G_{2l}, \cdots, G_{kl}$ be the induced multigraphs corresponding to the components $G_1, G_2, \cdots, G_k$ which are joined together at u and v.

As we observed earlier these multigraphs G_{jl} are either Eulerian or nearly-Eulerian with respect to u and v, and furthermore an even number of these multigraphs are nearly-Eulerian.

Let the subgraphs G_i have q_i lines and the multigraphs G_{il} have q_{il} lines. Then $q = q_1 + q_2 + \cdots + q_k$ and

$$l = q + m(G) = q_{1l} + q_{2l} + \cdots + q_{kl}.$$

Let $N_i = q_{il} - q_i$. Then $N_1 + N_2 + \cdots + N_k = m(G)$.

All that remains to show now is that either $N_i = l_i$ or $N_i = m_i$, for having done this we will have the contradiction

$$N_1 + N_2 + \cdots + N_k < \min \sum_{i=1}^{k} n_i,$$

where each $n_i \in \{l_i, m_i\}$ and an even number of the n_i's are l_i's.

Assume that G_{il} is Eulerian. Then by definition, $N_i \geqq m(G_i)$. But if $N_i > m(G_i)$, then we could add strictly fewer multilines to G_i to make an Eulerian multigraph, say G_{im}. This multigraph plus all the remaining multigraphs G_{jl} would then form an Eulerian multigraph having strictly fewer lines than G_l, contradicting the minimality of C_l. Thus $N_i = m(G_i)$, i.e., if G_{il} is Eulerian, then $N_i = m(G_i) = m_i$.

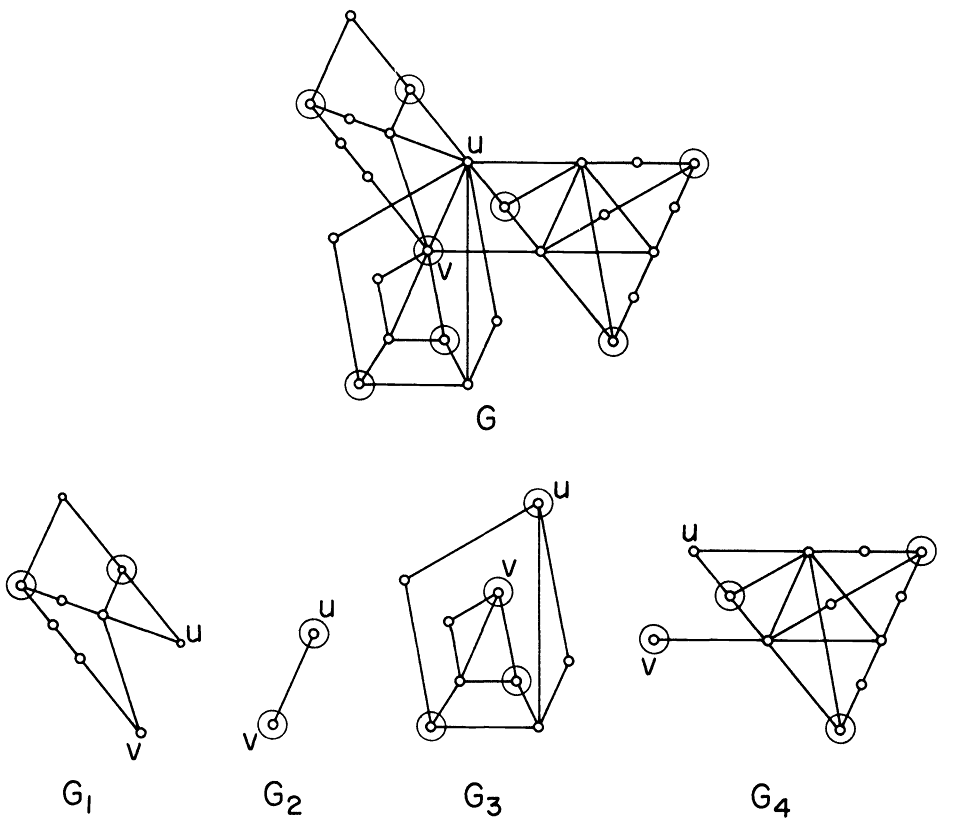
Assume finally that G_{il} is nearly-Eulerian with respect to u and v. The same kind of argument can be used to show that $N_i = l_i$, for if $N_i > l_i$, we can form a new nearly-Eulerian multigraph G_{im} having $q + l_i < q + N_i$ lines which when combined with the remaining multigraphs G_{jl} would form an Eulerian multigraph having strictly fewer lines than G_l, again a contradiction to the minimality of G_l. Thus

$$N_1 + N_2 + \cdots + N_k = \min \sum_{i=1}^{k} n_i,$$

which completes the proof.

We next present an example which illustrates the applicability of Theorem 7. The graph G in Fig. 2 has eight odd points. In order to find an m-set for G we note that G has two points u and v such that $G - \{u, v\}$ is disconnected. The components of G joined together at u and v are labeled G_1, G_2, G_3 and G_4. For each of these subgraphs of G we compute, by inspection, l_i and m_i. Recall that l_i is the least number of multilines we need to add to G_i in order to produce a nearly-Eulerian multigraph with respect to u and v, i.e., in order to produce a multigraph in which u and v are the only two points of odd degree. Recall also that m_i is the fewest number of additional multilines we need to add to G_i in order to produce a multigraph in which every point has even degree.

Having computed l_i and m_i, for $i = 1, 2, 3, 4$, we then find the minimum sum $\sum n_i$, as indicated in Fig. 2, such that there is an even number of n_i's and l_i's. Having selected, say m_1, m_2, l_3 and l_4, we then add the m_1 lines to G_1 which make



$\ell_1 = 4$ $\quad m_1 = 2$ $\qquad \ell_2 = 0$ $\quad m_2 = 1$ $\qquad \ell_3 = 2$ $\quad m_3 = 3$ $\qquad \ell_4 = 4$ $\quad m_4 = 5$

$$\min \sum_{i=1}^{4} n_i = m_1 + m_2 + \ell_3 + \ell_4 = 9$$
$$= m_1 + \ell_2 + m_3 + \ell_4 = 9$$

FIG. 2

G_1 Eulerian (G_{1l}), the m_2 lines to G_2 which make G_2 Eulerian (G_{2l}), the l_3 lines to G_3 which make G_3 nearly-Eulerian (G_{3l}), and the l_4 lines to G_4 which make G_4 nearly-Eulerian (G_{4l}). We then form $G_{1l} \cup G_{2l} \cup G_{3l} \cup G_{4l}$ and obtain the Eulerian multigraph G of Fig. 3. Thus we see that for the graph G of Fig. 2, $m(G) = 9$, $q = 41$ and $e(G) = q + m(G) = 50$.

5. Extensions of the general theory. We first comment that it seems possible that one can extend the cutpoint theorem and the 2-connected theorem to a 3- or even n-connected theorem. The only conceptual problem would seem to be the large number of different kinds of components that would need to be considered and the subsequent complexities of the minimum sums over these components.

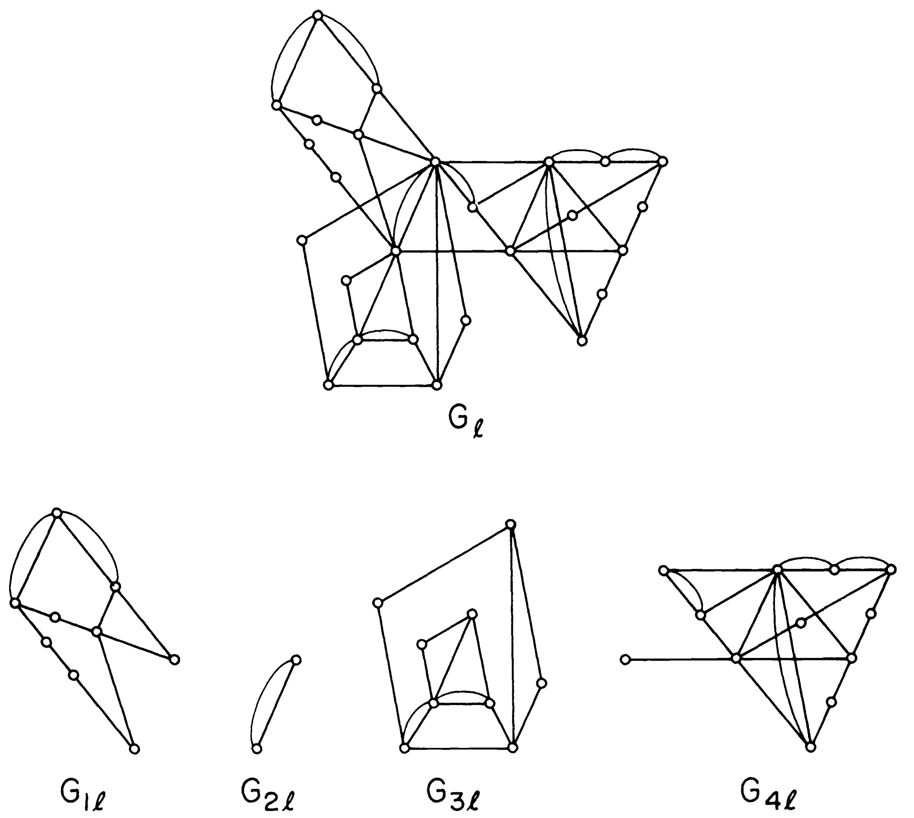


FIG. 3

We next suggest that a very natural next step to this work is to consider the problem of determining the minimum length of a cycle which has a homomorphism (not necessarily *full*) onto a given graph G. In other words, what is the length of a shortest closed walk which contains every *point* of a graph G? In intuitive terms this consideration is equivalent to asking the question: if a given graph is not Hamiltonian, then how close to being Hamiltonian is it? By studying *nearly-Hamiltonian* graphs one might expect to shed some light on the general Hamiltonian problem.

We close with a brief discussion of algorithmic procedures for finding Eulerian walks in an arbitrary graph. Mei-ko [3] has developed a very simple algorithm which works well for small graphs with few cycles, but which is not very feasible for large graphs or computer implementation. The problem can also be approached by recognizing that its basic combinatorial aspects are equivalent to finding a minimum weighted matching on K_{2n}. Each of the $2n$ points represents one of the odd points in the original graph G, and the lines are weighted with the corresponding distances from G. Edmonds [4] has developed a very sophisticated algorithm for handling weighted matching problems. We have constructed a simple branch and bound procedure which is very computer implementable and appears to be efficient for problems up to at least 20 odd points [5].

REFERENCES

[1] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, Mass., 1969.

[2] S. HEDETNIEMI, *On minimum walks in graphs*, Naval Res. Logist. Quart., 15 (1968), pp. 453–458.

[3] K. MEI-KO, *Graphic programming using odd or even points*, Chinese Math., 1 (1962), pp. 273–277.

[4] J. EDMONDS, *Maximum matching and a polyhedron with 0,1-vertices*, J. Res. Nat. Bur. Standards, 69B (1965), pp. 125–130.

[5] S. MITCHELL, S. HEDETNIEMI AND S. GOODMAN, *A branch and bound algorithm for weighted matchings*, DAMACS Tech. Rep. 4–73, University of Virginia, Charlottesville, 1973.

SIAM J. COMPUT.
Vol. 2, No. 1, March 1973

A NEW ALGORITHM FOR FINDING ALL SHORTEST PATHS IN A GRAPH OF POSITIVE ARCS IN AVERAGE TIME $O(n^2 \log^2 n)$ *

P. M. SPIRA†

Abstract. We present a new algorithm for finding the shortest path between each pair of nodes in a directed nonnegatively weighted graph. Our algorithm has average running time $O(n^2 \log^2 n)$ where the graph has n nodes in contrast to previous algorithms in the literature, all of whose average running times are $O(n^3)$ for this problem.

Key words. Shortest paths, algorithm, directed weighted graphs.

1. Introduction. The problem of finding shortest paths in directed weighted graphs has been investigated by many authors. An excellent summary of the history of this problem is given by Dreyfus [1]. In this paper we consider the problem of finding all shortest paths in such a graph under the restriction that all weights are nonnegative real numbers. Prior methods used in this problem—such as that of Dijkstra [2] or that of Floyd [3]—have running time which is $O(n^3)$ to find the paths. We give a new algorithm in this paper and show that if the arcs are independent identically distributed random variables from any distribution whatsoever then the expected running time of our algorithm is $O(n^2 \log^2 n)$. Also the standard deviation is at most $O(n^2 \log n)$.

2. Preliminaries to the result. Before proceeding to our algorithm we give several preliminaries and review certain well-known facts about sorting. Let $\{x_1, \cdots, x_n\}$ be a set of real numbers. We wish to produce an ordered list of them, sequentially identifying the minimum element in the set, the second minimum and so on. Then it is well known that [3] the minimum can be produced in $n - 1$ comparisons and each successive element in the ordered sequence can be produced in $\lceil \log_2 n \rceil$ additional comparisons. In order to review how this is done and for future use in the algorithm we have the following definition.

DEFINITION 2.1. A *played binary tree* is a binary tree some of whose terminal nodes are labeled with real numbers and some of whose terminal nodes are blank. In addition, if either or both successor nodes of a given node is labeled, then the given node is labeled with the smallest value of those on the successor nodes.

We give an example in Fig. 1. It easily follows that the value at the root of such a tree is the minimum value in the tree.

THEOREM 2.2. *There is an algorithm to extract the k smallest of a set* $\{x_1, \cdots, x_n\}$ *of n real numbers in* $n - 1 + (k - 1)\lceil \log_2 n \rceil$ *comparisons.*

Proof. We start with a complete binary tree with $2^{\lceil \log_2 n \rceil}$ leaves. Such a tree has depth $\lceil \log_2 n \rceil$. At the first n leaves place $x_1, \cdots, x_n$ respectively. Play the tree extracting the minimum in $n - 1$ comparisons. To get each successive element

* Received by the editors September 5, 1972.

† Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory, University of California, Berkeley, California 94720. This work was supported by the Joint Services Electronics Program under Contract F44620-71-C0087 and by the U.S. Army Research Office—Durham under Grant DA-ARO-D-31-124-71-G174.

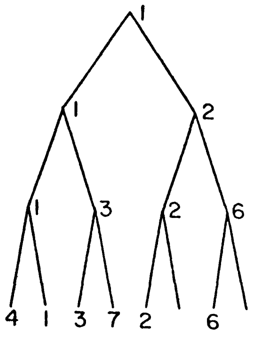


FIG. 1. *A played binary tree*

desired merely erase the path taken by the previous winner and fill in the tree once more. This completes the proof.

We note that one can sort the entire set this way in an asymptotically optimal number of comparisons. We shall use a slight variant of the above result.

THEOREM 2.3. *Let* $\{S_i\}$ *be a family of sets of real numbers such that*

(i) S_i *is a singleton,*

(ii) S_i *is obtained from* S_{i-1} *by deleting the minimum element in* S_{i-1} *and adding either one or two new elements.*

Then there is an algorithm which successively finds the minimum element of $S_1, S_2, \cdots, S_k$ *in at most* $2k\lceil \log_2 n\rceil$ *comparisons provided that* $|S_k| \leqq n$.

Proof. The reader can easily supply the variation of the algorithm in the preceding theorem.

We do not claim that the preceding results are new. In fact they are most likely classified as well-established folklore. But they are given here for convenience and to warm up the reader for what follows.

3. The algorithm. Let G be a directed nonnegatively weighted graph. We wish to find for all pairs (i, j), $1 \leqq i \neq j \leqq n$, the shortest path from i to j. We shall give a new algorithm which is easily understood provided that one is familiar with Dijkstra's algorithm for finding the shortest paths i to j, $1 \leqq j \neq i \leqq n$, for a fixed i (see [2]).

We make the following simple observation about Dijkstra's algorithm.

LEMMA 3.1. *Assume that the shortest paths from node* 1 *to nodes* $i_1, \cdots, i_k$ *have been found by Dijkstra's algorithm. Then the next node to be labeled will be the closest unlabeled node to one of the already labeled nodes or to node* 1.

Proof. Trivial.

This suggests the possibility of sorting the arcs before proceeding with the labeling. In the one origin case this does not pay since sorting is an $O(n^2 \log n)$ operation for a complete n node graph. However, if we want all $n(n - 1)$ shortest paths in the graph, it is possible to improve the existing $O(n^3)$ algorithms by first sorting the arcs in certain cases. This is what we have done.

The idea of the algorithm is simple. We first sort the arcs into n lists, one for the arcs emanating from each of the nodes. This is $O(n^2 \log n)$. We then choose an origin, say node 1. Then we label the closest node to node 1. Say it's node 2.

We then add the distance to node 2 to the length of the shortest arc from node 2 and compare that to the length of the shortest remaining node from node 1. We can permanently label the node indicated by this comparison unless it happens that the arc from node 2 to node 1 wins this contest. If it does we run another contest between the shortest remaining arc from 1 and the sum of the distance to node 2 plus the shortest remaining arc from node 2 now that the arc to node 1 has been bypassed.

Assume we have found the shortest paths to $k - 1$ nodes. With no loss of generality say they are nodes $2, 3, \cdots, k$ and have shortest path lengths $D_2, \cdots, D_k$ respectively. Note that also $D_{11} = 0$. Assume that the shortest remaining arc from node i is d_{im_i} for $1 \leqq i \leqq k$. Then we find

$$\min \{D_{1i} + d_{im_i} : 1 \leqq i \leqq k\};$$

say the minimum is $D_{1j} + d_{jm_j}$. We then label node m_j if it has not already been labeled. If m_j is labeled we take the next arc on the sorted list of arcs from node j, say it's $d_{jm_{j'}}$, add $D_{1j} + d_{jm_{j'}}$, and minimize the set obtained from the previous set by replacing $D_j + d_{jm_j}$ by this new value. If m_j is unlabeled we label it getting $D_{1m_{j'}} = D_{1j} + d_{jm_{j'}}$. We then compute $D_{1j} + d_{jm_{j'}}$ and $D_{1m_j} + d_{m_js}$ where we have assumed that d_{m_js} is the minimum arc on the list of arcs from node m_j. Next we minimize the new set obtained by deleting $D_{1j} + d_{m_j}$ from the previous set and adding these two new values to the set. By proper organization of the elements we are minimizing, Theorem 2.3 tells us that we can identify successive minima in at most $2\lceil \log_2 n \rceil$ comparisons.

We run the tournaments necessary to obtain all shortest paths from node 1 and then proceed in the same way with nodes $2, 3, \cdots, n$ as the origin. If the total number of tournaments is small compared to $n^3/\log n$ we shall have an improvement over existing algorithms. In the next section we discuss this.

We now give a formal description of the algorithm. Let $G = \langle \mathcal{N}, \mathcal{D} \rangle$ be a directed graph where $\mathcal{N} = \{①, ②, \cdots, ⓝ\}$ is the set of nodes and $\mathcal{D} = \{d_{ij}: 1 \leqq i \leqq n, 1 \leqq j \neq i \leqq n\}$ is the set of nonnegative arc lengths where d_{ij} is the distance from node i to node j. By convention $d_{ij} = \infty$ if no such arc exists. When the algorithm terminates D_{ij} will be the length of the shortest path from i to j for all n^2 values of (i, j) with $1 \leqq i \leqq n, 1 \leqq j \leqq n$. D_{ii} will be zero for $1 \leqq i \leqq n$.

To help the reader understand what follows note that $I(i, j)$ will be the index of the node on which the jth shortest arc from node i terminates, LABEL (i, j) will be an indicator of whether we have found D_{ij}, p_i will be a pointer to the first unused arc in the ordered list of arcs starting at node i, i will be the current origin, and S will be the set of paths from which we extract minima by use of the procedure of Theorem 2.3.

The algorithm proceeds as follows:

1. Sort in increasing order the n sets of arcs

$$\{d_{ij}: 1 \leqq j \neq i \leqq n\} \quad \text{for } i = 1, 2, \cdots, n.$$

Set $I(i, j) \leftarrow \ell$ where $d_{i\ell}$ is the jth element in the sorted set of arcs beginning at node i for all

$$1 \leqq i \leqq n, 1 \leqq j \neq i \leqq n.$$

```
    Set $D_{ii} \leftarrow 0$ for $1 \leqq i \leqq n$.
 2. Set LABEL $(i, j) \leftarrow$ NO for $1 \leqq i \leqq n, 1 \leqq j \neq i \leqq n$.
    Set LABEL $(i, i) \leftarrow$ YES for $1 \leqq i \leqq n$. Set $i \leftarrow 1$.
 3. Set $S \leftarrow \varnothing$. Set $p_i \leftarrow 1$ for $1 \leqq i \leqq n$. Set COUNT $\leftarrow 1$. Set $k \leftarrow i$.
 4. Set $s_k \leftarrow D_{ik} + d_{kI(k,\text{COUNT})}$,
    Set $p_k \leftarrow p_k + 1$. Set $S \leftarrow S \cup \{s_k\}$.
 5. Set $t \leftarrow t'$ where $s_{t'} = \min\{s_j : s_j \in S\}$.
    Set $p_t \leftarrow p_t + 1$. (Use algorithm of Theorem 2.3 to find $t$.)
 6. If $p_t = n + 1$ go to 8.
 7. Replace value of $s_t$ in $S$ by $s_t \leftarrow D_{it} + d_{tI(i,p_t)}$.
 8. If LABEL $(i, I(i, p_t - 1))$ = YES go to 5.
 9. Set LABEL $(i, I(i, p_t - 1)) \leftarrow$ YES.
    Set $D_{iI(i,p_t-1)} \leftarrow D_{it} + d_{tI(i,p_t-1)}$. Set $k \leftarrow I(i, p_t - 1)$.
    Set COUNT $\leftarrow$ COUNT + 1.
10. If COUNT $\leqq n$ go to 4.
11. Set $i \leftarrow i + 1$.
12. If $i \leqq n$ go to 3.
13. Stop.
```

4. Analysis of the algorithm. We now analyze the complexity of our algorithm as a function of the number of nodes in the graph. We note that the actual ordering of the arcs will effect the running time of the algorithm and so consider average complexity where the arcs are identically distributed nonnegative random variables.

THEOREM 4.1. *Let p be any probability density function of a real variable x such that $p(x) = 0$ for $x < 0$. Let $M_n(G)$ be the number of steps required by the algorithm to find the shortest path between all pairs of points in an n node graph G whose arcs are independent random variables chosen from p. Then averaged over all such G,*

$$\overline{M}_n \leqq O(n^2 \log^2 n)$$

and the standard deviation of M_n is at most $O(n^2 \log n)$ for any such p whatsoever.

Proof. We consider the number of times step 5 must be executed. It is easy to see that this step is dominant. Let N_{ij} be the number of times step 5 is executed when node i is the origin node and the first $j - 1$ shortest paths have been found. In each of the sorted lists of arcs a certain set has already been used. For the rest of the arcs their sinks are the $n - j$ unlabeled nodes of G plus perhaps some of the already labeled nodes. The order of the relative distances to these nodes is completely independent of the order of relative distances to nodes which are the sinks of arcs which have already been used. Furthermore all possible orderings of the remaining arcs are equally likely. This follows from the fact that G was constructed by independent trials from the probability density function p. Thus the probability that a new node will be labeled is at least $(n - j)/n$ no matter what list the last arc in the next path to win a tournament is drawn from. It follows that

$$\overline{N}_{ij} \leqq \frac{n-j}{n} \sum_{i=1}^{\infty} i(j/n)^{i-1} = \frac{n}{n-j}.$$

But this occurs for n origins and $j = 1, 2, \cdots, n - 1$. Hence since each execution of step 5 is $O(\log n)$ there is a constant C such that for any n,

$$\overline{M}_n \leqq Cn \log n \sum_{j=1}^{n-1} \frac{n}{n-j} = O(n^2 \log^2 n).$$

Now note that N_{ij} is a process whose statistics depend only upon the number of arcs which lead to unlabeled nodes which are left after the first $j - 1$ shortest paths have been found. Also

$$\operatorname{var}(N_{ij}) \leqq \overline{2N_{ij}^2} \leqq \frac{n-j}{n} \sum_{i=1}^{\infty} i^2 (j/n)^{i-1} \leqq \frac{2n^2}{(n-j)^2}.$$

Thus if the total number of executions of step 5 from origin i is N_i, we have

$$\operatorname{var}(N_i) \leqq n^2 \sum_{j=1}^{n-1} \frac{1}{(n-j)^2} \leqq 3n^2.$$

Now note that N_i and N_j are not, in general, independent for $i \neq j$. Nevertheless if N is the total number of executions of step 5 from all origins, we have

$$\operatorname{var}(N) \leqq \overline{N^2} = n\overline{N_1^2} + n(n-1)\overline{N_1 N_2} \leqq n^2 \overline{N_1^2} \leqq 3n^4,$$

proving the theorem.

We can, as usual, combine our algorithm with an $O(n^3)$ algorithm to hedge against our worst case of $O(n^3 \log n)$ operations. Since by Chebyshev's inequality [4, p. 219] the probability that our algorithm runs for more than k standard deviations beyond its mean value is at most $1/k^2$, we can choose k small and run a standard $O(n^3)$ algorithm if our algorithm is not done in $k + 1$ standard deviations. For example, if we choose $k = \sqrt{n}$ the mean will be $O(n^2 \log^2 n)$ and the worst case will still be $O(n^3)$.

Acknowledgment. The author gratefully acknowledges one of the referees for pointing out the need for greater exposition in the proof of Theorem 4.1 than the cursory treatment of an earlier version.

REFERENCES

[1] S. E. Dreyfus, *An appraisal of some shortest path algorithms*, Operations Res., 17 (1969), pp. 395–411.

[2] E. W. Dijkstra, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[3] R. W. Floyd, *Algorithm 97: Shortest path*, Comm. ACM, 5 (1962), p. 345.

[4] W. Feller, *An Introduction to Probability and Statistics*, vol. 1, John Wiley, New York, 1957.

SIAM J. COMPUT.
Vol. 2, No. 1, March 1973

BINARY SEARCH TREES OF BOUNDED BALANCE*

J. NIEVERGELT AND E. M. REINGOLD†

Abstract. A new class of binary search trees, called trees of bounded balance, is introduced. These trees are easy to maintain in their form despite insertions and deletions of nodes, and the search time is only moderately longer than in completely balanced trees. Trees of bounded balance differ from other classes of binary search trees in that they contain a parameter which can be varied so the compromise between short search time and infrequent restructuring can be chosen arbitrarily.

Key words. Balanced trees, lexicographic trees, binary search trees, table look-up.

Introduction. Binary search trees are an important technique for organizing large files because they are efficient for both random and sequential access of records in a file. Two main problems have received attention in the recent literature, each concerned with the search time in such trees.

The first has to do with trees on a fixed set of names (or keys) and associated probabilities. Knuth (1971) and Hu and Tucker (1971) have given algorithms for constructing optimal trees; see Knuth (1973). Bruno and Coffman (1972), and Walker and Gotlieb (1972) have given fast algorithms for constructing near-optimal trees. Nievergelt and Wong (1972) have shown that asymptotically, both optimal and balanced trees have the same average search time.

The second problem, which we consider to be of greater practical importance because of its more realistic assumptions, has to do with trees over a set of names which is dynamic, one which changes in time through insertions and deletions. Hibbard (1962) determined how the average search time behaves if trees are left to grow at random. To improve the search time over that of trees which have grown at random, one looks for trees which satisfy three conflicting requirements: they must be close to being balanced, so that the search time is short; one must be able to restructure them easily when they have become too unbalanced; and this restructuring should be required only rarely. Adel'son-Vel'skii and Landis (1962) (see also Foster (1965) and Knuth (1973)) described a class of trees, now known as AVL trees or height-balanced trees, which strike an elegant compromise between these conflicting requirements.

This paper is intended as a contribution to the second topic. A new class of binary search trees, called trees of bounded balance, or BB trees for short, is described. BB trees share with the height-balanced trees of Adel'son-Vel'skii and Landis (1962) the property that they are easy to maintain in their form despite insertions and deletions of nodes, and that search time is only moderately longer than in balanced trees. They differ from height-balanced trees in one important respect: They contain a parameter which can be varied so the compromise between short search time and frequency of restructuring can be chosen arbitrarily.

* Received by the editors July 5, 1972, and in revised form January 19, 1973.

† Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. This work was supported in part by the National Science Foundation under Grant GJ-31222.

Trees of bounded balance.

DEFINITION. The empty tree, T_0, of zero nodes is a binary tree. A *binary tree* T_n of $n \geqq 1$ nodes is an ordered triple (T_ℓ, v, T_r), where T_ℓ, T_r are binary trees of ℓ, r nodes respectively, $\ell \geqq 0$, $r \geqq 0$, $\ell + r = n - 1$, and v is a single node called the root of T_n.

DEFINITION. The *height* of a binary tree T_n of $n \geqq 1$ nodes is zero if $n = 1$, otherwise it is given by max (height of T_ℓ, height of T_r) + 1.

DEFINITION. The *internal path length* $|T_n|$ of a binary tree T_n is zero if $n \leqq 1$, otherwise it is given by $|T_n| = |T_\ell| + |T_r| + n - 1$.

DEFINITION. The *root-balance* $\rho(T_n)$ of a binary tree $T_n = (T_\ell, v, T_r)$ of $n \geqq 1$ nodes is $\rho(T_n) = (\ell + 1)/(n + 1)$.

DEFINITION. A binary tree T_n is said to be of *bounded balance* α, or in the set BB[α], for $0 \leqq \alpha \leqq 1/2$, if and only if either $n \leqq 1$ or, for $n > 1$ and $T_n = (T_\ell, v, T_r)$, the following hold:

1. $\alpha \leqq \rho(T_n) \leqq 1 - \alpha$, and
2. both T_ℓ and T_r are of bounded balance α.

The notion of root-balance is taken, with slight modification, from Nievergelt and Wong (1973). It is always in the range $0 < \rho(T_n) < 1$, and it indicates the relative number of nodes in the left and right subtrees of T_n. Thus the completely balanced trees T_n of $n = 2^k - 1$ nodes are in BB[1/2], while the Fibonacci trees defined by

$$F_0 = \text{empty}, \quad F_1 = \bullet, \qquad F_{i+2} = \begin{tikzcd}[column sep=small] & \bullet \arrow[dl, dash] \arrow[dr, dash] & \\ F_i & & F_{i+1} \end{tikzcd}$$

can be shown to be in BB[1/3].

It is interesting to note that there is a "gap" in the balance of trees.

THEOREM 1. *For all* α *in the range* $1/3 < \alpha < 1/2$, BB[α] = BB[1/2].

Proof. If T is not completely balanced, i.e., not in BB[1/2], consider a minimal subtree T' of T which is not in BB[1/2]. T' must be of the form (T_ℓ, v, T_r), where both T_ℓ and T_r are in BB[1/2], i.e., $l = 2^s - 1$ and $r = 2^t - 1$, but $s \neq t$ (say $s < t$). Then

$$\rho(T') = \frac{1}{1 + 2^{t-s}} \leqq 1/3,$$

which puts T in BB[α] for some $\alpha \leqq 1/3$.

Search time in BB trees. The height of T_n is a measure of the worst case time required to search T_n. Since the internal path length T_n can be expressed as the sum, over all nodes, of the length of the (unique) path from the root of T_n to each node, it is clear that $|T_n|/n$ is a measure of the average time required to search T_n.

The following theorem is due to Nievergelt and Wong (1973).

THEOREM 2. *If* T_n *is in* BB[α], *then*

$$|T_n| \leqq \frac{1}{H(\alpha)}(n + 1)\log(n + 1) - 2n,$$

where[1]

$$H(\alpha) = -\alpha \log \alpha - (1 - \alpha)\log(1 - \alpha).$$

[1] Throughout this paper, all logarithms are taken base 2.

It is not difficult to show the following theorem.

THEOREM 3. *If T_n is in* BB[α], *then the height of T_n is at most*

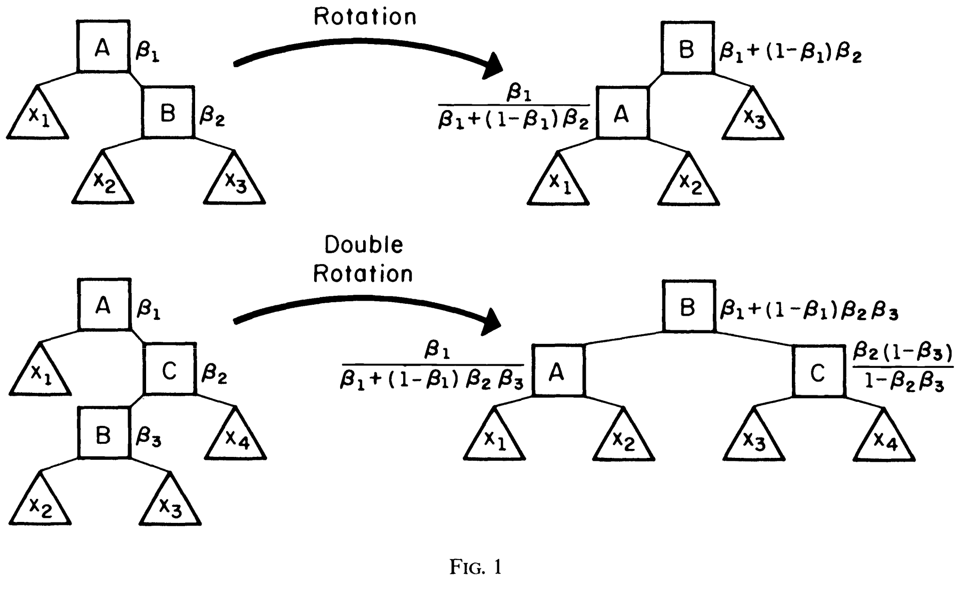
$$\frac{\log(n+1)-1}{\log(1/(1-\alpha))}.$$

Proof. The proof is by induction on n.

The bounds in both of these theorems are sharp for any tree all of whose subtrees have root-balance α. Since the root-balance of a terminal node is 1/2, the only trees with this property are the completely balanced trees of $n = 2^k - 1$ nodes. It is clear, however, that for trees all of whose root-balances are close to α, these bounds are reasonably tight.

These theorems provide bounds on average and worst case search times for trees in BB[α], for any α. For example, trees in BB[1/3] look "sparse," but their internal path length is at most 9% longer than it is for completely balanced trees with the same number of nodes; this follows immediately from Theorem 2 since $1/H(1/3) \approx 1.09$. Hence, searching a tree in BB[1/3] will take, on the average, at most 9% longer than searching a completely balanced tree with the same number of nodes. Similarly, it follows from Theorem 3 that searching a tree in BB[1/3] will take, in the worst case, at most 70% longer.

Rebalancing BB trees. If upon the addition or deletion of a node to a tree in BB[α] the tree becomes unbalanced relative to α, that is, some subtree of T_n has root-balance outside the range [α, $1-\alpha$], then that subtree can be rebalanced by certain *tree transformations* which are of two types (ignoring symmetrical variants), shown in Fig. 1. In Fig. 1 we have used squares to represent nodes, and triangles to represent subtrees; the root-balance is given beside each node.



Rotation
A
β_1
B
β_2
X_1
X_2
X_3
B
$\beta_1+(1-\beta_1)\beta_2$
A
$\frac{\beta_1}{\beta_1+(1-\beta_1)\beta_2}$
X_1
X_2
X_3
Double Rotation
A
β_1
C
β_2
B
β_3
X_1
X_2
X_3
X_4
B
$\beta_1+(1-\beta_1)\beta_2\beta_3$
A
$\frac{\beta_1}{\beta_1+(1-\beta_1)\beta_2\beta_3}$
C
$\frac{\beta_2(1-\beta_3)}{1-\beta_2\beta_3}$
X_1
X_2
X_3
X_4

FIG. 1

THEOREM 4. *If* $\alpha \leqq 1 - \sqrt{2}/2$ *and the insertion or deletion of a node in a tree in* BB[α] *causes a subtree* T *of that tree to have root-balance less than* α, T *can be rebalanced by performing one of the two transformations shown above. More precisely, let* β_2 *denote the balance of the right subtree of* T *after the insertion or deletion has been done. If* $\beta_2 < (1 - 2\alpha)/(1 - \alpha)$, *then a rotation will rebalance* T, *otherwise a double rotation will rebalance* T.

Sketch of proof. Under the various hypotheses, it can be shown that after the transformation has been applied, the new balances are all in the range $[\alpha, 1 - \alpha]$.

If the balance of a subtree goes above $1 - \alpha$, then we use the mirror images of these transformations, and a corresponding theorem. Introducing additional transformations will probably increase the allowable range of α. However, since $1 - \sqrt{2}/2 \approx .2928$ and BB[α] − BB[1/2] is empty for $1/2 > \alpha > 1/3$, $\alpha \leqq 1 - \sqrt{2}/2$ is a reasonable choice. By Theorems 2 and 3 we know that the average search time will be no worse than 15% longer for a BB[$1 - \sqrt{2}/2$] tree than for a completely balanced tree with the same number of nodes, while the worst case search time will be at most twice as long. Considering the ease with which nodes can be added and deleted from BB trees, this moderate increase in search time is justifiable.

Insertion and deletion in BB trees. Assume that each node N of the tree has the form

LLINK	DATA	SIZE	RLINK

and that the DATA field of every node in the left subtree of N is lexicographically before DATA(N), while the DATA field of every node in the right subtree is legicographically after DATA(N); thus the tree is a search tree relative to the lexicographic ordering. SIZE(N) is the number of nodes in the subtree whose root is the node N.[2]

The following algorithm, given in detail in the Appendix, inserts the name NEW to the tree, preserving both the balance and the ordering: Follow links down through the tree going left if NEW is less than the node and right otherwise. If NEW is found to be equal to a name in the tree, then carry out the procedure described in the next paragraph. At each stage of the search, check to see whether the addition of a node to the subtree will unbalance the tree; if not, add one to the size field and continue down the tree. If the subtree does become unbalanced, then perform the appropriate transformation before continuing down the tree.

Notice that we may be modifying the tree for nothing in the event that we discover that NEW is already in the tree after modifications have been made. In that case we retrace the path down the tree correcting the SIZE fields, but *not* restructuring the tree: the restructuring which has been done, albeit unnecessarily, has improved the balance of the tree.

[2] C. A. Crane and D. E. Knuth have suggested that it may be more useful to have SIZE(N) be one plus the size of the left subtree of node N. This has the effect of simplifying the arithmetic in some algorithms (e.g. in finding the kth data item) while making it slightly more complicated in others. See Crane (1972) and Knuth (1973).

Deletion of a node is similar: Follow links down through the tree as before, subtracting one from each SIZE field. If a subtree thus becomes unbalanced, perform the appropriate transformation and continue down the tree. When we arrive at the node N to be deleted, one of three cases arises. If N is a leaf, simply delete it. If N has only one son, link the father of N to the son of N, thus deleting N. Otherwise, find the postorder successor of N (or predecessor, depending which most improves the balance) and promote it to the place of N, taking care to adjust the appropriate SIZE fields and links. Again, if transformations have been made and we find that the node to be deleted is not in the tree, we correct the size fields in a second top-down pass, but do not restructure the tree.

The time required by the insertion and deletion algorithms is clearly proportional to the search time; thus Theorems 2 and 3 demonstrate that insertion and deletion require $O(\log n)$ time. Of obvious interest is the coefficient of the $\log n$. This coefficient will be the same for insertion and deletion as it is for searching, plus whatever time is required to do the rebalancings. Hence it is important to know the expected number of transformations which must be performed during insertion or deletion.

In order to proceed with such an analysis, we must assume some sort of distribution of root-balances in trees of n nodes. Given a tree in BB[α], insertions and deletions have the effect of shifting the root-balance around in the interval $[\alpha, 1 - \alpha]$. The behavior of the root-balance under insertions and deletions is quite similar to a discrete, one-dimensional random walk with reflecting barriers—when a step would take the root-balance outside the interval, a transformation is applied and the root-balance moves closer to 1/2. According to probability theory (see Feller (1968, p. 391)), the distribution of positions for such a random walk is *uniform* over the interval and hence *this is the assumption we will make.* This assumption is weak, however, since the barriers of the random walk corresponding to the shifting of the root-balance are what might be called "repulsing;" they do not just reflect the particle back the same distance that it tried to go forward, but rather they repulse the particle (quite strongly) to send it closer to 1/2. It is thus likely that a more accurate assumption would be a truncated normal distribution centered at 1/2, and hence that the expected number of transformations is even smaller than the following theorem indicates.

THEOREM 5. *Under the (weak) assumption that distribution of root-balances in a* BB[α] *tree is uniform over* $[\alpha, 1 - \alpha]$, *the expected number of tree transformations required for insertion or deletion of a node is less than* $2/(1 - 2\alpha)$.

Sketch of proof. If T_n is a tree of n nodes in BB[α] with ℓ and r nodes in its left and right subtrees, respectively, then

$$\alpha(n + 1) - 1 \leqq \ell, \qquad r \leqq (1 - \alpha)(n + 1) - 1.$$

For simplicity, we shall approximate these lower and upper bounds by αn and $(1 - \alpha)n$, respectively. Since the root-balances are uniformly distributed in $[\alpha, 1 - \alpha]$, each of the $(1 - \alpha)n - \alpha n = (1 - 2\alpha)n$ possible values for ℓ and r is equally likely to occur as the number of nodes in the left and right subtrees, respectively, of T_n. Of all the $(1 - 2\alpha)n$ possible values, only two are critical for insertion or deletion, the largest and the smallest; only for these balances can insertion or deletion cause the tree to go out of BB[α]. Thus the probability, p_n, of causing the root-balance of T_n to go out of the interval $[\alpha, 1 - \alpha]$ is[3]

$$p_n = \frac{1}{(1 - 2\alpha)n},$$

and this is also the probability of having to apply a transformation at the root of T_n during insertion or deletion.

Now the expected number of nodes in the two subtrees of a tree in BB[α] with m nodes is $m/2$ since the average root-balance is 1/2 by the uniform distribution assumption. Hence the expected number of nodes whose root-balances will go out of $[\alpha, 1 - \alpha]$ and will thus need rebalancing is, assuming for simplicity that n is a power of two,

$$p_n + p_{n/2} + p_{n/4} + \cdots + p_{n/n} = \sum_{i=0}^{\log n} \frac{1}{(1 - 2\alpha)n/2^i} = \frac{1}{(1 - 2\alpha)n} \sum_{i=0}^{\log n} 2^i$$
$$= \frac{2n - 1}{(1 - 2\alpha)n} < \frac{2}{1 - 2\alpha}.$$

This completes the sketch of the proof.

It is remarkable that this bound on the expected number of rebalancings is *independent of the size of the tree.* For example, we find that on the average no more than 4.85 transformations will be necessary to insert or delete a node when the tree is in BB$[1 - \sqrt{2}/2]$.

Comparison with height-balanced trees. Height-balanced trees are characterized by the fact that the difference between the heights of the left and right subtrees of any node is at most one. For example, the Fibonacci trees described earlier are a special case of height-balanced trees. The same two transformations serve to restructure a height-balanced tree which has been upset by an insertion or deletion. Adel'son-Vel'skii and Landis (1962) have shown that one transformation is sufficient to rebalance a height-balanced tree which has been unbalanced by an insertion; the expected number being about 0.3 (see Knuth (1973)). Deletion from height-balanced trees may require rebalancing at each level of the tree, but an argument similar to that in Theorem 5 shows that the expected number of transformations needed is constant.

Height-balanced trees cannot be described as BB[α] for any α.

THEOREM 6. *There are trees in* BB[1/3] *which are not height-balanced and for all* $\varepsilon > 0$ *there is a height-balanced tree with root-balance less than* ε.

Proof. The first part of the theorem is shown by considering a tree such as that in Fig. 2 which is in BB[1/3] but which is not height-balanced. The second part is shown by considering a tree whose left subtree is the Fibonacci tree of height h and whose right subtree is the completely balanced tree of height h. As $h \to \infty$ the balance of such a tree, which is height-balanced, goes to zero. This completes the proof.

[3] This implicitly assumes that the node is inserted in (deleted from) either subtree of T_n with probability 1/2, regardless of the size of these subtrees. If one assumed that the probability of insertion or deletion from a subtree is proportional to the size of this subtree, then this analysis would become less favorable for insertion, and more favorable for deletion.

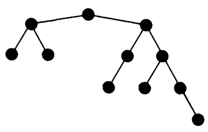


FIG. 2

The search time for height-balanced trees is somewhat better than for BB trees. The worst case search time for height-balanced trees is about 1.44 log n comparisons. The average search time can be as bad as 1.05 log n comparisons; this is the average search time for Fibonacci trees; the exact bound is unknown. In contrast, the search times for a tree in BB[α] are bounded by Theorems 2 and 3. For example, when $\alpha = 1 - \sqrt{2}/2$, the worst case search time is about 2 log n and the average search time is less than 1.15 log n.

Insertion and deletion of nodes in height-balanced trees require a top-down pass over the path from the root to the node to be inserted or deleted, followed by a bottom-up pass over that same path (for insertion, the second pass can instead be top-down). Typically, the bottom-up pass is accomplished by the use of a pushdown stack. The use of a stack can be eliminated but only at some cost in time or memory (e.g., every node could store an upward pointer to its father. Alternatively, the return path to the root can be retained by reversing the links on descent, and restoring them on the way back up; an additional bit would be necessary to indicate whether the right link or the left link had been reversed). In most cases, insertion and deletion of nodes in BB trees is accomplished by a single top-down pass over the path. In the event of a redundant insertion or deletion, a second top-down pass is necessary. Unlike a bottom-up pass, an additional top-down pass does not require the use of a pushdown stack or its equivalent.

A height-balanced tree requires at least 2 bits of storage for every node to indicate which of the three possible conditions holds between the heights of its two subtrees. By comparison, a node in a BB-tree requires more storage because it has to hold the size of the tree rooted at this node. However, some important benefits compensate for this. Such important operations as finding the kth data element, or the qth quantile, or how many elements there are lexicographically between x and y, can all be done in time $O(\log n)$, while they seem to require time $O(n)$ if the size information is not explicitly stored.

Table 1 summarizes the comparison of random trees (BB[0]), height-balanced trees, BB[$1 - \sqrt{2}/2$] trees, and completely balanced trees (BB[1/2] if we ignore semileaves).

The important advantage BB trees have over height-balanced trees is that the trade-off between search time and insertion/deletion time can be specified by the appropriate choice of α, the bound on the balance. Thus when insertions and deletions are rare α could be chosen close to $1 - \sqrt{2}/2$, while if insertions and deletions are very frequent, α could be chosen closer to zero. It is not easy to generalize height-balanced trees to include a parameter which has the function of α. The obvious choice would be to define a tree T to be of *height-balance h* if and only if for every node N of T, the heights of the two subtrees of N differ by at most h (the special case $h = 1$ yields the conventional height-balanced trees); see Foster (1972). This suffers from the fact that the smallest possible change in h

TABLE 1

	Number of comparisons needed to search the tree in the worst possible case	Expected number of comparisons needed to search an average tree	Time required to return an unbalanced tree to its class
Random trees of n nodes (BB[0])	n	$1.39 \log(n + 1)$*	0
AVL trees of n nodes	$1.44 \log(n + 1)$	$\log(n + 1) + 0.25$†	$O(\log n)$
BB[$1 - \sqrt{2}/2$] trees of n nodes	$2 \log(n + 1)$	$1.05 \log(n + 1)$†	$O(\log n)$
Completely balanced trees of n nodes (BB[1/2], ignoring semileaves)	$\log(n + 1)$	$\log(n + 1)$	$O(n)$

* Due to Hibbard (1962).
† Based on empirical evidence; the exact bound is unknown.

(say from $h = 1$ to $h = 2$) changes the class of trees very drastically, and thus the compromise between search time and rebalancing time cannot be finely tuned.

Appendix. The insertion algorithm. The algorithm presented in this Appendix is given in sufficient detail to make its implementation fairly easy; we have implemented and tested it in SNOBOL.

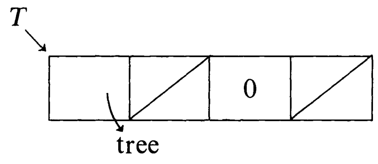
Assume that each node of the tree has the form

LLINK	DATA	SIZE	RLINK

with the four fields as previously described. To simplify notation, if T is a pointer to a tree whose root is such a node, then

$$\|T\| = \begin{cases} 0 & \text{if } T \text{ is empty,} \\ \text{SIZE}(T) & \text{otherwise.} \end{cases}$$

The name NEW is to be added to the BB[α] tree pointed to by a header node:



R is a pointer which will be used in the search through the tree to find out where NEW should be added. RP is always one step behind R in the tree; that is, RP will point to the father of the node pointed to by R. S is a variable whose value is either "L" or "R" and $S \cdot$ LINK is either LLINK or RLINK according to the value of S. For example,

$$S = \text{“}L\text{”},$$

$$S \cdot \text{LINK}(P) \leftarrow P$$

has the same effect as

$$\text{LLINK}(P) \leftarrow P.$$

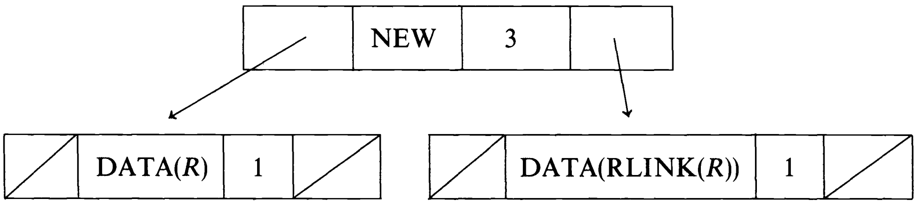
The value of S together with the value of RP tell us which pointer has to be modified when we rebalance a subtree.

Step 1 (Initialize). Set $RP \leftarrow T$ and $S \leftarrow$ "L". Now the pointer which is one step behind points to the header node of the tree.

Step 2 (Small tree?). Set $R = S \cdot \text{LINK}(RP)$; this moves us down one level in the tree. If $1/(\|R\| + 2) \geqq \alpha$ then insert NEW in the subtree pointed to by R using the obvious method, i.e. without any rebalancing; we can do this if the tree is small enough. If in doing this insertion we discover that NEW is already in the tree, then go to Step 9.

Step 3 (Compare). Compare NEW to DATA(R). If NEW = DATA(R), then the name is already in the tree, so we go to Step 9. If NEW < DATA(R), go to Step 7.[4]

Step 4 (Rotation no help?). If $\|R\| = 2$, RLINK(R) is not null, and DATA(RLINK(R)) > NEW, then set $S \cdot \text{LINK}(RP)$ to point to the structure



and stop. When $\alpha < 1/4$ this keeps the insertion algorithm from going into an infinite loop on a subtree of two nodes when the name to be inserted lies *between* them, e.g.,

$$\begin{tikzcd}[column sep=small] \text{“}A\text{”} \arrow[dr, no head] & \\ & \text{“}C\text{”} \end{tikzcd}$$

when we try to insert "B".

Step 5 (Add to right subtree). Compute what the new balance of R will be after insertion:

$$v = \frac{\|\text{LLINK}(R)\| + 1}{\|R\| + 2}.$$

If $\alpha \leqq v \leqq 1 - \alpha$, then no rebalancing is needed at this level, so set

$$\text{SIZE}(R) \leftarrow \text{SIZE}(R) + 1,$$

$$S \leftarrow \text{“}R\text{”},$$

$$RP \leftarrow R,$$

$$R \leftarrow \text{RLINK}(R)$$

and go to Step 2.

[4] The two cases are *not* handled symmetrically. Step 4 is needed to prevent an infinite loop under the conditions described; if the symmetrical case arises, the algorithm performs a rotation in Step 7 and returns, eventually, to Step 4.

Step 6 (Rebalance from right to left). Rebalance, using the transformations given in the figure *before* adding the name. If $\|R\| = 2$, then use rotation. Otherwise, compute the value β_2 will have *after* the insertion of NEW. If $\beta_2 < (1 - 2\alpha)/(1 - \alpha)$, then use rotation, otherwise use double rotation. Set $S \cdot \text{LINK}(RP)$ to point to the rebalanced subtree and go to Step 2.

Step 7 (Add to left subtree). Compute what the new balance of R will be *after* insertion:

$$v = \frac{\|\text{LLINK}(R)\| + 2}{\|R\| + 2}.$$

If $\alpha \leqq v \leqq 1 - \alpha$, then no rebalancing is needed at this level, so set

$$\text{SIZE}(R) \leftarrow \text{SIZE}(R) + 1,$$
$$S \leftarrow \text{“}L\text{”},$$
$$RP \leftarrow R,$$
$$R \leftarrow \text{LLINK}(R),$$

and go to Step 2.

Step 8 (Rebalance from left to right). Rebalance, using the mirror images of the transformations in the figure, *before* adding the name. If $\|R\| = 2$, then use rotation. Otherwise compute the value β_2 will have *after* the insertion of NEW. If $1 - \beta_2 < (1 - 2\alpha)/(1 - \alpha)$, then use rotation, otherwise use split-rotation. Set $S \cdot \text{LINK}(RP)$ to point to the rebalanced subtree and go to Step 2.

Step 9 (Duplicate name). We have found that NEW is already in the tree, so a second top-down pass is needed to correct the size fields. Set $R \leftarrow \text{LLINK}(T)$ so it points to the top of the tree.

Step 10 (Correct size field). Compare NEW to DATA(R). If NEW = DATA(R) we are done. Otherwise set $\text{SIZE}(R) \leftarrow \text{SIZE}(R) - 1$. Then, if NEW > DATA($R$) set $R \leftarrow \text{RLINK}(R)$, otherwise set $R \leftarrow \text{LLINK}(R)$. Repeat Step 10.

Acknowledgment. We are grateful to C. A. Crane and D. E. Knuth for very helpful comments which were based on an earlier version of this paper.

REFERENCES

G. M. Adel'son-Vel'skii and Ye. M. Landis (1962), *An algorithm for the organization of information*, Dokl. Akad. Nauk SSSR, 146, pp. 263–266 = Soviet Math. Dokl., 3, pp. 1259–1263.

J. Bruno and E. G. Coffman (1972), *Nearly optimal binary search trees*, Information Processing 71, vol. 1, North Holland, Amsterdam, pp. 99–103.

C. A. Crane (1972), *Linear lists and priority queues as balanced binary trees*, Doctoral thesis, Stanford Univ., Stanford, Calif.

W. Feller (1968), *An Introduction to Probability Theory and its Application*, vol. 1, 3rd ed., John Wiley, New York.

C. C. Foster (1965), *Information storage and retrieval using AVL trees*, Proc. ACM 20th National Conference, pp. 192–205.

——— (1972), *A generalization of AVL trees*, Tech. Note TN/CS/00033, Computer and Information Sciences Department, Univ. of Massachusetts, Amherst; Comm. ACM, to appear.

T. HIBBARD (1962), *Some combinatorial properties of certain trees*, J. Assoc. Comput. Mach., 9, pp. 13–28.

T. C. HU AND A. C. TUCKER (1971), *Optimal computer search trees and variable-length alphabetical codes*, SIAM J. Appl. Math., 21, pp. 514–532.

D. E. KNUTH (1971), *Optimum binary search trees*, Acta Informatica, 1, pp. 14–25.

——— (1973), *The Art of Computer Programming. Volume III: Sorting and Searching*, Addison-Wesley, Reading, Mass.

J. NIEVERGELT AND C. K. WONG (1972), *On binary search trees*, Information Processing 71, vol. 1, North Holland, Amsterdam, pp. 91–98.

——— (1973), *Upper bounds for the total path length of binary trees*, J. Assoc. Comput. Mach., 20, pp. 1–6.

W. A. WALKER AND C. C. GOTLIEB (1972), *A top down algorithm for constructing nearly-optimal lexicographic trees*, Graph Theory and Computing, R. C. Read, ed., Academic Press, New York, pp. 303–323.

SIAM J. Comput.
Vol. 2, No. 1, March 1973

ISOMORPH REJECTION AND A THEOREM OF DE BRUIJN[*]

S. G. WILLIAMSON[†]

Abstract. Let G be a group acting on a finite set S. Let L be a G-stable subset of S and denote by Δ a transversal for the orbit partition of G acting on L. The problem of devising efficient generating algorithms for Δ is called the "isomorph rejection" problem. Given a field F of characteristic zero, let F^S denote all functions from S to F. Note that F^S is an algebra under the operations of pointwise addition and multiplication. The problem of generating Δ or I_Δ (the indicator or characteristic function of Δ) may be regarded as a special case of the problem of generating and representing functions φ in F^S (in case $\varphi = I_\Delta$). In this context, instead of attempting to generate I_Δ directly one may instead choose a pair of linear operators (T_0, T_1) and attempt to construct functions $v_1, \cdots, v_p$ in F^S such that $T_0(v_1 + \cdots + v_p) = T_1(I_\Delta)$. The construction of $(v_1, \cdots, v_p)$ may be regarded as a "weak solution" to the isomorph rejection problem (which becomes an actual solution if $T_0 = T_1 =$ identity). In this paper properties of such constructions are considered in the interesting case where $S = R^D$ is itself a finite function set and the operators T_0 and T_1 are constructed from representations of the group G as operators on F^S. The extraction of information from such weak solutions is carried out by means of linear functionals on F^S. In this setting certain functionals yield information about the cardinality of Δ in the form of various well-known identities due to deBruijn and Pólya. Some examples of weak solutions to isomorph rejection problems are given.

Key words. Isomorph rejection, tensor algebra, generating functions, Pólya's counting theorem.

1. Introduction. Let L be a set and let G be a group acting on L. All sets are assumed to be finite. We are concerned with the problem of generating a transversal Δ for the orbit partition of the action of G on L. Such problems are often called "isomorph rejection problems" [4]. The idea, of course, is to construct an efficient algorithm for the generation of Δ. The nature of this problem as manifested by various examples that occur in practice is of considerable interest. One trivial consequence of such an algorithm is that we know the cardinality $|\Delta|$ of Δ. Thus we have "solved" an enumeration problem. However, having *only* solved an enumeration problem we may be far from having solved the corresponding isomorph rejection problem in any meaningful way. Having the list Δ stored in a computer in some form gives in general far more information than simply knowing the cardinality $|\Delta|$. Thus the very worst accepted procedure for computing $|\Delta|$ should in some sense be no more complicated than the best known generating algorithm for constructing Δ. The problem of computing $|\Delta|$ (and related questions) has been given considerable attention. In what follows we discuss certain interrelationships between the construction problem and results concerned with the computation of $|\Delta|$, particularly the work of de Bruijn [1], [2].

2. The set Δ mentioned above is a subset of L. For any finite sets R and D let R^D denote the set of all functions with domain D and range R. Suppose the sets L under consideration are all subsets of some set S. Let I_L denote the indicator or characteristic function of L. The problem of generating and storing Δ in machine computation is trivially equivalent to generating and storing $I_\Delta \in \{0, 1\}^S$. It is

* Received by the editors September 6, 1972.

† Department of Mathematics, University of California, San Diego, La Jolla, California 92037. This work was supported by the Air Force Office of Scientific Research, Air Force Systems Command, U.S.A.F., under AFOSR Contract/Grant 71-2089.

frequently convenient (in connection with obtaining generating functions for example [1], [2], [5]) to consider subsets L of S where a "weight" or element from a field F is assigned to each element of L. Thus instead of considering the space of characteristic functions $\{0, 1\}^S$, we replace $\{0, 1\}$ by a field F of characteristic zero. We thus consider the problem of generating and storing functions $\varphi \in F^S$, which includes the case $\varphi = I_\Delta$. We remark that F^S is an F-algebra of functions of dimension $|S|$ under the usual operations of pointwise addition and multiplication. If $S = R^D$ is itself a set of functions with domain D, $|D| = d$, then we say that the corresponding F-algebra has *rank d*. In any case, the indicator functions of the one point subsets $I_{\{x\}}$, $x \in S$, form what is usually called the "standard basis" of the algebra. For example, if $S = \{1, \cdots, n\}$, then $I_{\{k\}}$ is the function which assigns 1 to k and 0 to all $j \neq k$. That is, $I_{\{k\}} = \begin{pmatrix} 1 \cdots k \cdots n \\ 0 \cdots 1 \cdots 0 \end{pmatrix}$ or, more briefly, $I_{\{k\}} = (0, \cdots, 1, \cdots, 0)$ is the basis vector for the space of n-tuples that assigns a "1" to the kth "coordinate" and "0" to all other "coordinates." In the case where $S = R^D$, $|R| = r$, $|D| = d$, F^S has dimension r^d and the standard basis is the set $\{I_{\{f\}}, f \in R^D\}$. For convenience of notation we write simply I_f for the indicator function of $\{f\}$, $R = \{1, \cdots, r\}$, $D = \{1, \cdots, d\}$.

We now describe certain special features of the algebra F^S when $S = R^D$. Let $M_{d,r}(F)$ denote the set of $d \times r$ matrices over F. For $A \in M_{d,r}(F)$ let $\hat{A}$ denote the element of F^S defined at each function $f \in S$ by $\hat{A}(f) = \prod_{t=1}^d a_{tf(t)}$, where a_{ij} denotes the i, jth entry of A. If $S = \{1, 2\}^{\{1,2\}}$ and if $A = \begin{pmatrix} 1 & 3 \\ -1 & 2 \end{pmatrix}$ and $f(1) = 1$, $f(2) = 1 \left(\text{i.e., } f = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}\right)$, then $\hat{A}(f) = -1$. Let $H_{d,r} = \{\hat{A} : A \in M_{d,r}(F)\}$ denote the set of all such functions in F^S. We shall call $H_{d,r}$ the set of *homogeneous functions of rank d*. We leave it to the reader to verify that $\hat{A} = 0$ (the identically zero function) if and only if some row $A_{(i)}$ of the matrix A is zero. Also the reader should check that $\hat{A} = \hat{B} \neq 0$ if and only if there exist $c_i \in F$, $i = 1, \cdots, d$, such that for each i the ith row $A_{(i)} = c_i B_{(i)}$ and $\prod_{i=1}^d c_i = 1$. These two facts interpret the equality of functions in $H_{d,r}$ in matrix terms. The reader should also check that the standard basis elements are all in $H_{d,r}$. In the example above where $f = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}$ we have $I_f = \hat{A}$, where $A = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$ as the reader may easily check from the definitions. Note also that if $\alpha \in F$ and $\hat{A} \in H_{d,r}$ then $\alpha\hat{A} = \hat{B} \in H_{d,r}$, where B is obtained from A by multiplying any row (no matter which!) by α. Thus in our example $6\hat{A} = \hat{B}$, where $B = \begin{pmatrix} 6 & 18 \\ -1 & 2 \end{pmatrix}$ or $B = \begin{pmatrix} 1 & 3 \\ -6 & 12 \end{pmatrix}$. Since F^S is an algebra it should make sense to compute the product $\hat{A}\hat{B}$. Observe that $\hat{A}\hat{B}(f) = \hat{A}(f)\hat{B}(f)$ by definition of multiplication in F^S. The latter expression is

$$\prod_{t=1}^{d} a_{tf(t)} \prod_{t=1}^{d} b_{tf(t)} = \prod_{t=1}^{d} a_{tf(t)} = \hat{C}(f),$$

where $C = (c_{ij})$ with $c_{ij} = a_{ij}b_{ij}$. Thus $C = A \cdot B$ is the Schur product (Hadamard product) of the two matrices A and B obtained by componentwise multiplication. Addition of elements of $H_{d,r}$ is more involved. The reader should check for example that if $A = \begin{pmatrix} 1 & -1 \\ 1 & 2 \end{pmatrix}$, $B = \begin{pmatrix} 2 & 3 \\ 1 & 2 \end{pmatrix}$, then $\hat{A} + \hat{B} = \hat{C}$, where $C = \begin{pmatrix} 3 & 2 \\ 1 & 2 \end{pmatrix}$ (take $R = D = \{1, 2\}$ as above). In general, if $A_{(i)} = B_{(i)} = C_{(i)}$ for all $i \neq j$ and if $C_{(j)} = A_{(j)} + B_{(j)}$, then $\hat{A} + \hat{B} = \hat{C}$. Unless storage space is a problem this rule can be ignored in most combinatorial computations described below. Note that one would expect to have to specify r^d elements of the field F to describe an "arbitrary" function in F^S where $S = R^D$, $|R| = r$, $|D| = d$. However, any element of $H_{d,r}$ is specified by only rd elements of F. What is more, since the standard basis of F^S is contained in $H_{d,r}$, any $\varphi \in F^S$ can be written as a finite sum $\hat{A}_1 + \cdots + \hat{A}_p = \varphi$ of elements of $H_{d,r}$. There is a minimal p for which this statement is valid, although the corresponding functions $\hat{A}_i$, $i = 1, \cdots, p$, may not be elements of the standard basis. The computation of this p and the corresponding $\hat{A}_i$ functions can be extremely difficult. However, even suboptimal solutions to this problem can represent some quite compact and curious ways of representing functions φ or indicator functions I_A or I_Δ in F^S, as we shall see below.

We now consider an elementary example. Let $D = \{1, 2, 3, 4\}$ and $R = \{1, 2, \cdots, r\}$. Think of D as the four vertices of a square as shown in Fig. 1(a). A function $f \in R^D$, for example $f = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 1 \end{pmatrix}$, represents a "labeling" of the vertices with integers from R as shown in Fig. 1 (b).

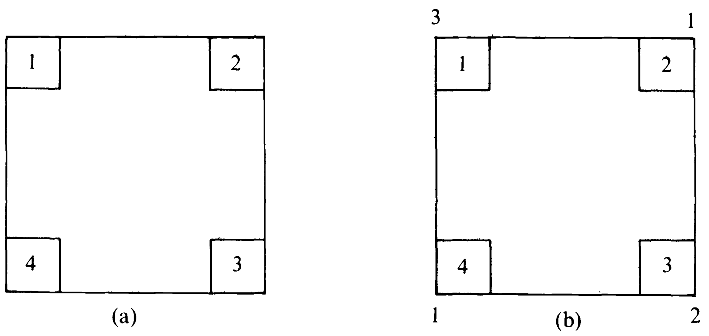


FIG. 1

Now, let $S = R^D$ and let $L \subset S$ denote all functions $f \in R^D$ which have the property that if a "1" appears at a vertex the diagonally opposite vertex is also a "1". It is easy to check that the indicator $I_L = \hat{A} + \hat{B} + \hat{C} + \hat{D}$, where A, B, C, and D are $4 \times r$ matrices given by

$$A = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 0 & 1 & \cdots & 1 \\ 0 & 1 & \cdots & 1 \\ 0 & 1 & \cdots & 1 \end{bmatrix}.$$

Let $G = C_4$, the cyclic group generated by the cycle $\tau = (1\quad 2\quad 3\quad 4)$, act on D. For $\sigma \in G, f \in R^D$, define $(\sigma f) = f \circ \sigma^{-1}$ (f composed with σ^{-1}). Thus G acts on S. Clearly L is invariant under the action of G. Let Δ be a transversal for this action. We are interested in I_Δ. The set Δ represents all squares with vertices labeled with integers $\{1, \cdots, r\}$, where two squares are regarded as the same up to rotations in the plane, and where any "1" must have a "1" diagonally opposite to it. The following remarks represent one way of looking at this question which extends to many other situations in an interesting way. For $\sigma \in G$ define linear operators $\bar{\sigma}$ and $\tilde{\sigma}$ on the standard basis of F^S by $\bar{\sigma} I_f = I_{(\sigma f)}$ and $\tilde{\sigma} I_f = I_f \chi(\sigma f = f)$. Here χ (*statement*) is 1 if *statement* is true and 0 if *statement* is false. We define $\bar{\sigma}$ and $\tilde{\sigma}$ on all of F^S by linear extension from the standard basis. If X is any $d \times r$ matrix, then $\bar{\sigma}\hat{X} = \hat{Y}$, where the ith row $Y_i = X_{(\sigma-1i)}$. That is Y is X with the rows permuted according to σ^{-1}. For example $\tau\hat{B} = \hat{C}$, $\tau^2\hat{B} = \hat{B}$, $\tau^3\hat{B} = \hat{C}$ in the representation of I_L given above. We define two linear operators

(2.1) $$T_G = \frac{1}{|G|}\sum_{\sigma \in G} \bar{\sigma}$$

and

(2.2) $$Q_G = \frac{1}{|G|}\sum_{\sigma \in G} \tilde{\sigma}.$$

Observe that we have two representations of I_L as a sum of elements in $H_{d,r}$. Trivially,

(2.3) $$I_L = \sum_{f \in L} I_f,$$

and from the above,

(2.4) $$I_L = \hat{A} + \hat{B} + \hat{C} + \hat{D}.$$

Using the operator T_G we may write

(2.5) $$I_L = \sum_{f \in \Delta} \frac{|G|}{|G_f|} T_G(I_f) = T_G\left(\sum_{f \in \Delta} \frac{|G|}{|G_f|} I_f\right),$$

where G_f is the stabilizer of G at f and

(2.6) $$I_L = T_G(\hat{A} + 2\hat{B} + \hat{D}).$$

From the definition of Q_G one easily checks that $Q_G(I_f) = (|G_f|/|G|)I_f$ and that $T_G Q_G = Q_G T_G$. Thus from (2.5) and (2.6) we obtain

(2.7) $$T_G Q_G(\hat{A} + 2\hat{B} + \hat{D}) = T_G\left(\sum_{f \in \Delta} I_f\right).$$

For convenience we make the following definition.

Definition 2.8. Let $N \subset S = R^D$ and let T_0 and T_1 be linear operators on F^S. A sequence of elements $(\hat{A}_1, \cdots, \hat{A}_q)$ in $H_{d,r}$ will be called *a* (T_0, T_1) *composition or pre-list of length* q *of* N if $T_0(\hat{A}_1 + \cdots + \hat{A}_q) = T_1(\sum_{f \in N} I_f)$.

We remark that if $T_0 = T_1 =$ identity, then $\hat{A}_1 + \cdots + \hat{A}_q = I_N$ and finding $(\hat{A}_1, \cdots, \hat{A}_q)$ is equivalent to knowing the list N. We may thus say that $(\hat{A}, 2\hat{B}, \hat{D})$ is a (T_GQ_G, T_G) composition or pre-list of Δ. The construction of the set Δ or equivalently $I_\Delta = \sum_{f \in \Delta} I_f$ is the desired output from an algorithm designed to solve the isomorph rejection problem for the list L under the action of the group G. Thus to construct $(\hat{A}, 2\hat{B}, \hat{D})$ is to construct the preimage under T_GQ_G of the image of the desired list under T_G. Setting $p = (1 \quad 0 \cdots 0)$, $q = (0 \quad 1 \cdots 1)$, $s = (1, \cdots, 1)$, we may represent $(\hat{A}, 2\hat{B}, \hat{D})$ by the table of Fig. 2(a). The top row represents the coefficients of the $\hat{A}$, $\hat{B}$ and $\hat{D}$. The vectors $\hat{A}$, $\hat{B}$, and $\hat{D}$ are in turn specified by the columns.

$(\hat{A}, 2\hat{B}, \hat{D}) =$

1	2	1
p	p	q
p	q	q
p	p	q
p	q	q

(a)

1	-4	$+4$
s	p	p
s	s	p
s	q	q
s	s	q

(b)

FIG. 2. *Two (T_GQ_G, T_G) compositions or pre-lists of* Δ ($p = (1 \quad 0 \cdots 0)$, $q = (0 \quad 1 \cdots 1)$ $s = (1 \quad 1 \ldots 1)$).

This table (some remarks on Fig. 2(b) are made below) is not a "solution" to the isomorph rejection problem but still retains much of the desired information about Δ. For example, if we define l to be the linear functional on F^S such that $l(I_f) = 1$ for all f, then $l(\sum_{f \in \Delta} I_f) = |\Delta|$ is the number of elements in the list Δ. It is trivial to check that for any $\hat{X} = (\hat{x}_{ij})$ in $H_{d,r}$, $l(\hat{X})$ is the product of the row sums of $\hat{X}$. Thus $lT_G\hat{X} = l\hat{X}$ for any $\hat{X}$, this product being independent of the order of the rows. Thus $lT_G = l$ as operators and $lT_GQ_G\hat{X} = lQ_G\hat{X}$. To compute $lQ_G\hat{X}$ we must compute $l\bar{\sigma}\hat{X}$ for each $\sigma \in G$. The rule for such a computation is a special case of Theorem 3.2 below but is easy to state:

(a) Construct $D\sigma$, the partition of D induced by the cycles of σ (i.e., the orbit classes of the cyclic group $\langle\sigma\rangle$ generated by σ as $\langle\sigma\rangle$ acts on D).

(b)
$$l\bar{\sigma}\hat{X} = \prod_{a \in D\sigma} \sum_{j=1}^{r} \left(\prod_{t \in a} x_{tj} \right).$$

For example, let

$$\hat{X} = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \end{pmatrix}$$

and let $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$. Then $D\sigma = \{\{1, \ 3\}, \{2, \ 4\}\}$ since the cycle decomposition of $\sigma = (13)(24)$. For $a \in D\sigma$ we have either $a = \{1, \ 3\}$ or $a = \{2, \ 4\}$. As $r = 3$, (b) becomes

$$l\tilde{\sigma}\hat{X} = (x_{11}x_{31} + x_{12}x_{32} + x_{13}x_{33})(x_{21}x_{41} + x_{22}x_{42} + x_{23}x_{43}).$$

With these observations it is a trivial matter to compute l of the left-hand side of (2.7) (to obtain l of the right-hand side which is $|\Delta|$). We consider $\sigma = e, \tau, \tau^2$ and τ^3. Note that $l\tilde{\tau} = l\tilde{\tau}^3$ and that the row sums of p and q are 1 and $r - 1$ respectively. We compute, referring to (2.7) and Fig. 2(a),

$$\begin{aligned} l\tilde{e}(\hat{A} + 2\hat{B} + \hat{D}) &= 1 + 2(r-1)^2 + (r-1)^4, \\ 2l\tilde{\tau}(\hat{A} + 2\hat{B} + \hat{D}) &= 2(1 + 0 + (r-1)), \\ l\tilde{\tau}^2(\hat{A} + 2\hat{B} + \hat{D}) &= 1 + 2(r-1) + (r-1)^2. \end{aligned}$$

Thus we have, referring to (2.2),

$$|\Delta| = 1 + (r-1) + \tfrac{3}{4}(r-1)^2 + \tfrac{1}{4}(r-1)^4 \tag{2.9}$$

as a polynomial in r. (The same procedure applied to Fig. 2(b) yields $|\Delta| = r^4/4 - r^3 + \frac{9}{4}r^2 - \frac{3}{2}r + 1$ which is easily seen to be the same polynomial as specified by (2.9)). For example when $r = 2$; $|\Delta| = 3$ and when $r = 3$, $|\Delta| = 10$. These lists are shown in Fig. 3.

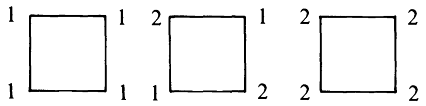


FIG. 3(a). $r = 2$, $|\Delta| = 3$

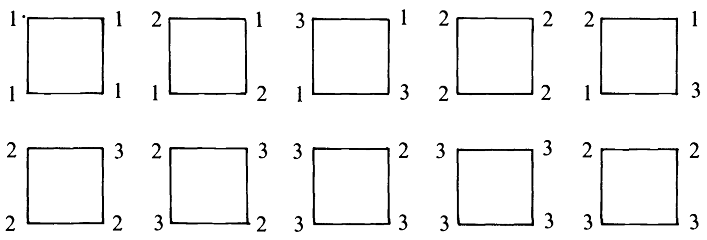


FIG. 3(b). $r = 3$, $|\Delta| = 10$

There are many possibilities for the linear functional l other than the "counting functional" and l need not satisfy $lT_G = l$. We have said little about the field F, which might be taken to be the field of rational functions in a variable z. We might define $l(I_f) = z^{|f^{-1}(2)|}$, for example, which tells us how many times the symbol "2" appears in the image of f. It is easy to check (by writing $\hat{X} = \sum_f (\prod x_{tf(t)})\hat{I}_f$ in terms of the standard basis $\hat{I}_f$ and applying l) that $l(\hat{X}) = (x_{11} + x_{12}z + x_{13} + \cdots)$ $(x_{21} + x_{22}z + x_{23} + \cdots) \cdots (x_{d1} + x_{d2}z + x_{d3} + \cdots)$. Thus $lT_G = l$ in this case. Applying l to the right-hand side of (2.7) thus gives a polynomial $g(z) = g_0 + g_1 z + g_2 z^2 + g_3 z^3 + g_4 z^4$ where the coefficient g_t is the number of elements of Δ that have exactly t 2's in the image of f. Of course, in general the way to find the actual values of these coefficients is by applying l to the left-hand side of (2.7) (or to Fig. 2(a) or (b)). The rule for computing $l\tilde{\sigma}\hat{X}$ is now given, for any $\sigma \in G$, by

$$l\tilde{\sigma}\hat{X} = \prod_{a \in D\sigma} \left(\prod_{t \in a} x_{t1} + \prod_{t \in a} x_{t2} z + \prod_{t \in a} x_{t3} + \cdots \right).$$

As in the computation of (2.9) we have (setting $\rho = r - 2$),

$$l\tilde{e}(\hat{A} + 2\hat{B} + \hat{D}) = 1 + 2(\rho + z)^2 + (\rho + z)^4 = q_1(z),$$
$$2l\tilde{\tau}(\hat{A} + 2\hat{B} + \hat{D}) = 2(1 + (\rho + z^4)) = q_2(z),$$
$$l\tilde{\tau}^2(\tilde{A} + 2\tilde{B} + \hat{D}) = 1 + 2(\rho + z^2) + (\rho + z^2)^2 = q_3(z).$$

Computing

$$g(z) = \tfrac{1}{4}(q_1(z) + q_2(z) + q_3(z))$$

gives

$$(2.10) \quad g(z) = (1 + \rho + \tfrac{3}{4}\rho^2 + \tfrac{1}{4}\rho^4) + (\rho + \rho^3)z + (1 + \tfrac{1}{2}\rho + \tfrac{3}{2}\rho^2)z^2 + \rho z^3 + z^4.$$

The first coefficient checks with (2.9) (with r replaced by $(r - 1)$) and the remaining coefficients are easily checked from the definition of the problem in this rather simple case. The reader might be interested in constructing pre-lists for other problems of more complexity than the above illustration or in writing computer programs to do so. A $(T_G Q_G, T_G)$ composition of the list Δ' of all regular dodecagons with vertices labeled with symbols $\{1, \cdots, r\}$ such that any "1" symbol has another "1" symbol within a distance of two vertices is shown in Fig. 5. Fig. 4(a) shows an element of Δ'; Fig. 4(b) shows an element not in Δ'. The group G is taken to be the dihedral group of order 24 acting on $D = \{1, \cdots, 12\}$. Notice the appearance of negative coefficients. The "principle of inclusion-exclusion" was used to prepare this pre-list [5], [6], and this particular composition was computed by means of a general algorithm due to Mr. David Perlman of the University of California, San Diego Computer Center. Perlman's algorithm yields quite compact representations of various types of lists, but the question of optimality (with regard to the length of the compositions) is not at all clear. The list in Fig. 2(b) (in this case easily computed by hand computation) is the output of Perlman's algorithm for the example discussed above in connection with the $(T_G Q_G, T_G)$ composition shown in Fig. 2(a).

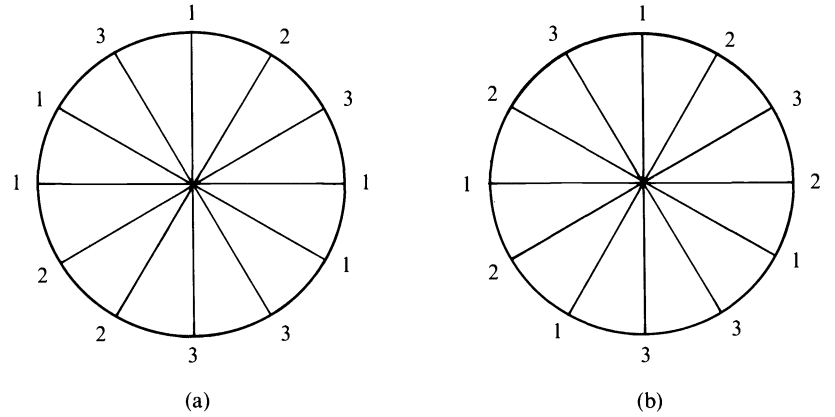


FIG. 4. *The structure (a) is in Δ'; (b) is not*

We have considered the above elementary example in detail to illustrate the two basic aspects of a "solution" to the isomorph rejection problem in the sense described above:

(i) construction of the pre-list or (T_0, T_1) composition of the desired answer for a pair of linear operators T_0 and T_1;

(ii) the extraction of information from the pre-list constructed in (1).

Solutions to (i) have not been studied in any systematic way although some results appear in [5] and [6]. Basically (i) represents a problem in computational combinatorics which amounts to the classical isomorph rejection problem when $T_0 = T_1 =$ identity. In many ways (i) is perhaps the most interesting and basic aspect of the problem, but we shall not deal with it here. We have indicated some results in the above example relating to (ii), the extraction of information given

$+1$	-12	$+12$	$+12$	$+12$	$+6$	-12	-24	-12	$+3$
s	q	q	q	q	q	q	q	q	q
s	q	q	q	q	q	q	q	q	q
s	p	p	p	p	p	p	p	p	p
s	q	q	q	q	q	q	q	q	q
s	q	q	q	q	q	q	q	q	q
s	s	s	s	s	s	s	q	q	p
s	s	s	s	s	q	q	q	p	q
s	s	s	s	q	q	q	p	q	q
s	s	s	q	q	p	p	q	q	p
s	s	q	q	p	q	q	q	q	q
s	s	q	p	q	q	q	q	p	q
s	s	p	q	q	s	p	p	q	p

FIG. 5. *A $(T_G Q_G, T_G)$ composition of Δ', the list of all dodecagons with restricted positions regarded as equivalent up to the action of the dihedral group.*

the pre-list. In the case of $(T_G Q_G, T_G)$ compositions the basic question is the evaluation of $l\tilde{\sigma}\hat{X}$ for $\hat{X} \in H_{d,r}$. Observe that if G acts on $L = S = R^D$ and Δ is a transversal for this action, then (i) is trivial. A $(T_G Q_G, T_G)$ composition (or (Q_G, T_G) composition) of Δ is $\hat{J}$, where J is the $d \times r$ matrix with every entry equal to 1. The question of the evaluation of $l\tilde{\sigma}\hat{X}$ or $lQ_G\hat{X}$ specializes to the evaluation of $lQ_G\hat{J}$ and, as one might by now guess, is closely related to Polyá-de Bruijn type formulas for enumeration under group actions. In the next section we shall consider formulas for the evaluation of $lQ_G\hat{X}$, $\hat{X} \in H_{d,r}$, and develop the relationship between our methods and those of de Bruijn [1], [2].

Explanation of Fig. 5. This table represents 10 functions $\alpha\hat{A} \in F^{(R^D)}$, $\alpha \in F$, where $D = \{1, \cdots, 12\}$, $R = \{1, \cdots, r\}$. Each column represents a $12 \times r$ matrix A. The numbers at the top are the coefficients α. As in Fig. 2, $q = (0 \quad 1 \quad 1 \cdots 1)$, $p = (1 \quad 0 \cdots 0)$, $s = (1 \cdots 1)$ are r-vectors representing the rows of the matrices A as indicated in the table.

3. The evaluation of $l\tilde{\sigma}\hat{X}$ and Pólya-de Bruijn-type counting theorems. As we saw in the previous section, the evaluation of $l\tilde{\sigma}\hat{X}$ was the key to the extraction of certain classes of information from $(T_G Q_G, T_G)$ compositions. In the previous section the group G acted on D and hence on R^D by $(\sigma f) = f \circ \sigma^{-1}$. We had in mind applications to labeled structures such as in the example of the square whose vertices ($D = \{1, 2, 3, 4\}$) were labeled with symbols from $R = \{1, 2, \cdots, r\}$. In the example we took $G = C_4$, the cyclic group of order 4, acting on D. We could just as well have taken $G = D_4$, the dihedral group of order 8, acting on D. Probably the most general group action of interest in this class of combinatorial computations is illustrated by the following example.

Let the vertices of a square be labeled with symbols λ, μ, ρ, where μ denotes an arrow pointing to the midpoint of the square, λ denotes an arrow pointing to the midpoint of the edge to the left of μ and ρ an arrow pointing to the midpoint of the edge to the right of μ. Figure 6 shows an example of such a "λ, μ, ρ-diagram" and the corresponding figure.

Consideration of the isometries of such a figure leads one to observe that D_4 acts on functions $f \in R^D$, where $D = \{1, 2, 3, 4\}$ and $R = \{\lambda, \mu, \rho\}$, but this action

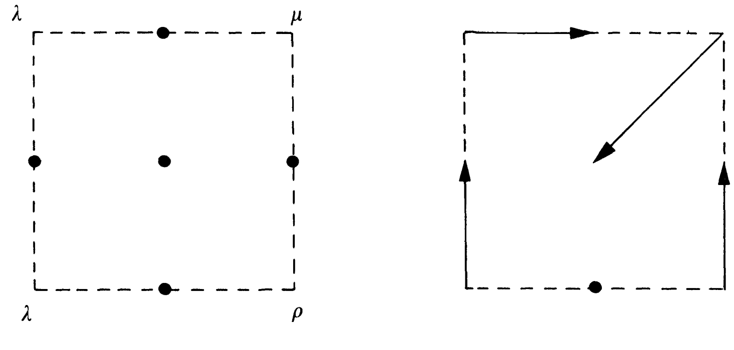


FIG. 6

occurs both on the domain and the range of the function set. The action on D is the standard action of the dihedral group D_4 generated by the permutations $\tau = (1\ \ 2\ \ 3\ \ 4)$ and $\sigma = (12)(34)$ (the vertices of the square being labeled clockwise from the upper left-hand corner). The action on R is given by $\{e, (\lambda\rho)\}$. All rotations of the figure act as the identity on R and reflections transpose the symbols λ and ρ.

Actions of the above type can be described as follows: Let a group G be given and let G act on D and on R. (Formally (see [2]) we are given homomorphisms $\mu: G \to$ symmetric group on D, $\nu: G \to$ symmetric group on R.) Define the action of G on $D \times R$ by $\sigma(i, j) = (\sigma i, \sigma j)$. Given $f \in R^D$ we observe that this action is such that $\sigma(\text{graph} f) = \text{graph } g$ for some $g \in R^D$ ($\sigma(S)$ for $S \subset D \times R$ is defined to be the set of all $(\sigma i, \sigma j)$ where $(i, j) \in S$). In particular, $(\sigma i, \sigma j) \in \text{graph } g \Leftrightarrow \sigma j = g\sigma(i) \Leftrightarrow j = \sigma^{-1}g\sigma(i)$. If $(i, j) \in \text{graph} f$ this implies that $j = f(i)$ so $g = \sigma f \sigma^{-1}$, where juxtaposition of symbols denotes function composition.

In the above example, if $f = \begin{pmatrix} 1 & 2 & 3 & 4 \\ \lambda & \mu & \rho & \lambda \end{pmatrix}$ and if $\sigma = (1\ \ 2)(3\ \ 4)$ on D and $(\lambda\rho)$ on R, then $\sigma f \sigma^{-1} = g = \begin{pmatrix} 1 & 2 & 3 & 4 \\ \mu & \rho & \rho & \lambda \end{pmatrix}$. Observe that in the algebra $F^{(R^D)}$, σ induces a linear operator $\bar{\sigma}$ defined on the standard basis by $\bar{\sigma}(I_f) = I_g$. As in the previous section we write I_f and I_g as $\hat{X}$ and $\hat{Y}$, where X and Y are 4×3 matrices (replace λ by "1", μ by "2", and ρ by "3" if one wishes to simplify indexing of the matrices). In this way we write $I_f = \hat{X}$ and $I_g = \hat{Y}$, where

$$X = \begin{array}{c|ccc|} & \lambda & \mu & \rho \\ 1 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \\ 4 & 1 & 0 & 0 \end{array} \quad \text{and} \quad Y = \begin{array}{c|ccc|} & \lambda & \mu & \rho \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 3 & 0 & 0 & 1 \\ 4 & 1 & 0 & 0 \end{array}.$$

Observe that $X(i, j) = Y(\sigma i, \sigma j)$ where for any matrix A, $A(i, j)$ denotes the i, jth entry. This statement corresponds to the fact that $\text{graph } g = \sigma(\text{graph} f)$. In general given $X = (x_{ij})$, a $d \times r$ matrix, then $\hat{X} = \sum_{f \in R^D} (\prod_{t=1}^{d} x_{tf(t)}) I_f$ (take $D = \{1, \cdots, d\}$, $R = \{1, \cdots, r\}$). Using this rule and the definition of $\bar{\sigma}$ given above one can check that in general $\bar{\sigma}\hat{X} = \hat{Y}$ where, as above, $X(i, j) = Y(\sigma i, \sigma j)$ or, equivalently, $Y(i,j) = X(\sigma^{-1}i, \sigma^{-1}j)$ for all (i, j). Notice that if G acts as the identity on R, then this rule reduces to the rule for computing $\bar{\sigma}\hat{X}$ given in the previous section. We define the linear operator $\tilde{\sigma}$ on $F^{(R^D)}$ as in § 2: $\tilde{\sigma} I_f = I_f \chi(\sigma f \sigma^{-1} = f)$. $\chi(\textit{statement}) = 1$ if *statement* = true, 0 if *statement* = false. The linear operators

$$T_G = \frac{1}{|G|} \sum_{\sigma \in G} \bar{\sigma} \quad \text{and} \quad Q_G = \frac{1}{|G|} \sum_{\sigma \in G} \tilde{\sigma}$$

are defined as in § 2 ((2.1) and (2.2)). If a sequence of elements $(\hat{A}_1, \cdots, \hat{A}_q)$ is a

$(T_G Q_G, T_G)$ composition of $N \subset S = R^D$ as in Definition 2.8, and l is the functional $l(I_f) = 1$ for all $f \in R^D$, then to evaluate the cardinality $|N|$ we shall describe the evaluation of $lQ_G(\hat{A}_1 + \cdots + \hat{A}_q)$. That is, we evaluate $l\tilde{\sigma}\hat{A}_t$ for $t = 1, \cdots, q, \sigma \in G$. (As in § 2, $lT_G = l$, and hence $lT_G Q_G(\hat{A}_1 + \cdots + \hat{A}_q) = lQ_G I_N$ becomes $lQ_G\hat{A}_1 + \cdots + lQ_G\hat{A}_q = |N|$.)

We introduce some notation. Given $\sigma \in G$, let $D\sigma$ and $R\sigma$ denote the partitions of D and R induced by the cycles of σ in D and R respectively. If $\sigma = (12)(34)$ as in the above example then $D\sigma = \{\{1,2\},\{3,4\}\}$ and $R\sigma = \{\{u\},\{\lambda,\rho\}\} \equiv \{\{2\},\{1,3\}\}$. The construction of the set $R\sigma$ follows from the fact that σ is a reflection and thus acts on R as the transposition $(\lambda\ \ \rho)$. For $X \in M_{d,r}(F)$ let $X[a|b]$ denote the submatrix of X obtained by keeping the rows of X with row indices in a and the columns of X with column indices in b. We define the circulant of $X[a|b]$ with respect to σ to be

$$\text{cir}\, X[a|b] = \sum_{j=0}^{|b|-1} \prod_{k=0}^{|a|-1} X(\sigma^k i_a, \sigma^{k+j} i_b) \quad \text{if} \quad |b|\,\big|\,|a| \tag{3.1}$$

and cir $X[a|b] = 0$ otherwise, where $i_a \in a$, $i_b \in b$. This computation is represented by Fig. 7, in the case $|a| = 6$, $|b| = 3$.

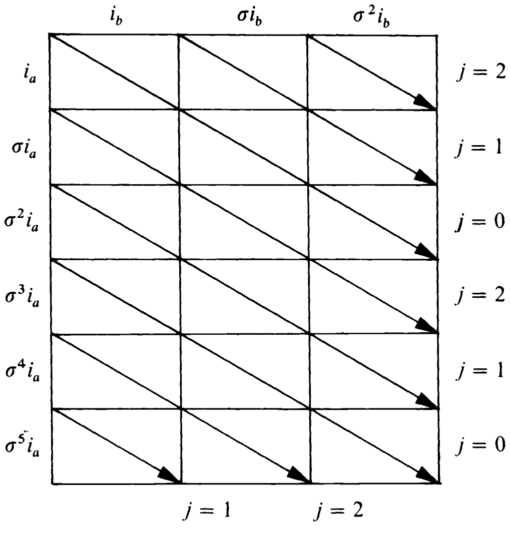


FIG. 7. *The computation of* cir $X[a|b]$

Explanation of Fig. 7. The (s, t)th box of this 6×3 array contains the $(\sigma^s i_a\, \sigma^t i_b)$th entry of X. For $j = 0, 1, 2$ take the product of the entries in the boxes (6 terms, one entry per box) along the appropriate diagonal corresponding to each j. Sum these products over all j. Here $s \in \{0, \cdots, 5\}$, $t \in \{0, 1, 2\}$.

The reader should note that if $d = r$ and $\sigma = (1, 2, \cdots, r)$ on both D and R (i.e., σ is the full cycle) then cir $X[a|b]$ is the standard circulant of the $d \times d$ matrix X. We now state the basic result.

THEOREM 3.2. *Let l denote the trivial functional on $F^{(R^D)}$ defined by $l(I_f) = 1$ for all f and let $\sigma \in G$. Using the notations defined above we have*

$$l\tilde{\sigma}\hat{X} = \prod_{a \in D\sigma} \left(\sum_{b \in R\sigma} \operatorname{cir} X[a|b] \right).$$

Before proving Theorem 3.2 we make some observations. The result of Theorem 3.2 gives immediately the rule for computing $lQ_G\hat{X} = (1/|G|)\sum_{\sigma \in G} l\tilde{\sigma}\hat{X}$. Observe that for any $Y = (y_{ij}) \in M_{d,r}(F)$ the functional $l' = l\hat{Y}$ defined by composition of the trivial functional l with the endomorphism of left multiplication by $\hat{Y}$ (i.e., $l\hat{Y}(I_f) = l(\hat{Y}I_f) = \prod_{t=1}^{d} y_{tf(t)}$) can be evaluated at $\tilde{\sigma}\hat{X}$ by use of Theorem 3.2. One need only show (we leave this to the reader) that $\tilde{\sigma}$ and $\hat{Y}$ commute for any such Y. Thus if $l' = l\hat{Y}$, we have the following corollary.

COROLLARY 3.3.

$$l'\tilde{\sigma}\hat{X} = \prod_{a \in D\sigma} \left(\sum_{b \in R\sigma} \operatorname{cir} Z[a|b] \right),$$

where $Z = Y \cdot X$ is the Schur product of Y and X.

As an example of Corollary 3.3 consider $l' = l\hat{Y}$, where

$$Y = \begin{bmatrix} w(1) & w(2) & \cdots & w(r) \\ w(1) & w(2) & \cdots & w(r) \\ \vdots & & & \\ w(1) & w(2) & \cdots & w(r) \end{bmatrix}.$$

Here $w: R \to F$ may be taken to be any function and all rows of Y are identical. Note that if Δ is a transversal (s.d.r.) for the orbits of G acting on R^D then (from the definition of T_G)

$$T_G\left(\sum_{f \in \Delta} \frac{|G|}{|G_f|} I_f \right) = \sum_{f \in R^D} I_f.$$

But $Q_G(I_f) = (|G_f|/|G|)I_f$, and hence,

$$(3.4) \qquad Q_G\hat{J} = T_G\left(\sum_{f \in \Delta} I_f \right).$$

Here $\sum_{f \in R^D} I_f = \hat{J}$, where J is the $d \times r$ matrix with every entry 1. This $\hat{J}$ is a (Q_G, T_G) composition of Δ of length 1. Thus in this case the construction of a (Q_G, T_G) or, equivalently, a (T_GQ_G, T_G) composition is trivial.

Assume that G acts as the identity on the range R and hence for each σ, $R\sigma$ is the discrete partition. In this case,

$$\prod_{a \in D\sigma} \left(\sum_{b \in R\sigma} \operatorname{cir} Y[a|b] \right) = \left(\sum_{i=1}^{r} w(i) \right)^{c(\sigma,1)} \left(\sum_{i=1}^{r} w^2(i) \right)^{c(\sigma,2)} \cdots \left(\sum_{i=1}^{r} w^d(i) \right)^{c(\sigma,d)},$$

where $c(\sigma, k)$ is the number of cycles of σ of length k. Note that since all rows of Y are the same, $l'T_G = l'$ and thus applying l' to both sides of (3.4) using (3.3) we obtain, writing $W(f) = \prod_{t=1}^{d} w(f(t))$,

$$P_G\left(\sum_{i=1}^{r} w(i), \cdots, \sum_{i=1}^{r} w^d(i)\right) = \sum_{f \in \Delta} W(f), \tag{3.5}$$

where

$$P_G(x_1, \cdots, x_d) = \frac{1}{|G|} \sum_{\sigma \in G} x_1^{c(\sigma,1)} \cdots x_d^{c(\sigma,d)}$$

is the standard cycle index polynomial. The identity (3.5) is the classical Pólya identity [1], [2].

As another example of identities which may be derived by evaluating functionals on (3.4) let us assume again that G acts on both D and R and use the trivial functional l. Thus l of the right-hand side becomes $|\Delta|$. Note that in this case, $X = J$, we have simply cir $J[a|b] = |b|\chi(|b|\,|\,|a|)$, from (3.1). Let $\delta(\sigma, k)$ and $\rho(\sigma, k)$ denote the number of cycles of length k of σ in D and R respectively. Thus

$$\prod_{a \in D\sigma} \left(\sum_{b \in R\sigma} \chi(|b|\,|\,|a|)|b| \right) = \prod_{k=1}^{d} \left(\sum_{j|k} j\rho(\sigma, j) \right)^{\delta(\sigma,k)}. \tag{3.6}$$

But for any integers p, q we have $(\partial/\partial z)_0^q e^{pz} = p^q$, where $(\partial/\partial z)_0^q$ denotes the qth derivative evaluated at $z = 0$. Using this observation we write

$$\prod_{k=1}^{d} \left(\sum_{j|k} j\rho(\sigma, j) \right)^{\delta(\sigma,k)} = \prod_{k=1}^{d} \left(\frac{\partial}{\partial z_k} \right)_0^{\delta(\sigma,k)} \prod_{k=1}^{d} \exp\left[\left(\sum_{j|k} j\rho(\sigma, j) \right) z_k \right].$$

But

$$\exp\left[\sum_k \sum_{j|k} j\rho(\sigma, j) z_k \right] = \exp\left[\sum_j \sum_t j\rho(\sigma, j) z_{jt} \right] = \prod_{j=1}^{r} \left[\exp\left(j \sum_t z_{tj} \right) \right]^{\rho(\sigma,j)}.$$

Setting $\xi_j = \exp(j \sum_t z_{tj})$ we obtain

$$|\Delta| = \frac{1}{|G|} \sum_{\sigma \in G} \prod_{k=1}^{d} \left(\frac{\partial}{\partial z_k} \right)_0^{\delta(\sigma,k)} \prod_{j=1}^{r} \xi_t^{\rho(\sigma,j)}. \tag{3.7}$$

This is one of a number of results of this form, expressed in terms of differential operators, due to de Bruijn [2]. The most well-known identity of this type is obtained in the special case where G is replaced by a direct product $G = K \times H$, where $K:D$, $H:R$. The action is defined by $(\sigma, \varphi)f = \varphi \circ f \circ \sigma^{-1}$ as above. But in this case the sum over $\sigma \in G$ occurring in (3.7) may be replaced by the double sum over $\sigma \in K$, $\varphi \in H$. Hence we have

$$|\Delta| = P_K\left(\frac{\partial}{\partial z_1}, \cdots, \frac{\partial}{\partial z_d} \right)_0 P_H(\xi_1, \cdots, \xi_d) \tag{3.8}$$

(see [1], [2]).

The point of these derivations from the circulant formulation given in Theorem 3.2 and Corollary 3.3 is not to give still another proof of these identities but to point out the relationship between these identities and the more general problems described in § 2. Most of the computational advantages of these identities (such as (3.6), (3.7) and (3.8)) are already contained in the identities in Theorem 3.2 and Corollary 3.3. There seems to be little need in most computations to bring in differential operators at all. Identity (3.8) does have a conceptual advantage in some situations (see [1]) in that it shows quite clearly how the computation relates to the cycle index polynomials of the component subgroups. One should recall that (3.6), (3.7) and (3.8) were derived from the application of Theorem 3.2 and Corollary 3.3 to the trivial list R^D of *all* functions (hence $X = J$). The same techniques can be applied to any $(T_G Q_G, T_G)$ composition $(\hat{A}_1, \cdots, \hat{A}_q)$ of a list N. Whether or not the evaluation of appropriate functionals can be formulated in terms of (3.6), (3.7) and (3.8), one can always use Theorem 3.2 or Corollary 3.3. Analogous derivations of various other extensions of Pólya type theorems [2] are interesting and may be developed in a manner parallel to the above discussion. We shall not consider these extensions here. Rather, we shall conclude below with some elementary remarks about the algebra $F^{(R^D)}$ and the proof of Theorem 3.2.

4. The algebra $F^{(R^D)}$. In § 2 we pointed out how the algebra $F^{(R^D)}$ is the natural setting for the class of combinatorial problems discussed above. Assume, for simplicity of notation, that $D = \{1, \cdots, d\}$ and $R = \{1, \cdots, r\}$. A function $\varphi \in F^{(R^D)}$ is a "rule" which assigns to each $f = \begin{pmatrix} 1 & 2 & \cdots & d \\ i_1 & i_2 & \cdots & i_d \end{pmatrix} \in R^D$ an element $\varphi(f) = T_f \equiv T_{i_1 i_2 \cdots i_d}$ of the field F. Physicists will recognize that φ is a tensor of rank d, or $F^{(R^D)}$ is the space of (covariant) tensors of rank d. The space $F^{(R^D)}$ is simply the canonical finite-dimensional vector space regarded trivially as an algebra by componentwise multiplication. The special structure of the domain of φ as a set of functions, R^D, relates to $H_{d,r}$, the homogeneous functions of rank d. We now remark briefly on the algebraic notion of a tensor product of vector spaces and how it relates to the material of the previous sections. Let V be a vector space, dim $V = r$. Choose a basis $\alpha = \{e_1, \cdots, e_r\}$ for V. For $(v_1, \cdots, v_d) \in \times^d V$, define $\psi_\alpha(v_1, \cdots, v_d) = \hat{A}$, where A is the $d \times r$ matrix with ith row $A_{(i)} = (a_{i1}, \cdots, a_{ir})$, $i = 1, \cdots, d$, defined by the relation $v_i = a_{i1}e_1 + \cdots + a_{ir}e_r$.

$$\begin{tikzcd} \overset{d}{\times} V \arrow[r, "\psi"] \arrow[dr, "\mu"'] & F^{(R^D)} \arrow[d, "\mu_0"] \\ & W \end{tikzcd}$$

FIG. 8

We observe that $(\psi_\alpha, F^{(R^D)})$ is a dth tensor product of V [3]. That is, we claim that:

(i) The space spanned by Im ψ_α is $F^{(R^D)}$;

(ii) For any vector space W over F and any multilinear μ there is a linear μ_0 such that the diagram of Fig. 8 commutes.

Note that Im $\psi_\alpha = H_{d,r}$ as defined in § 2. Since $H_{d,r}$ contains the standard basis $\{I_f : f \in R^D\}$, (i) above is immediate. Define μ_0 on this basis by $\mu_0(I_f) = \mu(e_{f(1)}, \cdots,$

$e_{f(d)}$) and extend μ_0 linearly to $F^{(R^D)}$. Let A and $v_1, \cdots, v_d$ be as above. Then

$$\mu_0(\hat{A}) = \mu_0\left(\sum_f \prod_t a_{tf(t)} I_f\right) = \sum_f \prod_t a_{tf(t)}\mu(e_{f(1)}, \cdots, e_{f(d)}) = \mu(v_1, \cdots, v_d).$$

Thus $\mu_0\psi_\alpha = \mu$, and the above diagram commutes. One should note that ψ_α is multilinear. Of course, the map ψ_α is not "canonical" or basis independent, but this is exactly the useful feature in the combinatorial computations described above. Using conventional notations we would write $\hat{A} = v_1 \otimes \cdots \otimes v_d$. Thus the functions $\hat{A}$ of $H_{d,r}$ are the homogeneous tensors and may be regarded, if one so desires, as the tensor product of their row vectors. The product rule (Schur product) $\hat{A}\hat{B} = \hat{C}$, where $C = A \cdot B$ is the rule $(v_1 \otimes \cdots \otimes v_d)(v'_1 \otimes \cdots \otimes v'_d) = (v_1v'_1) \otimes \cdots \otimes (v_dv'_d)$, where $v_tv'_t$ denotes componentwise multiplication. Certain other aspects of the above material will be transparent from this algebraic point of view (and equally transparent from any other point of view!).

We should also remark that many of the standard operators of finite-dimensional multilinear algebra (derivations, contractions, Kronecker powers, induced transformations) seem to relate in various ways to the combinatorial problems mentioned above. We have not made any systematic study of such questions.

5. Proof of Theorem 3.2. We wish to show that $l\tilde{\sigma}\hat{X} = \prod_{a\in D}(\sum_{b\in R\sigma} \text{cir}\, X[a|b])$. From the definition of $\tilde{\sigma}$ we have that $\tilde{\sigma}\hat{X} = \sum_{f\in R^D}(\prod_{(i,t)\in \text{graph} f} X(i, t))\chi(\sigma f\sigma^{-1} = f)I_f$. Let $\langle\sigma\rangle \subset G$ denote the cyclic group generated by σ. The condition $\sigma f\sigma^{-1} = f$ is equivalent to the condition $\sigma(\text{graph} f) = \text{graph} f$. That is, $\langle\sigma\rangle$ acts on graph f. Let $\{i_a : a \in D\sigma\}$ denote a fixed transversal for the orbit partition of $\langle\sigma\rangle$ on D. Similarly define $\{i_b : b \in R\sigma\}$. Clearly, if $\sigma f\sigma^{-1} = f$, then $\{(i_a, f(i_a)) : a \in D\sigma\}$ is a transversal for the action of $\langle\sigma\rangle$ on graph f. Thus we may write

$$\prod_{(i,j)\in \text{graph} f} X(i,j) = \prod_{a\in D\sigma}\prod_{k=0}^{|a|-1} X(\sigma^k i_a, \sigma^k f(i_a)).$$

Note that $\sigma^{|a|}(i_a, f(i_a)) = (i_a, \sigma^{|a|}f(i_a))$ and thus $\sigma^{|a|}f(i_a) = f(i_a)$. This implies that if $f(i_a)$ is in the cycle b of $R\sigma$, then $|b|\,|\,|a|$. Conversely, we observe that if to each i_a we associate some t_a, where t_a is in a cycle b of $R\sigma$, $|b|\,|\,|a|$, then the set

$$Q = \{(\sigma^k i_a, \sigma^k t_a) : 0 \leqq k < |a|, a \in D\sigma\}$$

is clearly the graph of a function f such that $\sigma(\text{graph} f) = \text{graph} f$. For each $a \in D\sigma$ define

$$U_a = \{t : t \text{ is in some } b \in R\sigma, |b|\,|\,|a|\}.$$

Let $Z_{a,t} = \prod_{k=0}^{|a|-1} X(\sigma^k i_a, \sigma^k t)$ be defined for $a \in D\sigma$, $t \in U_a$. By the above observations we have, setting $W = \times_{a\in D\sigma} U_a$,

$$l\tilde{\sigma}\hat{X} = \sum_{\varphi\in W}\prod_{a\in D\sigma} Z_{a,\varphi(a)}.$$

But, interchanging sum and product (this interchange is where much of the computational utility is picked up) we obtain

$$l\tilde{\sigma}\hat{X} = \prod_{a \in D\sigma} \left(\sum_{t \in U_a} Z_{a,t} \right).$$

Note that

$$\sum_{t \in U_a} Z_{a,t} = \sum_{b \in R\sigma} \left(\sum_{j=0}^{|b|-1} Z_{a,\sigma^j i_b} \right) \chi(|b| \,|\, |a|) = \sum_{b \in R\sigma} \operatorname{cir} X[a|b].$$

This completes the proof.

Acknowledgment. The author wishes to thank Mr. Chris Parrish and Mr. Dennis White for many helpful suggestions in the preparation of this manuscript. The examples given of weak solutions to isomorph rejection problems and other examples are generated by a very interesting algorithm and computer program developed by Mr. David Perlman at the University of California, San Diego, Computer Center, in his doctoral thesis.

REFERENCES

[1] N. G. DE BRUIJN, *Pólya's theory of counting*, Applied Combinatorial Mathematics, E. F. Beckenbach, ed., John Wiley, New York, 1964, pp. 144–184.

[2] ———, *A survey of generalizations of Pólya's enumeration theorem*, Nieuw Arch. Wisk. (2), 19 (1971), pp. 89–112.

[3] W. H. GREUB, *Multilinear Algebra*, Springer-Verlag, New York, 1967.

[4] M. B. WELLS, *Elements of Combinatorial Computing*, Pergamon Press, New York, 1972.

[5] S. G. WILLIAMSON, *The combinatorial analysis of patterns and the principle of inclusion–exclusion*, Discrete Math., 1 (1972), pp. 357–388.

[6] ———, *Tensor compositions of combinatorial structures*, Linear and Multilinear Algebra, to appear.

SIAM J. COMPUT.
Vol. 2, No. 1, March 1973

ON THE NUMBER OF NONSCALAR MULTIPLICATIONS NECESSARY TO EVALUATE POLYNOMIALS*

MICHAEL S. PATERSON† AND LARRY J. STOCKMEYER‡

Abstract. We present algorithms which use only $O(\sqrt{n})$ nonscalar multiplications (i.e. multiplications involving "x" on both sides) to evaluate polynomials of degree n, and proofs that at least $\sqrt{n}$ are required. These results have practical application in the evaluation of matrix polynomials with scalar coefficients, since the "matrix $\times$ matrix" multiplications are relatively expensive, and also in determining how many multiplications are needed for polynomials with rational coefficients, since multiplications by integers can in principle be replaced by several additions.

Key words. Polynomial evaluation, nonscalar multiplications, rational coefficients, matrix polynomial.

1. Introduction. A well-known result given by Motzkin [2] and Winograd [6] is that, even with preliminary adaptation of the coefficients, at least $n/2$ multiplications are required to evaluate a polynomial of degree n if the coefficients of the polynomial are algebraically independent. However we frequently wish to evaluate polynomials with rational or integer coefficients for which this result does not apply, and so we are led to investigate the number of multiplications required to evaluate rational polynomials. Our main theorem is that $\sqrt{n}$ are necessary, and we present algorithms to demonstrate that $O(\sqrt{n})$ are sufficient.

Apart from providing a satisfactory answer to a theoretical problem our results have some practical applications. Because multiplication by an integer can be replaced by repeated additions, the only multiplications which are counted in the above results are those where the indeterminate of the polynomial appears in both multiplicands, that is the *nonscalar* multiplications. However in some other applications, such as the evaluation of matrix polynomials with scalar coefficients, we are again concerned to minimize the number of nonscalar multiplications because these may be much more expensive than additions and subtractions, or multiplication by a scalar. For practical purposes therefore our most important contributions are the algorithms in § 3, which use only $O(\sqrt{n})$ nonscalar multiplications.

We define an *algorithm* α *over a scalar field* S as $\alpha = \alpha(1), \alpha(2), \cdots, \alpha(k)$, where

(i) $\alpha(1) \in \mathrm{S} \cup \{x\}$ and

(ii) either $\alpha(r) \in \mathrm{S} \cup \{x\}$ or else $\alpha(r) = (\circledcirc, i, j)$, where $1 \leqq i, j < r$, and $\circledcirc \in \{+, -, \times, \div\}$ for $1 < r \leqq k$.

We say that $\alpha(r)$ defines a *nonscalar multiplication/division* if

$\alpha(r) = (\times, i, j)$ and neither $\alpha(i) \in \mathrm{S}$ nor $\alpha(j) \in \mathrm{S}$, or

$\alpha(r) = (\div, i, j)$ and $\alpha(j) \notin \mathrm{S}$.

* Received by the editors December 3, 1971, and in revised form January 23, 1973. This work was done, in part, at Project MAC, an M.I.T. research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract N000-70-A-0362-001.

† School of Computer Science, University of Warwick, Coventry, Warwickshire, England.

‡ Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

We define the associated elements $\lambda_1, \cdots, \lambda_k \in \mathrm{S}(x)$ as

$$\lambda_r = \begin{cases} \alpha(r) & \text{if } \alpha(r) \in \mathrm{S} \cup \{x\} \\ \lambda_i \circledcirc \lambda_j & \text{if } \alpha(r) = (\circledcirc, i, j) \end{cases} \quad \text{for } 1 \leqq r \leqq k.$$

We say that α computes the polynomial $p(x) \in \mathrm{S}[x]$ if $\lambda_r = p(x)$ for some r, $1 \leqq r \leqq k$.

2. Lower bounds. If a polynomial $p(x)$ can be computed with k nonscalar multiplications/divisions, then the algorithm can be expressed by the following scheme $\mathfrak{A}_k$, where the m_{ij}, $\tilde{m}_{ij}$ are in S and $\circledcirc_r$ is $\times$ or $\div$.

$\underline{\mathfrak{A}_k}$.

$$\mu_{-1} = 1, \qquad \mu_0 = x.$$

For $r = 1, \cdots, k$,

$$\mu_r = \left(\sum_{i=-1}^{r-1} m_{r,i}\mu_i\right) \circledcirc_r \left(\sum_{i=-1}^{r-1} \tilde{m}_{r,i}\mu_i\right).$$

Finally,

$$p(x) = \sum_{i=-1}^{k} m_{0,i}\mu_i.$$

The m_{ij}, $\tilde{m}_{ij}$ will be called the *parameters* of the algorithm. It is useful to think of $\mathfrak{A}_k$ as a parameterization mapping:

$$\mathfrak{A}_k : \mathrm{S}^{M(k)} \to \{p(x) \in \mathrm{S}(x) | p(x) \text{ can be computed in } \leqq k$$
$$\text{nonscalar multiplication/divisions}\},$$

where $M(k)$ = the number of parameters in $\mathfrak{A}_k$.

For the time being we shall consider algorithms without divisions.

THEOREM 1. *For any $n > 2$, there are rational polynomials of degree n which require $\sqrt{n}$ nonscalar multiplications for their evaluation by any algorithm over* **R** *without divisions.*

Proof. Since we are considering algorithms without divisions and since $(m + \Sigma) \times (\tilde{m} + \tilde{\Sigma}) = \Sigma \times \tilde{\Sigma} + m \cdot \tilde{\Sigma} + \tilde{m} \cdot \Sigma + m \cdot \tilde{m}$ we can take $m_{r,-1} = \tilde{m}_{r,-1} = 0$ for $r = 1, \cdots, k$. The constant terms are thus removed from the multiplicands and replaced subsequently by scalar multiplications and additions. A further reduction of $\mathfrak{A}_k$ is easily seen. By suitable scalar multiplication we can arrange that in any multiplicand the first nonzero parameter is 1, and of course we can assume without loss of generality that there is some nonzero parameter.

After these reductions,

$$\mu_1 = x \times x$$

and

$$\mu_2 = (ax^2 + bx) \times (cx^2 + dx).$$

The reader may readily verify that since $\mu_1 = x^2$ has already been computed, μ_2 may as well be in the form $\mu_2 = (m_{21}\mu_1 + m_{20}\mu_0) \times \mu_1$ where $m_{21} = 1$, or $m_{21} = 0$

and $m_{20} = 1$. Counting up the number of parameters in such a reduced form of $\mathfrak{A}_k$ we get, for $k \geqq 2$,

$$\begin{aligned} M(k) &\leqq 1 + (4 + 6 + \cdots + 2(k-1)) + k + 2 \\ &= k^2 + 1. \end{aligned}$$

Suppose $p(x) = q_n x^n + \cdots + q_1 x + q_0$, $q_i \in \mathbf{Q}$, can be evaluated in k nonscalar multiplications by some algorithm $\mathfrak{A}_k^*$, being a reduced form of $\mathfrak{A}_k$. Carry out the operations specified by the algorithm $\mathfrak{A}_k^*$ formally, and view the result as a polynomial in x whose coefficients are integer polynomials in the parameters, that is,

$$p(x) = a_n(\vec{m})x^n + \cdots + a_1(\vec{m})x + a_0(\vec{m}),$$

where $\vec{m}$ denotes the vector of parameters of length $M(k) = k^2 + 1$ and $a_i \in \mathbf{Z}[\vec{m}]$, $i = 0, \cdots, n$. Regarding the $\vec{m}$ as indeterminates, the a_i are in $\mathbf{Q}(\vec{m})$ which has degree of transcendence $M(k)$ over $\mathbf{Q}$. Now suppose $M(k) < n + 1$; then the set $\{a_i | i = 0, \cdots, n\}$ must be algebraically dependent and so satisfy a nontrivial polynomial relation. For further discussion of this see, for example, [5, § 64]. That is, for some nontrivial integer polynomial P^*,

$$P^*(a_n(\vec{m}), \cdots, a_0(\vec{m})) \equiv 0.$$

There are $\frac{1}{3}(k+1)!(k-1)!$ reduced forms $\mathfrak{A}_k^*$ of $\mathfrak{A}_k$. Let $P(a_n, \cdots, a_0) \equiv \prod_* P^*(a_n, \cdots, a_0)$. If all rational polynomials of degree n were computable by the algorithm scheme $\mathfrak{A}_k$, we should have

$$P(\mathbf{Q}^{n+1}) \equiv 0.$$

However, since P is continuous and the rationals are dense this would imply that $P \equiv 0$, contrary to assumption. Therefore,

$$k^2 + 1 \geqq n + 1 \quad \text{or} \quad k \geqq \sqrt{n}.$$

This completes the proof.

Note that we have actually shown that $\{(q_n, \cdots, q_0) \in \mathbf{Q}^{n+1} | q_n x^n + \cdots + q_1 x + q_0$ requires $\sqrt{n}$ nonscalar multiplications$\}$ is a dense subset of $\mathbf{R}^{n+1}$.

The problem of evaluating integer polynomials is related to the problem of evaluating rational polynomials because if $p(x) \in \mathbf{Q}[x]$ can be evaluated in $\leqq k$ nonscalar multiplications by an algorithm over $\mathbf{Q}$ then, for some $z_0 \in \mathbf{Z}$, $z_0 \cdot p(x) \in \mathbf{Z}[x]$ can be evaluated in $\leqq k$ nonscalar multiplications by an algorithm over $\mathbf{Z}$. Hence Theorem 2 (with $\sqrt{n} - 1$ in place of $\sqrt{n} - \frac{1}{2}$) may be deduced as a corollary of Theorem 1, though we know of no useful implication in the opposite direction. We give here a different, combinatorial, proof of Theorem 2.

THEOREM 2. *For any $n > 1$, there are integer polynomials of degree n which require at least $\sqrt{n} - \frac{1}{2}$ nonscalar multiplications for their evaluation by any algorithm over $\mathbf{Z}$ without divisions.*

Proof. Consider the finite ring $\mathbf{F} = \{0, 1\}$ and the ring homomorphism $H: \mathbf{Z} \to \mathbf{F}$ given by

$$H(z) = \begin{cases} 1 & \text{if } z \text{ odd}, \\ 0 & \text{if } z \text{ even}. \end{cases}$$

If $z_nx^n + \cdots + z_1x + z_0 \in \mathbf{Z}[x]$ can be evaluated by an algorithm over $\mathbf{Z}$ using k nonscalar multiplications, then certainly $w_nx^n + \cdots + w_1x + w_0 \in \mathbf{F}[x]$, where $w_i = H(z_i)$ for $i = 0, \cdots, n$, can be evaluated by an algorithm over $\mathbf{F}$ using k nonscalar multiplications.

As in the proof of Theorem 1, we can assume that algorithms over $\mathbf{F}$ without divisions can be expressed as a certain reduced form of $\mathfrak{A}_k$. Again, we can assume that no constant terms appear in the multiplicands so that the rth multiplication has the form

$$\mu_r = \Sigma \times \tilde{\Sigma},$$

where $\Sigma = \sum_{i=0}^{r-1} m_{r,i}\mu_i$ and $\tilde{\Sigma} = \sum_{i=0}^{r-1} \tilde{m}_{r,i}\mu_i$. Since $\Sigma \times \tilde{\Sigma} = \tilde{\Sigma} \times \Sigma$, and since neither Σ nor $\tilde{\Sigma}$ should be identically zero, there are at most

$$\tfrac{1}{2}(2^r - 1)2^r < 2^{2r-1}$$

effectively distinct and useful choices for the rth multiplication for $r \geqq 2$. Therefore the number of different polynomials in $\mathbf{F}[x]$ computable by algorithms over $\mathbf{F}$ with k nonscalar multiplications and no divisions is less than

$$\left(\prod_{r=2}^{k} 2^{2r-1}\right)2^{k+2} = 2^{k^2+k+1}.$$

(More careful counting would yield 2^{k^2+k-2} if desired.) Since there are 2^{n+1} polynomials in $\mathbf{F}[x]$ of degree n or less, if $k^2 + k \leqq n$ then some of these cannot be computed using only k multiplications. The result follows.

For algorithms which use divisions as well we obtain a result similar to Theorem 1.

THEOREM 3. *For any n, there are rational polynomials of degree n which require $\sqrt{n} - 2$ nonscalar multiplications/divisions for their evaluation by an algorithm over* $\mathbf{R}$.

Proof. Consider each of the 2^k algorithmic forms we obtain from $\mathfrak{A}_k$ by choosing a multiplication or division at each nonscalar step. Each introduces at most $k^2 + 4k + 2$ parameters. If we suppose that

$$k^2 + 4k + 2 < n + 1$$

then there is a nontrivial polynomial $P_i \in \mathbf{Q}[x_{n+1}]$ for each form, such that if $q_nx^n + \cdots + q_0 \in \mathbf{Q}[x]$ is computed by the ith algorithmic form, then $P_i(q_n, \cdots, q_0) = 0$. If

$$P(q_n, \cdots, q_0) = \prod_{i=1}^{2^k} P_i(q_n, \cdots, q_0)$$

and all nth degree rational polynomials can be evaluated using k nonscalar multiplications/divisions, we have $P(\mathbf{Q}^{n+1}) \equiv 0$. But $P \not\equiv 0$ and P is continuous which gives a contradiction, proving the result.

3. Algorithms.

3.1. A fast algorithm which uses rational preprocessing. Before giving algorithms which use $O(\sqrt{n})$ nonscalar multiplications, we present an algorithm which will be used later but which is also interesting in its own right. The algorithm is designed to work for real polynomials with algebraically independent coefficients and will therefore use at least $n/2$ multiplications.

Motzkin [2], Eve [1] and Pan [3] have exhibited algorithms which use $n/2 + O(1)$ multiplications. However, the preprocessing used by these algorithms involves finding the roots of polynomials of large degree and this may be computationally difficult in itself. Therefore, we are led to investigate how close we can get to $n/2$ multiplications by using algorithms with rational preprocessing. The best known result is given by the following.

THEOREM 4. *Any polynomial of degree n can be evaluated using $n/2 + O(\log n)$ multiplications. Moreover, the scalars used by the algorithm are rational functions of the coefficients of the polynomial.*

Proof.

ALGORITHM A. Assume $n = 2^m - 1$. First compute $x^2, x^4, x^8, \cdots, x^{2^{m-1}}$. This requires $\log_2 n$ multiplications. Let $N(d)$ = number of multiplications used by the algorithm to evaluate a monic polynomial of degree d given that we have $x^2, x^4, \cdots, x^{2^{m-1}}$. $N(1) = 0$ because $x + a_0$ requires no multiplications.

Now $x^{2p-1} + a_{2p-2}x^{2p-2} + \cdots + a_1x + a_0 = (x^p + c)(x^{p-1} + a_{2p-2}x^{p-2} + \cdots + a_{p+1}x + a_p) + x^{p-1} + b_{p-2}x^{p-2} + \cdots + b_1x + b_0$, where $c = a_{p-1} - 1$ and $b_j = a_j - ca_{p+j}$ for $j = 0, \cdots, p-2$. Therefore $N(2^i - 1) = 2N(2^{i-1} - 1) + 1$ which yields

$$N(2^i - 1) = 2^{i-1} - 1 \quad \text{for } i = 1, \cdots, m.$$

So,

$$N(n) = N(2^{m-1}) = 2^{m-1} - 1 = (n + 1)/2 - 1.$$

Allowing one more multiplication for the monic division, we have

$$N(n) = (n + 1)/2.$$

Total multiplications $= (n + 1)/2 + \log_2 n$ if $n = 2^m - 1$. For general n, we can break the polynomial into pieces of length $2^i - 1$, evaluate them separately and put them back together using the powers $x^2, x^4, \cdots, x^{2\lfloor\log_2 n\rfloor}$. The putting-back-together can require at most another $\log_2 n$ multiplications for a total of $n/2 + 2\log_2 n$. This completes the proof.

Essentially the same algorithm was discovered independently by Rabin and Winograd and is given in [4].

3.2. Algorithms which use $O(\sqrt{n})$ nonscalar multiplications. We now present two algorithms which use $O(\sqrt{n})$ nonscalar multiplications. These algorithms can in theory be used to evaluate any rational polynomial $p(x)$ by evaluating $z_0 \cdot p(x)$ by an algorithm over $\mathbf{Z}$ for some appropriate $z_0 \in \mathbf{Z}$. As mentioned before, we can perform an integer scalar multiplication by successive additions.

First we describe a technique for producing $O(\sqrt{n})$ nonscalar multiplications algorithms from $O(n)$ algorithms in which we count all multiplications. The idea is to evaluate $x^2, x^3, x^4, \cdots, x^k$ for some k and then view a polynomial of degree $n = km$ in x as a polynomial of degree m in x^k whose coefficients are themselves polynomials of degree $k - 1$. These polynomials of degree $k - 1$ act like scalars because once we have $x^2, x^3, \cdots, x^{k-1}$ we can compute them free of charge. The only complication which arises is that these polynomial "scalars" become polynomials of larger degree when multiplied whereas numerical scalars remain single numbers when multiplied.

As an example of this method applied to Horner's rule, we obtain the following $O(\sqrt{n})$ algorithm.

ALGORITHM B.

$$\begin{aligned}
a_{km-1}x^{km-1} &+ \cdots + a_1x + a_0 \\
&= (\cdots((a_{km-1}x^{k-1} + \cdots + a_{k(m-1)})x^k \\
&\quad + a_{k(m-1)-1}x^{k-1} + \cdots + a_{k(m-2)})x^k \\
&\quad \vdots \\
&\quad + a_{2k-1}x^{k-1} + \cdots + a_{k+1}x + a_k)x^k \\
&\quad + a_{k-1}x^{k-1} + \cdots + a_1x + a_0.
\end{aligned}$$

This requires about $k + m = k + n/k$ nonscalar multiplications. Minimizing $k + n/k$ gives $k = \sqrt{n}$, or $2\sqrt{n}$ nonscalar multiplications. Note that this algorithm uses about n additions and $n - \sqrt{n}$ scalar multiplications.

The best known coefficient of $\sqrt{n}$ is obtained by applying the technique to the $n/2 + O(\log n)$ algorithm of Theorem 4.

THEOREM 5. *Any polynomial of degree n can be evaluated using $\sqrt{2n} + O(\log n)$ nonscalar multiplications. Moreover, the scalars used by the algorithm are rational functions of the coefficients of the polynomial.*

Proof.

ALGORITHM C. Assume $n = k(2^{m-1})$.

(i) Compute $x^2, x^3, x^4, \cdots, x^k$ (k multiplications).

(ii) Compute $x^{2k}, x^{4k}, x^{8k}, \cdots, x^{2^{m-1}k}$ ($\log_2(n/k) = m$ multiplications).

Let $N(d)$ = number of nonscalar multiplications used by the algorithm to evaluate a monic polynomial of degree d given that we have the powers computed in (i) and (ii) above. $N(k) = 0$ because scalar multiplication is free.

We split a monic polynomial up in an analogous way to that of Algorithm A, but the derivation is a little more complicated.

Let $p(x)$ be a monic polynomial of degree $k(2p - 1)$ which we express in the form

$$p(x) = q(x)\cdot x^{kp} + r(x),$$

where q is monic,

$$\deg(q) = k(p-1), \qquad \deg(r) \leqq kp - 1.$$

Formally dividing the polynomial $r(x) - x^{k(p-1)}$ by $q(x)$ we obtain rational polynomials $c(x)$ and $s(x)$ satisfying:

$$r(x) - x^{k(p-1)} = c(x)\cdot q(x) + s(x),$$

$$\deg(c) \leqq k - 1, \qquad \deg(s) \leqq k(p-1) - 1.$$

Then

$$p(x) \equiv (x^{kp} + c(x)) \cdot q(x) + x^{k(p-1)} + s(x).$$

Hence, taking $p = 2^{i-1}$, where $i \leqq m$, we get

$$N(k(2^i - 1)) = 2N(k(2^{i-1} - 1)) + 1$$

which yields

$$N(k(2^m - 1)) = 2^{m-1} - 1 \approx n/2k.$$

The total number of nonscalar multiplications is

$$n/2k + k + \log_2 (n/k).$$

Minimizing with respect to k gives $k \approx \sqrt{n/2}$, or $\sqrt{2n} + \log_2 \sqrt{2n}$ nonscalar multiplications.

As before, for general n, this algorithm may require an extra $\log \sqrt{2n}$ multiplications giving a final total of

$$\sqrt{2n} + \log_2 n + O(1) \text{ nonscalar multiplications}.$$

This completes the proof.

Note that this algorithm uses about $n + \sqrt{n/2}$ additions and $n - \sqrt{n/2}$ scalar multiplications.

Acknowledgment. We gratefully acknowledge the many helpful discussions with Albert Meyer concerning this work.

REFERENCES

[1] J. EVE, *The evaluation of polynomials*, Numer. Math., 6 (1964), pp. 17–21.

[2] T. S. MOTZKIN, *Evaluation of polynomials and evaluation of rational functions*, Bull. Amer. Math. Soc., 61 (1955), p. 163.

[3] V. YA. PAN, *Methods of computing values of polynomials*, Uspekhi Mat. Nauk, 21 (1966), no. 6; English transl., Russian Math. Surveys., 21 (1966), pp. 105–136.

[4] M. O. RABIN AND S. WINOGRAD, *Fast evaluation of polynomials by rational preparation*, IBM Res. Rep. RC 3645, Yorktown Heights, N.Y., 1971.

[5] B. L. VAN DER WAERDEN, *Modern Algebra*, Frederick Ungar, New York, 1949.

[6] S. WINOGRAD, *On the number of multiplications necessary to compute certain functions*, Comm. Pure Appl. Math., 23 (1970), pp. 165–179.

SIAM J. COMPUT.
Vol. 2, No. 2, June 1973

ON THE EQUIVALENCE OF ASYNCHRONOUS CONTROL STRUCTURES*

J. ROBERT JUMP AND P. S. THIAGARAJAN†

Abstract. This paper is concerned with the problem of detecting when two asynchronous control systems are equivalent. The systems investigated in the paper are first represented by means of a formal model called an asynchronous control structure (ACS). This model specifies the constraints imposed on the generation of control signals by a system by means of a simple graphical model called a marked graph. Behavioral equivalence is then characterized in terms of the set of all possible sequences of control signals that can be generated by the system. These sequences are represented by means of another (infinite) marked graph, called a behavior graph. Finally, it is shown that two control systems are equivalent if and only if their behavior graph representations have identical (finite) generating sets.

Key words. Control structure models, asynchronous control systems, concurrent events, marked graphs, behavioral equivalence.

1. Introduction.

In this paper we present a formal model for a class of asynchronous control systems and develop a complete characterization of "behavioral equivalence" based on this model. This characterization is then used to show the existence of a decision algorithm which can be used to determine whether or not two given systems are equivalent.

Abstractly, a control system may be viewed as a device that enforces certain specified constraints on the order of occurrence of "events." We shall not give an explicit definition of an event but shall assume that occurrences of events have the following characteristics.

1. An occurrence is initiated by a control signal called a *ready signal.*
2. Once initiated, an occurrence requires a finite but unbounded period of time.
3. When an occurrence terminates, an *acknowledge signal* is generated.
4. Each event may occur repeatedly, and several different events may occur concurrently.

A control system communicates with its environment through communication *links*, and events are classified as either *input events* or *output events.* An input event is one that is initiated by the environment by sending a ready signal to the control system through an input link. When the event is terminated, an acknowledge signal is transmitted to the environment over the same link. Output events are those that are initiated by the control system when a ready signal is generated by the system on one of its output links. The environment then signals the completion of the event by generating an acknowledge signal.

The ready and acknowledge signals associated with a link will be referred to as *link signals.* If the last link signal transmitted through a link was an acknowledge signal, the link is said to be *idle.* A link is *active* if the last link signal was a ready signal. Initially, all links of a system are idle. A *communication cycle* on a link consists of a ready signal, which activates the link, followed by an acknowledge signal, which sets the link idle again.

* Received by the editors July 11, 1972, and in revised form February 13, 1973.

† Department of Electrical Engineering, Rice University, Houston, Texas 77001. This research was supported by the National Science Foundation under Grant GJ-750.

Control systems of this general type have been proposed by several authors as a means of coordinating parallel computations (e.g., [2], [4], [5], [10], [11], [16], [17]). In this case, the events correspond to the different types of operations executed during the computation, and the constraints imposed by the control system ensure that the computation is determinate (i.e., independent of variations in the time required to execute the operations of the computation).

The usual method for representing constraints on the generation of control signals is by means of a directed graph. The model presented in this paper is also based on a directed graph, called a marked graph [8], [9], in which the state of the system is represented by placing markers on some of the edges. A change of state is then simulated by the movement of markers in the graph. Although marked graphs have been primarily used to study problems of scheduling and optimization of resources in multiprocessor systems, they are also useful tools for representing control systems. Indeed, a more general version, called a Petri net, has been used for this purpose [6], [15].

We have restricted our attention to marked graphs because they are more amenable to analysis than Petri nets while still representing a useful and powerful class of systems. In fact, by assigning the same event to more than one vertex in the marked graph we can model systems in which different occurrences of the same event can have different effects on the subsequent activity of the system. In this way, we can represent a certain type of "pseudo-conflict" which is usually represented by means of the more general Petri net [15].

In the next section, we review the theory of marked graphs. The control system model is presented and its basic properties are developed in § 3. The problem of characterizing and detecting equivalence of control systems is formulated and solved in §§ 4 and 5. Finally, in § 6, we offer a more detailed comparison of the models and results presented in this paper with other related work which has appeared in the literature.

2. Marked graphs. This section contains a brief summary of those basic definitions and properties of marked graphs that will be used in this paper. A more detailed and complete development of the theory of marked graphs can be found in the report by Holt and Commoner [8]. We begin by summarizing the graph theory terminology that will be used.

A *graph* is an ordered pair (T, P), where T is a countable set of *vertices* and $P \subseteq T \times T$ is the set of edges. A path π in the graph (T, P) is a sequence, $\pi = t_0, t_1, \cdots, t_n$, of vertices such that $(t_i, t_{i+1}) \in P$ for $0 \leqq i < n$. For such a path π we also say that:

(a) The *length* of π is n.

(b) π extends from t_0 to t_n.

(c) t_0 and t_n are *end points* and t_i, for $1 \leqq i < n$, are the *inner vertices* of π.

(d) π is a *cycle* if $t_0 = t_n$.

(e) π is *elementary* if all its inner vertices are distinct.

A graph (T, P) is *strongly connected* if, for any two vertices t_1 and t_2 in T, there is a path which extends from t_1 to t_2. Given a vertex t, $I(t)$ denotes the set of all edges directed into t and $O(t)$ denotes the set of all edges directed out of t.

A *marking* of a graph (T, P) is a function M from P into the set of nonnegative integers (i.e., $M: P \to \{0, 1, 2, \cdots\}$).

DEFINITION. A *marked graph* is a triple (T, P, M), where (T, P) is a graph[1] in which $I(t)$ and $O(t)$ are finite sets for all $t \in T$, and M is a marking of the graph (T, P).

For the marked graph (T, P, M), the elements of T are also called *transitions* and M is said to be its *initial marking*. An example of a marked graph is shown in Fig. 2.1. Its marking M is represented by placing *markers* (darkened circles) on the

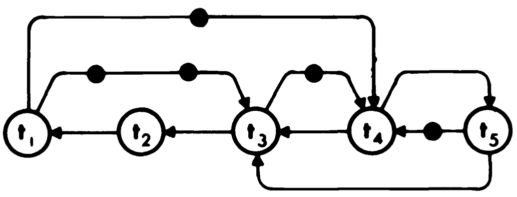


FIG. 2.1. *Example of a marked graph*

edges such that edge e contains $M(e)$ markers. Thus $M(t_1, t_3) = 2$, $M(t_3, t_4) = 1$, $M(t_2, t_1) = 0$, etc. Because of this representation, we shall refer to $M'(e)$ as the number of markers on edge e under marking M', where M' can be any marking and e any edge.

Given a graph (T, P), a transition $t \in T$ is *firable* under the marking M' if $M'(e) > 0$ for all $e \in I(t)$. Furthermore, when t is fired, a new marking M'' is produced, where M'' is defined by

$$M''(e) = \begin{cases} M'(e) - 1 & \text{if } e \in I(t) - O(t), \\ M'(e) + 1 & \text{if } e \in O(t) - I(t), \\ M'(e) & \text{otherwise.} \end{cases}$$

Hence, the operation of firing a transition t can be represented by removing one marker from each edge in $I(t)$ and adding one marker to each edge in $O(t)$. In Fig. 2.1, transition t_4 is the only one that is firable under the indicated marking.

Let Q be the set of all possible markings of the graph (T, P). Then the *next-marking fuction* δ is a partial function from $Q \times T$ into Q defined as follows:

(a) $\delta(M', t) = M''$ if t is firable under M' and M'' is the next marking produced when t fires.

(b) δ is undefined at (M', t) if t is not firable under M'.

δ is extended to $Q \times T^*$ in the usual way.[2] Let $\bar{t}$ be a string in T^*. Then

(a) $\delta(M', \bar{t}) = M''$ if $\bar{t} = t_1 t_2 \cdots t_n$ with $n > 0$, and there is a sequence of markings $M_0, M_1, \cdots, M_n$ such that

  (1) $M_0 = M'$ and $M_n = M''$, and

  (2) t_i is firable under M_{i-1} and $\delta(M_{i-1}, t_i) = M_i$ for $1 \leqq i \leqq n$.

(b) $\delta(M', \bar{t}) = M'$ if $\bar{t} = \lambda$, the string of length zero.

(c) $\delta(M', \bar{t})$ is undefined otherwise.

If δ is defined at $(M', \bar{t})$, we say that the string $\bar{t}$ is firable under M'. Thus λ is firable under any marking.

[1] In this paper, the class of marked graphs without multiple edges is sufficient.

[2] T^* denotes the free monoid generated by the set T.

DEFINITION. Let (T, P, M) be a marked graph. Then $\bar{t} \in T^*$ is said to be a *firing sequence* if $\bar{t}$ is firable under the initial marking M.

For example, in Fig. 2.1, $\bar{t} = t_4t_5t_3t_2t_1$ is a firing sequence and $\delta(M, \bar{t}) = M$. A marking M' of the marked graph (T, P, M) is said to be *reachable* if there is a firing sequence $\bar{t}$ such that $\delta(M, \bar{t}) = M'$.

The following proposition, proved by Holt and Commoner [8], will be used extensively in the later sections. We first introduce the following notations.

(a) Let $\bar{u}$ be a string of symbols and x a single symbol. Then $\#(x|\bar{u})$ denotes the number of times x appears in $\bar{u}$.

(b) Let $\pi = t_0, t_1, \cdots, t_n$ be a path in the graph (T, P) and M' a marking. Then $\Sigma(M'|\pi)$ denotes the number of markers on π under M'. That is,

$$\Sigma(M'|\pi) = \sum_{i=1}^{n} M'(t_{i-1}, t_i).$$

PROPOSITION 2.1. *Let (T, P, M) be a marked graph and $\pi = t_0, t_1, \cdots, t_n$ an elementary path of this graph. Furthermore, let $\bar{u}$ be a firable string of transitions under marking M' and let $M'' = \delta(M', \bar{u})$. Then*

$$\Sigma(M''|\pi) = \Sigma(M'|\pi) + \#(t_0|\bar{u}) - \#(t_n|\bar{u}).$$

Thus the number of markers left on a path π, after the firing of a sequence of transitions, is uniquely determined by the number of markers on π before the sequence fires and the number of times the endpoints of π fire.

Two useful restrictions that may be imposed on a marked graph are that it be live and safe. A transition is *live* if it appears in at least one firing sequence. A marked graph is *live* if all of its transitions are live. Liveness of finite marked graphs has been characterized as follows.

PROPOSITION 2.2 (Theorem 1 in [9]). *A finite marked graph is live if and only if every cycle in the graph contains at least one edge e such that $M(e) > 0$.*

A marked graph (T, P, M) is said to be *safe* if, for all reachable markings M' and edges $e \in P$, $M'(e) \leqq 1$. Safeness in finite marked graphs has been characterized by means of a distinguished type of cycle called a synchronizing loop.[3]

DEFINITION. A *synchronizing loop* of a marked graph (T, P, M) is an elementary cycle that contains exactly one marker under the initial marking M.

PROPOSITION 2.3 (Theorem 2 in [9]). *A finite and live marked graph is safe if and only if every edge is contained in a synchronizing loop.*

For every edge e and reachable marking M' of a safe marked graph (T, P, M), $M'(e) \in \{0, 1\}$. Hence, for safe marked graphs, we will also use M' to denote the set of edges that contain a marker under M'. We will therefore write $e \in M'$ whenever $M'(e) = 1$ and $e \notin M'$ if $M'(e) = 0$.

Let π be a synchronizing loop such that (u, v) is the edge of π in M. Then since the number of markers on a cycle does not change by transition firings [9, Lemma 1], the firing of the transitions in the loop is totally ordered. Moreover, this ordering induces a total ordering on any subset of edges of the loop in a natural way. Indeed, if X is a subset of the edge set of the synchronizing loop π, and $(t, t') \in X$, then we define the *loop order* of (t, t'), *relative to* X, to be $i - 1$, where i is the number

[3] Holt and Commoner use the term "basic circuit" [8].

of edges in X that lie on the subpath of π from u to t'. In Fig. 2.1, $\pi = t_1, t_4, t_5, t_3, t_2, t_1$ is a synchronizing loop. Let $X = \{(t_4, t_5), (t_3, t_2), t_2, t_1)\}$. Then the loop order of (t_4, t_5) is 0, that of (t_3, t_2) is 1, and that of (t_2, t_1) is 2.

3. Asynchronous control structures. In this section we present a formal model for the class of control systems studied in this paper. This model is obtained by augmenting the marked graph model to explicitly represent the input and output signals of the system. We then define the behavior of these systems by means of the sequences of signals that satisfy the constraints of the system as represented by this formal model.

We begin by associating the links of a control system with the edges of a marked graph in the following way.

DEFINITION. Let $G = (T, P, M)$ be a marked graph and let α be a partial function from P onto the set $\{1, 2, \cdots, L\}$ for some positive integer L. Then α is called a *link assignment* for G if the set $\alpha^{-1}(i) = \{e \in P | \alpha(e) = i\}$ is contained in at least one synchronizing loop of G, for $1 \leqq i \leqq L$.

The link assignment α assigns edge e to link $\alpha(e)$. If none of the edges in $\alpha^{-1}(i)$ contains a marker under M, then the elements of $\alpha^{-1}(i)$ are called *output edges*; if exactly one contains a marker, then they are said to be *input edges*. Note that more than one edge may be assigned to the same link and that there is at least one edge assigned to every link. The following restrictions on link assignments will be needed later to establish the desired relationship between the formal marked graph model and the class of control systems it models.

DEFINITION. A link assignment α, for the marked graph $G = (T, P, M)$, is said to be *valid* if it satisfies the following two restrictions:

(1) No transition in T is the endpoint of more than one edge in dom α.[4]

(2) If (t', t) is an edge in dom α, then $|I(t)| = 1$.[5]

We can now define an abstract model for the class of control systems studied in this paper.

DEFINITION. An *asynchronous control structure* (ACS) is a triple $C = (G, \alpha, L)$, where

(1) G is a finite marked graph that is live, safe, and strongly connected;

(2) α is a valid link assignment for G;

(3) the range of α is $\{1, 2, \cdots, L\}$.

In order to introduce link signals into the ACS model, we shall denote the ready signal on link i by r_i and the acknowledge signal by a_i. These signals will be called *external signals* since they represent the interaction of the control system with its environment. The set $\{r_i, a_i | 1 \leqq i \leqq L\}$ of all external signals will be denoted by the symbol S_E. Since α is not necessarily one-to-one, different occurrences of an external signal may be associated with different edges in the marked graph. In order to distinguish these different occurrences, we define an *internal signal* to be one of the form r_i^j or a_i^j, where $1 \leqq i \leqq L$ and $0 \leqq j < |\alpha^{-1}(i)| = \hat{\imath}$.[6] The set of all internal signals will be denoted by the symbol S_I.

[4] dom α denotes the domain of the partial function α.

[5] $|I(t)|$ denotes the cardinality of the set $I(t)$.

[6] Notation: The integer $|\alpha^{-1}(i)|$ appears in numerous expressions throughout the paper. It will therefore be denoted by the symbol $\hat{\imath}$ in order to simplify these expressions.

We make the assignment of internal signals to transitions in G as follows.

DEFINITION. Let $C = (G, \alpha, L)$ be an ACS with $G = (T, P, M)$. Then the *signal assignment* for C is a function β from T into $T \cup S_I$ defined as follows:

(1) $\beta(t) = t$ if t is not the endpoint of an edge in dom α.

(2) Let $e = (t, t')$ be an edge in dom α with $\alpha(e) = i$ and loop order j, relative to $\alpha^{-1}(i)$.

(a) $\beta(t) = r_i^j$ and $\beta(t') = a_i^j$ if e is an output edge, and

(b) $\beta(t) = a_i^{R[j-1/\hat{\imath}]}$ and $\beta(t') = r_i^j$ if e is an input edge.[7]

Note that β is completely determined by the link assignment α. Moreover, it is well-defined since α is a valid link assignment (condition (1)). Since no two edges in $\alpha^{-1}(i)$ have the same loop order, the function β is one-to-one and onto.

Although the behavior of a control system can be represented and characterized by the ACS model and its signal assignment, we have found the following model, which combines these two concepts, to be more useful.

DEFINITION. Let $C = (G, \alpha, L)$ be an asynchronous control structure with $G = (T, P, M)$ and β its signal assignment. Then the *signal graph* for C is the marked graph $G_c = (T_c, P_c, M_c)$, where

(1) $T_c = \beta(T) = \{\beta(t) | t \in T\}$,

(2) $P_c = \beta(P) = \{(\beta(t), \beta(t')) | (t, t') \in P\}$,

(3) $M_c = \beta(M) = \{(\beta(t), \beta(t')) | (t, t') \in M\}$.

The set $T_I = T_c - S_I$ is called the set of *internal transitions* of G_c. Thus the signal graph is obtained from the ACS by renaming all transitions which are assigned an internal signal with the signal itself. The correspondence between the formal model and control system can now be established by means of this graph.

An input link i is idle if one of the edges in $\alpha^{-1}(i)$ contains a marker, otherwise it is active. The interpretation for output links is reversed so that output link i is idle when none of the edges in $\alpha^{-1}(i)$ contains a marker, and active otherwise. The restriction that $\alpha^{-1}(i)$ be contained in a synchronizing loop and the definition of input and output edges ensure that the system starts with all links idle under the initial marking M.

The activity of a control system is simulated by the movement of markers in the signal graph. When a signal vertex x_i^j fires, this is interpreted as the generation of the external signal x_i. Recall that a communication cycle on a link consists of a ready signal followed by an acknowledge signal which causes the link state to go from idle to active and back to idle again. If link i is an idle input link, then there is an input edge $(a_i^{R[j-1/\hat{\imath}]}, r_i^j)$ in the signal graph which contains a marker. When vertex r_i^j fires, generating the external signal r_i, the marker is removed, signifying the change of the link state from idle to active. The link remains active until a marker is placed on edge $(a_i^j, r_i^{R[j+1/\hat{\imath}]})$ by vertex a_i^j firing and generating signal a_i. Hence a communication cycle on an input link i is simulated by a marker leaving an input edge in $\alpha^{-1}(i)$ with loop order j, followed by a marker entering the edge in $\alpha^{-1}(i)$ with loop order $R[j + 1/i]$. In a similar way, a communication cycle on an output link is simulated by a marker passing through an output edge.

[7] $R[j/\hat{\imath}]$ denotes the remainder and $Q[j/\hat{\imath}]$ the quotient obtained by dividing j by $\hat{\imath}$. Thus $j = Q[j/\hat{\imath}]\hat{\imath} + R[j/\hat{\imath}]$.

Note that the condition that $\alpha^{-1}(i)$ be contained in a synchronizing loop will ensure that ready and acknowledge signals alternate on each link. Since the loop order has been used to assign superscripts to the internal signals, the firing of signal vertex x_i^j may be interpreted as the jth (modulo $\hat{\imath}$) generation of external signal x_i. Finally, the two restrictions that are required of valid link assignments have the following interpretation. The restriction that no transition be the endpoint of two edges in dom α ensures that no two signals will be constrained to occur simultaneously. The second restriction guarantees that the only constraint placed on the generation of signals by the environment of a control system is that ready and acknowledge signals on the same link alternate.

Fig. 3.1 is an example of the signal graph of an ACS. It has two input edges (both assigned to link 1) and three output edges (all assigned to link 2). The first

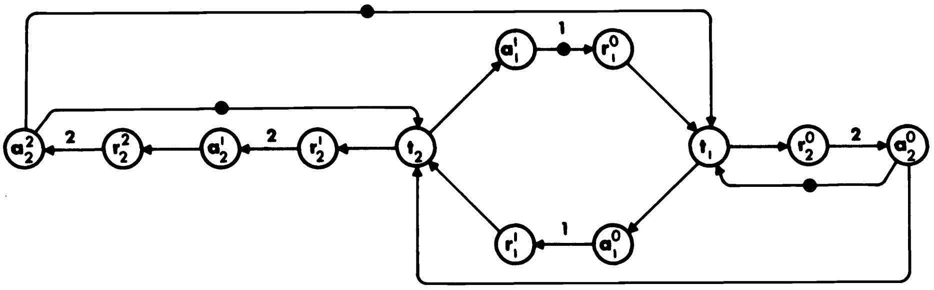


FIG. 3.1. *Example of a signal graph*

communication cycle on link 1 corresponds to the movement of a marker through transition r_1^0, t_1 and a_1^0. After this, the second cycle on link 1 can be initiated (by firing vertex r_1^1), but it will not be completed until the first cycle on link 2 is complete. The initiation of the second cycle on link 1 and the termination of the first cycle on link 2 will initiate a sequence of two cycles on link 2 by firing t_2. After t_2 fires, the second cycle on link 1 will also be completed allowing the initiation of the third cycle on that link. After two cycles on link 1 and three on link 2, the system behavior repeats.

The behavior of a control system is completely characterized by the set of all possible sequences of signals on its links. Due to the correspondence between the generation of link signals and the firing of transitions established above, the behavior of an ACS will be characterized by means of the set of all possible firing sequences of its signal graph.

DEFINITION. Let $\bar{u}$ be a firing sequence for the signal graph G_c. Then the corresponding *reduced firing sequence*, denoted by $\rho(\bar{u})$, is the sequence of internal signals obtained by deleting all of the internal transitions (elements of T_I) from $\bar{u}$.

Note that if $\bar{u} \in T_I^*$, then $\rho(\bar{u}) = \lambda$. As an example, consider the firing sequence $\bar{u} = r_1^0 t_1 r_2^0 a_1^0 r_1^1 a_2^0 t_2 a_1^1$ for the signal graph in Fig. 3.1. Then $\rho(\bar{u})$ is the sequence $r_1^0 r_2^0 a_1^0 r_1^1 a_2^0 a_1^1$.

DEFINITION. Let $\bar{v}$ be a reduced firing sequence for the signal graph G_c. Then the corresponding *signal sequence*, denoted by $\sigma(\bar{v})$, is the sequence of external signals obtained by deleting the superscripts from all of the symbols in $\bar{v}$.

For the reduced firing sequence $\bar{v} = r_1^0 r_2^0 a_1^0 r_1^1 a_2^0 a_1^1$ for the signal graph of Fig. 3.1, $\sigma(\bar{v}) = r_1 r_2 a_1 r_1 a_2 a_1$. We can now define the behavior of an ACS C as the set of all signal sequences for G_c.

DEFINITION. Let C be an ACS and G_c its signal graph. Then the *behavior* of C is the set $B_c = \{\sigma(\rho(\bar{u})) | \bar{u}$ is a firing sequence of $G_c\}$.

The main goal of this paper is to develop an effective method for determining when two control structures have equal behaviors.

4. Behavior graphs. Although the asynchronous control structure model is a concise representation for the behavior of control systems, it is not unique. Therefore, in order to develop a test for equivalence of control systems, it is first necessary to introduce an alternative marked graph representation called a behavior graph. In this section, we show how the behavior graph for a control system can be obtained directly from its signal graph. We then develop the basic properties of behavior graphs that will be needed. In the next section, we show that a reduced behavior graph (i.e., one with no redundant edges) is a unique behavioral representation. We then present a finite test for equivalence based on this model.

In order to develop the behavior graph model, we shall use the following terminology. An elementary path in a signal graph is said to be *signal-free* if none of its inner vertices are internal signals. A path π from x to y is called *marker-minimal* if $\Sigma(M_c|\pi') \geqq \Sigma(M_c|\pi)$ for all paths π' from x to y. Clearly every subpath of a marker-minimal path is also marker-minimal. Finally, a signal-free path is said to be *maximal* if it is not a subpath of any other signal-free path. Note that the terminal endpoint of a maximal signal-free path is an internal signal and that any path from one internal signal to another can be decomposed into a sequence of maximal signal-free subpaths.

DEFINITION. Let C be an ACS and G_c its signal graph. Then the *constraint relation* of C is the ternary relation $\gamma_c \subseteq S_I \times S_I \times \{0, 1, 2, \cdots\}$ defined by: $(x, y, m) \in \gamma_c$ if and only if there is a marker-minimal, signal-free path π from x to y such that $\Sigma(M_c|\pi) = m$.

For example, if C denotes the ACS given in Fig. 3.1, then

$$\{(a_2^0, r_2^0, 1), (a_2^0, a_1^0, 1), (a_2^0, a_1^1, 0), (a_2^0, r_2^1, 0)\} \subseteq \gamma_c$$

is the set of all elements in γ_c whose first component is a_2^0.

The constraint relation γ_c represents the constraints, on the order of occurrence of internal signals, that are enforced by the control structure C. It may be interpreted as follows. If $(x, y, m) \in \gamma_c$, then the ith occurrence of the signal x must precede the $(i + m)$th occurrence of y in any reduced firing sequence for G_c. We can now define the behavior graph model.

DEFINITION. Let γ_c denote the constraint relation and S_E the set of external signals of an asynchronous control structure C. Then the *behavior graph* for C is the (infinite) marked graph $\bar{G}_c = (\bar{T}_c, \bar{P}_c, \bar{M}_c)$, where

(1) $\bar{T}_c = \{x_i^j | x_i \in S_E \text{ and } j \in Z\}$,[8]

(2) $\bar{P}_c = \{(x_i^{j+ni}, y_k^{l+(n+m)k}) | (x_i^j, y_k^l, m) \in \gamma_c \text{ and } n \in Z\}$,

(3) $\bar{M}_c = \{(x_i^j, y_k^l) \in \bar{P}_c | j < 0 \text{ and } l \geqq 0\}$.

[8] Z denotes the set $\{\cdots, -2, -1, 0, 1, 2, \cdots\}$.

The firing of transition x_i^j in $\bar{G}_c$ corresponds to the jth occurrence of the link signal x_i. Moreover, there is exactly one transition in $\bar{G}_c$ for every occurrence of every link signal. Hence the behavior graph for a system explicitly represents the constraints on the generation of control signals by the system. Figure 4.1 shows a portion of the behavior graph for the ACS given in Fig. 3.1.

The following result provides a more useful characterization of the relation $\bar{P}_c$.

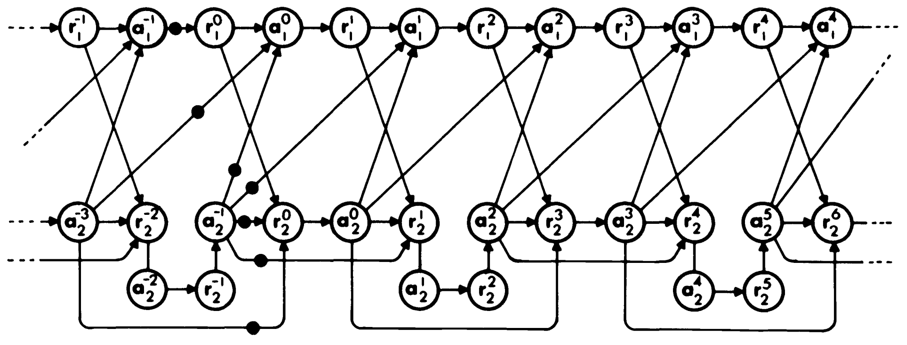


FIG. 4.1. *Behavior graph for the ACS of Fig.* 3.1

PROPOSITION 4.1. $(x_i^j, y_k^l) \in \bar{P}_c$ *if and only if*

$$(x_i^{R[j/\hat{\imath}]}, y_k^{R[l/\hat{k}]}, Q[l/\hat{k}] - Q[j/\hat{\imath}]) \in \gamma_c.$$

Proof. $(x_i^j, y_k^l) \in \bar{P}_c$ if and only if there is a triple $(x_i^p, y_k^q, m) \in \gamma_c$ such that $j = p + n\hat{\imath}$ and $l = q + (m + n)\hat{k}$ for some $n \in Z$. But $p < \hat{\imath}$ and $q < \hat{k}$ so that $p = R[j/\hat{\imath}], q = R[l/\hat{k}]$, and $Q[l/\hat{k}] - Q[j/\hat{\imath}] = m$. □

We now establish the relationship between paths in the signal graph and paths in the behavior graph of an ACS.

LEMMA 4.1. *Let G_c be the signal graph of an ACS and let x and y be internal signals of G_c. If π is a path from x to y in G_c, then there exists a path π' from x to y in G_c such that*

(1) $\Sigma(M_c|\pi') = \Sigma(M_c|\pi)$, *and*

(2) *every signal-free subpath of π' is marker-minimal.*

Proof. Let π_1 be a marker-minimal path from x to y in G_c and let $n = \Sigma(M_c|\pi) - \Sigma(M_c|\pi_1)$. By definition of G_c, y is contained in a synchronizing loop, say π_2, which is a marker-minimal cycle. Hence the path π', obtained by composing π_1 with n copies of π_2, satisfies conditions (1) and (2). □

THEOREM 4.1. *Let G_c be the signal graph and $\bar{G}_c$ the behavior graph of an ACS C. Let x_i^j and y_k^l be any two vertices of $\bar{G}_c$. Then there is a path from x_i^j to y_k^l in $\bar{G}_c$ if and only if there is a path π from $x_i^{R[j/\hat{\imath}]}$ to $y_k^{R[l/\hat{k}]}$ in G_c such that $\Sigma(M_c|\pi) = Q[l/\hat{k}] - Q[j/\hat{\imath}]$.*

Proof. Assume that there is a path π from $X_i^{R[j/\hat{\imath}]}$ to $y_k^{R[l/\hat{k}]}$ in G_c such that $\Sigma(M_c|\pi) = Q[l/\hat{k}] - Q[j/\hat{\imath}]$. Without loss of generality, we may assume that every signal-free subpath of π is marker-minimal (Lemma 4.1). We show that there is a path from x_i^j to y_k^l in $\bar{G}_c$ by induction on n, the number of internal signals on the path π.

Step 1. $n = 2$. The required result follows at once from Proposition 4.1.

Step 2. $n > 2$. Assume the result holds for all paths in G_c that contain n internal signals and assume that π contains $n + 1$ internal signals. Then π can be decomposed into a subpath π_1, containing n internal signals, that extends from $x_i^{R[j/\hat{\imath}]}$ to z_p^u, and a signal-free path π_2 that extends from z_p^u to $y_k^{R[l/\hat{k}]}$, where $z_p^u \in S_I$. Let

$$q = (\Sigma(M_c|\pi_1) + Q[j/\hat{\imath}])\hat{p} + u.$$

Then, since $0 \leqq u < \hat{p}$, we have that

$$u = R[q/\hat{p}] \quad \text{and} \quad \Sigma(M_c|\pi_1) = Q[q/\hat{p}] - Q[j/\hat{\imath}].$$

Hence, by the induction hypothesis, there is a path from x_i^j to z_p^q in $\bar{G}_c$. Furthermore,

$$\begin{aligned}\Sigma(M_c|\pi_2) &= \Sigma(M_c|\pi) - \Sigma(M_c|\pi_1)\\ &= Q[l/\hat{k}] - Q[j/\hat{\imath}] - Q[q/\hat{p}] + Q[j/\hat{\imath}]\\ &= Q[l/\hat{k}] - Q[q/\hat{p}].\end{aligned}$$

By the basis step, therefore, there is a path from z_p^q to y_k^l in G_c. This establishes the existence of a path from x_i^j to y_k^l in $\bar{G}_c$.

Conversely, assume that there is a path π from x_i^j to y_k^l in $\bar{G}_c$. We shall show that there is a path π' from $x_i^{R[j/\hat{\imath}]}$ to $y_k^{R[l/\hat{k}]}$ in G_c and that $\Sigma(M_c|\pi') = Q[l/\hat{k}] - Q[j/\hat{\imath}]$. The proof is by induction on n, the length of π.

Step 1. $n = 1$. The result follows at once from Proposition 4.1.

Step 2. $n \geqq 2$. Assume the result for all paths of length n and let π be a path of length $n + 1$ from x_i^j to y_k^l in $\bar{G}_c$. Then there is a path π_1 of length n from x_i^j to z_p^q and a path of length 1 from z_p^q to y_k^l, where $z_p^q \in \bar{T}_c$. By the induction hypothesis, there is a path π_1' in G_c from $x_i^{R[j/\hat{\imath}]}$ to $z_p^{R[q/\hat{p}]}$ such that

$$\Sigma(M_c|\pi_1') = Q[q/\hat{p}] - Q[j/\hat{\imath}].$$

Furthermore, there is a path π_2' in G_c from $z_p^{R[q/\hat{p}]}$ to $y_k^{R[l/\hat{k}]}$ such that

$$\Sigma(M_c|\pi_2') = Q[l/\hat{k}] - Q[q/\hat{p}].$$

Clearly π' can be formed by composing π_1' and π_2'. □

Since G_c is by definition a live marked graph, every cycle in G_c contains at least one initial marker. Hence it can be easily shown, using Theorem 4.1, that $\bar{G}_c$ *is acyclic*. The following corollaries develop two other useful properties of behavior graphs.

COROLLARY 4.1. *Let x_i and y_k be external signals for an ACS C and let $j \in Z$. Then there is a path from x_i^j to y_k^l in $\bar{G}_c$ for some $l \in Z$, and a path from x_i^j to x_i^l for any $l > j$.*

Proof. This result follows easily from the requirements that G_c be strongly connected and that the set $\{x_i^j | 0 \leqq j < \hat{\imath}\}$ be contained in a synchronizing loop. □

COROLLARY 4.2. *Let $\bar{G}_c = (\bar{T}_c, \bar{P}_c, \bar{M}_c)$ be the behavior graph of an ACS, $x_i^j \in \bar{T}_c$, and $\bar{w}$ a firing sequence for $\bar{G}_c$. Then $0 \leqq \#(x_i^j|\bar{w}) \leqq 1$. Moreover, $\#(x_i^j|\bar{w}) = 0$ if $j < 0$.*

Proof. We first show that x_i^j is live if and only if $j \geqq 0$. From Corollary 4.1, it can be easily seen that the in-degree of every vertex in $\bar{G}_c$ is greater than zero. Hence every finite path of $\bar{G}_c$ is a terminal subpath of some infinite path of $\bar{G}_c$. Using Theorem E4 in [8], it follows that x_i^j is live if and only if every infinite path directed into x_i^j contains at least one initial marker.

Assume that $j < 0$ and let π be an infinite path directed into x_i^j. Let y_k^l be any other vertex on π. Then there is a path from y_k^l to x_i^j in $\bar{G}_c$. Hence, by Theorem 4.1, there is a path π' in G_c from $y_k^{R[l/\hat{k}]}$ to $x_i^{R[j/\hat{\imath}]}$ such that

$$\Sigma(M_c|\pi') = Q[j/\hat{\imath}] - Q[l/\hat{k}] \geqq 0.$$

Since $j < 0$, we have $Q[j/\hat{\imath}] < 0$, so that $l < 0$ also. Now from the definition of $\bar{M}_c$ and the fact that y_k^l was chosen arbitrarily, we have that $\Sigma(\bar{M}_c|\pi) = 0$. Hence x_i^j is not live.

Conversely, assume that $j \geqq 0$ and that y_k^l is any other vertex on an infinite path π directed into x_i^j. Using Theorem 4.1 as before, it can be shown that $l < (Q[j/\hat{\imath}] + 1)\hat{k}$. Hence the number of vertices on π, of the form y_k^l with $l \geqq 0$, is finite. Since π is infinite and $\bar{G}_c$ is acyclic, there exists a vertex z_p^q on π such that $q < 0$. From the definition of $\bar{M}_c$, it follows that the subpath of π from z_p^q to x_i^j contains at least one initial marker. Hence x_i^j is live.

We now show that if x_i^j is live, it can fire only once. To this end, let π be a path from x_i^{-1} to x_i^j, where $j > 0$ (Corollary 4.1). From the definition of $\bar{M}_c$, we have $\Sigma(\bar{M}_c|\pi) \geqq 1$. If there is more than one initial marker on π, then there must be an edge (z_p^q, y_k^l) of π such that $l < 0$ and $q \geqq 0$. Then $Q[q/\hat{p}] \geqq 0$ and $Q[l/\hat{k}] < 0$ so that $Q[l/\hat{k}] - Q[q/\hat{p}] < 0$. But this contradicts Proposition 4.1. Hence $\Sigma(\bar{M}_c|\pi) = 1$. Let $M' = \delta(\bar{M}_c|\bar{w})$. Then

$$\begin{aligned}\Sigma(M'|\pi) &= \Sigma(\bar{M}_c|\pi) + \#(x_i^{-1}|\bar{w}) - \#(x_i^j|\bar{w})\\ &= 1 - \#(x_i^j|\bar{w}),\end{aligned}$$

since x_i^{-1} never fires. Since $\Sigma(M'|\pi) \geqq 0$, we have that $\#(x_i^j|\bar{w}) \leqq 1$. $\square$

We shall associate signal sequences with firing sequences in the same way that it was done for signal graphs. Thus if $\bar{u}$ is a firing sequence of a behavior graph, then $\sigma(\bar{u})$ denotes the signal sequence obtained by deleting the superscripts from all of the symbols in $\bar{u}$. Note that it is not necessary to first define a reduced firing sequence since every transition in a behavior graph is associated with a signal. We now characterize the set of firing sequences and signal sequences of a behavior graph.

LEMMA 4.2. *Let $\bar{w}$ be a firing sequence of the behavior graph $\bar{G}_c = (\bar{T}_c, \bar{P}_c, \bar{M}_c)$ and let $x_i^j \in \bar{T}_c$. Then $\bar{w}x_i^j$ is a firing sequence for $\bar{G}_c$ if and only if $\#(y_k^l|\bar{w}) - \#(x_i^j|\bar{w}) \geqq 1 - \bar{M}_c(e)$ for all edges $e = (y_k^l, x_i^j) \in \bar{P}_c$.*

Proof. Assume that $\bar{w}x_i^j$ is a firing sequence of $\bar{G}_c$ and let $e = (y_k^l, x_i^j)$ be an element of $\bar{P}_c$ and $M' = \delta(\bar{M}_c, \bar{w})$. Then

$$M'(e) = \bar{M}_c(e) + \#(y_k^l|\bar{w}) - \#(x_i^j|\bar{w})$$

(Proposition 2.1). But $M'(e) > 0$ since $I(x_i^j) \subseteq M'$. Hence

$$\#(y_k^l|\bar{w}) - \#(x_i^j|\bar{w}) \geqq 1 - \bar{M}_c(e).$$

Conversely, assume that $\#(z_p^q|\bar{w}) - \#(x_i^j|\bar{w}) \geqq 1 - \bar{M}_c(e)$ for all $e' = (z_p^q, x_i^j)$ in $\bar{P}_c$. Let $e = (y_k^l, x_i^j)$ be an edge in $\bar{P}_c$ and $M' = \delta(\bar{M}_c, \bar{w})$. Then

$$M'(e) = \bar{M}_c(e) + \#(y_k^l|\bar{w}) - \#(x_i^j|\bar{w}).$$

But by hypothesis,

$$\#(y_k^l|\bar{w}) - \#(x_i^j|\bar{w}) \geqq 1 - \bar{M}_c(e).$$

Hence $M'(e) > 0$ so that $e \in M'$. But this implies that $I(x_i^j) \subseteq M'$. Therefore, x_i^j is firable under M' and $\bar{w}x_i^j$ is a firing sequence for $\bar{G}_c$. □

PROPOSITION 4.2. *Let $\bar{w}$ be a firing sequence for the behavior graph $\bar{G}_c = (\bar{T}_c, \bar{P}_c, \bar{M}_c)$. Let $x_i^j \in \bar{T}_c$ such that $\#(x_i^j|\bar{w}) = 0$ and $j > 0$. Then $\bar{w}x_i^j$ is a firing sequence for $\bar{G}_c$ if and only if $\#(y_k^l|\bar{w}) = 1$ for all y_k^l such that $l \geqq 0$ and there is a path from y_k^l to x_i^j in $\bar{G}_c$.*

Proof. Assume that $\bar{w}x_i^j$ is a firing sequence for $\bar{G}_c$ and that $y_k^l \in \bar{T}_c$. Further assume that $l \geqq 0$ and that there is a path π from y_k^l to x_i^j in $\bar{G}_c$. Let $M' = \delta(\bar{M}_c, \bar{w})$. Then

$$\#(y_k^l|\bar{w}) = \Sigma(M'|\pi) - \Sigma(\bar{M}_c|\pi) + \#(x_i^j|\bar{w}).$$

But $\Sigma(M'|\pi) > 0$ since $I(x_i^j) \subseteq M'$ and $\Sigma(\bar{M}_c|\pi) = 0$ since $l \geqq 0$. Moreover, it is given that $\#(x_i^j|\bar{w}) = 0$ so that $\#(y_k^l|\bar{w}) > 0$. But $\#(y_k^l|\bar{w}) \leqq 1$ (Corollary 4.2) so that $\#(y_k^l|\bar{w}) = 1$.

To prove the converse, assume that $\#(y_k^l|\bar{w}) = 1$ for all y_k^l such that $l \geqq 0$ and there is a path from y_k^l to x_i^j in $\bar{G}_c$. Let $e = (z_p^q, x_i^j)$ be an edge in $\bar{P}_c$.

Case 1. $q \geqq 0$. Then by hypothesis, $\#(z_p^q|\bar{w}) = 1$, $\#(x_i^j|\bar{w}) = 0$, and $\bar{M}_c(e) = 0$. Hence

$$\#(z_p^q|\bar{w}) - \#(x_i^j|\bar{w}) \geqq 1 - \bar{M}_c(e).$$

Case 2. $q < 0$. Then $\#(x_i^j|\bar{w}) = 0$, $\bar{M}_c(e) = 1$, and $\#(z_p^q|\bar{w}) = 0$ (Corollary 4.2). Hence

$$\#(z_p^q|\bar{w}) - \#(x_i^j|\bar{w}) \geqq 1 - \bar{M}_c(e).$$

In both cases, the result follows from Lemma 4.2. □

PROPOSITION 4.3. *Let $\bar{u}$ be a signal sequence of $\bar{G}_c$ and let $\bar{w}$ be a firing sequence of $\bar{G}_c$ such that $\sigma(\bar{w}) = \bar{u}$. Then for any external signal x_i, $\bar{u}x_i$ is a signal sequence if and only if $\bar{w}x_i^j$ is a firing sequence, where $j = \#(x_i|\bar{u})$.*

Proof. Assume that $\bar{u}x_i$ is a signal sequence. Then $\bar{w}x_i^j$ is a firing sequence for some $j \geqq 0$. Let l be an integer such that $0 \leqq l < j$. By Corollary 4.1, there is a path from x_i^l to x_i^j in $\bar{G}_c$. Hence $\#(x_i^l|\bar{w}) = 1$ (Proposition 4.2). Also, $\#(x_i^j|\bar{w}) = 0$ since $\bar{w}x_i^j$ is a firing sequence (Corollary 4.2). On the other hand, if $l > j$, then $\#(x_i^l|\bar{w}) = 0$ (Corollary 4.1 and Proposition 4.2). But this implies that $\#(x_i^l|\bar{w}) = 1$ if and only if $0 \leqq l < j$. Hence $\#(x_i|\bar{u}) = j$. The converse is obvious.

COROLLARY 4.3. *There is a one-to-one correspondence between the set of all firing sequences and the set of all signal sequences for a behavior graph $\bar{G}_c$.*

Proof. Let $\bar{u}$ and $\bar{v}$ be two firing sequences such that $\sigma(\bar{u}) = \sigma(\bar{v})$. If $\bar{u} \neq \bar{v}$, then there must exist a firing sequence $\bar{w}$ and two distinct transitions x_i^j and y_k^l such that $\bar{w}x_i^j$ and $\bar{w}y_k^l$ are firing sequences and $\sigma(\bar{w}x_i^j) = \sigma(\bar{w}y_k^l)$. But

$$\sigma(\bar{w}x_i^j) = \sigma(\bar{w})x_i = \sigma(\bar{w}y_k^l) = \sigma(\bar{w})y_k.$$

Hence $x_i = y_k$. But then

$$j = \#(x_i|\bar{w}) = \#(y_k|\bar{w}) = l$$

so that $x_i^j = y_k^l$. Thus $\bar{u} = \bar{v}$ and σ is one-to-one. □

The behavior graph of an ACS may contain redundant edges. These can be removed in the following way.

DEFINITION. Let C be an ACS and $\bar{G}_c = (\bar{T}_c, \bar{P}_c, \bar{M}_c)$ be its behavior graph. Then the *reduced behavior graph* for C is the marked graph $\bar{\bar{G}}_c = (\bar{\bar{T}}_c, \bar{\bar{P}}_c, \bar{\bar{M}}_c)$, where:

(1) $\bar{\bar{T}}_c = \bar{T}_c$.

(2) $\bar{\bar{P}}_c$ is obtained from $\bar{P}_c$ as follows: Let $(x_i^j, y_k^l) \in \bar{P}_c$. Then $(x_i^j, y_k^l) \in \bar{\bar{P}}_c$ if there is no path of length greater than 1 from x_i^j to y_k^l in $\bar{G}_c$.

(3) $\bar{\bar{M}}_c = \bar{M}_c \cap \bar{\bar{P}}_c$.

This definition is illustrated in Fig. 4.2. It can be easily shown that all of the properties of behavior graphs developed above also hold for reduced behavior

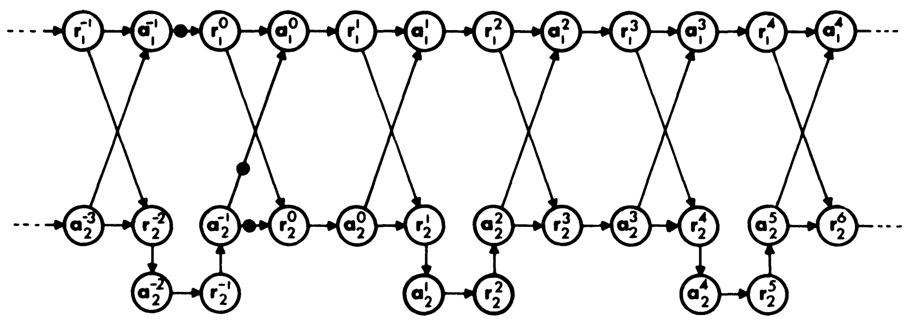


FIG. 4.2. *Reduced behavior graph for the ACS of Fig.* 3.1

graphs. In the next section, it will be shown that two control structures are equivalent if and only if they have identical reduced behavior graphs.

Both the behavior graph and the reduced behavior graph of an ACS have the same set of signal sequences as its signal graph. Hence they are valid representations for the behavior of an ACS. This result can be proved using the characterizations of firing sequences and signal sequences in Propositions 4.3 and 4.4. The details are left to the reader.

5. Equivalence. In this section, we develop an effective test for equivalence between control structures. We begin by showing that two control structures are equivalent if and only if their reduced behavior graphs are identical. Although a reduced behavior graph is infinite, it is completely determined by a finite subset of its edge set called a generating set. Two control structures can therefore be tested for equivalence by comparing certain subsets of these finite generating sets.

We will use the following definition of control structure equivalence.

DEFINITION. Two asynchronous control structures are *equivalent* if they have the same behavior.

Recall that the behavior of an ACS was defined as the set of all possible signal sequences generated by its signal graph (or equivalently, by its behavior graph). As a result, this definition imposes the constraint that two equivalent control systems have the same number of input links and the same number of output links. It further requires that the correspondence between the links of the two systems be established before the equivalence test is applied.

We now show that two asynchronous control structures are equivalent just in case they have identical reduced behavior graphs.

LEMMA 5.1. *Let x_i^j and y_k^l be two transitions of the reduced behavior graph $\bar{\bar{G}}_c$, with $j, l \geqq 0$. If there are no paths from x_i^j to y_k^l, then there is a firing sequence $\bar{w}y_k^l$ such that $\#(x_i^j|\bar{w}) = 0$.*

Proof. By Corollary 4.2, there is a firing sequence for $\bar{\bar{G}}_c$ of the form $\bar{w}y_k^l$. If $\#(x_i^j|\bar{w}) = 1$, then $\bar{w}$ can be written in the form $\bar{w} = \bar{w}_1 x_i^j \bar{w}_2$. Assume that there is a path from x_i^j to some transition that appears in $\bar{w}_2$. Then there is a transition z_p^q and two sequences of transitions $\bar{u}_1$ and $\bar{u}_2$ such that $\bar{w}_2 = \bar{u}_1 z_p^q \bar{u}_2$ and there is a path from x_i^j to z_p^q but no path from z_p^q to any transition that appears in $\bar{u}_2$. By hypothesis, there is no path from z_p^q to y_k^l. Using Proposition 4.2, it can be shown that $\bar{w}_1 x_i^j \bar{u}_1 \bar{u}_2 y_k^l$ is also a firing sequence for $\bar{\bar{G}}_c$. This procedure can be used repeatedly to remove all of the transitions from $\bar{w}_2$ that are the terminal endpoints of paths starting at x_i^j. Finally, it can be used to remove x_i^j. □

THEOREM 5.1. *Let* C1 *and* C2 *be two asynchronous control structures. Then* C1 *and* C2 *are equivalent if and only if* $\bar{\bar{P}}_{c1} = \bar{\bar{P}}_{c2}$.

Proof. If $\bar{\bar{P}}_{c1} = \bar{\bar{P}}_{c2}$, then C1 is clearly equivalent to C2. Therefore assume that $\bar{\bar{P}}_{c1} \neq \bar{\bar{P}}_{c2}$. Without loss of generality, let $(x_i^j, y_k^l) \in \bar{\bar{P}}_{c1} - \bar{\bar{P}}_{c2}$ and $j, l \geqq 0$. We shall show that there is a string $\bar{w} \in T_{c1}^*$ that is either a firing sequence for $\bar{\bar{G}}_{c1}$ or $\bar{\bar{G}}_{c2}$, but not both.

Case 1. There is no path from x_i^j to y_k^l in $\bar{\bar{G}}_{c2}$. Then there is a firing sequence wy_k^l such that $\#(x_i^j|\bar{w}) = 0$ (Lemma 5.1). But every firing sequence for $\bar{\bar{G}}_{c1}$ of the form $\bar{w}y_k^l$ has $\#(x_i^j|\bar{w}) = 1$ since $(x_i^j, y_k^l) \in \bar{\bar{P}}_{c1}$ (Proposition 4.2). Hence $\bar{w}y_k^l$ is not a firing sequence for $\bar{\bar{G}}_{c1}$.

Case 2. There is a path π from x_i^j to y_k^l in $\bar{\bar{G}}_{c2}$. Note that since $(x_i^j, y_k^l) \notin \bar{\bar{P}}_{c2}$, the length of π is greater than 1. Since $\bar{\bar{G}}_{c1}$ is reduced, there is no such path in $\bar{\bar{G}}_{c1}$. Let z_p^q be an inner vertex of π. Then either there is no path from z_p^q to y_k^l or there is no path from x_i^j to z_p^q in $\bar{\bar{G}}_{c1}$. If there is no path from z_p^q to y_k^l, then there is a firing sequence $\bar{w}y_k^l$ for $\bar{\bar{G}}_{c1}$ with $\#(z_p^q|\bar{w}) = 0$. Similarly, if there is no path from x_i^j to z_p^q, then there is a firing sequence $\bar{u}z_p^q$ for $\bar{\bar{G}}_{c1}$ such that $\#(x_i^j|\bar{u}) = 0$. As in Case 1, $\bar{w}y_k^l$ and $\bar{u}z_p^q$ are not firing sequences for $\bar{\bar{G}}_{c2}$.

In both cases, we can find a string of transitions that is a firing sequence for one of the behavior graphs but not the other. This implies that C1 and C2 are not equivalent since there is a one-to-one correspondence between firing sequences and signal sequences of a behavior graph. □

Although Theorem 5.1 characterizes equivalent control structures, it does not provide a finite test since it requires the comparison of two infinite graphs. In the remainder of this section we show that it is only necessary to compare a finite section of these graphs. We first introduce the concept of a generating set for behavior graphs.

DEFINITION. Let $C = (G, \alpha, L)$ be an ACS, $\mathbf{w}$ an L-tuple in N^L,[9] and let $\bar{P}_c/\mathbf{w}$ denote the set $\{(x_i^j, y_k^l) \in \bar{P}_c | 0 \leqq j < w_i\}$. Then the *subset generated* by $\bar{P}_c/\mathbf{w}$ is defined by

$$\langle \bar{P}_c/\mathbf{w} \rangle = \{(x_i^{j+nw_i}, y_k^{l+nw_k}) | (x_i^j, y_k^l) \in \bar{P}_c/\mathbf{w} \text{ and } n \in Z\}.$$

$\bar{P}_c/\mathbf{w}$ is said to be a *generating set* if $\langle \bar{P}_c/\mathbf{w} \rangle = \bar{P}_c$.

Thus we see that if $\bar{P}_c/\mathbf{w}$ is a generating set, then all of the edges in $\bar{P}_c$ can be obtained from the edges in $\bar{P}_c/\mathbf{w}$ by incrementing and decrementing the superscripts of their endpoints by a fixed amount. The following result shows that all paths in $\bar{P}_c$ can be generated, in a similar way, from paths whose initial edge is in $\bar{P}_c/\mathbf{w}$.

PROPOSITION 5.1. *Let $C = (G, \alpha, L)$ be an ACS and $\mathbf{w} \in N^L$ such that $\bar{P}_c/\mathbf{w}$ is a generating set for $\bar{P}_c$. Then there is a path of length n from x_i^j to y_k^l in $\bar{G}_c$ if and only if there is a path of length n from $x_i^{R[j/w_i]}$ to $y_k^{l-Q[j/w_i]w_k}$ in $\bar{G}_c$.*

Proof. The proof is by induction on n.

Step 1. $n = 1$. In this case the proposition follows easily from the definition of a generating set.

Step 2. $n > 1$. Assume the proposition holds for all paths of length n. There is a path π of length $n + 1$ from x_i^j to y_k^l if and only if there is a path π_1 of length n from x_i^j to z_p^q and a path π_2 of length 1 from z_p^q to y_k^l, for some $z_p^q \in \bar{T}_c$. But, by the induction hypothesis, π_1 exists if and only if there is a path π_1' of length n from $x_i^{R[j/w_i]}$ to $z_p^{q-Q[j/w_i]w_p}$. It can be easily seen that π_2 exists if and only if there is a path π_2' of length 1 from $z_p^{q-Q[j/w_i]w_p}$ to $y_k^{l-Q[j/w_i]w_k}$. Finally, π_1' and π_2' exist if and only if there is a path π' of length $n + 1$ from $x_i^{R[j/w_i]}$ to $y_k^{l-Q[j/w_i]w_k}$. □

The following corollary identifies a simple and useful generating set for any behavior graph.

COROLLARY 5.1. *Let $C = (G, \alpha, L)$ be an ACS and let $\mathbf{w} \in N^L$ such that $w_i = \hat{\imath}$ for $i = 1, 2, \cdots, L$. Then $\bar{P}_c/\mathbf{w}$ is a generating set for $\bar{P}_c$.*

Proof. Clearly $\langle \bar{P}_c/\mathbf{w} \rangle \subseteq \bar{P}_c$. Let $(x_i^j, y_k^l) \in \bar{P}_c$. Then $(x_i^{R[j/\hat{\imath}]}, y_k^{R[l/\hat{k}]}, Q[l/\hat{k}] - Q[j/\hat{\imath}]) \in \gamma_c$ by Proposition 4.1. But this implies that

$$(x_i^{R[j/\hat{\imath}]}, y_k^{R[l/\hat{k}] + (Q[l/\hat{k}] - Q[j/\hat{\imath}])\hat{k}}) \in \bar{P}_c/\mathbf{w}.$$

Hence $(x_i^{R[j/\hat{\imath}]}, y_k^{l-Q[j/\hat{\imath}]\hat{k}}) \in \bar{P}_c/\mathbf{w}$ so that $(x_i^j, y_k^l) \in \langle \bar{P}_c/\mathbf{w} \rangle$ by Proposition 5.1. □

We now extend the definition of generating sets to reduced behavior graphs as follows.

DEFINITION. Let $C = (G, \alpha, L)$ be an ACS and $\mathbf{w} \in N^L$. Then $\bar{\bar{P}}_c/\mathbf{w} = \bar{P}_c/\mathbf{w} \cap \bar{\bar{P}}_c$. As before, $\bar{\bar{P}}_c/\mathbf{w}$ is said to be a generating set for $\bar{\bar{P}}_c$ if $\langle \bar{\bar{P}}_c/\mathbf{w} \rangle = \bar{\bar{P}}_c$.

This definition leads immediately to the following corollary to Proposition 5.1.

COROLLARY 5.2. *If $\bar{P}_c/\mathbf{w}$ is a generating set for $\bar{P}_c$, then $\bar{\bar{P}}_c/\mathbf{w}$ is a generating set for $\bar{\bar{P}}_c$.*

When $w_i = \hat{\imath}$, for $i = 1, 2, \cdots, L$, then we call the generating sets $\bar{P}_c/\mathbf{w}$ and $\bar{\bar{P}}_c/\mathbf{w}$ the *standard generating sets* for $\bar{P}_c$ and $\bar{\bar{P}}_c$ respectively. Figure 5.1 shows the standard generating set for the behavior graph of Fig. 4.1. If the broken edges are

[9] N denotes the set of natural numbers $\{1, 2, 3, \cdots\}$ and N^L is the L-fold Cartesian product of N with itself.

removed, then the standard generating set for the reduced behavior graph of Fig. 4.2 is obtained. Figure 5.1 also illustrates that a behavior graph is essentially a repetition of the pattern given by any one of its generating sets.

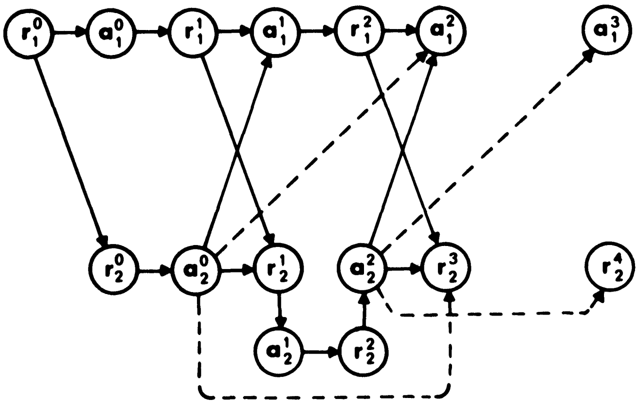


FIG. 5.1. *Standard generating sets for the ACS of Fig.* 3.1

The following lemma will be used to test a subset of a generating set to see if it is also a generating set for $\overline{\overline{P}}_c$.

LEMMA 5.2. *Let $C = (G, \alpha, L)$ be an ACS and $\overline{\overline{P}}_c/\mathbf{u}$ a generating set for $\overline{\overline{P}}_c$. Let $\mathbf{w}$ be an element of N^L and p an element of N, such that $u_i = pw_i$ for $i = 1, 2, \cdots, L$. Then $\overline{\overline{P}}_c/\mathbf{w}$ is a generating set for $\overline{\overline{P}}_c$ if and only if*

$$\overline{\overline{P}}^c/\mathbf{u} = \{(x_i^{j+nw_i}, y_k^{l+nw_k}) | (x_i^j, y_k^l) \in \overline{\overline{P}}_c/\mathbf{w} \text{ and } 0 \leqq n < p\}.$$

Proof. Let X denote the set

$$\{(x_i^{j+nw_i}, y_k^{l+nw_k}) | (x_i^j, y_k^l) \in \overline{\overline{P}}_c/\mathbf{w} \text{ and } 0 \leqq n < p\}.$$

We first assume that $\overline{\overline{P}}_c/\mathbf{w}$ is a generating set for $\overline{\overline{P}}_c$, and show that $\overline{\overline{P}}_c/\mathbf{u} = X$. If $(x_i^r, y_k^s) \in X$, then $0 \leqq r < w_i + (p-1)w_i = pw_i = u_i$. Hence X is a subset of $\overline{\overline{P}}_c/\mathbf{u}$. To show that $\overline{\overline{P}}_c/\mathbf{u} \subseteq X$, let $(x_i^r, y_k^s) \in \overline{\overline{P}}_c/\mathbf{u}$. Since $\overline{\overline{P}}_c/\mathbf{u} \subseteq \overline{\overline{P}}_c$ and $\overline{\overline{P}}_c/\mathbf{w}$ generates $\overline{\overline{P}}_c$, there is an edge $(x_i^j, y_k^l) \in \overline{\overline{P}}_c/\mathbf{w}$ such that $r = j + nw_i$ and $s = l + nw_k$, for some $n \in Z$. But $0 \leqq j < w_i$ and $0 \leqq r < u_i = pw_i$, so that $0 \leqq n < p$. Hence $(x_i^r, y_k^s) \in X$.

To prove the converse, assume that $\overline{\overline{P}}_c/\mathbf{u} = X$. We shall show that $\langle \overline{\overline{P}}_c/\mathbf{w} \rangle = \overline{\overline{P}}_c$. Clearly $X \subseteq \langle \overline{\overline{P}}_c/\mathbf{w} \rangle$, so that $\overline{\overline{P}}_c/\mathbf{u} \subseteq \langle \overline{\overline{P}}_c/\mathbf{w} \rangle$. Since $\overline{\overline{P}}_c/\mathbf{u}$ generates $\overline{\overline{P}}_c$ and $u_i = pw_i$, for $i = 1, 2, \cdots, L$, we have that $\overline{\overline{P}}_c \subseteq \langle \overline{\overline{P}}_c/\mathbf{w} \rangle$. To show that $\langle \overline{\overline{P}}_c/\mathbf{w} \rangle \subseteq \overline{\overline{P}}_c$, let $(x_i^r, y_k^s) \in \langle \overline{\overline{P}}_c/\mathbf{w} \rangle$. Then there are integers j, l and n such that $r = j + nw_i$, $s = l + nw_k$, and $(x_i^j, y_k^l) \in \overline{\overline{P}}_c/w$. Now

$$j + nw_i = j + (R[n/p] + Q[n/p]p)w_i = j + R[n/p]w_i + Q[n/p]u_i.$$

Similarly,

$$l + nw_k = l + R[n/p]w_k + Q[n/p]u_k.$$

But

$$(x_i^{j+R[n/p]w_i}, y_k^{l+R[n/p]w_k}) \in X \quad \text{and} \quad X = \overline{\overline{P}}_c/\mathbf{u}.$$

Hence

$$(x_i^r, y_k^s) \in \langle \overline{\overline{P}}_c/\mathbf{u} \rangle = \overline{\overline{P}}_c. \qquad \square$$

In other words, Lemma 5.2 shows that a set $\overline{\overline{P}}_c/\mathbf{w}$ is a generating set for $\overline{\overline{P}}_c$ if there is a known generating set $\overline{\overline{P}}_c/\mathbf{u}$, with $\mathbf{u} = p\mathbf{w}$ for some $p \in N$, such that $\overline{\overline{P}}_c/\mathbf{w}$ "generates" $\overline{\overline{P}}_c/\mathbf{u}$.

As an example of the use of Lemma 5.2, consider the ACS given in Fig. 5.2.

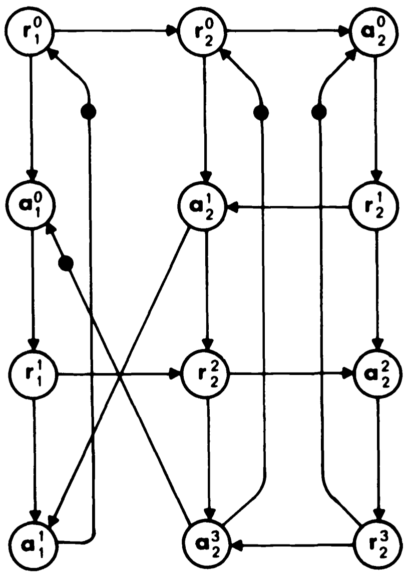


FIG. 5.2. *The ACS* C1

Let $\mathbf{u} = (2, 4)$ and $\mathbf{w} = (1, 2)$. Then $\overline{\overline{P}}_{c1}/\mathbf{u}$ is the standard generating set for $\overline{\overline{P}}_{c1}$ and $\mathbf{u} = 2\mathbf{w}$. The sets $\overline{\overline{P}}_{c1}/\mathbf{u}$ and $\overline{\overline{P}}_{c1}/\mathbf{w}$ are specified as follows:

$$\overline{\overline{P}}_{c1}/\mathbf{u} = \left\{ \begin{array}{l} (r_1^0, a_1^0), (a_1^0, r_1^1), (r_1^0, r_2^0), (r_2^0, a_2^0), \\ (a_2^0, r_2^1), (r_2^1, a_2^1), (a_2^1, r_2^2), (a_2^1, a_1^1), \\ (r_1^1, a_1^1), (a_1^1, r_1^2), (r_1^1, r_2^2), (r_2^2, a_2^2), \\ (a_2^2, r_2^3), (r_2^3, a_2^3), (a_2^3, r_2^4), (a_2^3, a_1^2) \end{array} \right\},$$

$$\overline{\overline{P}}_{c1}/\mathbf{w} = \left\{ \begin{array}{l} (r_1^0, a_1^0), (a_1^0, r_1^1), (r_1^0, r_2^0), (r_2^0, a_2^0), \\ (a_2^0, r_2^1), (r_2^1, a_2^1), (a_2^1, r_2^2), (a_2^1, a_1^1) \end{array} \right\}.$$

The set $\{(x_i^{j+n}, y_k^{l+2n}) | (x_i^j, y_k^l) \in \overline{\overline{P}}_{c1}/\mathbf{w} \text{ and } 0 \leqq n < 2\}$ is equal to

$$\overline{\overline{P}}_{c1}/\mathbf{w} \cup \left\{ \begin{array}{l} (r_1^{0+1}, a_1^{0+1}), (a_1^{0+1}, r_1^{1+1}), (r_1^{0+1}, r_2^{0+2}), (r_2^{0+2}, a_2^{0+2}), \\ (a_2^{0+2}, r_2^{1+2}), (r_2^{1+2}, a_2^{1+2}), (a_2^{1+2}, r_2^{2+2}), (a_2^{1+2}, a_1^{1+1}) \end{array} \right\}.$$

But this set is equal to the standard generating set $\overline{\overline{P}}_{c1}/\mathbf{u}$, so that $\overline{\overline{P}}_{c1}/\mathbf{w}$ is also a generating set for $\overline{\overline{P}}_{c1}$.

We can now characterize equivalent control structures in terms of generating sets. Since it is always possible to find a finite generating set for an ACS (Corollary 5.1), this characterization provides an effective test for equivalence. We first introduce the following notation.

NOTATION. Let C1 $= (G_1, \alpha_1, L)$ and C2 $= (G_2, \alpha_2, L)$ be two asynchronous control structures with the same number of links. Then

(a) $\hat{\imath}_1 = |\alpha_1^{-1}(i)|$, (b) $\hat{\imath}_2 = |\alpha_2^{-1}(i)|$,

(c) $\hat{\imath} = \text{g.c.d.}(\hat{\imath}_1, \hat{\imath}_2)$,

(d) $i_1 = Q[\hat{\imath}_1/\hat{\imath}]$, and (e) $i_2 = Q[\hat{\imath}_2/\hat{\imath}]$

for $i = 1, 2, \cdots, L$. If C1 is the ACS in Fig. 5.2 and C2 the one in Fig. 5.3, then

(a) $\hat{1}_1 = 2$, (b) $\hat{1}_2 = 3$,
$\hat{2}_1 = 4$; $\hat{2}_2 = 6$;

(c) $\hat{1} = 1$,
$\hat{2} = 2$;

(d) $1_1 = 2$, (e) $1_2 = 3$,
$2_1 = 2$; $2_2 = 3$.

For the control structures in Figs. 5.2 and 5.3, we see that $i_1 = j_1$ and $i_2 = j_2$, for $0 \leqq i, j \leqq L$. The next result shows that this relationship always holds when the two control structures are equivalent.

LEMMA 5.3. *Let* C1 *and* C2 *be two equivalent control structures with L links. Then* $i_1 = j_1$ *and* $i_2 = j_2$ *for* $1 \leqq i, j \leqq L$.

Proof. Let l be the least integer such that there is a path π_1 from $x_i^{\hat{\imath}_1\hat{\imath}_2}$ to y_k^l in $\bar{\bar{G}}_{c1}$. The existence of π_1 is assured by Corollary 4.1. Since $\bar{\bar{P}}_{c1} = \bar{\bar{P}}_{c2}$, l must also be the least such integer for $\bar{\bar{G}}_{c2}$. The corresponding path in $\bar{\bar{G}}_{c2}$ is denoted by π_2. By Proposition 5.1, the existence of π_1 and π_2 imply that there is a path from x_i^0 to $y_k^{l-\hat{\imath}_2\hat{k}_1}$ in $\bar{\bar{G}}_{c1}$ and a path from x_i^0 to $y_k^{l-\hat{\imath}_1\hat{k}_2}$ in $\bar{\bar{G}}_{c2}$. Moreover, $l' = l - \hat{\imath}_2\hat{k}_1$ and $l'' = l - \hat{\imath}_1\hat{k}_2$ are the least such integers for $\bar{\bar{G}}_{c1}$ and $\bar{\bar{G}}_{c2}$. Since $\bar{\bar{P}}_{c1} = \bar{\bar{P}}_{c2}$, we have that $l' = l''$. Hence $\hat{\imath}_1\hat{k}_2 = \hat{\imath}_2\hat{k}_1$. But this implies that $i_1 j_2 = j_1 i_2$. Since g.c.d. $(i_1, i_2) = 1$ and g.c.d. $(j_1, j_2) = 1$, we have $i_1 = j_1$, and $i_2 = j_2$. □

We can now state and prove the main result of the paper which characterizes equivalent control structures in terms of a common finite generating set.

THEOREM 5.2. *Let* C1 $= (G_1, \alpha_1, L)$ *and* C2 $= (G_2, \alpha_2, L)$ *be two asynchronous control structures, and let* $\mathbf{w} \in N^L$ *such that* $w_i = \hat{\imath} = \text{g.c.d.}(\hat{\imath}_1, \hat{\imath}_2)$ *for* $i = 1, 2, \cdots, L$. *Then* C1 *and* C2 *are equivalent if and only if* $\bar{\bar{P}}_{c1}/\mathbf{w}$ (*or equivalently* $\bar{\bar{P}}_{c2}/\mathbf{w}$) *is a generating set for both* $\bar{\bar{P}}_{c1}$ *and* $\bar{\bar{P}}_{c2}$.

Proof. If $\bar{\bar{P}}_{c1}/\mathbf{w}$ is a generating set for both $\bar{\bar{P}}_{c1}$ and $\bar{\bar{P}}_{c2}$, then clearly $\bar{\bar{P}}_{c1} = \bar{\bar{P}}_{c2}$ and C1 is equivalent to C2 by Theorem 5.1. To prove the converse assume that $\bar{\bar{P}}_{c1} = \bar{\bar{P}}_{c2}$. We shall use Lemma 5.2 to show that $\bar{\bar{P}}_{c1}/\mathbf{w}$ is a generating set for $\bar{\bar{P}}_{c1}$ and $\bar{\bar{P}}_{c2}$. To this end, let X denote the set

$$\{(x_i^{j+nw_i}, y_k^{l+nw_k}) | (x_i^j, y_k^l) \in \bar{\bar{P}}_{c1}/\mathbf{w} \text{ and } 0 \leqq n < i_1\}$$

and $\mathbf{u}$ be the element of N^L defined by $u_i = \hat{\imath}_1$, for $i = 1, 2, \cdots, L$. We must show that $\bar{\bar{P}}_{c1}/\mathbf{u} = X$.

Let $(x_i^j, y_k^l) \in \bar{\bar{P}}_{c1}/\mathbf{u}$. Then $(x_i^j, y_k^l) \in \bar{\bar{P}}_{c1}$ and $0 \leqq j < \hat{\imath}_1$. Let a and b be two integers such that $ai_1 + bi_2 = \text{g.c.d.}(i_1, i_2) = 1$ and let $q = -Q[j/\hat{\imath}]\hat{\imath}$. By Proposition 5.1, the edge $(x_i^{j+aq\hat{\imath}_1}, y_k^{l+aq\hat{k}_1})$ is in $\bar{\bar{P}}_{c1}$. Since $\bar{\bar{P}}_{c1} = \bar{\bar{P}}_{c2}$, $(x_i^{j+aq\hat{\imath}_1+bq\hat{\imath}_2},$

$y_k^{l+aq\hat{k}_1+bq\hat{k}_2})$ is in both $\overline{\overline{P}}_{c1}$ and $\overline{\overline{P}}_{c2}$. But

$$\begin{aligned} j + aq\hat{\imath}_1 + bq\hat{\imath}_2 &= j + q\hat{\imath}(ai_1 + bi_2) \\ &= j + q\hat{\imath} \\ &= j - Q[j/\hat{\imath}]\hat{\imath} \\ &= R[j/\hat{\imath}] \\ &< w_i. \end{aligned}$$

Similarly,

$$\begin{aligned} l + aq\hat{k}_1 + bq\hat{k}_2 &= l + q\hat{k}(ak_1 + bk_2) \\ &= l + q\hat{k}(ai_1 + bi_2) \quad \text{(by Lemma 5.3)} \\ &= l - Q[j/\hat{\imath}]w_k. \end{aligned}$$

Hence $(x_i^{R[j/\hat{\imath}]}, y_k^{l-Q[j/\hat{\imath}]w_k}) \in \overline{\overline{P}}_{c1}/\mathbf{w}$. But $Q[j/\hat{\imath}] < i_1$, since $j < \hat{\imath}_1$. Also, $R[j/\hat{\imath}] + Q[j/\hat{\imath}]w_i = j$ and $l - Q[j/\hat{\imath}]w_k + Q[j/\hat{\imath}]w_k = l$. Therefore $(x_i^j, y_k^l) \in X$ so that $\overline{\overline{P}}_{c1}/\mathbf{u} \subseteq X$.

Let $(x_i^j, y_k^l) \in X$. Then there are integers r, s and n such that $0 \leqq n < i_1$, $j = r + n\hat{\imath}$, $l = s + n\hat{k}$, and $(x_i^r, y_k^s) \in \overline{\overline{P}}_{c1}/\mathbf{w}$. Let a and b be two integers such that $ai_1 + bi_2 = \text{g.c.d.}(i_1, i_2) = 1$. Since $\overline{\overline{P}}_{c1} = \overline{\overline{P}}_{c2}$, then

$$(x_i^{r+na\hat{\imath}_1+nb\hat{\imath}_2}, y_k^{s+na\hat{k}_1+nb\hat{k}_2}) \in \overline{\overline{P}}_{c1}.$$

But $r + na\hat{\imath}_1 + nb\hat{\imath}_2 = j$ and $s + na\hat{k}_1 + nb\hat{k}_2 = l$. Hence $(x_i^j, y_k^l) \in \overline{\overline{P}}_{c1}$. Since $0 \leqq n < i_1$, we have $0 \leqq j < u_i$. Therefore $(x_i^j, y_k^l) \in \overline{\overline{P}}_{c1}/\mathbf{u}$, so that $X \subseteq \overline{\overline{P}}_{c1}/\mathbf{u}$. □

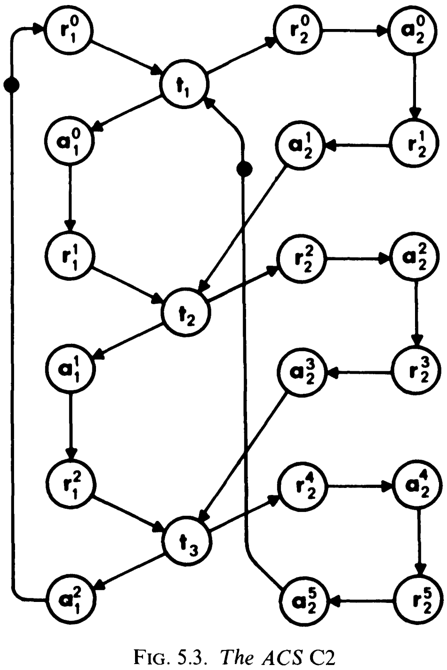


FIG. 5.3. *The ACS* C2

In summary, two asynchronous control structures $C1 = (G_1, \alpha_1, L)$ and $C2 = (G_2, \alpha_2, L)$ are equivalent if and only if

(1) $i_{\underline{1}} = j_1$ and $i_2 = j_2$ for $1 \leqq i, j \leqq \underline{L}$, and

(2) $\overline{\overline{P}}_{c1}/\mathbf{w}$ is a generating set for both $\overline{\overline{P}}_{c1}$ and $\overline{\overline{P}}_{c2}$, where $w_{\underline{i}} = \hat{\imath}$ for $1 \leqq i \leqq L$.

Lemma 5.2 can be used to determine whether or not the set $\overline{\overline{P}}_{c1}/\mathbf{w}$ is a generating set for $\overline{\overline{P}}_{c1}$ and $\overline{\overline{P}}_{c2}$. Hence the design of an algorithm to check for the equivalence of two control structures, using conditions (1) and (2) above, is straightforward. To illustrate this procedure, consider the control structures in Figs. 5.2 and 5.3. We have already seen that condition (1) is satisfied and that $\overline{\overline{P}}_{c1}/\mathbf{w}$ is a generating set for $\overline{\overline{P}}_{c1}$. Lemma 5.2 can be used to show that $\overline{\overline{P}}_{c1}/\mathbf{w}$ also generates $\overline{\overline{P}}_{c2}$. Hence the two control structures are equivalent.

6. Summary. In this paper we have modeled a class of asynchronous control systems with marked graphs. By means of this model, the equivalence problem has been formulated and solved. In the process, two different marked graph representations, the signal graph and the behavior graph, were introduced. In this final section, we compare these two models with previous work in this field.

The signal graph is closely related to the models proposed by Dennis [6] and Patil [15]. In fact, the method of representing the generation of link signals in the signal graph is a combination of the techniques used in these two papers. Although these two models are more general, since conflict can be explicitly represented, the signal graph is more amenable to analysis. On the other hand, by allowing many-to-one link assignments, certain types of pseudo-conflicts can be represented with a signal graph. For example, well-formed networks composed of WYE, SEQUENCE, JUNCTION, UNION, and Muller's SWITCH modules [6], [3], [14] can be modeled if the UNION condition [1] is enforced by the network itself.

Ordering relations similar to the one represented by a behavior graph have been studied by Muller [12], [13] and Holt [7]. Indeed, the Hasse diagram of the partially ordered set of C-states, used by Muller to investigate asynchronous switching circuits, is constructed in much the same way as the behavior graph. The major difference is due to the intended application of the two concepts. Muller used the diagram of C-states to characterize the different types of speed-independent circuits where the elements of the circuit were arbitrary logical elements. Moreover, every element is represented by a component of the C-state and the emphasis is on sequences of states rather than input and output signals. Since the behavior diagram has been developed to characterize system equivalence, as seen by the environment of the system, only sequences of link signals are explicitly represented.

Holt, on the other hand, models concurrent systems as infinite collections of occurrence graphs called occurrence systems. For a certain class of systems he derives the Petri net as a finite representation of the occurrence system. Obtaining the behavior graph from the signal graph may be viewed as the inverse of this process. Indeed, with suitable interpretations, a generating set of a behavior graph may be considered as an o-cycle and the corresponding control system as an occurrence system which is the cd-closure of this o-cycle.

Finally, the behavior graph can be viewed as a formalization and generalization of the p-nets and section graphs used by Dennis [6], Patil [15], and Bruno and

Altman [3] to specify the behavior of the asynchronous control modules. In fact, a nonconditional module can be modeled as an ACS. The graph of a minimal generating set of the corresponding behavior graph would closely resemble the p-net for that module. Certain networks of these modules can also be modeled with an ACS. In this case, a generating set for the behavior graph can be considered as an extended p-net representation for the network.

REFERENCES

[1] S. M. ALTMAN AND P. J. DENNING, *Decomposition of control networks*, Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York, 1970, pp. 81–92.

[2] J. L. BAER, D. P. BOVER AND G. ESTRIN, *Legality and other properties of graph models of computation*, J. Assoc. Comput. Mach., 17 (1970), pp. 543–554.

[3] J. BRUNO AND S. M. ALTMAN, *A theory of asynchronous control networks*, IEEE Trans. Computers, C-20 (1971), pp. 629–638.

[4] V. CERF, E. FERNANDEZ, K. GOSTELOW AND S. VOLANSKY, *Formal controlflow properties of a model of computation*, Tech. Rep. UCLA-10P14-105, Computer Science Department, University of California, Los Angeles, 1971.

[5] J. B. DENNIS, *Computation structures*, notes for subject 6.232, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass., 1970.

[6] ———, *Modular asynchronous control structures for a high performance processor*, Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York, 1970, pp. 55–80.

[7] A. W. HOLT, R. SHAPIRO, H. SAINT AND S. WARSHALL, *Information system theory project*, Tech. Rep. for Rome Air Development Center, RADC-68-305, prepared by Applied Data Research, Princeton, N.J., 1968.

[8] A. W. HOLT AND F. COMMONER, *Events and conditions*, Information Theory Project, Research Rep. of Applied Data Research, Princeton, N.J., 1970.

[9] A. W. HOLT, F. COMMONER, S. EVEN AND A. PNUELI, *Marked directed graphs*, J. Comput. System Sci., 5 (1971), pp. 511–523.

[10] R. M. KARP AND R. E. MILLER, *Parallel program schemata*, Ibid., 3 (1969), pp. 147–195.

[11] T. L. LUCONI, *Output functional computational structures*, IEEE Ninth Annual Symposium on Switching and Automata Theory, IEEE, New York, 1968, pp. 76–84.

[12] R. E. MILLER, *Speed independent switching circuit theory*, Switching Theory, vol. II, John Wiley, New York, 1965, Chap. 10.

[13] D. E. MULLER AND W. S. BARTKY, *A theory of asynchronous circuits*, Proc. International Symposium on the Theory of Switching, vol. 29 of the Annals of the Computation Laboratory of Harvard University, Harvard University Press, Cambridge, Mass., 1959, pp. 204–243.

[14] D. E. MULLER, *Asynchronous logics and applications to information processing*, Switching Theory in Space Technology, H. Aiken and W. F. Main, eds., Stanford University Press, Stanford, Calif., 1963, pp. 289–297.

[15] S. S. PATIL, *Coordination of asynchronous events*, Ph.D. thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass., 1970.

[16] J. E. RODRIGUEZ, *A graph model for parallel computation*, Ph.D. thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass., 1967.

[17] D. R. SLUTZ, *The flow graph schemata model of parallel computation*, Ph.D. thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass., 1968.

SIAM J. COMPUT.
Vol. 2, No. 2, June 1973

GENETIC ALGORITHMS AND THE OPTIMAL ALLOCATION OF TRIALS*

JOHN H. HOLLAND†

Abstract. This study gives a formal setting to the difficult optimization problems characterized by the conjunction of (1) substantial complexity and initial uncertainty, (2) the necessity of acquiring new information rapidly to reduce the uncertainty, and (3) a requirement that the new information be exploited as acquired so that average performance increases at a rate consistent with the rate of acquisition of information. The setting has as its basis a set $\mathscr{A}$ of structures to be searched or tried and a performance function $\mu:\mathscr{A} \to$ real numbers. Within this setting it is determined how to allocate trials to a set of random variables so as to maximize expected performance. This result is then transformed into a criterion against which to measure the performance of a robust and easily implemented set of algorithms called *reproductive plans.* It is shown that reproductive plans can in fact surpass the criterion because of a phenomenon called *intrinsic parallelism*—a single trial (individual $A \in \mathscr{A}$) simultaneously tests and exploits many random variables.

1. Introduction. There is an extensive and difficult class of optimization problems characterized by:

(1) substantial complexity and initial uncertainty;
(2) the necessity of acquiring new information rapidly to reduce the uncertainty;
(3) a requirement that the new information be exploited as acquired so that average performance increases at a rate consistent with the rate of acquisition of information.

These problems derive from a whole range of long-standing questions, such as the following.

How is the productivity of a plant or process to be improved, while it is operating, when many of the interactions between its variables are unknown?

How does one improve performance in successive plays of a complex game (such as Chess or Go or a management game) when the solution of the game is unknown (and probably too complex to implement even if it were known)?

How does evolution produce increasingly fit organisms under environmental conditions which perforce involve a great deal of uncertainty vis-à-vis individual organisms?

How can the performance of an economy be upgraded when its mechanisms are only partially known and relevant data is incomplete?

In each case rapid improvement in performance is highly desirable (or essential), though the combination of complexity and uncertainty makes a direct approach to optimization unfeasible. Often the complexity and uncertainty are great enough that the optimum will be attained, if at all, only after an extensive

* Received by the editors August 3, 1972.

† Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, Michigan 48104. This research was supported in part by the National Science Foundation under Grant GJ-29989X.

period of trial and calculation. Problems of this kind will be referred to here as "problems of adaptation," a usage similar to that of Tsypkin [8], though broader.

Problems of adaptation can be given a more precise formulation along the following lines. Let $\mathscr{A}$ be the set of objects or *structures* (control policies, game strategies, chromosomes, mixes of goods, etc.) to be searched or tried. Generally $\mathscr{A}$ will be so large that it cannot be tried one at a time over any feasible time period. Let $\mu : \mathscr{A} \to [r_0, r_1]$, where $[r_0, r_1]$ is an interval of real numbers, be a "performance measure" (error, payoff, fitness, utility, etc.) which assigns a level of *performance* $\mu(A)$ to each structure $A \in \mathscr{A}$. Conditions (2) and (3) above then reduce to:

(2′) Obtain new information about μ by trying previously untried structures in $\mathscr{A}$ (it being assumed that the outcome of trying $A \in \mathscr{A}$ is the information $\mu(A)$).

(3′) Assure that $\bar{\mu}_t$, the average of the outcomes of the first t trials, increases rapidly whenever the search of $\mathscr{A}$ reveals an A with $\mu(A) > \bar{\mu}_t$.

With no more formalization than this, a dilemma comes into sharp focus. The rate of search is maximized when each successive trial is of a previously untried $A \in \mathscr{A}$. On the other hand, repeated trials of any A' for which $\mu(A') > \bar{\mu}_t$ will increase $\bar{\mu}_t$ more rapidly. In other words, if the rate of search is maximized, the information cannot be exploited, whereas if information obtained is maximally exploited, no new information is acquired.

The basic problem then is to find a resolution of this dilemma. The simplest precise version of the dilemma arises when we restrict our attention to two random variables ξ_1, ξ_2, defined so that the outcome of a trial of ξ_i is a performance $\mu(\xi_i, t)$. The object then is to discover a procedure for distributing some arbitrary number of trials, N, between ξ_1 and ξ_2 so as to maximize the expected payoff over the N trials. If for each ξ_i we know the mean and variance (μ_i, σ_i) of its distribution, the problem has a trivial solution (namely, allocate all trials to the random variable with maximal mean). The dilemma asserts itself, however, if we inject just a bit more uncertainty. Thus we can know the mean-variance pairs but not which variable is described by which pair; i.e., we know pairs (μ, σ) and (μ', σ') but not which pair describes ξ_1. (This is a version of the much studied 2-armed bandit problem, a prototype of important decision problems. See, for example, Bellman [2] and Hellman and Cover [4].) If it could be determined through observation which of ξ_1 and ξ_2 has the higher mean, then from that point on, all trials could be allocated to that random variable. Unfortunately, unless the distributions are nonoverlapping, no finite number of observations will establish *with certainty* which random variable has the higher mean. Here the tradeoff between gathering information and exploiting it appears in its simplest terms. Gathering information requires trials of *both* random variables, with a consequent decrement in average payoff (because the average performance of one of the random variables is less than maximal). On the other hand, premature exploitation of the apparent best random variable, by allocation of most or all trials thereto, runs the risk of a large loss (because there is a nonzero probability that the apparent best is really second best).

A procedure for allocating trials between ξ_1 and ξ_2 will be said to optimally satisfy the conditions (2′) and (3′) if it maximizes the expected performance over N trials. It should be noted that any increase in the uncertainty, such as not knowing some of the means or variances, can only result in a lower expected performance.

Thus a procedure which solves the given problem yields an upper bound on expected performance under increased uncertainty—a criterion against which to measure the performance of various feasible algorithms.

In these terms the objective of this paper is two-fold. First, making the natural extension to an arbitrary number r of random variables determine an upper bound on expected performance under uncertainty. Second, compare the expected performance of the general class of algorithms known as *reproductive plans* (see Holland [5] and later in this paper) to this criterion. It will be shown that, even under conditions of maximum uncertainty, reproductive plans closely follow the criterion. Moreover, reproductive plans can use a single trial to test many random variables simultaneously, a property designated *intrinsic parallelism.* Thus, reproductive plans can exceed the optimum for one-at-a-time testing of random variables. The latter part of the paper will show how this advantage is attained and that it increases in direct proportion to the number of random variables r.

2. Definition of the problem. Several definitions will have to be added to give precise mathematical form to the problem of searching $\mathscr{A}$ under conditions (1), (2′) and (3′). (This is only a partial formalization of problems of adaptation, sufficient for present purposes—a more complete formulation is given in Holland [5].) First of all, let the elements of $\mathscr{A}$ be represented by strings of length l over a set of symbols $\Sigma = \{\sigma_1, \cdots, \sigma_k\}$; i.e., each $A \in \mathscr{A}$ is represented by (or designates) a string of symbols (alleles, weights, etc.) $\sigma_{i_1}\sigma_{i_2} \cdots \sigma_{i_l}$, where $\sigma_{i_h} \in \Sigma$. For simplicity in what follows, $\mathscr{A}$ will simply be taken to *be* the set of strings (rather than the abstract elements represented by the strings). Let $\sigma_1\square\square \cdots \square$ designate the set of all elements of $\mathscr{A}$ beginning with the symbol σ_1. (For example, $\sigma_1\sigma_1\sigma_4$, $\sigma_1\sigma_2\sigma_1$, and $\sigma_1\sigma_3\sigma_3$ would belong to $\sigma_1\square\square$, but $\sigma_2\sigma_1\sigma_1$ would not.) More generally, let any string ξ of length l over the augmented set $\Sigma \cup \{\square\} = \{\sigma_1, \cdots, \sigma_k, \square\}$ designate a subset of $\mathscr{A}$ as follows: $A \in \mathscr{A}$ belongs to the subset designated by $\xi = \delta_{i_1}\delta_{i_2} \cdots \delta_{i_l}$ if and only if (i) whenever $\delta_{i_j} \in \Sigma$ the string A has the symbol δ_{i_j} at the jth position, and (ii) whenever $\delta_{i_j} = \square$ any symbol from Σ may occur at the jth position in A. (For example, the strings $\sigma_1\sigma_1\sigma_1\sigma_3$ and $\sigma_3\sigma_1\sigma_2\sigma_3$ belong to $\square\sigma_1\square\sigma_3$ but $\sigma_1\sigma_1\sigma_1\sigma_2$ does not.) The set of $(k + 1)^l$ strings defined over $\Sigma \cup \{\square\}$ will be called the set Ξ of *schemata*; they amount to a decomposition of $\mathscr{A}$ into a large number of subsets based on the representation in terms of Σ.

If now there is a probability distribution P over $\mathscr{A}$, say the probability $P(A)$ that $A \in \mathscr{A}$ will be selected for trial, $\mathscr{A}$ can be treated as a sample space and each schema ξ designates an event on $\mathscr{A}$. Accordingly, the performance measure μ becomes a random variable, the elementary event A occurring with probability $P(A)$ and yielding payoff $\mu(A)$. Moreover, the restriction $\mu|\xi$ of μ to a particular subset ξ is also a random variable, $A \in \xi$ being chosen with probability $(P(A))/(\sum_{A\in\xi} P(A))$ and yielding payoff $\mu(A)$. In what follows, ξ will be of interest only in its role of designating the random variable $\mu|\xi$; therefore ξ will be used to designate both an element of Ξ and the corresponding random variable with sample space ξ, distribution $(P(A))/(\sum_{A\in\xi} P(A))$, and values $\mu(A)$. As a random variable, ξ has a well-defined average μ_ξ and variance σ_ξ^2; intuitively, μ_ξ is the payoff expected when an element of ξ is randomly selected (under the distribution P).

Using the decomposition of $\mathscr{A}$ into random variables gives by P and Ξ, it is possible to formalize the earlier discussion concerning optimal allocation of trials.

For the 2-schemata case, let $n_{(2)}$ be the number of trials allocated to the schema with the lowest *observed* payoff rate at the end of N trials. Let $q(n_{(2)})$ be the probability that the schema with the highest observed payoff rate is actually second best. (In detail $q(n_{(2)})$ is actually a function of $n_{(2)}$, N, (μ, σ), and (μ', σ'); hence the necessity of knowing the mean-variance pairs.) If μ_1 is the mean of ξ_1 and μ_2 the mean of ξ_2, then the expected payoff rate for the allocation $(N - n_{(2)}, n_{(2)})$ is

$$\max(\mu_1, \mu_2) - [(N - n_{(2)})q(n_{(2)}) + n_{(2)}(1 - q(n_{(2)}))] \cdot |\mu_1 - \mu_2| \cdot \frac{1}{N}.$$

An optimum value for $n_{(2)}$ can be obtained from this expression by standard techniques. (The development below is somewhat more complicated since, in general, one cannot guarantee a priori that the random variable with the highest observed payoff rate at the *end* of the N trials will have received a predetermined number of trials by that time.)

Though the derivations are much more intricate, the extension from the 2-schemata case to the r-schemata case is conceptually straightforward. It is possible to obtain useful bounds on the payoff rate as a function of the total number of trials, N, together with bounds on the total number of trials which should be allocated to the schema with the highest observed payoff rate. Because the upper and lower bounds so obtained are close to one another (relative to N), the action of the optimal procedure is pretty clearly defined. Using this information it is possible to define a realizable procedure which approaches the optimum as N increases.

If the r-schemata must be tested one at a time, it is clear that one can do no better than the procedure just outlined. If, on the other hand, information about *several* schemata can be obtained from trial of a single individual, $A \in \mathscr{A}$, the rate of improvement could exceed the optimal rate for the one-schema-at-a-time procedure. It should be remarked at once that, for this improvement to take place, the information must not only be obtained but *used* to generate subsequent individuals ($A \in \mathscr{A}$) for trial—each of which will reveal further information about a variety of schemata. The second part of this paper studies a specific set of reproductive plans, the *genetic plans*, which can do just this. It will be shown that genetic plans follow the general course of the optimal procedure when artificially constrained to one-schema-at-a-time searches, but advance much more rapidly when not so constrained.

3. Optimal allocation of trials. For notational convenience in the 2-schemata case, let ξ_1 be the schema with highest mean, ξ_2 the schema with lowest mean. (The observer, of course, does not know this.) Let $\xi_{(1)}(\tau, N)$ be the schema with the highest *observed* payoff rate (average per trial) after an allocation of N trials according to plan τ; let $\xi_{(2)}(\tau, N)$ designate the schema with lowest *observed* rate. Note that for any number of trials n, $0 \leqq n \leqq N$, allocated to $\xi_{(2)}(\tau, N)$, there is a positive probability, $q(N - n, n)$, that $\xi_{(2)}(\tau, N) \neq \xi_2$ (assuming overlapping distributions). Equivalently, $q(N - n, n)$ is the probability that the observed best is actually second best.

THEOREM 1. *Given N trials to be allocated to two random variables ξ_1 and ξ_2, with means $\mu_1 > \mu_2$ and variances σ_1, σ_2 respectively, the minimum expected loss results when the number of trials allocated ξ_2 is*

$$n^* \sim \left(\frac{\sigma_2}{\mu_1 - \mu_2}\right)^2 \ln\left[\left(\frac{\mu_1 - \mu_2}{\sigma_2}\right)^4\left(\frac{N^2}{8\pi \ln N^2}\right)\right].$$

The corresponding expected loss per trial is

$$l^*(N) \sim \frac{\sigma_2^2}{(\mu_1 - \mu_2)N}\left[2 + \ln\left[\left(\frac{\mu_1 - \mu_2}{\sigma_2}\right)^4\left(\frac{N^2}{8\pi \ln N^2}\right)\right]\right].$$

Proof. (Given two arbitrary functions, $Y(t)$ and $Z(t)$, of the same variable t, "$Y(t) \sim Z(t)$" will be used to mean $\lim_{t\to\infty} (Y(t)/Z(t)) = 1$ while "$Y(t) \cong Z(t)$" means that under stated conditions the difference $Y(t) - Z(t)$ is negligible).

In determining the expected payoff rate of a plan τ over N trials, two possible sources of loss must be taken into account: (1) The *observed* best $\xi_{(1)}(\tau, N)$ is really second best, whence the $N - n$ trials given $\xi_{(1)}(\tau, N)$ incur an (expected) cumulative loss $|\mu_1 - \mu_2| \cdot (N - n)$; this occurs with probability $q(N - n, n)$. (2) The observed best is in fact the best, whence the n trials given $\xi_{(2)}(\tau, N)$ incur a loss $|\mu_1 - \mu_2| \cdot n$; this occurs with probability $q(N - n, n)$. The expected loss $l(N)$ over N trials is thus

$$|\mu_1 - \mu_2| \cdot [(N - n)q(N - n, n) + n(1 - q(N - n, n))].$$

In order to select an n which minimizes the expected loss, it is necessary first to write $q(N - n, n)$ as an explicit function of n. To derive this function let S_2 be the sum of the outcomes (payoffs) of n trials of ξ_2 and let S_1 be the corresponding sum for the $N - n$ trials of ξ_1. Then $q(N - n, n)$ is just the probability that $S_2/n < S_1/(N - n)$ or, equivalently, the probability that $S_1/(N - n) - S_2/n < 0$. By the central limit theorem, S_2/n approaches a normal distribution with mean μ_2 and variance σ_2^2/n; similarly, $S_1/(N - n)$ has mean μ_1 and variance $\sigma_1^2/(N - n)$. The distribution of $S_1/(N - n) - S_2/n$ is by definition the sum (convolution) of the distributions of $S_1/(N - n)$ and $-(S_2/n)$; by an elementary theorem (on the convolution of normal distributions), this is a normal distribution with mean $\mu_1 - \mu_2$ and variance $\sigma_1^2/(N - n) + \sigma_2^2/n$. Thus the probability $\Pr\{S_1/(N - n) - S_2/n < 0\}$ is the tail $1 - \Phi(x_0)$ of a normal distribution $\Phi(x)$ in standard form so that

$$x = \frac{y - (\mu_1 - \mu_2)}{\sqrt{\sigma_1^2/(N - n) + \sigma_2^2/n}}$$

and $-x_0$ is the value of x when $y = 0$.

The tail of a normal distribution is well approximated by

$$\Phi(-x) = 1 - \Phi(x) \lesssim \frac{1}{\sqrt{2\pi}} \cdot \frac{e^{-x^2/2}}{x}.$$

Thus

$$q(N-n,n) \lesssim \frac{1}{\sqrt{2\pi}} \cdot \frac{e^{-x_0^2/2}}{x_0}$$

$$= \frac{1}{\sqrt{2\pi}} \frac{\sqrt{\sigma_1^2/(N-n)+\sigma_2^2/n}}{\mu_1-\mu_2} \exp\frac{1}{2}\left[\frac{-(\mu_1-\mu_2)^2}{\sigma_1^2/(N-n)+\sigma_2^2/n}\right]$$

(from which we see that q is a function of the variances and means as well as the total number of trials, N, and the number of trials, n, given ξ_2). Upon noting that q decreases exponentially as a function of n, it becomes clear that, to minimize loss as N increases, the number of trials allocated the observed best, $N-n$, should be increased dramatically relative to n. This observation (which will be verified in detail shortly) enables us to simplify the expression for x_0. Whatever the value of σ_1, there will be an N_0 such that, for any $N > N_0$, $\sigma_1^2/(N-n) \ll \sigma_2^2/n$, for n close to its optimal value. (In most cases of interest this occurs even for small numbers of trials since, usually, σ_2 is at worst an order of magnitude or two larger than σ_1.) Using this we see that, for n close to its optimal value,

$$x_0 \lesssim \frac{(\mu_1-\mu_2)\sqrt{n}}{\sigma_2}, \qquad N > N_0.$$

We can now proceed to determine what value of n will minimize the loss $l(n)$ by taking the derivative of l with respect to n:

$$\frac{dl}{dn} = |\mu_1-\mu_2| \cdot \left[-q + (N-n)\frac{dq}{dn} + 1 - q - n\frac{dq}{dn}\right]$$

$$= |\mu_1-\mu_2| \cdot \left[(1-2q) + (N-2n)\frac{dq}{dn}\right],$$

where

$$\frac{dq}{dn} \lesssim \frac{1}{\sqrt{2\pi}}\left[-\frac{e^{-x_0^2/2}}{x_0^2} - e^{-x_0^2/2}\right]\frac{dx_0}{dn} = -\left[\frac{q}{x_0} + x_0 q\right]\frac{dx_0}{dn}$$

and

$$\frac{dx_0}{dn} \lesssim \frac{\mu_1-\mu_2}{2\sigma_2\sqrt{n}} = \frac{x_0}{2n}.$$

Thus

$$\frac{dl}{dn} \lesssim |\mu_1-\mu_2| \cdot \left[(1-2q) - (N-2n)\cdot q \cdot \frac{x_0^2+1}{2n}\right].$$

n^*, the optimal value of n, satisfies $dl/dn = 0$, whence we obtain a bound on n^* as follows:

$$0 \lesssim (1-2q) - \left(\frac{N}{2n^*} - 1\right) \cdot q \cdot (x_0^2+1)$$

or

$$\frac{N}{2n^*} - 1 \lesssim \frac{1-2q}{q\cdot(x_0^2+1)}.$$

Noting that $1/(x_0^2+1) \lesssim 1/x_0^2$ and that $1-2q$ rapidly approaches 1 because q decreases exponentially with n, we see that $(N-2n^*)/n^* \lesssim 2/(x_0^2 q)$, where the error rapidly approaches zero as N increases. Thus the observation of the preceding paragraph is verified, the ratio of trials of the observed best to trials of second-best growing exponentially.

Finally, to obtain n^* as an explicit function of N, q must be written in terms of n^*:

$$\frac{N-2n^*}{n^*} \lesssim \frac{2\sqrt{2\pi}\sigma_2}{\mu_1-\mu_2}\cdot\frac{1}{\sqrt{n^*}}\cdot\exp\left[\frac{(\mu_1-\mu_2)^2 n^*}{2\sigma_2^2}\right].$$

Introducing $b = (\mu_1-\mu_2)/\sigma_2$ and $N_1 = N - n^*$ for simplification, we obtain

$$N_1 \lesssim \frac{\sqrt{8\pi}}{b}\cdot\exp\left[\frac{b^2 n^* + \ln n^*}{2}\right]$$

or

$$n^* + \frac{\ln n^*}{b^2} \gtrsim \frac{2}{b^2}\cdot\ln\left(\frac{b}{\sqrt{8\pi}}\cdot N_1\right),$$

where the fact that $(N-2n^*) \sim (N-n^*)$ has been used, with the inequality generally holding as soon as N_1 exceeds n^* by a small integer. (To get numerical bounds on σ_2 when it is not explicity known, note that for bounded payoff (all $A \in \mathscr{A}$, $r_0 \leqq \mu(A) \leqq r_1$) the maximum variance occurs when all payoff is concentrated at the two extremes. That is,

$$P(r_0) = P(r_1) = \tfrac{1}{2} \quad\text{and}\quad \sigma_2^2 \leqq \sigma_{\max}^2 = (\tfrac{1}{2}r_1^2 + \tfrac{1}{2}r_0^2) - (\tfrac{1}{2}r_1 + \tfrac{1}{2}r_0)^2 = \left(\frac{r_1-r_0}{2}\right)^2.)$$

We obtain a recursion for an ever better approximation to n^* as a function of N_1 by rewriting this as

$$n^* \gtrsim b^{-2}\ln\left[\frac{(bN_1)^2}{8\pi n^*}\right].$$

Thus

$$\begin{aligned} n^* &\gtrsim b^{-2}\ln\left[\frac{(bN_1)^2}{8\pi(b^{-2}\ln((bN_1)^2/8\pi n^*))}\right] \\ &\gtrsim b^{-2}\ln\left[\frac{b^4N_1^2}{8\pi}\cdot\frac{1}{\ln((bN_1)^2/8\pi) - \ln n^*}\right] \\ &\gtrsim b^{-2}\ln\left[\frac{b^4N_1^2}{8\pi(\ln N_1^2 - \ln(b^2/8\pi))}\right] \\ &\gtrsim b^{-2}\ln\left[\frac{b^4N_1^2}{8\pi\ln N_1^2}\right], \end{aligned}$$

where, again, the error rapidly approaches zero as N increases. Finally, where it is desirable to have n^* approximated by an explicit function of N, the steps here can be redone in terms of N/n^*, noting that N_1/n^* rapidly approaches N/n^* as N increases. Then

$$n^* \sim b^{-2} \ln \left[\frac{b^4 N^2}{8\pi \ln N^2}\right],$$

where, still, the error rapidly approaches zero as N increases.

The expected loss per trial $l^*(N)$ when n^* trials have been allocated to $\xi_{(2)}(\tau, N)$ is

$$\begin{aligned}
l^*(N) &= \frac{1}{N}|\mu_1 - \mu_2| \cdot [(N - n^*)q(N - n, n^*) + n^*(1 - q(N - n^*, n^*))] \\
&= |\mu_1 - \mu_2| \cdot \left[\frac{N - 2n^*}{N} q(N - n^*, n^*) + \frac{n^*}{N}\right] \\
&\gtrsim |\mu_1 - \mu_2| \cdot \left[\frac{2n^*}{Nx_0^2} + \frac{n^*}{N}\right] \\
&\gtrsim \frac{|\mu_1 - \mu_2|}{b^2 N} \cdot \left[2 + \ln\left(\frac{b^4 N^2}{8\pi \ln N^2}\right)\right]. \qquad \text{Q.E.D.}
\end{aligned}$$

The expression for n^* (and hence the one for $l^*(N)$) was obtained on the assumption that the n^* trials were allocated to $\xi_{(2)}(\tau, N)$. However, there is *no* realizable sequential algorithm which can "foresee" in all cases which of the two schemata will be $\xi_{(2)}(\tau, N)$. There will always be observational sequences wherein *each* schema has a positive probability of being $\xi_{(2)}(\tau, N)$ even after τ has allocated $n > n^*$ trials to one. (For example, τ may have allocated exactly n^* trials to each and must decide where to allocate the next trial even though each schema has a positive probability of being $\xi_{(2)}(\tau, N)$.) Thus, no matter what the plan τ, it will in some cases allocate $n > n^*$ trials to a schema ξ (on the assumption that ξ will turn out to be $\xi_{(1)}(\tau, N)$) only to be confronted with the fact that $\xi = \xi_{(2)}(\tau, N)$. For these sequences the loss will perforce exceed the optimum. Hence $l^*(N)$ is not attainable by any realizable sequential algorithm τ—there will always be outcome sequences which lead τ to allocate too many trials to $\xi_{(2)}(\tau, N)$.

There is, however, a realizable plan τ_0 for which the expected loss per trial $l(\tau_0, N)$ quickly approaches $l^*(N)$; i.e.,

$$\lim_{N \to \infty} \frac{l(\tau_0, N)}{l^*(N)} = 1.$$

τ_0 initially allocates n^* trials to each schema (in any order) and then allocates the remaining $N - 2n^*$ trials to the schema with the highest observed payoff rate at the end of the $2n^*$ trials.

COROLLARY 1.1. *Given N trials, τ_0's expected loss, $l(\tau_0, N)$, approaches the optimum $l^*(N)$. That is, $l(\tau_0, N) \sim l^*(N)$.*

Proof. The expected loss per trial $l(\tau_0, N)$ for τ_0 is determined by applying the earlier discussion of sources of loss to the present case:

$$l(\tau_0, N) = \frac{1}{N} \cdot |\mu_1 - \mu_2| \cdot [(N - n^*)q(n^*, n^*) + n^*(1 - q(n^*, n^*))],$$

where q is the same function as before, but here the probability of error is irrevocably determined after only n^* trials have been allocated to *each* schema; i.e.,

$$q(n^*, n^*) \sim \frac{1}{\sqrt{2\pi}} \frac{\sqrt{\sigma_1^2/n^* + \sigma_2^2/n^*}}{\mu_1 - \mu_2} \exp\left[\frac{-(\mu_1 - \mu_2)^2}{\sigma_1^2/n^* + \sigma_2^2/n^*}\right].$$

(Note that n^* is *not* being redetermined for τ_0; n^* is the number of trials determined above.) Rewriting $l(\tau_0, N)$ we have

$$l(\tau_0, N) = |\mu_1 - \mu_2| \cdot \left[\frac{N - 2n^*}{N} q(n^*, n^*) + \frac{n^*}{N}\right].$$

Since, asymptotically, q decreases as rapidly as N^{-1}, it is clear that the second term in the brackets will dominate as N grows. Inspecting the earlier expression for $l^*(N)$ we see the same holds there. Thus, since the two second terms are identical,

$$\lim_{N \to \infty} \frac{l(\tau_0, N)}{l^*(N)} = 1. \qquad \text{Q.E.D.}$$

To treat the case of r schemata we need a new determination of the probability that the observed best is not the schema with the highest mean. To proceed to this determination let the r schemata be $\xi_1, \xi_2, \cdots, \xi_r$ and let $\mu_1 > \mu_2 > \cdots > \mu_r$ (again, without the observer knowing that this ordering holds).

THEOREM 2. *Given N trials to be allocated to r random variables $\{\xi_1, \xi_2, \cdots, \xi_r\}$, with means $\mu_1 > \mu_2 > \cdots > \mu_r$ and variances $\sigma_1, \sigma_2, \cdots, \sigma_r$ respectively, the minimum expected loss per trial $l_r^*(N)$ is bounded above and below by $l''_{N,r}$ and $l'_{N,r}$, respectively, where*

$$l'_{N,r} \sim \frac{(r - 1)\sigma_2^2}{(\mu_1 - \mu_2)N}\left[2 + \ln\left[\left(\frac{\mu_1 - \mu_2}{\sigma_2}\right)^4 \left(\frac{N^2}{8\pi(r - 1)^2 \ln N^2}\right)\right]\right]$$

and

$$l''_{N,r} \sim \frac{(r - 1)\mu_1\sigma_2^2}{(\mu_1 - \mu_2)^2 N}\left[2 + \ln\left[\left(\frac{\mu_1 - \mu_2}{\sigma_2}\right)^4 \left(\frac{N^2}{8\pi \ln N^2}\right)\right]\right].$$

Proof. We are interested in the probability q_r that the average of the observations of any $\xi_j, j > 1$, exceeds the average for ξ_1; that is, the probability of error

$$q_r(n_1, \cdots, n_r) = \Pr\left\{\left(\frac{S_2}{n_2} > \frac{S_1}{n_1}\right) \text{ or } \left(\frac{S_3}{n_3} > \frac{S_1}{n_1}\right) \text{ or } \cdots \text{ or } \left(\frac{S_r}{n_r} > \frac{S_1}{n_1}\right)\right\}.$$

When a given number of trials $n_0 = \sum_{i=2}^r n_i$ has been allocated to $\xi_2, \xi_3, \cdots, \xi_r$ to minimize the probability of error q_r, that error will clearly be largest if $\mu_2 = \mu_3 = \cdots = \mu_r$. (In other words, when $\mu_j \lneqq \mu_2$, an allocation of $n_j < n_2$ trials to ξ_j will yield

$$\Pr\left\{\left(\frac{S_j}{n_j} > \frac{S_1}{n_1}\right)\right\} < \Pr\left\{\left(\frac{S_2}{n_2} > \frac{S_1}{n_1}\right)\right\};$$

hence for a given number of trials, a greater reduction in q_r can be achieved if the means μ_j are not all equal.) Moreover, for those cases where $\mu_2 = \mu_3 = \cdots = \mu_r$, the worst case occurs when the largest of the variances $\sigma_2, \sigma_3, \cdots, \sigma_r$ is in fact the common variance of each of $\xi_2, \xi_3, \cdots, \xi_r$. Given this worst case, $(\mu_2, \sigma_2) = (\mu_3, \sigma_3) = \cdots = (\mu_r, \sigma_r)$, q_r will be minimized for an allocation of n_0 trials to $\xi_2, \cdots, \xi_r$ if (as nearly as possible) equal numbers of trials are given each schema (since each schema contributes equally to the probability of error).

From these observations we can obtain bounds on q_r. As before, let

$$q(n_1, n_2) = \Pr\left\{\frac{S_2}{n_2} > \frac{S_1}{n_1}\right\}.$$

When $\sum_{i=2}^{r} n_i = (r-1)m$ trials are allocated to $\xi_2, \cdots, \xi_r$ so as to minimize the probability of error, we have

$$\Pr\left\{\left(\frac{S_2}{n_2} > \frac{S_1}{N-(r-1)m}\right) \text{ or } \cdots \text{ or } \left(\frac{S_r}{n_r} > \frac{S_1}{N-(r-1)m}\right)\right\}$$
$$< \binom{r-1}{1} q(N-(r-1)m, m) - \binom{r-1}{1} 2!\, q(N-(r-1)m, m) + \cdots,$$

where the right-hand side is obtained by noting that the events $\{(S_i/n_i > S_1/(N-(r-1)m)), i = 2, \cdots, r\}$ are independent and, under the best allocation in the worst case, $n_i = m$ for $i = 2, \cdots, r$, so that

$$\Pr\left\{\frac{S_i}{m} > \frac{S_1}{N-(r-1)m}\right\} = q(N-(r-1)m, m).$$

Thus, when $(r-1)m$ trials are allocated to $\xi_2, \cdots, \xi_r$ to minimize the probability of error, we have the following bounds:

$$q(N-(r-1)m, m) < q_r(n_1, \cdots, n_r)$$
$$= \Pr\left\{\left(\frac{S_2}{n_2} > \frac{S_1}{N-(r-1)m}\right) \text{ or } \cdots \text{ or } \left(\frac{S_r}{n_r} > \frac{S_1}{N-(r-1)m}\right)\right\}$$
$$< (r-1)q(N-(r-1)m, m).$$

Using the upper and lower bounds on q_r thus obtained, we can proceed to upper and lower bounds, $l^{\mathrm{II}}_{N,r}$ and $l^{\mathrm{I}}_{N,r}$ respectively, on the expected loss $l_N(n_2, \cdots, n_r)$ for N trials:

$$l^{\mathrm{I}}_{N,r}(m) = (\mu_1 - \mu_2)[(N-(r-1)m)q + (r-1)m(1-q)] < l_N(n_2, \cdots, n_r) < l^{\mathrm{II}}_{N,r}(m)$$
$$= \mu_1[(N-(r-1)m)(r-1)q + (r-1)m(1-(r-1)q)],$$

using the earlier discussion of sources of loss and the fact that $(\mu_1 - \mu_2) < (\mu_1 - \mu_j) < \mu_1$ for all j. As before we can determine the optimal value of m for these bounds, m^{**} and m^* respectively, by setting $dl/dm = 0$. Letting $q'' = (r-1)q$ and $q' = q$, we have

$$\frac{dl^{(i)}}{dm} = \mu^{(i)}\left[(N-2(r-1)m)\frac{dq^{(i)}}{dm} - 2(r-1)q^{(i)} + (r-1)\right] = 0.$$

Or, noting that $1 - 2q^{(i)}$ rapidly approaches 1, we have

$$m^{(i)} \sim \frac{N}{2(r-1)} + \frac{1}{2}\left(\frac{dq^{(i)}}{dm}\right)^{-1}.$$

$q^{(i)}$ decreases with m, so dq/dm is a negative quantity. Since $dq''/dm = (r-1)dq'/dm$, we have $dq''/dm < dq_r/dm < dq'/dm$ and

$$m^{**} \sim \frac{N}{2(r-1)} + \frac{1}{2}\left(\frac{dq''}{dm}\right)^{-1} = \frac{N}{2(r-1)} + \frac{1}{2(r-1)}\left(\frac{dq'}{dm}\right)^{-1}$$
$$> \frac{N}{2(r-1)} + \frac{1}{2}\left(\frac{dq'}{dm}\right)^{-1} \sim m^{*}.$$

That is, $(r-1)m^{**}(N) > n_{\text{opt}} = \sum_{i=2}^{r} n_{i,\text{opt}} > (r-1)m^{*}(N)$. Thus by determining the optimal m for each of the bounds we obtain bounds on n_{opt}, the number of trials which should be allocated to schemata other than ξ_1 in order to minimize expected loss.

m^* is directly obtained from the previous 2-schemata derivation by using $(r-1)m$ in place of n and taking the derivative of q with respect to m instead of n. The result is

$$m^* \sim b^{-2}\ln\left(\frac{b^4N_1^2}{8\pi(r-1)^2\ln N_1^2}\right) \sim b^{-2}\ln\left(\frac{b^4N^2}{8\pi(r-1)^2\ln N^2}\right),$$

where $N_1 = N - (r-1)m$ now.

m^{**} is similarly obtained using $(r-1)q$ for q throughout. The result is

$$m^{**} \sim b^{-2}\ln\left(\frac{b^4N^2}{8\pi\ln N^2}\right) \sim m^* + \frac{2\ln(r-1)}{b^2}.$$

The corresponding upper and lower bounds on the expected loss per trial are

$$l''_{N,r}(m^{**}) \sim \mu_1\cdot\frac{r-1}{b^2N}\left[2 + \ln\left(\frac{b^4N^2}{8\pi\ln N^2}\right)\right]$$

and

$$l'_{N,r}(m^{*}) \sim (\mu_1 - \mu_2)\cdot\frac{r-1}{b^2N}\left[2 + \ln\left(\frac{b^4N^2}{8\pi(r-1)^2\ln N^2}\right)\right]. \qquad \text{Q.E.D.}$$

4. Allocation of trials by genetic plans. We now have bounds on the best possible performance (in terms of minimizing expected loss) of any plan which tests one random variable (schema) at a time. The objective now is to obtain a measure of the performance of genetic plans in similar circumstances so that a comparison can be made with this criterion. This comparison will reveal two things: (1) Even when the genetic plan is constrained to test one schema at a time, losses decrease at a rate proportional to that decreed by the criterion (though, initially, the plan does not have information about the means and variances required to calculate an optimal allocation of trials). (2) Intrinsic parallelism (tests of many schemata with a single trial) is used to advantage by genetic plans, enabling them to greatly surpass the one-schema-at-a-time criterion. Because both of these points came through convincingly under approximations less severe than "$\sim$", weaker approximations will be used wherever they substantially simplify the derivation.

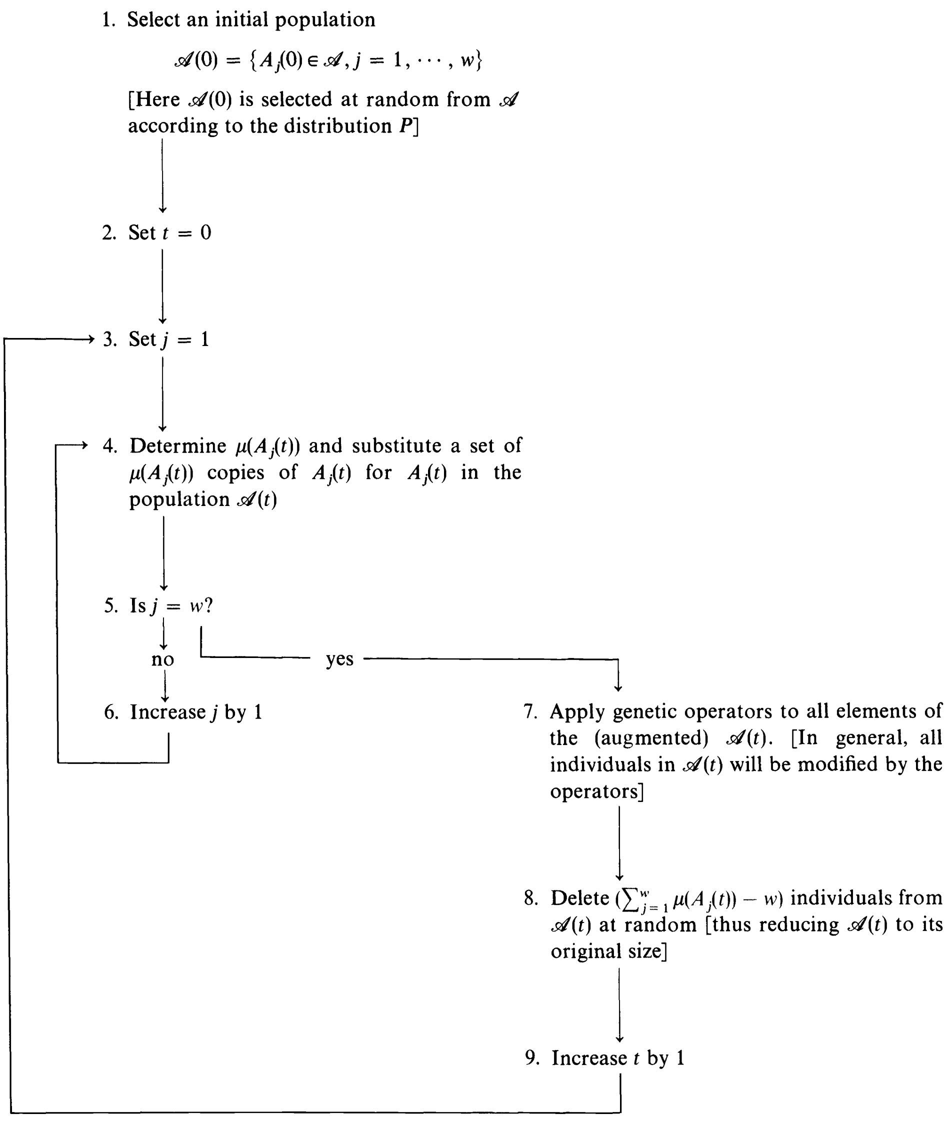
Specifically, let us consider reproductive plans using genetic operators on a nonincreasing population (i.e., for all t, the average effective payoff rate of the population, $\bar{\mu}'_t$, is 1). Such plans can be diagrammed as in Diagram 1. The genetic operators, $\omega \in \Omega$, of step 7 are either of the form

$$\omega : \mathscr{A} \times \mathscr{A} \to \mathscr{A} \times \mathscr{A}$$

or else

$$\omega : \mathscr{A} \to \mathscr{A}.$$

Diagram 1



1. Select an initial population

   $\mathscr{A}(0) = \{A_j(0) \in \mathscr{A}, j = 1, \cdots, w\}$

   [Here $\mathscr{A}(0)$ is selected at random from $\mathscr{A}$ according to the distribution P]

2. Set $t = 0$

3. Set $j = 1$

4. Determine $\mu(A_j(t))$ and substitute a set of $\mu(A_j(t))$ copies of $A_j(t)$ for $A_j(t)$ in the population $\mathscr{A}(t)$

5. Is $j = w$?

   no

   yes

6. Increase j by 1

7. Apply genetic operators to all elements of the (augmented) $\mathscr{A}(t)$. [In general, all individuals in $\mathscr{A}(t)$ will be modified by the operators]

8. Delete $(\sum_{j=1}^{w} \mu(A_j(t)) - w)$ individuals from $\mathscr{A}(t)$ at random [thus reducing $\mathscr{A}(t)$ to its original size]

9. Increase t by 1

The intended interpretation is that (pairs of) individuals selected from the population $\mathscr{A}(t)$ are transformed by the operator into new (pairs of) individuals. The operators are conservative in the sense that they do not alter the size of the population. Formal definitions of various genetic operators can be found in Holland [6], but for the analysis below, it is necessary to know only that (i) arguments for the operators are chosen at random from $\mathscr{A}(t)$, and (ii) the conditional probability $o_{\xi t}$ that $A \in \xi$, once selected, will be transformed to some $A' \notin \xi$, is generally small and decreases to a value negligibly different from zero as the proportion of ξ in $\mathscr{A}(t)$ approaches 1.

It should be noted that the reproductive plan modifies the distribution P (over $\mathscr{A}$) as the number of trials increases. As a consequence the marginal distributions for the schemata of interest $\{\xi\}$, hence the means $\{\mu_\xi\}$, may change as N grows large. However, the central limit theorem holds for sequences of independent random variables with variable distribution as long as they are uniformly bounded. This condition holds for all schemata when there is an upper bound on performance ($\text{l.u.b.}_{A \in \mathscr{A}} \{\mu(A)\} < \infty$), and we shall proceed accordingly.

THEOREM 3. *Given N trials, a reproductive plan with genetic operators can be expected to allocate $N_{\xi_{(1)}}$ trials to the schema (random variable) $\xi_{(1)}$ with the best observed average payoff, where*

$$N_{\xi_{(1)}} \cong (>) \frac{N_{\xi_{(1)}}(0)}{\hat{z}_{(1)}} \exp\left[\frac{\hat{z}_{(1)} n_\rho}{n_\rho(0)}\right],$$

with $\hat{z}_{(1)}$ being the average of the logarithms of the observed payoffs for $\xi_{(1)}$, $N_{\xi(1)}(0)$ being the trials allocated to $\xi_{(1)}$ at the outset, and $n_\rho = N - N_{\xi_{(1)}}$.

Proof. The increase in the number of instances of schema ξ during step 4 of the plan is given by

$$\mathscr{N}'_\xi(t) = \sum_{A \in \xi(t)} \mu(A),$$

where $\xi(t)$ is the set of instances of ξ in the population $\mathscr{A}(t)$. The instances of ξ in $\mathscr{A}(t)$ constitute a sample of ξ under the modified distribution P_t holding at time t. The value of $\mathscr{N}'_\xi(t)$ can be written in terms of $\mathscr{N}_\xi(t)$, the expected number of individuals in $\xi(t)$, by using the average

$$\hat{\mu}_{\xi t} \overset{\text{def}}{=} \frac{1}{\mathscr{N}_\xi(t)} \sum_{A \in \xi(t)} \mu(A)$$

of the observations of ξ at time t. In these terms

$$\mathscr{N}'_\xi(t) = \hat{\mu}_{\xi t} \mathscr{N}_\xi(t).$$

We will concentrate here on (typical) reproductive plans wherein the operators in step 7 are applied to elements of $\mathscr{A}(t)$ independently of their identity (representation). As mentioned earlier, the common characteristic of these operators is such that the conditional probability $o_{\xi t}$ of $A \in \mathscr{A}(t)$ being transformed to $A' \notin \xi(t)$ decreases to a negligibly small value as the size of $\xi(t)$ approaches that of $\mathscr{A}(t)$. Hence at the end of step 7 we can expect

$$\mathscr{N}''_\xi(t) \cong (1 - o_{\xi t}) \hat{\mu}_{\xi t} \mathscr{N}_\xi(t),$$

where the factor $(1 - o_{\xi t}) \to 1$ as $\mathscr{N}_\xi(t) \to w$. (This ignores new instances of ξ formed by the operators from other $A \notin \xi(t)$.) Finally, after the deletions of step 8 (which are again uniform over $\mathscr{A}(t)$), we expect

$$\mathscr{N}_\xi(t+1) \cong \frac{1}{\bar{\mu}_t}(1 - o_{\xi t})\hat{\mu}_{\xi t}\mathscr{N}_\xi(t),$$

since the expected value of the increase, $\sum_{j=1}^{w} \mu(A_j(t))$, is just $\bar{\mu}_t = \sum_{A \in \mathscr{A}} \mu(A)P_t(A)$, where P_t is the (modified) distribution over $\mathscr{A}$ at time t. Putting this recursion in explicit form we get

$$\begin{aligned}\mathscr{N}_\xi(t+1) &\cong \left(\prod_{t'=0}^{t} \hat{\mu}'_{\xi t'}\right)\mathscr{N}_\xi(0) = \mathscr{N}_\xi(0)\exp\left[\ln\left(\prod^{t} \hat{\mu}'_{\xi t'}\right)\right] \\ &= \mathscr{N}_\xi(0)\exp\left[\sum^{t} \ln(\hat{\mu}'_{\xi t'})\right] \\ &= \mathscr{N}_\xi(0)\, e^{\hat{z}_t t},\end{aligned}$$

where $\hat{\mu}'_{\xi t} = \hat{\mu}_{\xi t}(1 - o_{\xi t})/\bar{\mu}_t$ and $\hat{z}_t = \sum^{t} \ln(\hat{\mu}'_{\xi t'})/t$. Using the fact that

$$\sum_{t'=1}^{t} f(t') \geqq \int_{t'=0}^{t} f(t')\, dt'$$

for monotone increasing functions, the *total* number of trials of ξ to time t, $N_\xi(t)$, can be approximated by

$$N_\xi(t) \cong \mathscr{N}_\xi(0) + \mathscr{N}_\xi(0)\sum_{t'=1}^{t} e^{\hat{z}_t t'} \geqq \mathscr{N}_\xi(0) + \mathscr{N}_\xi(0)\int_{t'=0}^{t} e^{\hat{z}_t t'}\, dt',$$

assuming that at each time t, $\hat{\mu}'_{\xi t}$ (the rate of increase of $\xi(t)$) is in excess of 1. (It should be noted that this approximation to $N_\xi(t)$ assumes $\hat{z}_1 \cong \hat{z}_2 \cong \cdots \cong \hat{z}_t$. For small t, there may be a considerable error if $\hat{z}_{t'}$, $t' = 1, \cdots, t$, swings over a wide range, though at worst the error (as a fraction of the true value) will be considerably less than $e^{-\hat{z}_t}$. Moreover, as t increases, $\hat{z}_t$ changes more and more slowly because it is an average, while earlier terms in the sum being approximated are swamped by the larger later terms. Thus the probability of a given error steadily decreases as t increases.)

$$\begin{aligned}N_\xi(t) &\cong \mathscr{N}_\xi(0) + \frac{\mathscr{N}_\xi(0)}{\hat{z}_t}[e^{\hat{z}_t t} - 1] \\ &\cong \mathscr{N}_\xi(0)\left[\frac{e^{\hat{z}_t t}}{\hat{z}_t} + \left(1 - \frac{1}{\hat{z}_t}\right)\right].\end{aligned}$$

Therefore the total number of trials of a schema ξ increases exponentially as a function of time (assuming the performance of ξ is consistently better than the average).

Let $\xi_{(1)}$ be the schema receiving the greatest number of trials over the interval t, and let $\xi_{(1)}(t')$ designate the set of instances of $\xi_{(1)}$ present in the population $\mathscr{A}(t')$ at time t'. Let $n_p(t)$ be the total trials allocated to all other *individuals* $\{\mathscr{A}(t') - \xi_{(1)}(t')\}$ from $t' = 0$ through $t' = t$. Since for all t' the number of individuals in

$\mathscr{A}(t')$ remains constant, the total number of trials $N(t) = N(0)\cdot t$. It follows that

$$\frac{n_\rho(t)}{n_\rho(0)} = \frac{N(t) - N_{\xi_{(1)}}(t)}{N(0) - N_{\xi_{(1)}}(0)} \leq \frac{N(t)}{N(0)} = t.$$

Hence

$$N_{\xi_{(1)}} \cong (>)\mathscr{N}_{\xi_{(1)}}(0)\left[\frac{1}{\hat{z}_{(1)}}\cdot \exp\left(\frac{\hat{z}_{(1)}n_\rho(t)}{n_\rho(0)}\right)\right],$$

where $\hat{z}_{(1)}$ is the observed $\hat{z}_t$ for $\xi_{(1)}$. Or

$$n_\rho \cong (<)(n_\rho(0)/\hat{z}_{(1)})\ln[\hat{z}_{(1)}N_{\xi_{(1)}}/\mathscr{N}_{\xi_{(1)}}(0)]. \qquad \text{Q.E.D.}$$

The following correspondence allows comparison of this result to the one obtained earlier for optimal allocation.

	"Optimal" [*]	"Reproductive" [ρ]
$N_\bullet$, trials allocated to $\xi_{(1)}$	$N_{1*} = N_1$	$N_{1\rho} = N_{\xi_{(1)}}$
$n_\bullet$ trials allocated to other schemata	$n_* = (r-1)m^*$	$n_\rho = n_\rho$

Thus we have

$$N_{1*} \sim \frac{(r-1)\sqrt{8\pi}}{b}\exp\left[\frac{1}{2}\left(\frac{b^2 n_*}{r-1}\right) + \frac{1}{2}\ln\left(\frac{n_*}{r-1}\right)\right],$$

where $b = (\mu_1 - \mu_2)/\sigma_2$, vs.

$$N_{1\rho} \cong (>)\frac{N_{1\rho}(0)}{\hat{z}_{(1)}}\exp\left[\frac{\hat{z}_{(1)}n_\rho}{n_\rho(0)}\right],$$

where $\hat{z}_{(1)} = \sum^t \ln(\hat{\mu}'_{\xi_{(1)}t'})/t$.

Clearly the two plans behave in roughly the same way, the number of trials allocated to the "best" in each case increasing exponentially as a function of the total number of trials allocated to all other schemata. However, a comparison of expected loss per trial yields much more interesting information. For the reproductive plan the expected loss per trial is bounded above by

$$l''_\rho = \frac{\mu_1}{N}[N_{1\rho}r'q(N_{1\rho}, n'_\rho) + (1 - r'q(N_{1\rho}, n'_\rho))n_\rho],$$

where r' is the number of schemata which have received n'_ρ (or more) trials under the reproductive plan. It is critical to what follows that $r'\cdot n'_\rho$ need *not* be equal to n_ρ. Each $A \in \mathscr{A}$ is a trial of 2^l distinct schemata. As $\mathscr{A}(t)$ is transformed into $\mathscr{A}(t+1)$ by the reproductive plan, *each* schema ξ having instances in $\mathscr{A}(t)$ can be expected to have $(1 - o_\omega)\mu_\xi/\hat{\mu}_t$ instances in $\mathscr{A}(t+1)$. Thus, over the course of several time steps, the number of schemata r' receiving n'_ρ trials will be much, much greater than the number of trials n_ρ allocated to *individuals* A (where $A \in \mathscr{A}$ but $A \notin \xi_1$) even when n'_ρ approaches or exceeds n_ρ. (As a simple example consider a set of three trials $\{\sigma_1\sigma_1\sigma_1\sigma_2, \sigma_2\sigma_1\sigma_1\sigma_1, \sigma_2\sigma_2\sigma_2\sigma_1\}$. Each of the 6 schemata $\{\sigma_2\square\square\square, \square\sigma_1\square\square, \square\square\sigma_1\square, \square\square\square\sigma_1, \sigma_2\square\square\sigma_1, \square\sigma_1\sigma_1\square\}$ receives 2 trials, so that for $n'_\rho = 2$ we have $r' = 6$ and $n'_\rho \cdot r' = 12$ though n_ρ is clearly 3 (or less). (See below.) This observation, that generally $r'n'_\rho \gg n_\rho$, is an explicit consequence of the reproductive plan's *intrinsic parallelism* (each trial of an individual $A \in \mathscr{A}$ is a useful trial of a great many schemata).

THEOREM 4. *The ratio of the upper bound on the expected loss per trial for a reproductive plan, l''_ρ, to the corresponding lower bound for optimal allocation, l'_*, varies inversely as the number, r', of schemata being tried. Specifically,*

$$\frac{l''_\rho}{l'_*} \rightarrow \frac{(\mu_1 - \mu_2)^2 n_\rho(0)}{2\sigma_2^2 \hat{z}_{(1)}} \left(\frac{1}{r' - 1}\right),$$

where the parameters are as defined in the statements of the previous theorems.

Proof. Substituting the earlier expressions for $N_{1\rho}$ and $q(N_{1\rho}, n'_\rho)$ in l''_ρ, and noting that $(1 - r'q(N_{1\rho}, n'_\rho))n_\rho < n_\rho$, gives

$$l''_\rho \lesssim \frac{\mu_1}{N}\left[\frac{r'N_{1\rho}(0)}{\hat{z}_{(1)}b\sqrt{2\pi}} \exp\left[\frac{\hat{z}_{(1)}n_\rho}{n_\rho(0)} - \frac{b^2 n'_\rho + \ln n'_\rho}{2}\right] + n_\rho\right].$$

If $b^2 n'_\rho/2 \geqq \hat{z}_{(1)} n_\rho / n_\rho(0)$, it is clear that the first term decreases as n_ρ increases, but the second term, n_ρ, increases. In other words, if n'_ρ increases at a rate proportional to the rate of increase of n_ρ, the expected loss per trial will soon depend almost entirely on the second term, as was the case for optimal allocation. Thus, for n'_ρ so specified, we can compare losses by taking the ratio of the respective second terms:

$$\frac{l''_\rho}{l'_*} = \frac{n_\rho}{(r - 1)m^*}.$$

(A quick comparison of the first terms of l''_ρ and l'_* also shows that the above condition on n'_ρ is sufficient to assure that the first term of l''_ρ is always less than the first term of l'_*.) This comparison is conservative in the sense that the *upper* bound on the reproductive plan's losses is compared to the *lower* bound on the optimal allocation's losses.

To proceed, let the reproductive plan's loss per trial over N trials be compared to that of an optimal allocation of N trials to the r'-schemata which received n'_ρ or more trials under the reproductive plan. (It should be noted that the above condition on n'_ρ can be made as weak as desired by simply choosing $n_\rho(0)$ large enough.) Substituting the explicit expressions derived earlier for n_ρ and m^* as a function of N gives

$$\frac{l''_\rho}{l'_*} \lesssim \frac{b^2 n_\rho(0) \ln [\hat{z}_{(1)}(N - n_\rho)/N_{1\rho}(0)]}{(r' - 1)(\ln [b^4 N^2/8\pi(r' - 1)^2 \ln N^2])\hat{z}_{(1)}}.$$

Simplifying and deleting terms which do not affect the direction of the inequality, we get

$$\frac{l''_\rho}{l'_*} \lesssim \frac{b^2 n_\rho(0) \ln (\hat{z}_{(1)}N)}{(r' - 1)\hat{z}_{(1)}[2 \ln b^2 N - \ln (8\pi(r' - 1)^2 \ln N^2)]}.$$

Or, as N grows,

$$\frac{l''_p}{l'_*} \to \frac{b^2 n_p(0)}{2\hat{z}_{(1)}}\left(\frac{1}{r'-1}\right). \qquad \text{Q.E.D.}$$

Thus the reproductive plan effectively exploits its intrinsic parallelism—its losses for a given number of trials N, in relation to an optimal (one-schema-at-a-time) allocation, are reduced by the factor r'. We can get some idea of how large this reduction is by looking more closely at the relation between N, n'_p and r'. This relation in turn is more easily approached if we first look more closely at schemata. A schema will be said to be *defined on* the set of positions $\{j_1, \cdots, j_h\}$ at which $\delta_{i_j} \neq \square$. Given Σ with k symbols, there are k^h distinct schemata defined on any given set of $h \leqq l$ positions; moreover, no matter what set of positions is chosen, *every* $A \in \mathscr{A}$ is an instance of one of these k^h schemata. That is, the set of schemata so defined partitions $\mathscr{A}$, and any distinct set of positions gives rise to a different partition of $\mathscr{A}$. (For example, given the alphabet $\Sigma = \{\sigma_1, \sigma_2\}$ and strings of length $l = 4$, the set of schemata defined on position 1 is $\{\sigma_1\square\square\square, \sigma_2\square\square\square\}$. It is clear that every string in $\mathscr{A}$ begins either with the symbol σ_1 or else the symbol σ_2, hence the given set partitions $\mathscr{A}$. Similarly the set defined on position 2, $\{\square\sigma_1\square\square, \square\sigma_2\square\square\}$, partitions $\mathscr{A}$, and the set defined on positions 2 and 4, $\{\square\sigma_1\square\sigma_1, \square\sigma_1\square\sigma_2, \square\sigma_2\square\sigma_1, \square\sigma_2\square\sigma_2\}$, is still a different partition of $\mathscr{A}$, a refinement of the one just previous.) There are $\binom{l}{h}$ distinct ways of choosing h positions $\{1 \leqq j_1 < j_2 < \cdots < j_h \leqq l\}$ along a string of length l, and h can be any number between 1 and l. Thus there are $\sum_{h=1}^{l} \binom{l}{h} = 2^l$ distinct partitions induced on $\mathscr{A}$ by these sets of schemata. It follows that when the reproductive plan generates N trials, they will be simultaneously distributed over each of these partitions. That is, *each* of the 2^l *sets* of schemata (defined on the 2^l distinct choices of positions) receives N trials.

We can get a *rough* estimate of the number, r', of schemata receiving n'_p or more trials by assuming the N trials are distributed uniformly and independently over each partition. Two factors perturb the estimate: (1) Given a uniform initial distribution P, the reproductive plan will make the distribution increasingly non-uniform as n_p increases. However, until N gets fairly large relative to n_p the departure is small enough to make the estimate useful. (2) When a given schema defined on h positions receives n'_p or more trials, then so must every schemata of which it is a subset. (For example, let $\square\sigma_2\square\sigma_1$ receive 2 trials, say $\{\sigma_2\sigma_2\sigma_2\sigma_1, \sigma_1\sigma_2\sigma_1\sigma_1\}$. These are at the same time trials of $\square\sigma_2\square\square$, and also of $\square\square\square\sigma_1$. Hence $\square\sigma_2\square\square$ and $\square\square\square\sigma_1$ also receive at least 2 trials.) Similarly, if a given schema receives less than n'_p trials, then so will every schema of which it is a superset. These are clearly violations of the assumption of independence. Nevertheless, when N is small relative to k^l (so that only a small fraction of schemata have been tried), departures from independence are small enough to allow a useful estimate. Some thought about the number of dependencies relative to the total number of schemata tried, or a small Monte Carlo simulation, are convincing in this respect. Though the estimate is rough, the value of r' obtained for typical values of N, b, $\hat{z}_{(1)}$, etc. is clearly of the right order of magnitude.

The average number of trials per schema for a set of schemata defined on h positions is N/k^h. Under the assumption of uniform, independent trials, the Poisson distribution gives the number of schemata receiving n'_p or more trials:

$$\sum_{n=n'_p}^{\infty} \left(\frac{N}{k^h}\right)^n \left(\frac{1}{n!}\right) e^{-N/k^h}.$$

There are $\binom{l}{h}$ distinct sets of k^h schemata defined on h positions, so that the number r' of schemata in Ξ, $h = 1, \cdots, l$, receiving at least n'_p trials is then

$$\sum_{h=1}^{l} \binom{l}{k} k^h \sum_{n=n'_p}^{\infty} \left(\frac{N}{k^h}\right)^n \left(\frac{1}{n!}\right) e^{-N/k^h}.$$

This is a very large number as long as n'_p is smaller than $N/2$, as it always would be in practice. Even when N is quite small (so that the estimate is good), the number is substantial. For example, if the representations are of length $l = 32$ with two symbols in Σ (so that $\mathscr{A}$ contains $2^{32} \cong 4 \times 10^9$ elements) and if $N = 16$ with $n'_p = 8$, then $r' > 700$ schemata can be expected to receive in excess of n'_p trials. The numbers chosen here are clearly very conservative—if $N = 32$, $r' > 9000$ for l and n'_p as given; any increase in l produces even more dramatic increases in r'.

The advantages implied by this analysis have been observed in a variety of computer tests (Bagley [1], Cavicchio [3], Hollstien [7]).

5. Conclusion. Intrinsic parallelism in the search of schemata offers a tremendous advantage to any optimization procedure which can exploit it. Reproductive plans with genetic operators (genetic algorithms) are the only procedures so far studied which exhibit this phenomenon. They have the additional desirable properties of easy implementation, compact storage and automatic use of the large amounts of relevant information encountered during operation, and robustness (efficient operation under maximal uncertainty). For these reasons it is recommended that genetic algorithms be given serious consideration whenever a problem of natural or artificial adaptation arises.

REFERENCES

[1] J. D. Bagley, *The behavior of adaptive systems which employ genetic and correlation algorithms*, Ph.D. dissertation, University of Michigan, Ann Arbor, 1967.

[2] R. Bellman, *Adaptive Control Processes*, Princeton University Press, Princeton, N.J., 1961.

[3] D. J. Cavicchio, *Adaptive search using simulated evolution*, Ph.D. dissertation, University of Michigan, Ann Arbor, 1970.

[4] M. E. Hellman and J. M. Cover, *Learning with finite memory*, Ann. Math. Statist., 41 (1970), pp. 765–782.

[5] J. H. Holland, *A new kind of turnpike theorem*, Bull. Amer. Math. Soc., 75 (1969), pp. 1311–1317.

[6] ———, *Processing and processors for schemata*, Associative Information Techniques, E. L. Jacks, ed., Elsevier, New York, 1971, pp. 127–146.

[7] R. B. Hollstien, *Artificial genetic adaptation in computer control systems*, Ph.D. dissertation, University of Michigan, Ann Arbor, 1971.

[8] Ya Z. Tsypkin, *Adaptation and Learning in Automatic Systems*, Academic Press, New York, 1971.

SIAM J. COMPUT.
Vol. 2, No. 2, June 1973

A TECHNIQUE FOR SPEEDING UP LR(k) PARSERS*

A. V. AHO† AND J. D. ULLMAN‡

Abstract. We present a new transformation that reduces the size and increases the speed of LR(k) parsers. This transformation can be applied to all LR(k) parsers including those produced by Knuth's and DeRemer's techniques. The transformation causes the parser to avoid reductions by productions of the form $A \rightarrow B$, where A and B are nonterminals.

Key words. Parsing, compiling, parser optimization, LR(k) parsing, single productions

1. Introduction. The LR(k) grammars are the largest known class of unambiguous context-free grammars for which deterministic left-to-right (no backtrack) bottom-up parsers can be mechanically generated. This class of grammars is capable of describing virtually all of the syntax of programming languages that can be specified by context-free grammars.

The technique proposed by Knuth [17] for the automatic generation of a parser for an LR(k) grammar results in parsers that are too large for practical use. However, new techniques for generating LR(k) parsers have been developed and these techniques produce parsers of practical size (Korenjak [19], DeRemer [9], [10], Aho and Ullman [5]).

In this paper we present a new transformation on LR(k) parsers. The effect of this transformation is twofold. First, it makes a reduction in the number of rows (states) of a parser even if the parser is produced by one of the new techniques. Second, it allows the parser to skip many reductions by single productions. (A *single production* is a production of the form $A \rightarrow B$, where A and B are nonterminals.)

This transformation incorporates into the LR(k) framework the efficiency of operator precedence parsing (Floyd [12]), which also may be construed as ignoring reductions by single productions. In practical situations, single productions often have no significance in the generation of the translation for the input (see, e.g., Gray and Harrison [14]). For example, in various syntax directed translation schemes (Lewis and Stearns [20], Knuth [18], Aho and Ullman [3]), the translations of the nonterminal A are often equal to the translations of the nonterminal B when the underlying production is $A \rightarrow B$. Thus, our transformation can be applied to many parsers without affecting the translation normally performed by a parser in the process of compilation. Of course, should a single production have semantic significance, the production can be left intact.

Our principal results are the following. Our transformation can be applied to any LR(k) parser including the canonical LR(k) parser generated by Knuth's method [17] and the parser generated by DeRemer's simple LR method [10]. For the latter two parsers, all reductions by single productions can be eliminated if

* Received by the editors June 5, 1972, and in revised form February 16, 1973.

† Bell Telephone Laboratories, Inc., Murray Hill, New Jersey 07974.

‡ Department of Electrical Engineering, Princeton University, Princeton, New Jersey 08540. The work of this author was supported in part by the National Science Foundation under Grant GJ-35570.

the grammar has no more than one single production with a given nonterminal on the left side. Even if this latter condition is not satisfied, substantial savings in both time and the number of states are still possible, and we have found realistic situations where all single productions can be eliminated.

The problem of eliminating single productions in LR(k) parsers was first posed by DeRemer [9]. A straightforward solution which may increase the size of the parser is found in Anderson [7] and Anderson, Eve and Horning [8]. Pager [21] has recently extended our method to an arbitrary LR(k) grammar, but again, the size of the parser may increase. A technique for eliminating reductions by single productions in a simple precedence parser was implemented for ALGOL W [13].

2. Background.

In this section we shall review the basic notions of LR(k) grammars and parsing.

A context-free grammar (*grammar*) is a four-tuple $G = (N, \Sigma, P, S)$, where N and Σ are finite disjoint sets—*nonterminals* and *terminals*, respectively; S, in N, is the *start symbol*, and P is a finite set of *productions* of the form $A \rightarrow \alpha$, where A is in N and α in $(N \cup \Sigma)^*$. We assume the productions are numbered $1, 2, \cdots, p$ in some order.

Conventions. Let $G = (N, \Sigma, P, S)$ be a grammar.

(i) A, B and C denote nonterminals in N.
(ii) a, b and c denote terminals in Σ.
(iii) X, Y and Z denote nonterminals or terminals.
(iv) We use $u, v, \cdots, z$ for strings in Σ^* and $\alpha, \beta, \gamma, \cdots$ for strings in $(N \cup \Sigma)^*$.
(v) We use e for the empty string.

If $A \rightarrow \alpha$ is in P, then for all β and γ, we write $\beta A\gamma \underset{rm}{\Rightarrow} \beta\alpha\gamma$ and say $\beta A\gamma$ *directly derives* $\beta\alpha\gamma$. If γ is in Σ^*, then this derivation is said to be *rightmost*, and we write $\beta A\gamma \underset{rm}{\Rightarrow} \beta\alpha\gamma$. The relations $\overset{*}{\Rightarrow}$ and $\underset{rm}{\overset{*}{\Rightarrow}}$ denote the reflexive and transitive closure of $\Rightarrow$ and $\underset{rm}{\Rightarrow}$, respectively. Arrows may be subscripted by the name of the grammar, to resolve ambiguities.

The *language defined by G*, denoted $L(G)$, is $\{w | S \overset{*}{\Rightarrow} w\}$. It is well known that if $S \overset{*}{\Rightarrow} w$, then $S \underset{rm}{\overset{*}{\Rightarrow}} w$. That is, every sentence in the language has a rightmost derivation. A *right sentential form* of G is a string α such that $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha$. If $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha Aw \underset{rm}{\Rightarrow} \alpha\beta w$, then a prefix of $\alpha\beta$ is called a *viable prefix* of G.

Let $S = \gamma_0, \gamma_1, \cdots, \gamma_n = w$ be a sequence of right sentential forms such that for $0 \leqq i < n$, $\gamma_i = \alpha_i A_i w_i$, $\gamma_{i+1} = \alpha_i \beta_i w_i$ and $A_i \rightarrow \beta_i$ is production p_i of G. In this sequence we say that γ_{i+1} is *reduced* to γ_i using the production $A_i \rightarrow \beta_i$. The sequence of productions $p_{n-1}p_{n-2} \cdots p_1p_0$ which can be used to reduce w to S is called a *right parse* of w according to G. That is, a right parse is the reverse of the sequence of productions used in a rightmost derivation. A *parsing algorithm* (or *parser*) for G is a function that maps a sentence in $L(G)$ into a right parse and a word not in $L(G)$ into an error message.

Convention. Given a grammar $G = (N, \Sigma, P, S)$, we shall assume from here on that every symbol X in $(N \cup \Sigma)$ is useful in the sense that for each X there is some derivation of the form $S \overset{*}{\Rightarrow} \alpha X\beta \overset{*}{\Rightarrow} w$. Moreover, if X is a nonterminal that derives e, then we shall assume that X also derives a nonempty string. If X derives only e, we can eliminate X in the obvious way without impairing LR(k)-ness of the grammar.

We define $\mathrm{FIRST}_k^G(\alpha)$ as $\{w | \alpha \overset{*}{\Rightarrow} wx$ and either $|w|$[1] $= k$, or $|w| < k$ and $x = e\}$. If α is in Σ^*, then $\mathrm{FIRST}_k^G(\alpha)$ has one member, w, which is either α, if $|\alpha| \leqq k$, or the first k symbols of α. Note that $\mathrm{FIRST}_k^G(\alpha)$ is independent of G in this case. We delete G and/or k from FIRST when no ambiguity arises. Informally, FIRST(α) is the set of prefixes of length k of the terminal strings derivable from α. If α derives a terminal string w whose length is less than k, then w itself is included in FIRST(α).

A special case of the FIRST function, which is important in the context of LR(k) grammars, is EFF, the *e-free* FIRST function. Formally, $\mathrm{EFF}_k^G(\alpha)$ (or EFF(α) where G and k are understood) is $\{w | \alpha \underset{rm}{\overset{*}{\Rightarrow}} wx$ where $|w| = k$ or $|w| < k$ and $x = e$, and the last step in the derivation $\alpha \underset{rm}{\overset{*}{\Rightarrow}} wx$, if it exists,[2] does not use a production of the form $A \to e\}$.

For example, if G is a grammar with productions $S \to SaSb$ and $S \to e$, then $\mathrm{FIRST}_1(S) = \{e, a\}$ while $\mathrm{EFF}_1(S)$ is empty.

The *augmented grammar* associated with a grammar $G = (N, \Sigma, P, S)$ is the grammar $G' = (N \cup \{S'\}, \Sigma, P \cup \{S' \to S\}, S')$ obtained by adding a new starting production $S' \to S$ to P where S' is a new nonterminal not in $N \cup \Sigma$.

A grammar G is LR(k) if the two rightmost derivations in the associated augmented grammar, i.e.,

$$S' \underset{rm}{\overset{*}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta w, \tag{1}$$

and

$$S' \underset{rm}{\overset{*}{\Rightarrow}} \gamma B x \underset{rm}{\Rightarrow} \alpha\beta y, \tag{2}$$

together with the condition $\mathrm{FIRST}_k^G(w) = \mathrm{FIRST}_k^G(y)$, imply that $\gamma B x = \alpha A y$, i.e., that $\alpha = \gamma$, $A = B$ and $x = y$.

Informally, a grammar is LR(k) if given a right sentential form, say $\alpha\beta z$, where z could be w or y in (1) and (2) above, we can determine that β is the string introduced at the previous derivation step by examining only α, β and $\mathrm{FIRST}_k(z)$. Moreover, we can uniquely determine that the nonterminal which was replaced by β at that step was A.

One consequence of this definition which is not immediately obvious is that for each LR(k) grammar $G = (N, \Sigma, P, S)$, we can mechanically generate a deterministic parser that uses a pushdown list (stack) and a finite set of LR(k) tables as follows. At each step the parser decides whether to shift an LR(k) table on top of the pushdown list and read another input symbol or to call for a reduction using one of the productions in P. If a reduction is called for, there will be a string of LR(k) tables on top of the pushdown list, and this string of tables will correspond naturally to the right side of the given production. In the reduction this string is replaced by an LR(k) table.

An LR(k) *table*[3] is a pair of functions $T = (f, g)$ such that:

[1] $|\beta|$ stands for the length of β.

[2] That is, if α is not itself a terminal string.

[3] The term "state" is used in [9], [10], [17], [19].

(i) f, the *parsing action* function, maps *lookahead strings*, i.e., strings in Σ^* of length at most k, to the actions **shift**, **error**, **accept**, and **reduce** i, where i is the number of a production in P. The action function decides between shift and reduce at each step, with alternatives of halting the parse by accepting or declaring an error.

(ii) g, the *goto* function, maps $N \cup \Sigma$ to the set of tables and the word error. It is essentially the next state function of a finite automaton whose input symbols are $N \cup \Sigma$.

These tables are used to parse input strings as follows. The pushdown list holds a string of table names,[4] say $T_0T_1 \cdots T_m$, $m \geqq 0$, where the T_0 is a specific table, the *initial table*, and $T_i = (f_i, g_i)$ for $0 \leqq i \leqq m$.

Let us suppose that w is the unexpended suffix of the original input string and that we are observing the parser at a time either when $m = 0$, i.e., at the beginning, or after it has just performed a reduction. We determine the lookahead string u by finding $\mathrm{FIRST}_k(w)$. Then we apply to the lookahead string u the parsing action function f_m associated with T_m. If that action is shift, the next input symbol, say a,[5] is removed from the input and the table $g_m(a)$ is placed on top of the stack. The pushdown list thus becomes $T_0T_1 \cdots T_mT_{m+1}$ where $T_{m+1} = g_m(a)$. In the same manner, we remove symbols from the input and place tables on the pushdown list, as long as the action of the current table on top of the pushdown list applied to the next k remaining input symbols is shift. The actions error and accept have the obvious meaning.

Suppose that we arrive at a situation in which $T_0T_1 \cdots T_r$ is on the pushdown list and the lookahead string is u, such that the action of T_r on u is reduce i. Suppose production i is $A \to \alpha$. Then $|\alpha|$ tables are removed from the pushdown list,[6] leaving some table T_n, $0 \leqq n \leqq r$, on top. If $T_n = (f_n, g_n)$, then table $T = g_n(A)$ is placed on top of the pushdown list, which now contains $T_0T_1 \cdots T_nT$.

We are now "back where we started," having just performed a reduction. The process repeats, until either an error or accept action is called for.

Example 1. Consider the grammar G with the productions

$$\begin{aligned}&(1)\ S \to AA\\&(2)\ A \to \alpha A\\&(3)\ A \to b\end{aligned}$$

We display a set of LR(1) tables for an LR(1) parser for G in Fig. 1. T_0 is the initial table. LR(k) tables throughout this paper are shown as rows, with columns for the arguments of f and g. The following code is used for table entries:

$$x = \textbf{error},\ a = \textbf{accept},\ s = \textbf{shift},\ i = \textbf{reduce}\ i.$$

Each LR(1) table is one row of Fig. 1. For example, if T_0 is on top of the stack and a is the current lookahead symbol, then the parsing action called for is to shift (the "s" entry in the first column of the first row) and to place table T_3 on top of the stack (the sixth entry in the first row). An LR(1) parser using Fig. 1 would parse the input string abb as shown in Table 1.

[4] Many descriptions of LR(k) parsing place alternating grammar symbols and tables on the stack, but the grammar symbols are known to be superfluous.

[5] If $k > 0$, then a is the first symbol of u. If there is no next input symbol, the parser halts and reports error.

[6] If the length of α exceeds the number of tables on the stack, the parser halts and reports error, but this will never happen if the tables are created by the methods to be discussed.

Table	Parsing action function f			Goto function g			
	Lookahead String			Grammar symbol			
	a	b	e	S	A	a	b
T_0	s	s	x	T_1	T_2	T_3	T_4
T_1	x	x	a	x	x	x	x
T_2	s	s	x	x	T_5	T_6	T_7
T_3	s	s	x	x	T_8	T_3	T_4
T_4	3	3	x	x	x	x	x
T_5	x	x	1	x	x	x	x
T_6	s	s	x	x	T_9	T_6	T_7
T_7	x	x	3	x	x	x	x
T_8	2	2	x	x	x	x	x
T_9	x	x	2	x	x	x	x

FIG. 1. *Set of LR*(1) *tables*

We can construct a parser of this type for each LR(k) grammar G. To do so, we need to construct a set of LR(k) tables for G. There are several ways of generating a suitable set of LR(k) tables for G. We shall consider two techniques in this paper —Knuth's (canonical) method and DeRemer's simple LR method.

The *canonical* set of LR(k) tables can be constructed from G by generating what we call *sets of LR*(k) *items*.[7] An LR(k) *item* for G is a pair $[A \to \alpha \cdot \beta, u]$, where $A \to \alpha\beta$ is a production in the augmented grammar, and u is a string in Σ^*, with $|u| \leqq k$. Item $[A \to \alpha \cdot \beta, u]$ is said to be *valid* for viable prefix γ if there exists a derivation $S' \overset{*}{\Rightarrow} \delta A w \underset{rm}{\Rightarrow} \delta\alpha\beta w$ in the augmented grammar such that $\gamma = \delta\alpha$, and $u = \mathrm{FIRST}_k(w)$. Intuitively, if we have just seen some prefix of an input sentence that can be derived from γ and $[A \to \alpha{\cdot}\beta, u]$ is valid for γ, then we will expect to see next on the input a string that can be derived from βu.

TABLE 1

Stack	Input	Parsing action
T_0	abb	shift
T_0T_3	bb	shift
$T_0T_3T_4$	b	reduce by $A \to b$
$T_0T_3T_8$	b	reduce by $A \to aA$
T_0T_2	b	shift
$T_0T_2T_7$	e	reduce by $A \to b$
$T_0T_2T_5$	e	reduce by $S \to AA$
T_0T_1	e	accept

[7] Items are called "Partial states" in Knuth [17].

The key to constructing the canonical LR(k) parser for an LR(k) grammar G is to first construct the sets of valid LR(k) items for all the viable prefixes of G'. The number of distinct sets of items is clearly finite.

Let $\mathscr{A}$ be a set of LR(k) items. The *closure* of $\mathscr{A}$ is defined to be the least set $\mathscr{A}'$, satisfying:

(i) $\mathscr{A} \subseteq \mathscr{A}'$;

(ii) If $[A \to \alpha{\cdot}B\beta, u]$ is in $\mathscr{A}'$, then $[B \to {\cdot}\gamma, v]$ is in $\mathscr{A}'$, for all $B \to \gamma$ in P and v in $\mathrm{FIRST}_k(\beta u)$.

The function GOTO($\mathscr{A}, X$) is defined as follows. Let $\mathscr{A}$ be a set of items and let $\mathscr{A}'$ be the set of items $\{[B \to \alpha X \cdot \beta, u] | [B \to \alpha \cdot X\beta, u] \text{ is in } \mathscr{A}\}$. That is, find all items in $\mathscr{A}$ within an X immediately to the right of a dot. $\mathscr{A}'$ is this set of items with the dot shifted to the right of X. The set GOTO($\mathscr{A}, X$) is the closure of $\mathscr{A}'$.

LEMMA 1 (from [17], [4]). *If $\mathscr{A}$ is the set of LR(k) items valid for α, then* GOTO($\mathscr{A}, X$) *is the set of* LR(k) *items valid for αX.*

ALGORITHM 1 (Constructing the canonical collection). The *canonical collection* $\mathscr{S}$ of the *sets of valid LR(k) items* for an LR(k) grammar $G = (N, \Sigma, P, S)$ can be computed as follows.

1. Let $\mathscr{A}_0$ be constructed by taking the closure of the single item $[S' \to {\cdot}S, e]$, where $S' \to S$ is the starting production in the augmented grammar. $\mathscr{A}_0$ is called the *initial* set of items.

2. Begin with $\mathscr{S} = \{\mathscr{A}_0\}$. Apply step 3 until no new sets of items can be added to $\mathscr{S}$.

3. If $\mathscr{A}$ is in $\mathscr{S}$, add GOTO($\mathscr{A}, X$) to $\mathscr{S}$ for all X in $N \cup \Sigma$, if GOTO($\mathscr{A}, X$) is nonempty and not already in $\mathscr{S}$.

$\mathscr{S}$ is the canonical collection.

Example 2. Let us reconsider the grammar G with the productions

(1) $S \to AA$
(2) $A \to \alpha A$
(3) $A \to b$

We list the canonical collection of sets of LR(1) items (brackets deleted) for G. The notation $B \to \alpha{\cdot}\beta,\ c_1/c_2/\cdots/c_m$ is short for the items $[B \to \alpha{\cdot}\beta, c_1], \ldots, [B \to \alpha{\cdot}\beta, c_m]$.

$$\mathscr{A}_0: \begin{cases} S' \to {\cdot}S, & e \\ S \to {\cdot}AA, & e \\ A \to {\cdot}aA, & a/b \\ A \to {\cdot}b, & a/b \end{cases}$$

$$\mathscr{A}_1: \quad S' \to S{\cdot}, \quad e$$

$$\mathscr{A}_2: \begin{cases} S \to A{\cdot}A, & e \\ A \to {\cdot}aA, & e \\ A \to {\cdot}b, & e \end{cases}$$

$$\mathscr{A}_3: \begin{cases} A \to a{\cdot}A, & a/b \\ A \to {\cdot}aA, & a/b \\ A \to {\cdot}b, & a/b \end{cases}$$

$$\mathscr{A}_4\colon\ A \to b\cdot,\quad a/b$$

$$\mathscr{A}_5\colon\ S \to AA\cdot, e$$

$$\mathscr{A}_6\colon\begin{cases} A \to a\cdot A, & e \\ A \to \cdot aA, & e \\ A \to \cdot b, & e \end{cases}$$

$$\mathscr{A}_7\colon\ A \to b\cdot,\quad e$$

$$\mathscr{A}_8\colon\ A \to aA\cdot,\ a/b$$

$$\mathscr{A}_9\colon\ A \to aA\cdot,\ e$$

For example, $\mathscr{A}_0$ is computed by taking the closure of $[S' \to \cdot S, e]$. Since there is an S to the right of the dot, we add $[S \to \cdot AA, e]$ to $\mathscr{A}_0$. Since there is now an item with A to the right of the dot, we also add $[A \to \cdot aA, a/b]$ and $[A \to \cdot b, a/b]$ to $\mathscr{A}_0$. The second components of the items are a or b since A follows the A immediately to the right of the dot, and $\mathrm{FIRST}_1(A) = \{a, b\}$. $\mathscr{A}_1$ is $\mathrm{GOTO}(\mathscr{A}_0, S)$. $\mathscr{A}_2$ is $\mathrm{GOTO}(\mathscr{A}_0, A)$ and is found by taking the closure of $\{S \to A\cdot A, e\}$. This completes Example 2.

We can extend the GOTO function to strings in $(N \cup \Sigma)^*$ in the obvious way. That is,

(i) $\mathrm{GOTO}(\mathscr{A}, e) = \mathscr{A}$

(ii) $\mathrm{GOTO}(\mathscr{A}, \alpha X) = \mathrm{GOTO}(\mathrm{GOTO}(\mathscr{A}, \alpha), X)$.

It should be clear by Lemma 1 that if A_0 is the initial set of items, then $\mathrm{GOTO}(\mathscr{A}_0, \alpha)$ is nonempty if and only if α is a viable prefix to the grammar at hand.

ALGORITHM 2 (Constructing an LR(k) table from a set of LR(k) items). The following algorithm can be used to construct an LR(k) table $T = (f, g)$ from a set of LR(k) items $\mathscr{A}$. The parsing action function f is constructed as follows.

1. If $[A \to \alpha\cdot, u]$ is in $\mathscr{A}$, and $A \to \alpha$ is not $S' \to S$, then $f(u) =$ **reduce** i, where i is the number of production $A \to \alpha$.
2. If $[A \to \alpha\cdot\beta, u]$ is in $\mathscr{A}$, $\beta \neq e$, then $f(v) =$ **shift** for all v in $\mathrm{EFF}(\beta u)$.
3. If $[S' \to S\cdot, e]$ is in $\mathscr{A}$, then $f(e) =$ **accept**.
4. $f(u) =$ **error** otherwise.

If $f(u)$ is not uniquely defined for some u in Σ^* such that $|u| \leqq k$, then a *parsing action conflict* is generated by $\mathscr{A}$ and no table is produced for $\mathscr{A}$.

The GOTO function g is constructed as follows.

5. $g(X)$ is the name of the table constructed from $\mathrm{GOTO}(\mathscr{A}, X)$ whenever $\mathrm{GOTO}(\mathscr{A}, X)$ is not empty.
6. $g(x) =$ **error** if $\mathrm{GOTO}(\mathscr{A}, X)$ is empty.

The set of LR(k) tables constructed from the canonical collection of sets of LR(k) items for an LR(k) grammar G will be called the *canonical set of* LR(k) *tables* for G.

The table constructed from the initial set of items is called the *initial* table.

It is well known that a grammar G is LR(k) if and only if no parsing action conflicts are generated by any set in the canonical collection of sets of LR(k) items for G [17], [5].

Example 3. The canonical set of LR(1) tables for the grammar of Example 1 was given in Fig. 1.

The canonical set of LR(k) tables for a grammar is often impractically large, even if $k = 1$. As we have mentioned, several methods of producing smaller sets of LR(k) tables from an LR(k) grammar have evolved. We consider one of them here, the "simple LR" method (DeRemer [10]). The next algorithm defines the simple LR(k) method for the case $k = 1$.

DEFINITION. If $G = (N, \Sigma, P, S)$ is a grammar, and A is in N, then $\mathrm{FOLLOW}^G(A) = \{a | S \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w$ and $a = \mathrm{FIRST}_1(w)$ for some α and $w\}$. That is, $\mathrm{FOLLOW}^G(A)$ is the set of terminals which may follow A in a right sentential form, with the empty string e included if A can be the rightmost symbol of a right sentential form. We delete the G from FOLLOW^G when no confusion results.

ALGORITHM 3 (The SLR(1) method). The SLR(1) method of constructing a set of LR(1) tables for a grammar $G = (N, \Sigma, P, S)$ can be summarized as follows.

1. Using Algorithm 1, construct $\mathscr{S}_0$, the canonical collection of the sets of LR(0) items for G.

2. Replace every item of the form $[A \to \beta\cdot, e]$ in each set $\mathscr{A}$ in $\mathscr{S}$ by $[A \to \beta\cdot, a]$ for all a in FOLLOW(A). Let $\mathscr{S}$ be the resulting collection of sets of LR(1) items.

3. Using Algorithm 2, construct an LR(1) table for each $\mathscr{A}$ in $\mathscr{S}$. The GOTO entry for table T on symbol X is the table constructed from GOTO($\mathscr{A}'$, X), where $\mathscr{A}'$ is the set of LR(0) items from which T is constructed.

If no parsing action conflict occurs, then the resulting set of LR(1) tables is called the SLR(1) *set of tables* for G and the grammar G is called an SLR(1) *grammar*. (Thus, DeRemer's algorithm forms a definition of SLR grammars.)

If G is an SLR(1) grammar, then the SLR(1) set of tables for G is equivalent in the sense of [5] and [6] to the canonical set of LR(1) tables for G.

Example 4. Let us again consider the grammar

$$\begin{array}{ll}(1) & S \to AA \\ (2) & A \to aA \\ (3) & A \to b\end{array}$$

of Example 1. The canonical collection of sets of LR(0) items, with "e" deleted from each item, is:

$$\mathscr{B}_0: \begin{cases} S' \to \cdot S \\ S \to \cdot AA \\ A \to \cdot aA \\ A \to \cdot b \end{cases}$$

$$\mathscr{B}_1: \ S' \to S\cdot$$

$$\mathscr{B}_2: \begin{cases} S \to A\cdot A \\ A \to \cdot aA \\ A \to \cdot b \end{cases}$$

$$\mathscr{B}_3 : \begin{cases} A \to a{\cdot}A \\ A \to {\cdot}aA \\ A \to {\cdot}b \end{cases}$$

$$\mathscr{B}_4 : \ A \to b{\cdot}$$

$$\mathscr{B}_5 : \ S \to AA{\cdot}$$

$$\mathscr{B}_6 : \ A \to aA{\cdot}$$

The SLR(1) set of tables constructed from these sets of items is shown in Fig. 2.

Table	a	b	e	S	A	a	b
R_0	s	s	x	R_1	R_2	R_3	R_4
R_1	x	x	a	x	x	x	x
R_2	s	s	x	x	R_5	R_3	R_4
R_3	s	s	x	x	R_6	R_3	R_4
R_4	3	3	3	x	x	x	x
R_5	x	x	1	x	x	x	x
R_6	2	2	2	x	x	x	x

FIG. 2. *SLR*(1) *tables*

We shall now give a formal specification of the LR(k) parsing algorithm.

Let $\mathscr{T}$ be a set of LR(k) tables for a grammar $G = (N, \Sigma, P, S)$. We define the *action of the LR*(k) *parser using* $\mathscr{T}$ in the following manner. The LR(k) parser has a pushdown list and an input tape. A *configuration* of the parser is a pair $(T_0T_1 \cdots T_m, w)$, where T_0 is the initial table, $T_0, T_1, \cdots, T_m$ are in $\mathscr{T}$, and w is in Σ^*. The string $T_0T_1 \cdots T_m$ represents the pushdown list with table T_m on top. The string w represents the unexpended input. Let $T_i = (f_i, g_i)$ for $0 \leqq i \leqq m$, and let w represent the unexpended input. Let $T_i = (f_i, g_i)$ for $0 \leqq i \leqq m$, and let $u = \mathrm{FIRST}_k(w)$ be the current lookahead string.

1. If $f(u) =$ **shift**, and $g_m(a) = T$, where $w = aw'$, then we write $(T_0T_1 \cdots T_m, w) \vdash (T_0T_1 \cdots T_mT, w')$.

2. If $f(u) =$ **reduce** i, production i is $A \to \alpha$ and α is of length $r \geqq 0$, then we write $(T_0T_1 \cdots T_m, w) \vdash (T_0T_1 \cdots T_{m-r}T, w)$, where T is $g_{m-r}(A)$.

3. If $f(u)$ is **error** or **accept**, then there is no configuration C such that $(T_0T_1 \cdots T_m, w) \vdash C$.

4. If $f(u) =$ **accept**, $m = 1$, and $w = e$, then the configuration (T_0T_1, e) is said to be the *accepting configuration*. (It is easy to show that there is only one accepting configuration.)

An *initial configuration* of the parser is one of the form (T_0, w). Let $\vdash^*$ be the reflexive and transitive closure of $\vdash$, and let $\vdash^i$ be the composition of $\vdash$ with itself i times. A configuration C such that $(T_0, w) \vdash^* C$ for some w is said to be *accessible*.

A set of tables $\mathcal{T}$ is said to be *valid for G* if and only if the accepting configuration is accessible from (T_0, w) exactly when w is in $L(G)$.

It is straightforward to show that the canonical set of LR(k) tables for an LR(k) grammar G is valid for G. Likewise, the SLR(1) set of tables for an SLR(1) grammar G is valid for G. If $\mathcal{T}$ is a canonical or SLR(1) set of tables for G, then for each w in $L(G)$ the sequence of reductions made by the parser using $\mathcal{T}$ is the right parse for w.

We should note that a grammar G may have many valid sets of LR(k) tables in addition to the canonical and SLR sets.

3. "Don't care" entries.

Certain error entries in a set of LR(k) tables are never consulted by an LR(k) parser using $\mathcal{T}$. It is useful to distinguish these entries from those that can be consulted.

DEFINITION. We define *essential* entries as follows. Let $\mathcal{T}$ be a set of LR(k) tables for a grammar $G = (N, \Sigma, P, S)$. Let T_0 be the initial table in $\mathcal{T}$. Suppose $(T_0 T_1 \cdots T_m, w)$ is an accessible configuration of the LR(k) parser using $\mathcal{T}$, $\mathrm{FIRST}_k(w) = u$ and $T_i = (f_i, g_i)$ for $0 \leqq i \leqq m$.

(i) If $f_m(u) = \textbf{error}$, then $f_m(u)$ is an essential entry.

(ii) If $f_m(u) = \textbf{shift}$, $w = aw'$ and $g_m(a) = \textbf{error}$, then $g_m(a)$ is an essential entry.

(iii) If $f_m(u) = \textbf{reduce}\, i$, where production i is $A \to \alpha$, $|\alpha| = r$, and $g_{m-r}(A) = \textbf{error}$, then $g_{m-r}(A)$ is an essential entry.

If an error entry is not essential, then it is a *don't care entry* and from this point will be indicated by φ rather than x in an LR(k) table. Intuitively, a don't care entry in a set of LR(k) tables is one which may be changed arbitrarily without altering the action of the LR(k) parser using that set of tables. The following two lemmas pinpoint which error entries are don't cares in the canonical and SLR sets of tables.

LEMMA 2 [5], [6]. *In a canonical set of LR(1) tables:*

(a) *All error entries in the goto portion of each table are don't cares.*

(b) *An error entry in the action field of table T is essential if and only if either T is the initial table or the goto entry of some table on a terminal symbol.*

DEFINITION. We extend the GOTO function to tables and strings of grammar symbols as follows.

(i) $\mathrm{GOTO}(T, e) = T$ for all T.

(ii) $\mathrm{GOTO}(T, X_1 \cdots X_m)$ is that table T' which is the goto entry on X_m of the table $\mathrm{GOTO}(T, X_1 \cdots X_{m-1})$.

Some readers may wish to portray the GOTO function as a directed graph in which there is a path from node T to node T' that spells out the string $X_1 X_2 \cdots X_m$ if and only if $\mathrm{GOTO}(T, X_1 X_2 \cdots X_m) = T'$. The GOTO graph for the set of LR(1) tables in Fig. 1 is shown in Fig. 3.

Lemma 2 states that if $T' = \mathrm{GOTO}(T, a)$ for some terminal symbol a, then all error entries in the parsing action field of T' are essential.

We now define a function called NEXT, that will tell us, given a table T in a set of tables $\mathcal{T}$ and a production i, what tables in $\mathcal{T}$ can be on top of the stack immediately after the LR(k) parser using $\mathcal{T}$ has completed a reduce i action called for by T.

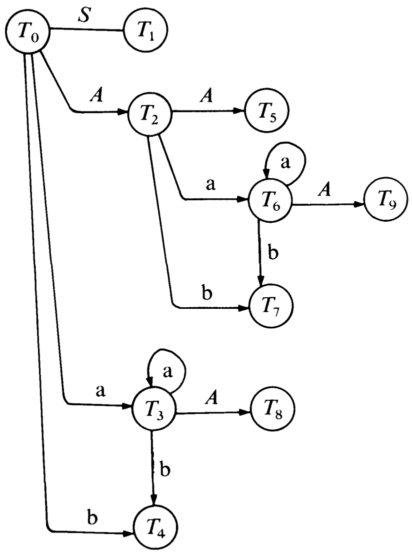


FIG. 3. GOTO *graph*

DEFINITION. Let T be a table in a set of tables $\mathcal{T}$ and $A \to \alpha$ the production whose number is i. Then NEXT(T, i) is $\{T'$ in $\mathcal{T} |$ there is a table T'' in $\mathcal{T}$ such that GOTO $(T'', \alpha) = T$ and GOTO$(T'', A) = T'\}$.

In Example 1, NEXT$(T_9, 2) = \{T_5, T_9\}$. That is, production 2 is $A \to aA$. T'' may be T_2 or T_6, since GOTO$(T_2, aA) =$ GOTO$(T_6, aA) = T_9$. GOTO$(T_2, A) = T_5$, while GOTO$(T_6, A) = T_9$.

LEMMA 3. *In an SLR*(1) *set of tables* $\mathcal{T}$,

(a) *All error entries in the goto field of each table in* $\mathcal{T}$ *are don't cares.*

(b) *An error entry on input symbol a in the action field of table T is essential if and only if one of the following conditions holds.*

(i) *T is the initial table.*

(ii) *There is a table* $T' = (f', g')$ *in* $\mathcal{T}$ *such that* $T = g'(b)$ *for some b in* Σ.

(iii) *There is some table T' with T in* NEXT(T', i) *such that the action of T' on a is reduce i.*[8]

Proof. Part (a) is similar to the proof of Lemma 2 in [5], and we omit it. Note that as a consequence all errors are detected by the parsing action functions. The "only if" portion of part (b) follows from the observation that the three ways listed comprise the only ways that table T can appear on top of the pushdown list with a as the current input symbol. The proof of the "if" portion is similar to Lemma 2 in cases (i) and (ii), so we must show that if (iii) holds, then the action entry of T on a is essential.

Let $G = (N, \Sigma, P, S)$ be the grammar for which the tables are constructed, and let production i be $A \to \alpha$. Since T is in NEXT(T', i), there is some table T_1 such that

[8] This is the only condition which differs from Lemma 2. The reason for its presence is that the SLR parser can call for a reduction at certain times when the canonical parser declares an error.

GOTO(T_1, A) = T. Let $\mathscr{A}_1$ be the set of LR(0) items from which T_1 is constructed by the SLR algorithm. Then $\mathscr{A}_1$ must contain an item with A to the right of the dot, since GOTO($\mathscr{A}_1$, A) is nonempty.

We assume that every nonterminal derives a nonempty string, so let $A \Rightarrow \beta \overset{*}{\Rightarrow} x$, where $x \neq e$. (β may be α.) Now $[A \rightarrow \cdot\beta, e]$ must be in $\mathscr{A}_1$, since $\mathscr{A}_1$ has an item with A to the right of the dot and $\mathscr{A}_1$ must be closed.

Next, we infer that since the action of T' on a is to reduce by production $A \rightarrow \alpha$, a must be in FOLLOW(A). Hence, if B is the last symbol of any step in the derivation $A \Rightarrow \beta \overset{*}{\Rightarrow} x$, we must have a in FOLLOW(B).

Now we put these observations together. Let $\mathscr{A}_1$ be the set of valid items for viable prefix γ, and let $\gamma \overset{*}{\Rightarrow} w$. Then there is some word wxy with rightmost derivation

$$S \underset{rm}{\overset{*}{\Rightarrow}} \gamma A y \underset{rm}{\Rightarrow} \gamma\beta y \underset{rm}{\overset{*}{\Rightarrow}} \gamma x y \underset{rm}{\overset{*}{\Rightarrow}} wxy.$$

Consider what happens when we attempt to parse wxa. We must, since $x \neq e$, trace out this derivation in reverse, at least until w is reduced to γ. Then, since a is in FOLLOW(B) for each nonterminal B which is the last symbol of any step in the derivation $A \underset{rm}{\Rightarrow} \beta \underset{rm}{\overset{*}{\Rightarrow}} x$, we know that x will be reduced to A even if a becomes the first (and only) input symbol. At this point, T is the table name on top of the stack and we exercise the error entry of T on input a. We thus have demonstrated this entry to be essential, and the proof is complete.

4. Eliminating reductions by single productions. A production of the form $A \rightarrow B$ where A and B are nonterminals is called a single production. Productions of this nature frequently arise when a context-free grammar is used to describe the precedence levels of operators in a programming language.

For example, consider the following grammar G_0 for arithmetic expressions containing the binary operators + and $*$. We assume $*$ has higher precedence than +, meaning that $a + a * a$ is to be interpreted as $a + (a * a)$ rather than $(a + a) * a$.

$G_0 = (\{E, T, F\}, \{a, +, *, (,)\}, P, E)$, where P is

(1) $E \rightarrow E + T$
(2) $E \rightarrow T$
(3) $T \rightarrow T * F$
(4) $T \rightarrow F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow a$

We can think of the nonterminals E, T and F generating expressions on different levels reflecting the precedence levels of the operators. E generates the first level of expressions. These are strings of T's separated by +'s. The operator + is on the first precedence level. T generates the second level of expressions consisting of F's separated by $*$'s. The third level of expressions are those generated by F, and these we can consider to be the primitive expressions.

When we parse the string $a + a * a$ according to G_0, we must first parse $a * a$ as a T before combining this T with the first a into an E.

The only function served by the two single productions $E \rightarrow T$ and $T \rightarrow F$ is to permit an expression on a higher level to be trivially reduced to an expression on a lower level.

Some programming languages have operators on twelve or more different precedence levels. If we are parsing according to a grammar that reflects many precedence levels, the parser will often make long sequences of reductions by single productions. We can speed up the parsing process considerably if we can eliminate these sequences of reductions.

Moreover, in most practical cases, single productions of this nature have no "semantic significance". That is, in syntax directed translation schemes as in [3], [18], or [20], when the underlying production is of the form $A \to B$, the translations of A on the left are often equal to the translations of B on the right. Thus, as far as the output of a compiler is concerned, it does not make the slightest difference whether the parser physically makes the reduction or not.

We shall now show how we can modify a set of LR(k) tables so that the LR(k) parser using this set of tables can avoid making reductions by some single productions. Let $\mathscr{T}$ be a set of LR(k) tables. We say two tables T and T' *disagree* in an entry if they have different non-φ values for that entry.

Suppose that an LR(k) parser using a set of tables $\mathscr{T}$ is in a configuration in which the two tables TT_2 are on top of the pushdown list (with T_2 topmost). Suppose that table T_2 calls for a reduction by production $A \to B$ in certain of its parsing action entries. Finally, suppose that the goto of table T on A is T_1 and that T_1 and T_2 never disagree except in those action entries where T_2 calls for a reduction by $A \to B$.

If we don't care whether the reduction of B to A is actually carried out or not, we could then create a new table T'_1, which agrees with T_1 when T_2 calls for a reduction by $A \to B$, and agrees with whichever of T_1 or T_2 is not φ otherwise. T'_1 becomes the goto of table T on both A and B, and replaces T_1 everywhere (which it may legitimately do, since it differs from T_1 only where T_1 is φ). As a result, T_2 never appears on top of T in the stack.

If we can do this modification for each table T which can appear below T_2 on the stack, then we have effectively eliminated T_2, since it will never be placed on the stack. Moreover, we have saved the time necessary to replace T_2 on the stack when it calls for a reduction by $A \to B$.

We show that it is possible, in this way, to eliminate sequences of reductions by single productions. Substantial savings, both in number of tables and in computation time, are possible in parsers for practical grammars.

ALGORITHM 4. Let $\mathscr{T}$ be a set of LR(1) tables for grammar $G = (N, \Sigma, P, S)$, and suppose $A \to B$ is a single production in P whose reductions we want to avoid. Let T be in $\mathscr{T}$, with $\mathrm{GOTO}(T, A) = T_1$ and $\mathrm{GOTO}(T, B) = T_2$. Let $T_1 = (f_1, g_1)$, $T_2 = (f_2, g_2)$, and suppose that the following conditions hold.

(i) If $g_1(X) \neq \varphi$ and $g_2(X) \neq \varphi$, then $g_1(X) = g_2(X)$.

(ii) If $f_1(a) \neq \varphi$ and $f_2(a)$ is neither φ nor **reduce** i, where production i is $A \to B$, then $f_1(a) = f_2(a)$. That is, T_1 and T_2 may disagree only where f_2 is **reduce** i.

Now modify T_1 by the following rules:

1. Set $f_1(a)$ equal to $f_2(a)$ whenever the latter is not φ or **reduce** i.
2. Set $g_1(X)$ to $g_2(X)$ whenever the latter is not φ.
3. Then, modify T by setting its goto on B to T_1.

Algorithm 4 causes the LR(k) parser to avoid reductions by $A \to B$ that were called for by table T_2 in those cases where table T is below T_2 on the stack. To see

Table	a	$+$	$*$	(	)	e	E	T	F	a	$+$	$*$	(	)
T_0	s	x	x	s	x	x	T_1	T_2	T_3	T_5	φ	φ	T_4	φ
T_1	φ	s	φ	φ	x	a	φ	φ	φ	φ	T_6	φ	φ	φ
T_2	φ	2	s	φ	2	2	φ	φ	φ	φ	φ	T_7	φ	φ
T_3	φ	4	4	φ	4	4	φ	φ	φ	φ	φ	φ	φ	φ
T_4	s	x	x	s	x	x	T_8	T_2	T_3	T_5	φ	φ	T_4	φ
T_5	x	6	6	x	6	6	φ	φ	φ	φ	φ	φ	φ	φ
T_6	s	x	x	s	x	x	φ	T_9	T_3	T_5	φ	φ	T_4	φ
T_7	s	x	x	s	x	x	φ	φ	T_{10}	T_5	φ	φ	T_4	φ
T_8	φ	s	φ	φ	s	x	φ	φ	φ	φ	T_6	φ	φ	T_{11}
T_9	φ	1	s	φ	1	1	φ	φ	φ	φ	φ	T_7	φ	φ
T_{10}	φ	3	3	φ	3	3	φ	φ	φ	φ	φ	φ	φ	φ
T_{11}	x	5	5	x	5	5	φ	φ	φ	φ	φ	φ	φ	φ

FIG. 4. *SLR*(1) *tables*

this, let us consider the effect of Algorithm 4. If the φ's truly represent don't cares, then the modified T_1 can substitute for T_2, and the only change in the parser's action will be that T_1 will not call for a reduction by $A \rightarrow B$ where T_2 would have done so. But since T appears below T_1 on the pushdown list in every situation where T_1 substitutes for T_2, we know that after T_2 called for a reduction by $A \rightarrow B$, T_1 would replace T_2 anyway. Thus, the only effect of Algorithm 1 is that a reduction by $A \rightarrow B$ has possibly been omitted.

Example 5. Let us consider the grammar G_0 introduced at the beginning of § 4. G_0 is SLR(1), and the SLR(1) set of tables with φ's indicating all don't cares is shown in Fig. 4.

The single productions are (2) $E \rightarrow T$ and (4) $T \rightarrow F$. The only table calling for a reduction by (4) is T_3. The tables which can appear below T_3 on the stack are T_0, T_4 and T_6, because these are the tables U for which $\text{GOTO}(U, F) = T_3$. Since $\text{GOTO}(T_0, T)$[9] $= \text{GOTO}(T_4, T) = T_2$ and $\text{GOTO}(T_6, T) = T_9$, we should compare T_3 with both T_2 and T_9. Since all actions of T_3 are φ or **reduce** 4, and all

Table	a	$+$	$*$	(	)	e	E	T	F	a	$+$	$*$	(	)
T_0	s	x	x	s	x	x	T_1	T_2	$\mathbf{T_2}$	T_5	φ	φ	T_4	φ
T_1	φ	s	φ	φ	x	a	φ	φ	φ	φ	T_6	φ	φ	φ
T_2	φ	2	s	φ	2	2	φ	φ	φ	φ	φ	T_7	φ	φ
T_4	s	x	x	s	x	x	T_8	T_2	$\mathbf{T_2}$	T_5	φ	φ	T_4	φ
T_5	x	6	6	x	6	6	φ	φ	φ	φ	φ	φ	φ	φ
T_6	s	x	x	s	x	x	φ	T_9	$\mathbf{T_9}$	T_5	φ	φ	T_4	φ
T_7	s	x	x	s	x	x	φ	φ	T_{10}	T_5	φ	φ	T_4	φ
T_8	φ	s	φ	φ	s	x	φ	φ	φ	φ	T_6	φ	φ	T_{11}
T_9	φ	1	s	φ	1	1	φ	φ	φ	φ	φ	T_7	φ	φ
T_{10}	φ	3	3	φ	3	3	φ	φ	φ	φ	φ	φ	φ	φ
T_{11}	x	5	5	x	5	5	φ	φ	φ	φ	φ	φ	φ	φ

FIG. 5. *First modification*

[9] Do not confuse the subscripted table names, e.g., T_0, with the nonterminal T which stands for "term."

goto entries are φ, the conditions of Algorithm 1 are met, and in fact, no modification to T_2 or T_9 is needed. We may thus replace T_3 by T_2 in the goto of T_0 and T_4 and replace T_3 by T_9 in the goto of T_6. Since T_3 no longer appears in any goto, it may be eliminated. The resulting tables appear in Fig. 5.

Now let us turn our attention to T_2, which calls for a reduction by production (2), $E \to T$. After such a reduction, either T_1 or T_8 could be the next table on top of the stack, and we must compare T_2 with these. On input $*$, the action of T_2 is **shift**, so we must check that T_1 and T_8 have φ there, which they do. We than replace the entries of T_1 and T_8 on $*$ by **shift**. Also, the goto of T_2 on $*$ is T_7, so we must check that the goto entries of T_1 and T_8 on $*$ are φ, which we already know. These entries are replaced by T_7. Then, the T_2 entries in the goto of T_0 are replaced by T_1 and those of T_4 are replaced by T_8. It is now possible to eliminate T_2. The resulting set of tables is shown in Fig. 6, and Example 5 is complete.

Table	a	$+$	$*$	(	)	e	E	T	F	a	$+$	$*$	(	)
T_0	s	x	x	s	x	x	T_1	$\mathbf{T_1}$	$\mathbf{T_1}$	T_5	φ	φ	T_4	φ
T_1	φ	s	**s**	φ	x	a	φ	φ	φ	φ	T_6	$\mathbf{T_7}$	φ	φ
T_4	s	x	x	s	x	x	T_8	$\mathbf{T_8}$	$\mathbf{T_8}$	T_5	φ	φ	T_4	φ
T_5	x	6	6	x	6	6	φ	φ	φ	φ	φ	φ	φ	φ
T_6	s	x	x	s	x	x	φ	T_9	$\mathbf{T_9}$	T_5	φ	φ	T_4	φ
T_7	s	x	x	s	x	x	φ	φ	T_{10}	T_5	φ	φ	T_4	φ
T_8	φ	s	**s**	φ	s	x	φ	φ	φ	φ	T_6	$\mathbf{T_7}$	φ	T_{11}
T_9	φ	1	s	φ	1	1	φ	φ	φ	φ	φ	T_7	φ	φ
T_{10}	φ	3	3	φ	3	3	φ	φ	φ	φ	φ	φ	φ	φ
T_{11}	φ	5	5	x	5	5	φ	φ	φ	φ	φ	φ	φ	φ

FIG. 6. *Second modification*

We could also treat production (6) $F \to a$ as a single production and eliminate table T_5 from Fig. 6 (because tables T_1, T_8, T_9 and T_{10} are all compatible with T_5). However, it is possible that production $F \to a$ is important when translation is done, so we might not want to make this elimination.

There is another simplification that is now possible in Fig. 6. The goto columns for E, T, F (and a if we had eliminated table T_5) are compatible and could be merged into a single column without affecting the behavior of the LR(1) parsing algorithm using this set of tables.

It should be observed that G_0 is an operator precedence grammar. It has been shown that for each operator precedence grammar, we can build an LR(1) parser with one column for the goto entry on all nonterminals (El Djabri [11]). However, this parser may not detect errors at the same point on the input that the canonical or SLR parsers do. In the case of G_0, we can obtain the parser of [11] using our transformation eliminating single productions and then merging the columns for nonterminals. Thus, no postponement of error detection occurs for G_0.

We do not believe that any general result along these lines can be shown. However, experiments with a small sample of practical grammars indicate that often there are several groups of mutually compatible goto columns for nonterminals. If there are k such groups we conjecture that we can regard the grammar as being k operator precedence grammars interacting in a simple way.

5. Eliminating single productions from canonical tables. In Example 5 we saw that Algorithm 4 completely eliminates reductions by single productions from the SLR(1) tables for G_0. This is no coincidence, as we shall presently show. In this section we shall concentrate on the canonical set of tables and show that they can always be modified by Algorithm 4 so that all reductions by single productions are eliminated, provided no nonterminal has more than one single production. The proof depends on showing that, in this case, T_1 and T_2 in Algorithm 4 can never disagree. In the next section we will prove the same result for SLR(1) sets of tables.

The first lemma shows that there can be no conflict in the goto fields of T_1 and T_2 of Algorithm 1.

LEMMA 4. *Let $G = (N, \Sigma, P, S)$ be an LR(1) grammar such that $A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_n \Rightarrow B$ is a derivation of single productions for $n \geqq 1$. Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be the sets of valid LR(1) items for viable prefixes γA_1 and γB, respectively. If $[C \to \alpha \cdot X\beta, a]$ is in $\mathscr{A}_1$ and $[D \to \alpha' \cdot Y\beta', b]$ is in $\mathscr{A}_2$, then $X \neq Y$.*

Proof. Suppose $X = Y$, and let $Y \overset{*}{\underset{rm}{\Rightarrow}} c\delta$ for some δ.[10] If the last step in this derivation does not use some production of the form $E \to e$, then c is in EFF($Y\beta'$), so the action of the table T_2 constructed from $\mathscr{A}_2$ must be to shift on c. If the derivation $Y \overset{*}{\underset{rm}{\Rightarrow}} c\gamma$ does use $E \to e$ at the last step, then the action of T_2 on c must be to reduce by $E \to e$ (i.e., $[E \to \cdot, c]$ is in $\mathscr{A}_2$).

Now let us consider the item $[C \to \alpha \cdot X\beta, a]$ in $\mathscr{A}_1$. First suppose $\alpha \neq e$. Then α ends in A_1 by the definition of valid item, i.e., $\alpha = \alpha_1 A_1$ for some α_1. Since $[C \to \alpha \cdot X\beta, a]$ is valid for γA_1, and c is in FIRST($X\beta a$), it follows that $[A_n \to B\cdot, c]$ is valid for γB. But then, the action of T_2 on c should be to reduce by $A_n \to B$. We have a contradiction of the LR(1)-ness of G in the case $\alpha = \alpha_1 A_1$.

Now suppose $\alpha = e$. Then in order for $[C \to \cdot X\beta, a]$ to be in $\mathscr{A}_1$, there must be some item of the form $[D \to \alpha'' A_1 \cdot F\beta'', d]$ in $\mathscr{A}_1$, such that $F \overset{*}{\underset{rm}{\Rightarrow}} Cw$ for some w. But since c is in FIRST($X\beta a$), we must have c in FIRST($F\beta'' d$), and so again we conclude that $[A_n \to B\cdot, c]$ is in T_2. As in the previous case, we contradict the LR(1)-ness of G, and so we may conclude the lemma.

LEMMA 5. *Let $G = (N, \Sigma, P, S)$ be an LR(1) grammar in which $A_1 \Rightarrow \cdots \Rightarrow A_n \Rightarrow B$ is a derivation of single productions, $n \geqq 1$. Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be the sets of valid LR(1) items for viable prefixes γA_1 and γB, respectively. Suppose $[C \to \alpha \cdot \beta, a]$ and $[D \to \alpha' \cdot \beta', b]$ are in $\mathscr{A}_1$ and $\mathscr{A}_2$, respectively, where $[D \to \alpha' \cdot \beta', b]$ is not $[A_n \to B\cdot, b]$. Then* EFF(βa) $\cap$ EFF($\beta' b$) *is empty.*

Proof. Suppose c is in EFF(βa) $\cap$ EFF($\beta' b$). Then the action on c of the table constructed from $\mathscr{A}_2$ is to shift if $\beta' \neq e$ and to reduce by $D \to \alpha'$ if $\beta' = e$. As in Lemma 4, we can argue that there must be an item $[F_2 \to \delta_1 A_1 \cdot \delta_2, d]$ in $\mathscr{A}_1$ (the case in which this item is $[C \to \alpha \cdot \beta, a]$ is not ruled out) such that c is in EFF($\delta_2 d$). Thus, there is a right sentential form $\gamma A_1 cw$ for some w. We may conclude that $[A_n \to B\cdot, c]$ is valid for γB, so the action of the table for $\mathscr{A}_2$ on c should be to reduce

[10] Recall that we are assuming that if Y is a nonterminal, then it derives at least one nonempty string.

by $A_n \to B$. We therefore have two "correct" actions for this table, and G is not LR(1).

Since an LR(k) grammar is unambiguous [17], [4], we can find an ordering of the nonterminals $A_1, A_2, \cdots, A_r$ such that if $A_i \to A_j$ is a single production, then $i < j$. (That is, if such an ordering were impossible, there would be some nonterminal A which derived itself in one or more steps, and the grammar would be ambiguous.) We can give an algorithm which uses such an order to apply Algorithm 4 repeatedly.

ALGORITHM 5. Let $\mathcal{T}$ be a set of LR(1) tables for grammar $G = (N, \Sigma, P, S)$ and let $A_1, \cdots, A_r$ be a linear order on N such that if $A_i \to A_j$ is a single production, then $i < j$. Then for $j = 1, 2, \cdots, r$ in turn, do the following.

Find pairs of tables T_1 and T_2 such that $A_i \to A_j$ is a single production, and for some table T, $\mathrm{GOTO}(T, A_i) = T_1$ and $\mathrm{GOTO}(T_1, A_j) = T_2$. If T_1 and T_2 can be merged by Algorithm 4, do so.

We shall now show that Algorithm 5 succeeds in eliminating all single productions in an important special case. We suspect that Algorithm 5 is quite good for eliminating single productions in the general case.

THEOREM 1. *Let Algorithm 5 be applied to the canonical set of tables for an LR(1) grammar $G = (N, \Sigma, P, S)$[11] in which no nonterminal has more than one single production. Then after application of Algorithm 5, no tables calling for reductions by single productions remain.*

Proof. A straightforward induction on the number of applications of Algorithm 4 shows that at all times during the execution of Algorithm 5, each table is the result of the merging by Algorithm 1 of a set of tables $U_1, \cdots, U_m$ (the case $m = 1$ is not ruled out) such that there is a string γ and nonterminals $A_{i_1}, \cdots, A_{i_m}$ with the following properties.

(i) U_j is the table constructed from the set of valid items for γA_{i_j}.

(ii) For $1 \leqq j < m$, there is a single production $A_{i_j} \to A_{i_{j+1}}$.

Thus, let T_1 and T_2 be subject to a possible application of Algorithm 4 when considered by Algorithm 5. Let T_1 be the result of merging $U_1, \cdots, U_m$ and let γ and the A_{i_j}'s be as above. By the order in which Algorithm 5 considers nonterminals, T_2 cannot be the result of mergers, but must be constructed from the set of valid items for γB, for some B. There must be some j such that $A_{i_j} \to B$ is a single production and by the hypothesis of the theorem and the order in which Algorithm 5 considers the nonterminals, we must have $j = m$.

As a consequence, we have $A_{i_l} \overset{*}{\Rightarrow} B$ for $1 \leqq l \leqq m$, and T_2 can be merged with any of $U_1, \cdots, U_m$ by Lemmas 2, 4, 5 and the definition of canonical tables. Thus, T_2 can be merged with T_1. Since T_1 and T_2 are arbitrary candidates for merger, we have shown that every T_2 which reduces by a single production is eliminated, and the proof is complete.

6. Eliminating reductions from SLR tables.

The same techniques used to eliminate reductions by single productions from canonical tables can be used for this purpose on SLR tables. Except in one case, the ideas are analogous to those

[11] Recall that here and throughout the paper we assume a grammar has no nonterminal which derives only the empty string.

of the previous section and we shall merely sketch them. The next two lemmas are analogous to Lemmas 4 and 5.

LEMMA 6. *Let $G = (N, \Sigma, P, S)$ be an SLR(1) grammar, and let $\mathscr{A}_1$ and $\mathscr{A}_2$ be the sets of valid LR(0) items for γA_1 and γB, respectively. Also, let $A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_n \Rightarrow B$ be a derivation of single productions, $n \geqq 1$. If $[C \to \alpha \cdot X\beta, e]$*[12] *is in $\mathscr{A}_1$ and $[D \to \alpha' \cdot Y\beta', e]$ is in $\mathscr{A}_2$, then $X \neq Y$.*

Proof. The proof is analogous to Lemma 4. We omit the details.

LEMMA 7. *Let $G = (N, \Sigma, P, S)$ be an SLR(1) grammar. Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be the sets of LR(0) items for γA_1 and γB, respectively, where $A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_l \Rightarrow B$ is a derivation of single productions, $n \geqq 1$. Let T_1 and T_2 be the tables constructed from $\mathscr{A}_1$ and $\mathscr{A}_2$ by the SLR(1) method. Suppose that the action of T_1 on lookahead a is not φ. Then the action of T_2 on a is* **reduce** *i, where production i is $A_n \to B$.*

Proof. First, we observe that if the action of T_1 on a is not φ, then a is in FOLLOW(A_1). The proof is straightforward if that action is **shift**, **reduce** or **accept**. The case in which the action is an essential error is covered by Lemma 3, when we note that T_1 cannot be the goto of some other table on a terminal symbol, since it comes from the set of valid items for an item ending in a nonterminal. That is, case (b)(iii) of Lemma 3 must pertain. This implies that reduction to A_1 occurs in some tables on input a. Consequently, a must be in FOLLOW(A_1) by the SLR construction rules.

By an argument used in Lemmas 4 and 5, since the set of LR(0) items for γA_1 is nonempty, it follows that $[A_n \to B\cdot, e]$ is valid for γB. Since a is in FOLLOW(A_1), it must be in FOLLOW(A_n), whereupon the action of T_2 on a must be **reduce** i, and the proof is complete.

THEOREM 2. *If Algorithm 5 is applied to the SLR(1) tables for SLR(1) grammar G, and no nonterminal of G has more than one single production, then all reductions by single productions are eliminated from the tables.*

Proof. The proof parallels that of Theorem 1 and is a straightforward consequence of Lemmas 3, 6 and 7. We omit the details.

7. Effect of the transformation. There are two potentially beneficial effects of our transformation. First, it reduces the number of rows in the parsing table. Let us consider the generalization of G_0, that is, a grammar such as the following.

$$\begin{aligned} &E_i \to E_i\theta_i E_{i+1} \qquad \text{for } 1 \leqq i < n \\ &E_i \to E_{i+1} \\ &E_n \to (E_1) \\ &E_n \to a \end{aligned}$$

representing expressions with operators (θ_i's) on n precedence levels. The SLR(1) parser has $3n + 3$ rows, of which $n - 1$ are deleted if we apply Algorithm 2. This gives a potential savings of 1/3 the rows as n becomes large.

However, the parsing table produced by Algorithm 5 has a higher density of entries which are neither **error** nor φ than does the canonical or SLR parser. Various formating techniques, e.g., [16], have been developed to condense the information in the parser. From experiments, it appears that increasing the density

[12] Recall that the second part of an LR(0) term must be e.

of information in the parser counteracts the shrinkage of rows, with the effect that the formatted parsers are of roughly the same size with or without elimination of single productions [1].

The second effect of the transformation is a parser speedup. If we consider the grammar for expressions with n precedence levels given above, we see that for large n, about half the productions are single productions. Thus, one might suspect speedup by a factor of 2 if the parser spends most of its time working on expressions or expressionlike constructs. However, since in practice short expressions occur more frequently than large ones, it is reasonable to expect that the single productions will receive more than their share of use. In fact, experiments at Toronto have shown that the parser for XPL is sped up about $2\frac{1}{2}$ times using our technique [15].

8. An observation regarding non-LR grammars. It is interesting to observe that the parsing table of Fig. 6, with the columns for E, T and F merged, can be obtained in an entirely different manner, a manner which suggests that it may be advantageous to consider the construction of LR-like parsing tables from non-LR or even ambiguous grammars.

Let us define the *skeletal grammar* for a grammar $G = (N, \Sigma, P, S)$ to be the grammar $G_s = (\{S\}, \Sigma, P', S)$, where P' consists of the productions of P with each instance of a nonterminal replaced by S. However, we do not place $S \to S$ in P'.

Example 6. The skeletal grammar G_{s0} for G_0 has the productions

$$\begin{aligned}&(1)\ E \to E + E\\ &(2)\ E \to E * E\\ &(3)\ E \to (E)\\ &(4)\ E \to a\end{aligned}$$

This grammar is not LR(1), in fact it is ambiguous. However, we can construct sets of items for G_{s0} exactly as if it were SLR(1). The sets of LR(0) items for G_{s0} are listed below.

$$\mathscr{A}_0 : \begin{cases} E' \to \cdot E \\ E \to \cdot E + E \\ E \to \cdot E * E \\ E \to \cdot (E) \\ E \to \cdot a \end{cases}$$

$$\mathscr{A}_1 : \begin{cases} E' \to E \cdot \\ E \to E \cdot + E \\ E \to E \cdot * E \end{cases}$$

$$\mathscr{A}_2 : \begin{cases} E \to (\cdot E) \\ E \to \cdot E + E \\ E \to \cdot E * E \\ E \to \cdot (E) \\ E \to \cdot a \end{cases}$$

$\mathscr{A}_3$: $E \to a\cdot$

$$\mathscr{A}_4: \begin{cases} E \to E + \cdot E \\ E \to \cdot E + E \\ E \to \cdot E * E \\ E \to \cdot (E) \\ E \to \cdot a \end{cases}$$

$$\mathscr{A}_5: \begin{cases} E \to E * \cdot E \\ E \to \cdot E + E \\ E \to \cdot E * E \\ E \to \cdot (E) \\ E \to \cdot a \end{cases}$$

$$\mathscr{A}_6: \begin{cases} E \to (E\cdot) \\ E \to E \cdot + E \\ E \to E \cdot * E \end{cases}$$

$$\mathscr{A}_7: \begin{cases} E \to E + E\cdot \\ E \to E \cdot + E \\ E \to E \cdot * E \end{cases}$$

$$\mathscr{A}_8: \begin{cases} E \to E * E\cdot \\ E \to E \cdot + E \\ E \to E \cdot * E \end{cases}$$

$\mathscr{A}_9$: $E \to (E)\cdot$

In G_{s0} we find FOLLOW(E') $= \{e\}$ and FOLLOW(E) $= \{e, +, *,)\}$. If we attempt to construct a parsing table, we find conflicts in $\mathscr{A}_7$ and $\mathscr{A}_8$ for inputs $+$ and $*$. However, if we recall that G_0 is an operator precedence grammar designed to force $*$ to take precedence over $+$ and for both $*$ and $+$ to associate from the left, we can correctly resolve the conflict in $\mathscr{A}_7$ by calling for reduction by $E \to E + E$ when the lookahead is $+$ and by shifting on lookahead $*$ or $+$. If we do this, we may construct the parsing table of Fig. 7, where row R_i is constructed from $\mathscr{A}_i$. It is easy to see that Fig. 7 is equivalent to Fig. 6 under the correspondence shown in Fig. 8, provided that we

(i) merge the columns for E, T and F in Fig. 6,

(ii) replace φ's by x's in Fig. 6, and

(iii) change to **error** the goto entries for R_7 on $+$ and R_8 on $+$ and $*$ in Fig. 7.
It is easy to show that these entries are never exercised.

Since G_0 happens to be an operator precedence grammar, the above relationship should not be surprising, especially in the light of [11]. Moreover, while a

Table	Action						Goto					
	a	+	*	(	)	e	E	a	+	*	(	)
R_0	s	x	x	s	x	x	R_1	R_3	x	x	R_2	x
R_1	x	s	s	x	x	a	x	x	R_4	R_5	x	x
R_2	s	x	x	s	x	x	R_6	R_3	x	x	R_2	x
R_3	x	4	4	x	4	4	x	x	x	x	x	x
R_4	s	x	x	s	x	x	R_7	R_3	x	x	R_2	x
R_5	s	x	x	s	x	x	R_8	R_3	x	x	R_2	x
R_6	x	s	s	x	s	x	x	x	R_4	R_5	x	R_9
R_7	x	1	s	x	1	1	x	x	R_4	R_5	x	x
R_8	x	2	2	x	2	2	x	x	R_4	R_5	x	x
R_9	x	3	3	x	3	3	x	x	x	x	x	x

FIG. 7. *SLR tables for G_{s0}*

Fig. 6	Fig. 7	Fig. 6	Fig. 7
T_0	R_0	1	1
T_1	R_1	3	2
T_4	R_2	5	3
T_5	R_3	6	4
T_6	R_4		
T_7	R_5		
T_8	R_6		
T_9	R_7		
T_{10}	R_8		
T_{11}	R_9		
(a) row names		(b) productions	

FIG. 8. *Correspondence between Fig. 6 and Fig. 7*

general theory has not yet emerged, some techniques for automatically generating LR-like parsing tables for ambiguous (and hence non LR) grammars have been developed and are reported in Aho, Johnson and Ullman [2]. It is interesting to conjecture that the technique developed in the present paper can be subsumed under the methods [2] or more general techniques for working directly from ambiguous grammars.

Acknowledgment. The authors would like to thank the referees for their perceptive comments on the original manuscript.

REFERENCES

[1] A. V. AHO AND S. C. JOHNSON, *LR parsing*, Tech. Rep., Bell Laboratories, Murray Hill, N.J., 1973.

[2] A. V. AHO, S. C. JOHNSON AND J. D. ULLMAN, *Mechanical parser generation for ambiguous grammars*, to appear in *Proc. SIGACT/SIGPLAN Symposium Principles of Programming Languages*, Boston, Mass., Oct. 1973.

[3] A. V. AHO AND J. D. ULLMAN, *Translations on a context-free grammar*, Information and Control, 19 (1971), pp. 439–475.

[4] ———, *The Theory of Parsing, Translation, and Compiling*, vol. 1, *Parsing*, Prentice-Hall, Englewood Cliffs, N.J., 1972.

[5] ———, *Optimization of LR(k) parsers*, J. Comput. System Sci., 6 (1972), pp. 573–602.

[6] ———, *The Theory of Parsing, Translation, and Compiling*, vol. 2, *Compiling*, Prentice-Hall, Englewood Cliffs, N.J., 1973.

[7] T. ANDERSON, *Syntactical analysis of LR(k) languages*. Ph.D. thesis, University of Newcastle upon Tyne, 1972.

[8] T. ANDERSON, J. EVE AND J. J. HORNING, *Efficient LR(1) parsers*, TR 24, Computing Laboratory, University of Newcastle upon Tyne, 1971.

[9] F. L. DEREMER, *Practical translators for LR(k) languages*, Ph.D. thesis, M.I.T., Cambridge, Mass., 1969.

[10] ———, *Simple LR(k) grammars*, Comm. ACM, 14 (1971), pp. 453–460.

[11] N. EL DJABRI, *LR(1) parsers for precedence grammars*, Computer Science Laboratory, Dept. of Electrical Engineering, Princeton University, Princeton, N.J., 1973.

[12] R. W. FLOYD, *Syntactic analysis and operator precedence*, J. Assoc. Comput. Mach., 10 (1963), pp. 316–333.

[13] S. L. GRAHAM, Private communication.

[14] J. N. GRAY AND M. A. HARRISON, *On the covering and reduction problems for context-free grammars*, J. Assoc. Comput. Mach., 19 (1972), pp. 675–698.

[15] J. J. HORNING, Private communication.

[16] M. JOLLIAT, *On the reduced matrix representation of LR(k) parser tables*, Ph.D. thesis, University of Toronto, 1973.

[17] D. E. KNUTH, *On the translation of languages from left to right*, Information and Control, 8 (1965), pp. 607–639.

[18] ———, *Semantics of context-free languages*, Math. Systems Theory, 2 (1968), pp. 127–146, and 5 (1971), pp. 95–96.

[19] A. J. KORENJAK, *A practical method for constructing LR(k) processors*, Comm. ACM, 12 (1969), pp. 613–623.

[20] P. M. LEWIS II AND R. E. STEARNS, *Syntax direct transductions*, J. Assoc. Comput. Mach., 15 (1968), pp. 464–488.

[21] D. PAGER, *On eliminating unit productions from LR(k) parsers*, Tech. Rep. PE 238, Information Sciences Program, University of Hawaii, Honolulu, 1972.

SIAM. J. COMPUT.
Vol. 2, No. 2, June 1973

TRANSPOSITION GRAPHS*

PHILLIP J. CHASE†

Abstract. The transposition graph $G(p_1, \cdots, p_k)$ is a regular graph having as vertices all sequences of $p_1 + \cdots + p_k$ symbols, of which p_1 are of kind $1, \cdots, p_k$ are of kind k. Two vertices are adjacent if transposing a pair of different symbols in one gives the other. We justify CACM Algorithms 382 and 383, which generate Hamilton paths in $G(p_1, \cdots, p_k)$. The main result is that $G(p_1, \cdots, p_k)$ is Hamiltonian.

1. Introduction. For $k \geqq 2$, let $p_1, \cdots, p_k$ be positive integers and let $n = p_1 + \cdots + p_k$. Let $\phi = (\theta_1, \cdots, \theta_k)$, where $\theta_1, \cdots, \theta_k$ are distinct symbols. When the symbols need to be explicit, we often take $\phi = (A, B, C, \cdots)$ or $\phi = (0, 1)$. Let $V = V(p_1, \cdots, p_k)$ be the set of sequences $x = (x_1, \cdots, x_n)$ for which p_1 of the x_i are $\theta_1, \cdots, p_k$ of the x_i are θ_k. If π is a permutation of $\{1, 2, \cdots, n\}$, then $\pi : V \to V$ according to $\pi(x_1, \cdots, x_n) = (x_{\pi(1)}, \cdots, x_{\pi(n)})$. For each $x \in V$, $\rho(x)$ is formed by replacing the subsequence of symbols θ_k by $(1, 2, \cdots, p_k)$. For example, $\rho(C\,A\,A\,C\,C\,B) = 1\,A\,A\,2\,3\,B$.

The *transposition graph* $G = G(p_1, \cdots, p_k)$ has vertex set $V(p_1, \cdots, p_k)$. For s and t in V, $\{s, t\}$ is an edge of G if $s \neq t$ and if there exists a transposition τ such that $\tau(s) = t$, in which case τ is unique and $\tau(t) = s$. Each vertex of G is seen to be adjacent to exactly $\frac{1}{2}[n^2 - (p_1^2 + \cdots + p_k^2)]$ other vertices; that is, G is regular of this degree. The *kernel graph* $H = H(p_1, \cdots, p_k)$ is a subgraph of G with the same vertex set, but with fewer edges. $\{s, t\}$ is an edge of H if it is an edge of G, and if for the transposition τ such that $\tau(s) = t$, we have $\rho\tau(s) = \tau\rho(s)$. This means that τ leaves invariant the order of the symbols of kind k. H is regular only when $p_k = 1$.

In Fig. 1, solid lines are edges of $H(2, 2)$ (hence of $G(2, 2)$ also), and dotted lines are the remaining edges of $G(2, 2)$.

In 1964, Donald E. Knuth [3] found an algorithm to generate, in $G(p, q)$ (the case $k = 2$), an exhaustive linear listing of vertices, without repeats, such that vertices adjacent in the listing are also adjacent in the graph. Such a listing is called a Hamilton path. Independently, Leo W. Lathroum [4] found another in 1965. The methods of Knuth and Lathroum are discussed in § 2. Edward P. Neuburg brought Lathroum's work to the author's attention, and this led to CACM Algorithms 382 and 383 ([1] and [2]).

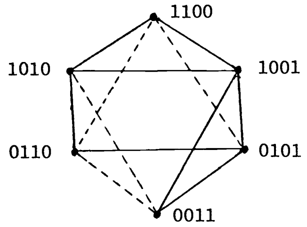


FIG. 1

* Received by the editors December 6, 1972.

† Department of Defense, Fort Meade, Maryland 20755.

Algorithm 382, named TWIDDLE and justified in § 2, is an ALGOL procedure which, by following a Hamilton path in $H(p, q)$, provides economy either (1) in generating all combinations of size q from $\{1, 2, \cdots, n\}$, or (2) in generating all binary sequences $(b_1, \cdots, b_n)$ containing p 0's and q 1's. Since the path is in H, as opposed to G, adjacent combinations $(c_1, \cdots, c_q)$, $c_1 < \cdots < c_q$, turn out to differ in just one component. Under usage (2), adjacent binary sequences differ by a transposed 0 and 1. Algorithm 383, named EXTENDED TWIDDLE and justified in § 3, essentially generates a Hamilton path in $H(p_1, \cdots, p_k)$.

Our principal result, in § 4, is that $H(p_1, \cdots, p_k)$, hence also $G(p_1, \cdots, p_k)$, is Hamiltonian except for $H(1, q)$ with $q > 1$. Even then, $G(1, q)$ is Hamiltonian. Recall that a graph is Hamiltonian if it has a Hamilton circuit, which is a Hamilton path in which also the first and last vertices are adjacent in the graph.

Figure 2 shows $H(1, 1, 2)$. The numbering of the vertices exhibits a Hamilton circuit.

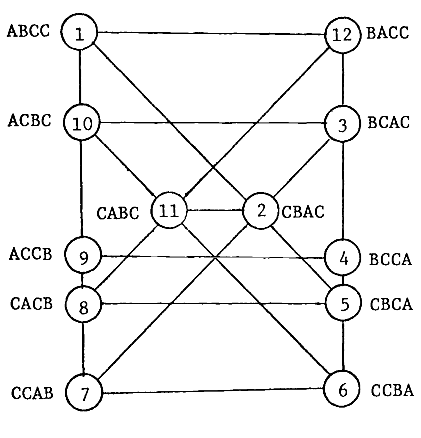


FIG. 2

2. The case $k = 2$. A *p, q signed sequence s* is a sequence of $n = p + q$ symbols of which p are signs, $+$ or $-$. The other q are numbers, giving the subsequence $1, 2, \cdots, q$. The set of all such signed sequences is denoted by $V'(p, q)$. The map $\alpha : V'(p, q) \to V(p, q)$ replaces every sign by 0 and every number by 1. Thus $\alpha(1\,2 - + + 3 + - 4 - +)$ is equal to $1\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0$. The map $\gamma : V'(p, q) \to V'(p, q)$ reverses signs, so that, for example, $\gamma(1\,2 - + + 3 + - 4 - +)$ equals $1\,2 + - - 3 - + 4 + -$.

Certain intervals in a signed sequence are called *blocks*. They are of two types. An *R(ight)-block* is a nondegenerate interval of $+$'s which is followed by a number but which is not preceded by a $+$. An *L(eft)-block* is a nondegenerate interval of signs of which the last, and only the last, is a $-$, and which is preceded by a number. The blocks in an example are as follows:

$$+ - 1\,2\,\underset{L}{\underline{-}}\,\underset{R}{\underline{+\,+}}\,3\,\underset{L}{\underline{+\,+\,-}}\,4\,+\,.$$

The sequence $- + - 1\,2 + +$ has no blocks at all. Blocks can never overlap.

The *successor* of a p, q signed sequence s which has blocks is a p, q signed sequence $\sigma(s)$ obtained as follows. Find the leftmost block in s. If it is an R-block, then slide the entire block one place to the right, putting the displaced number in the place vacated by the block's leftmost sign. If it is an L-block, slide it one place to the left, and put the displaced number in the place vacated by the block's rightmost sign. In either case, if a sign slides, then the next sign to the left (there may be intervening digits), if there is one, changes.

Example. Here each sequence is the successor of the sequence immediately above. The leftmost blocks are underlined.

$$\begin{array}{ll}
\underline{+\ +\ +}\ 1\,2 & 1\ \underline{-}\ +\ 2\ + \\
1\ \underline{-}\ -\ +\ 2 & -\ 1\ \underline{+}\ 2\ + \\
-\ 1\ \underline{-}\ +\ 2 & \underline{+}\ 1\,2\ +\ + \\
+\ -\ 1\ \underline{+}\ 2 & 1\ \underline{+}\ 2\ +\ + \\
\underline{+\ +}\ 1\,2\ + & 1\,2\ +\ +\ +
\end{array}$$

Immediate and important observations about σ are:

(i) σ preserves the subsequence of numbers,

(ii) $\alpha\sigma(s)$ and $\alpha(s)$ are adjacent in H.

Also, if $t = \sigma^i(s)$ for some i, then s and $\alpha(t)$ determine t. For, if in $\alpha(t)$, the jth 0 ($j = 2, \cdots, p$) is displaced an odd (resp. even) number of places from its position in $\alpha(s)$, then the $(j-1)$st sign is different (resp. the same) in t and s. The pth signs of s and t must be the same. Similarly, t and $\alpha(s)$ determine s. If the signs of s and t in $V'(p, q)$ are in this relation, then s and t are called *consistent*, and this is clearly an equivalence relation.

Let s have at least one block, the leftmost being I. Then σ transforms I into an interval I' of $\sigma(s)$. γ applied to $\sigma(s)$ transforms I' into a interval I'' of $\gamma\sigma(s)$ which turns out to be the leftmost block of $\gamma\sigma(s)$. I'' is an R-block or an L-block according as I is an L-block or an R-block, respectively. This gives the following lemma.

LEMMA 1. *If $t = \sigma(s)$, then $s = \gamma\sigma\gamma(t) = \sigma^{-1}(t)$.*

A related result is the following.

LEMMA 2. *Every succession $\gamma\sigma^j(s)$, $\gamma\sigma^{j-1}(s)$, $\cdots$, $\gamma\sigma(s)$, $\gamma(s)$ is generated by σ, i.e., $\sigma(\gamma\sigma^i) = \gamma\sigma^{i-1}$ for $i = 1, \cdots, j$.*

Proof. $\sigma(\gamma\sigma^i) = \gamma\,\gamma\,\sigma\,(\gamma\sigma^i) = \gamma\,(\gamma\,\sigma\,\gamma)\sigma^i = \gamma\,\sigma^{-1}\,\sigma^i = \gamma\,\sigma^{i-1}$.

It turns out that every succession $s, \sigma(s), \sigma^2(s), \cdots$ leads to a sequence t without blocks, which is therefore called *terminal.* By Lemma 2, the succession beginning with $\gamma(t)$ is at least as long, so we call sequences of the form $\gamma(t)$ *initial.* The initial p, q signed sequences are exactly those of the form (signs)^a $1\ 2 \cdots q(-)^b$, $a + b = p$, where, of the first a signs, the last must be $+$ (provided $a > 0$). The initial 2, 2 sequences are $+\ +\ 1\ 2$, $-\ +\ 1\ 2$, $+\ 1\ 2\ -$, and $1\ 2\ -\ -$. In general, they number 2^p.

THEOREM 1. *If s is an initial p, q sequence, then the succession s, $\sigma(s)$, $\sigma^2(s)$, $\cdots$ eventually ends with a terminal sequence. The corresponding succession $\alpha(s)$, $\alpha\sigma(s)$, $\alpha\sigma^2(s)$, $\cdots$ is a Hamilton path in $H(p, q)$.*

Proof. We proceed by induction on $n = p + q$, the desired conclusions being clear for $n = 2$. The last term of s is either the sign $-$ or the number q. Suppose it is q. Then $s = s'q$ where s' is an initial $p, q - 1$ sequence. Until q is affected, $\sigma^i(s'q)$ will be $\sigma^i(s')q$. By induction, therefore, beginning with $s'q$, and successively applying σ, we eventually reach a $t'q$, where t' is terminal, such that the corresponding images under α are a path T_1 in $H(p, q)$ containing all sequences with a terminal 1. Then $\sigma(t'q) = u' +$ where u' must be an initial $p - 1, q$ sequence. By induction, as before, σ leads eventually to a $v' +$ where v', hence also $v' +$, is terminal, and where the respective images under α are a path T_2 in $H(p, q)$ containing all vertices with a terminal 0. Together, T_1 and T_2 give a Hamilton path in $H(p, q)$. The case $s = s' -$ is argued similarly, completing the proof.

One consequence of Theorem 1 is that each $u \in V'(p, q)$ is of the form $\sigma^i(s)$ for exactly one initial sequence s. For $\alpha(u)$ occurs in each of the 2^p Hamilton paths in $H(p, q)$ generated by initial sequences. But, by Lemma 1, the initial sequence is recoverable from any signed sequence in its succession. So the 2^p occurrences of $\alpha(u)$ must correspond to 2^p different signed sequences. But there are only 2^p signed sequences t with $\alpha(t) = \alpha(u)$. So, as claimed, $u = \sigma^i(s)$ for a unique initial sequence s and a unique i less than $\binom{n}{p}$. It also follows that $u = \sigma^j(u)$ implies that $j = 0$.

In § 4, the following will be needed.

LEMMA 3. *$H(p, q)$ has a Hamilton path between A^pB^q and any vertex of the form $A^cB^qA^d$ with $d \geqq 1$ and $c + d = p$.*

Proof. Such a path will be generated by $\bar{s}^c s^{d-1} + 1\ 2 \cdots q$, where s is $+$ or $-$ according as q is even or odd.

Algorithm 382, TWIDDLE, simply follows the path generated by $+^p\ 1\ 2 \cdots q$, and is therefore justified by Theorem 1. Historically, Leo W. Lathroum was concerned with efficient generation of fixed density binary sequences. He discovered our sign device and applied it to the (single) initial sequence $+^p\ 1^q$. He did not consider the consequences of replacing $1^q = 1\ 1 \cdots 1$ by $1\ 2 \cdots q$. Lathroum's method was proved valid by Edward P. Neuburg, using an argument different from ours.

Donald E. Knuth's listing is recursively defined, giving a succession $K(q, p)$, beginning with $1^q\,0^p$ and ending with $0^p\,1^q$, by transpositions, as follows:

$$K(q, p) = K(q, p - 1)0, \bar{K}(q - 1, p - 1)01, K(q - 2, p)11,$$

where $\bar{K}$ is the reverse of K. Surprisingly, the resulting Hamilton path in $G(p, q)$ turns out to correspond to $1\ 2 \cdots q(-)^p$.

3. The case $k > 2$. Let $1 < j < k$, where $k > 2$. Let $V = V(p_1, \cdots, p_k)$ have symbols $\phi = (\theta_1, \cdots, \theta_k)$. Let $V_1 = V(p_1, \cdots, p_{j-1}, p_j + \cdots + p_k)$ have symbols $\phi_1 = (\theta_1, \cdots, \theta_j)$, and let $V_2 = V(p_j, \cdots, p_k)$ have symbols $\phi_2 = (\theta_j, \theta_{j+1}, \cdots, \theta_k)$. Then each $s \in V$ is obtained by putting together two sequences t and u, $t \in V_1$ and $u \in V_2$. Form t from s by replacing each symbol θ_i having $i \geqq j$ by θ_j. Form u from s by extracting the subsequence of symbols θ_i with $i \geqq j$. Then t and u determine s, and are determined by s, and so we write $s = [t, u]$.

Now let H, H_1, and H_2 be the kernel graphs corresponding to V, V_1, V_2, respectively. Let $T = (t_1, \cdots, t_a)$ be a path in H_1. For $u \in H_2$, let $[T, u] = ([t_1, u], [t_2, u], \cdots, [t_a, u])$. Then $[T, u]$ is a path in H including every $[t, u]$ with $t \in T$.

Let $J = (T_1, T_2, \cdots)$ be a denumberable sequence of Hamilton paths in H_1 such that the initial vertex of T_i is the terminal vertex of T_{i-1} for $i = 2, 3, \cdots$. Let $U = (u_1, \cdots, u_b)$ be a Hamilton path in H_2. Then $[T_1, u_1], [T_2, u_2], \cdots, [T_u, u_b]$, which we denote by $[J, U]$, defines a Hamilton path in H.

THEOREM 2. *$H(p_1, p_2, \cdots, p_k)$ has a Hamilton path.*

Proof. The case $k = 2$ is covered by Theorem 1. With the appropriate inductive hypothesis, there exists a Hamilton path $T = (t_1, \cdots, t_a)$ in $H(p_1, \cdots, p_{k-2}, p_{k-1} + p_k)$. Let $J = (T, \bar{T}, T, \bar{T}, \cdots)$, where $\bar{T} = (t_a, t_{a-1}, \cdots, t_1)$. Let U be a Hamilton path for $H(p_{k-1}, p_k)$. Then $[J, U]$ is a Hamilton path for $H(p_1, \cdots, p_k)$.

Algorithm 383 reduces inductively, by the above construction, using $H_1 = H(p_1, p_2 + \cdots + p_k), H_2 = H(p_2, \cdots, p_k)$, and $J = (T_1, T_2, T_3, \cdots)$, where each T_i corresponds to an initial signed p, q sequence (for $p = p_1$ and $q = p_2 + \cdots + p_k$). T_1 corresponds to $+^p 1 \cdots q$. T_{i+1} is determined by T_i. Let T_i correspond to the terminal sequence $s_1 \cdots s_a 1\, 2 \cdots q +^b$, $a + b = p$, where the s_i are signs with $s_a = -$ if $a > 0$. Then T_{i+1} corresponds to the initial sequence

$$s_1 \cdots s_{a-1}\, \bar{s}_a\, 1\, 2 \cdots q(-)^b$$

(noting that $\bar{s}_a = +$ if $a > 0$). Incidentally, the successive initial sequences so formed include all 2^p possibilities. The simpler sequence J used in the proof of Theorem 2 was not used for reasons of efficiency.

4. $H(p_1, \cdots, p_k)$ is Hamiltonian.

THEOREM 3. *Except in the case $H(1, q)$ with $q > 1$, $H(p_1, \cdots, p_k)$ is Hamiltonian.*

Proof. The proof will be by induction on k. There are initial cases to be treated for both $k = 2$ and $k = 3$. First, consider the case $k = 2$, $H(p, q)$. If $q = 1$, $H(p, q)$ is clearly Hamiltonian. When $q > 1$, $H(1, q)$ cannot be Hamiltonian because AB^q is adjacent only to BAB^{q-1}. This leaves the subcase $H(p, q)$, where $p > 1$ and $q > 1$. According to Lemma 3, there are Hamilton paths T in $H(p - 1, q)$ and U in $H(p, q - 1)$, T leading from $A^{p-1}B^q$ to B^qA^{p-1}, and U leading from $B^{q-1}A^p$ to $A^{p-1}B^{q-1}A$. If TA denotes the path in $H(p, q)$ formed by adjoining A to the end of each sequence in T, and if UB is formed similarly, then TA followed by UB is a Hamilton circuit in $H(p, q)$.

For $k \geqq 3$, we can take $p_1 \geqq \cdots \geqq p_{k-1}$. Since $H(1, q)$ is not Hamiltonian when $q > 1$, the cases $H(1, 1, q)$ and $H(p, 1, q)$, with p and q greater than 1, must be treated separately. For the case $H(1, 1, q)$, let $S = (s_1, \cdots, s_a)$ be a Hamilton path in $H(2, q)$ with $s_1 = X^2C^q$ and $s_a = C^qX^2$, which exists by Lemma 3. For each i in $\{2, \cdots, a\}$, there is a transposition τ_i such that $s_i = \tau_i s_{i-1}$. Form the path $T = (t_1, \cdots, t_a)$ in $H(1, 1, q)$ by taking $t_1 = ABC^q$, and, for $2 \leqq i \leqq a$, $t_i = \tau_i t_{i-1}$. Form $U = (u_1, \cdots, u_a)$ similarly, but with $u_1 = BAC^q$. Then T followed by $\bar{U} = (u_a, u_{a-1}, \cdots, u_1)$ is a Hamilton circuit in $H(1, 1, q)$. For an example, see Fig. 2.

The remaining initial case is $H(p, 1, q)$, where p and q are greater than 1. By Lemma 3, there is a Hamilton path T in $H(p, q + 1)$ from A^pX^{q+1} to $A^{p-1}X^{q+1}A$. Then $U = [T, BC^q]$ is a path in $H(p, 1, q)$ from A^pBC^q to $A^{p-1}BC^qA$. Suppose first that q is even. Let $J = (T, \bar{T}, T, \bar{T}, \cdots)$, and let W be the path $(C^qB, C^{q-1}BC, \cdots, CBC^{q-1})$ in $H(1, q)$. Since W is even with q, the last vertex in $[J, W]$ is A^pCBC^{q-1}, so that U followed by $[J, W]$ is a Hamilton circuit in $H(p, 1, q)$.

The subcase with odd $q \geqq 3$ remains. By Lemma 3, there are Hamilton paths T_1 and T_2 in $H(p, q + 1)$ leading, respectively, from $A^{p-1}X^{q+1}A$ to $X^{q+1}A^p$, and from $X^{q+1}A^p$ to A^pX^{q+1}. Let $J' = (T, T_1, T_2, T, \bar{T}, T, \bar{T}, \cdots)$. Since W is odd with q, U followed by $[J', W]$ gives a Hamilton circuit in $H(p, 1, q)$.

Now let $H(p_1, \cdots, p_k)$ be a case not so far covered. Then $k \geqq 3$ and we can make the inductive assumption that $H(p_2, \cdots, p_k)$ has a Hamilton circuit W. If $p_1 = 1$, then there are at least two 1's among $p_2, \cdots, p_k$, since $H(1, 1, q)$ with $q > 1$ has already been treated. But then W, which is of order $(p_2 + \cdots + p_k)!/(p_2! \cdots p_k!)$, would have to be even. Thus either W is even or $p_1 > 1$. Suppose first that W is even. Let T be any Hamilton path in $H(p_1, p_2 + \cdots + p_k)$, and let $J = (T, \bar{T}, T, \bar{T}, \cdots)$. Then $[J, W]$ is a Hamilton circuit in $H(p_1, \cdots, p_k)$. Next, suppose W is odd. Let $p = p_1$ and let $q = p_2 + \cdots + p_k$. As noted, it follows that $p > 1$, so that there exist Hamilton paths U, U_1, U_2 in $H(p, q)$ leading, respectively, from A^pB^q to $A^{p-1}B^qA$, $A^{p-1}B^qA$ to B^qA^p, B^qA^p to A^pB^q. For $J' = (U, U_1, U_2, U, \bar{U}, U, \bar{U} \cdots)$, $[J', V]$ is a Hamilton circuit in $H(p_1, \cdots, p_k)$, completing the proof.

COROLLARY. *Every transposition graph $G(p_1, \cdots, p_k)$ is Hamiltonian.*

Proof. Except for $G(1, q)$ with $q > 1$, this is implied by Theorem 3. But $G(1, q)$ is, in fact, complete and hence Hamiltonian.

REFERENCES

[1] P. J. CHASE, *Algorithm 382, combinations of M out of N objects*, Comm. ACM, 13 (1970), p. 368.
[2] ———, *Algorithm 383, permutations of a set with repetitions*, Ibid., 13 (1970), p. 368.
[3] DONALD E. KNUTH, Private communication.
[4] LEO W. LATHROUM, Private communication.
[5] EDWARD P. NEUBURG, Private communication.

SIAM J. COMPUT.
Vol. 2, No. 3, September 1973

DIVIDING A GRAPH INTO TRICONNECTED COMPONENTS*

J. E. HOPCROFT† AND R. E. TARJAN†

Abstract. An algorithm for dividing a graph into triconnected components is presented. When implemented on a random access computer, the algorithm requires $O(V + E)$ time and space to analyze a graph with V vertices and E edges. The algorithm is both theoretically optimal to within a constant factor and efficient in practice.

Key words. articulation point, connectivity, depth-first search, graph, separability, separation, triconnectivity

Introduction. The connectivity properties of graphs form an important part of graph theory. Efficient algorithms for determining some of these properties are both theoretically interesting and useful in a variety of applications. This paper considers the problem of dividing a graph into triconnected components. An algorithm for this purpose is useful for analyzing electrical circuits [1], for determining whether a graph is planar [2], and for determining whether two planar graphs are isomorphic [3]. An algorithm for planarity may be used in the design of printed circuit boards; an algorithm for isomorphism of planar graphs may be used to test structural isomorphism of chemical compounds [4].

One technique which has been used to solve connectivity problems is depth-first search. In [5] and [6], depth-first search is applied to give efficient algorithms for determining the biconnected components of an undirected graph and for determining the strongly connected components of a directed graph. The method has also been used in an efficient algorithm for planarity testing ([7], [8]) and in an algorithm to find dominators in a flow graph [9]. This paper applies depth-first search to the problem of finding the triconnected components of a graph. Old methods for determining these components require $O(V^3)$ steps or more, if the graph has V vertices ([1], [10]). The algorithm described here requires substantially less time, and it may be shown to be optimal to within a constant factor, assuming a suitable model of computation.

Four sections comprise the paper. The first section presents the necessary definitions and lemmas from graph theory, and it describes depth-first search. The second section intuitively explains the triconnectivity algorithm. The third section describes preliminary calculations and a simple test to find the separation pairs of a graph. The last section gives the heart of the triconnected components algorithm, including proofs of its correctness and the derivation of time and space bounds.

In deriving time bounds on algorithms, we assume a random-access computer model. A formal definition of such a model may be found in [11]. Intuitively, any logical, arithmetic, or control operation requires one step; all numbers must be integers whose absolute values are $O(V)$, if the problem graph has V vertices. (We

* Received by the editors July 27, 1972.

† Computer Science Department, Cornell University, Ithaca, New York 14850.

‡ This research was supported in part by the Hertz Foundation and the Office of Naval Research under Grant N00014-67-A-0077-0021.

use the following notation for specifying bounds: if f and g are functions of x, we say $f(x)$ is $O(g(x))$ if, for some constants k_1 and k_2, $|f(x)| \leqq k_1|g(x)| + k_2$ for all x.)

1. Graphs, connectivity, and depth-first search. The definitions used in this paper are more or less standard; see [12] and [13]. Triconnected components may be defined in several ways, all more or less equivalent. The results below, which we give without proof, follow from those of Saunders Maclaine [14]; our definitions are modified somewhat to make them more suitable for computer applications. Tutte [15] has also developed a theory of triconnected components; his definitions are equivalent to ours and to Maclaine's. The theory is also a special case of the more general theory of decomposing "clutters" into "chunks" due to Edmonds and Cunningham [16].

A *graph* $G = (\mathscr{V}, \mathscr{E})$ consists of a set $\mathscr{V}$ containing V *vertices* and a set $\mathscr{E}$ containing E *edges*. If the edges are ordered pairs (v, w) of distinct vertices, the graph is *directed*; v is called the *tail* and w the *head* of the edge. If the edges are unordered pairs of distinct vertices, also denoted by (v, w), the graph is *undirected*. If $\mathscr{E}$ is a multiset, that is, if any edge may occur several times, then G is a *multigraph*. If (v, w) is an edge of a multigraph G, vertices v and w are *adjacent*. Edge (v, w) is incident to vertices v and w; v and w are *incident* to (v, w). If $\mathscr{E}'$ is a set of edges in G, $\mathscr{V}(\mathscr{E}')$ is the set of vertices incident to one or more of the edges in $\mathscr{E}'$. If S is a set of vertices in G, $\mathscr{E}(S)$ is the set of edges incident to at least one vertex in S.

If G is a multigraph, a *path* $p: v \stackrel{*}{\Rightarrow} w$ in G is a sequence of vertices and edges leading from v to w. A path is *simple* if all its vertices are distinct. A path $p: v \stackrel{*}{\Rightarrow} v$ is a *cycle* if all its edges are distinct and the only vertex to occur twice on p is v, which occurs exactly twice. Two cycles which are cyclic permutations of each other are considered to be the same cycle. The *undirected version* of a directed multigraph is the multigraph formed by converting each edge of the directed multigraph into an undirected edge. An undirected multigraph is *connected* if every pair of vertices v and w in G is connected by a path. If $G = (\mathscr{V}, \mathscr{E})$ and $G' = (\mathscr{V}', \mathscr{E}')$ are two multigraphs such that $\mathscr{V}' \subseteq \mathscr{V}$ and $\mathscr{E}' \subseteq \mathscr{E}$, then G' is a *subgraph* of G. A multigraph having exactly two vertices v, w and one or more edges (v, w) is called a *bond*.

A (*directed*, *rooted*) *tree* T is a directed graph whose undirected version is connected, having one vertex (called the *root*) which is the head of no edges, and such that all vertices except the root are the head of exactly one edge. The relation "(v, w) is an edge of T" is denoted by $v \rightarrow w$. The relation "there is a path from v to w in T" is denoted by $v \stackrel{*}{\rightarrow} w$. If $v \rightarrow w$, v is the *father* of w and w is a *son* of v. If $v \stackrel{*}{\rightarrow} w$, v is an *ancestor* of w and w is a *descendant* of v. The set of descendants of a vertex v is denoted by $D(v)$. Every vertex is an ancestor and a descendant of itself. If G is a directed multigraph, a tree T is a *spanning tree* of G if T is a subgraph of G and T contains all the vertices of G.

Let P be a directed multigraph consisting of two disjoint sets of edges, denoted by $v \rightarrow w$ and $v - \rightarrow w$. Suppose P satisfies the following properties.

(i) The subgraph T containing the edges $v \rightarrow w$ is a spanning tree of P.
(ii) If $v - \rightarrow w$, then $w \stackrel{*}{\rightarrow} v$. That is, each edge not in the spanning tree T of P connects a vertex with one of its ancestors in T.

Then P is called a *palm tree*. The edges $v - \rightarrow w$ are called the *fronds* of P.

A connected multigraph G is *biconnected* if for each triple of distinct vertices v, w and a in V, there is a path $p:v \overset{*}{\Rightarrow} w$ such that a is not on the path p. If there is a distinct triple v, w, a such that a is on every path $p:v \overset{*}{\Rightarrow} w$, then a is called a *separation point* (or an *articulation point*) of G. We may partition the edges of G so that two edges are in the same block of the partition if and only if they belong to a common cycle. Let $G_i = (V_i, E_i)$ where E_i is the set of edges in the ith block of the partition, and $V_i = V(E_i)$. Then the following hold.

(i) Each G_i is biconnected.
(ii) No G_i is a proper subgraph of a biconnected subgraph of G.
(iii) Each vertex of G which is not an articulation point of G occurs exactly once among the V_i and each articulation point occurs at least twice.
(iv) For each i, j, $i \neq j$, $V_i \cap V_j$ contains at most one vertex; furthermore, this vertex (if any) is an articulation point.

The subgraphs G_i of G are called the *biconnected components* of G. The biconnected components of G are unique.

Let $\{a, b\}$ be a pair of vertices in a biconnected multigraph G. Suppose the edges of G are divided into equivalence classes $E_1, E_2, \cdots, E_n$ such that two edges which lie on a common path not containing any vertex of $\{a, b\}$ except as an endpoint are in the same class. The classes E_i are called the *separation classes* of G with respect to $\{a, b\}$. If there are at least two separation classes, then $\{a, b\}$ is a *separation pair* of G unless (i) there are exactly two separation classes, and one class consists of a single edge, or (ii) there are exactly three classes, each consisting of a single edge.

If G is a biconnected multigraph such that no pair $\{a, b\}$ is a separation pair of G, then G is *triconnected*. Let $\{a, b\}$ be a separation pair of G. Let the separation classes of G with respect to $\{a, b\}$ be $E_1, E_2, \cdots, E_n$. Let $E' = \bigcup_{i=1}^{k} E_i$ and $E'' = \bigcup_{i=k+1}^{n} E_i$ be such that $|E'| \geqq 2$, $|E''| \geqq 2$. Let $G_1 = (V(E'), E' \cup \{(a, b)\})$, $G_2 = (V(E''), E'' \cup \{(a, b)\})$. The graphs G_1 and G_2 are called *split graphs* of G with respect to $\{a, b\}$. Replacing a multigraph G by two split graphs is called *splitting* G. There may be many possible ways to split a graph, even with respect to a fixed separation pair $\{a, b\}$. A splitting operation is denoted by $s(a, b, i)$; i is a label distinguishing this split operation from other splits. The new edges (a, b) added to G_1 and G_2 are called *virtual edges*; they are labeled to identify them with the split. A virtual edge (a, b) associated with split $s(a, b, i)$ will be denoted by (a, b, i). If G is biconnected, then any split graph of G is also biconnected.

Suppose a multigraph G is split, the split graphs are split, and so on, until no more splits are possible (each graph remaining is triconnected). The graphs constructed in this way are called the *split components* of G. The split components of a multigraph are not necessarily unique.

LEMMA 1. *Let $G = (V, E)$ be a multigraph with $|E| \geqq 3$. Let $G_1, G_2, \cdots, G_m$ be the split components of G. Then the total number of edges in $G_1, G_2, \cdots, G_m$ is bounded by* $3|E| - 6$.

Proof. The lemma is proved by induction on the number of edges of G. If G has 3 edges, the lemma is immediate, because G cannot be split. Suppose the lemma is true for graphs with $n - 1$ edges and suppose G has n edges. If G cannot be split, the lemma is true for G. Suppose, on the other hand, that G can be split into G' and G'', where G' has $k + 1$ edges and G'' has $n - k + 1$ edges for some

$2 \leqq k \leqq n - 2$. By induction, the total number of edges in $G_1, G_2, \cdots, G_m$ must be bounded by $3(k + 1) - 6 + 3(n - k + 1) - 6 = 3n - 6$. Thus, by induction, Lemma 1 is true.

In order to get unique triconnected components, we must partially reassemble the split components. Suppose $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two split components, both containing a virtual edge (a, b, i). Let

$$G = (V_1 \cup V_2, (E_1 - \{(a, b, i)\}) \cup (E_2 - \{(a, b, i)\})).$$

Then G is called a *merge graph* of G_1 and G_2; the merge operation will be denoted by $m(a, b, i)$. Merging is the inverse of splitting; if we perform a sufficient number of merges on the split components of a multigraph, we recreate the original multigraph.

The split components of a multigraph are of three types: triple bonds of the form $(\{a, b\}, \{(a, b), (a, b), (a, b)\})$, triangles of the form $(\{a, b, c\}, \{(a, b), (a, c), (b, c)\})$, and triconnected graphs. Let G be a multigraph whose split components are a set of triple bonds $\mathscr{B}_3$, a set of triangles $\mathscr{T}$, and a set of triconnected graphs $\mathscr{G}$. Suppose the triple bonds $\mathscr{B}_3$ are merged as much as possible to give a set of bonds $\mathscr{B}$, and that the triangles $\mathscr{T}$ are merged as much as possible to give a set of polygons $\mathscr{P}$. Then the set of graphs $\mathscr{B} \cup \mathscr{P} \cup \mathscr{G}$ is the set of *triconnected components* of G. If G is an arbitrary multigraph, the triconnected components of the biconnected components of G are called the *triconnected components* of G.

LEMMA 2. *The triconnected components of a graph G are unique.*

Proof. See [14], [16] and [17].

Figure 1 illustrates a biconnected graph G with several separation pairs. Figure 2 gives the split components of G. The triconnected components of G are formed by merging triangle (1, 8, 4) and triangle (4, 5, 8).

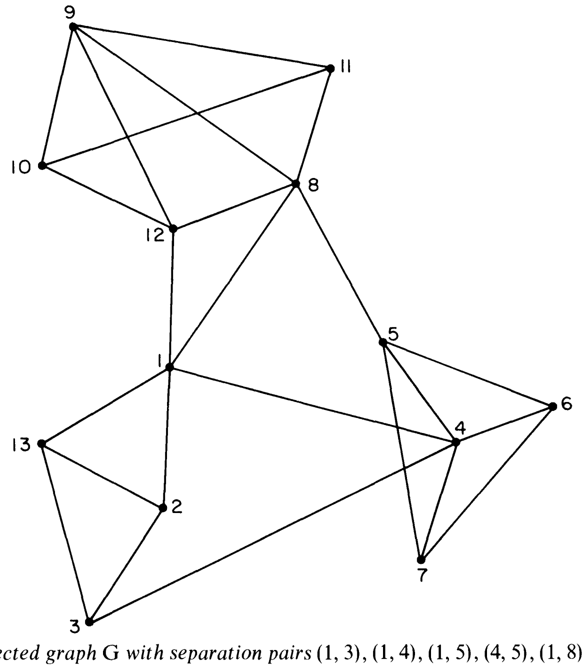


FIG. 1. *A biconnected graph* G *with separation pairs* (1, 3), (1, 4), (1, 5), (4, 5), (1, 8), (4, 8) and (8, 12)

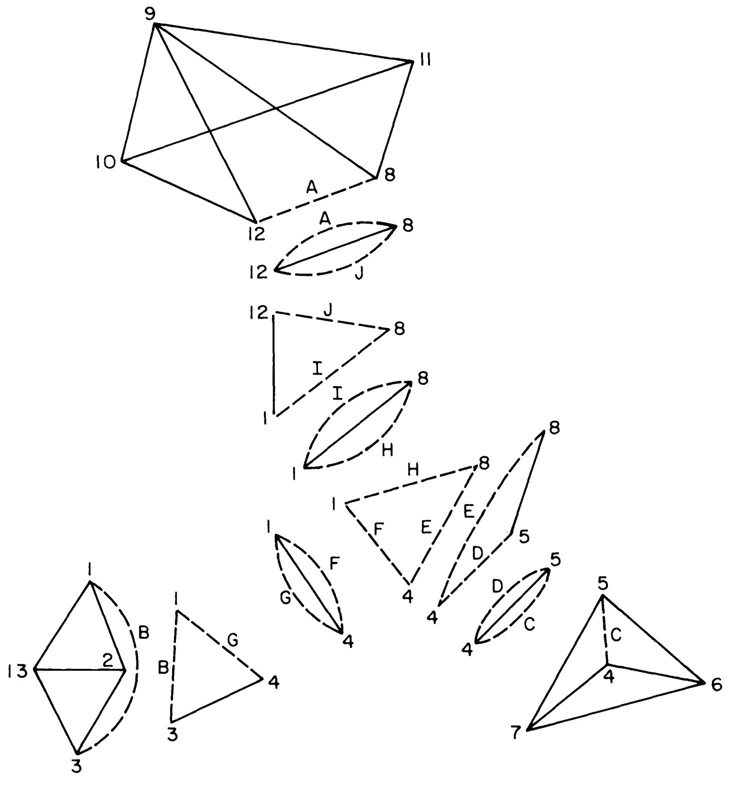


FIG. 2. *The split components of the graph* G *illustrated in Fig.* 1. *Triconnected components are formed by merging triangles* (1, 8, 4) *and* (4, 5, 8)

Graph algorithms require a systematic way of exploring a multigraph. We will use a method called *depth-first search.* To carry out a depth-first search of G, start from some vertex s and choose an edge leading from s to follow. Traversing the edge leads to a new vertex. Continue in this way, at each step selecting an unexplored edge leading from the most recently reached vertex which still has unexplored edges. If G is connected, each edge is traversed exactly once.

If G is undirected, a search of G imposes a direction on each edge of G given by the direction in which the edge is traversed during the search. Thus the search converts G into a directed multigraph G'.

LEMMA 3. *Let P be the directed multigraph generated by a depth-first search of a connected undirected multigraph G. Then P is a palm tree.*

Proof. See [5].

Depth-first search is important because the structure of paths in a palm tree is very simple. To implement a depth-first search of a multigraph, we use a simple recursive procedure which keeps a stack of the old vertices with possibly unexplored edges. To represent a multigraph, we use a set of *adjacency lists*, one for each vertex. If v is a vertex, adjacency list $A(v)$ contains all w such that (v, w) is an edge of G. These lists together comprise an *adjacency structure* for G. If G is undirected, each edge (v, w) is represented twice, once in $A(v)$ and once in $A(w)$. If G is directed, each edge is represented once.

Below is a recursive procedure to carry out a depth-first search. The exact search depends upon the order of edges in the adjacency lists. The procedure numbers the vertices from 1 to V in the order in which they are reached during the search, in addition to identifying tree arcs and fronds. Reference [5] gives a proof that the procedure is correct and requires $O(V + E)$ time to execute. It is easy to see that the vertices are numbered so that NUMBER(v) < NUMBER(w) if $v \xrightarrow{*} w$ in the generated spanning tree.

```
    PROCEDURE 1.
begin comment routine for depth-first search of a multigraph $G$ represented by
        adjacency lists $A(v)$. Variable $n$ denotes the last number assigned to a
        vertex;
    integer $n$;
    procedure DFS ($v, u$); begin comment vertex $u$ is the father of vertex $v$ in the
            spanning tree being constructed. The graph to be searched is
            represented by a set of adjacency lists $A(v)$;
        $n :=$ NUMBER ($v$) $:= n + 1$;
    $a$: comment dummy statement;
        for $w \in A(v)$ do begin
            if NUMBER ($w$) $= 0$ then begin
                    comment $w$ is a new vertex;
                    mark ($v, w$) as a tree arc;
                    DFS ($w, v$);
                $b$: comment dummy statement;
            end
            else if (NUMBER ($w$) < NUMBER ($v$)) and (($w \neq u$) or $\neg$ FLAG ($v$))
                then begin
                    comment the test is necessary to avoid exploring an edge
                        in both directions. FLAG ($v$) becomes false when the
                        entry in $A(v)$ corresponding to tree arc ($u, v$) is
                        examined;
                    mark ($v, w$) as a frond;
                $c$: comment dummy statement;
            end;
            if $w = u$ then FLAG ($v$) = false;
        end;
    end;
    $n := 0$;
    for $i := 1$ until $V$ do begin
        NUMBER ($i$) $:= 0$;
        FLAG ($i$) := true;
    end;
    comment the search starts at vertex $s$;
    DFS ($s$, 0);
end;
```

The dummy statements *a*, *b*, *c*, will be replaced when DFS is used to calculate other information about the graph. Figure 3 depicts the palm tree formed by applying DFS to the graph in Fig. 1.

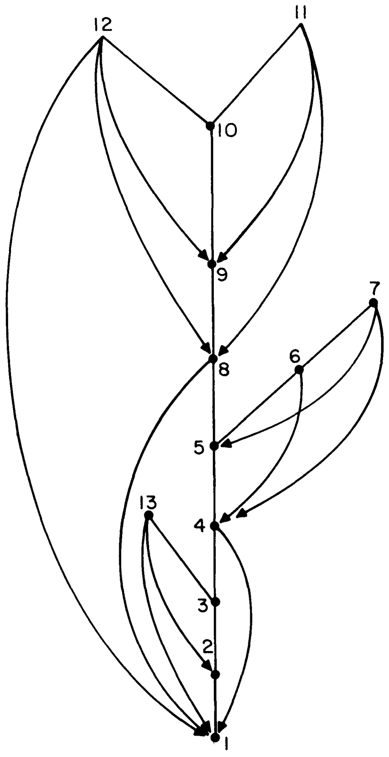


FIG. 3. *Palm tree produced by a depth-first search of graph* G *illustrated in Fig.* 1

2. An outline of the triconnectivity algorithm. This section sketches the ideas behind the triconnectivity algorithm. Later sections develop the detailed components. The algorithm is based on an idea of Auslander, Parter, and Goldstein ([18], [19]) for testing the planarity of graphs. Auslander, Parter, and Goldstein's idea gives rise to an $O(V)$ time algorithm for testing planarity, if depth-first search is used to order the calculations ([7], [8]). The same idea gives an $O(V + E)$ time algorithm for finding triconnected components.

Let G be an arbitrary biconnected multigraph. Suppose a cycle c is found in G. When the cycle is deleted from G, certain connected pieces remain; they are called *segments*. Auslander and Parter [18] show that G is planar if and only if

(i) any subgraph of G consisting of c plus a single segment is planar,

(ii) the segments may be combined consistently to give a planar embedding of the entire graph.

An efficient planarity algorithm may be developed from this result ([7], [8]). A similar result holds for the separation pairs of G, i.e., the following lemma.

LEMMA 4. *Let G be a biconnected multigraph and let c be a cycle in G. Let $S_1, \cdots, S_n$ be the subgraphs of $G - c$ such that e_1 and e_2 are edges of S_i if and only if some path p in G contains both e_1 and e_2, and no vertex of c lies between e_1 and e_2 in p. The segments S_i and the cycle c partition the edges of G. Let $\{a, b\}$ be a separation pair of G such that (a, b) is not a multiple edge. Then the following conclusions hold.*

(i) *Either a and b both lie on c, or a and b both lie in some segment S_i.*

(ii) *Suppose a and b both lie on c. Let p_1 and p_2 be the two paths comprising c which join a and b. Then either*

(a) *some segment S_i with at least two edges has only a and b in common with c, and some vertex v does not lie in S_i ($\{a, b\}$ is called a "type 1" separation pair), or*

(b) *no segment contains a vertex $v \neq a, b$ in p_1 and a vertex $w \neq a, b$ in p_2, and p_1 and p_2 each contain a vertex besides a and b ($\{a, b\}$ is called a "type 2" separation pair).*

(iii) *Conversely, any pair $\{a, b\}$ which satisfies* (a) *or* (b) *is a separation pair.*

It is easy to prove this lemma; a more technical version is proved in the next section. Lemma 4 gives rise to an efficient recursive algorithm for finding split components. We find a cycle in G and determine the segments formed when it is deleted. We test each segment for separation pairs by applying the algorithm recursively and we test the cycle for separation pairs by checking the criteria in Lemma 4. Recursive application of the algorithm requires finding cycles in subgraphs of G formed by combining a segment S_i and the initial cycle c.

We can make this algorithm very efficient by ordering the calculations using depth-first search. Each recursive call on the algorithm requires that we find a cycle in the piece of the graph to be tested for separation pairs. This cycle will consist of a simple path of edges not in previously found cycles plus a simple path of edges in old cycles. We use depth-first search to divide the graph into simple paths which may be assembled into these cycles. The first cycle c will consist of a sequence of tree arcs followed by one frond in P, the palm tree formed from G by depth-first search. The numbering of vertices is such that the vertices are in order by number along the cycle. Each segment will consist either of a single frond (v, w) or of a tree arc (v, w) plus a subtree with root w, plus all fronds which lead from the subtree. The search explores the segments in decreasing order of v and partitions each into simple paths consisting of a sequence of tree arcs followed by one frond.

Finding paths actually requires two searches because the pathfinding search must be carried out in a special order if it is to succeed, and certain preliminary calculations are necessary. The section on finding separation pairs describes the pathfinding process in detail and includes a version of Lemma 4 which characterizes separation pairs in terms of the generated paths. The section on finding split components indicates how these results may be used to determine the split components of a biconnected multigraph in $O(V + E)$ time.

To determine the triconnected components of an arbitrary multigraph, we eliminate multiple edges by splitting them off, creating a set of bonds with three edges. This requires $O(V + E)$ time if implemented correctly. Then we find the biconnected components of the resultant graph using the $O(V + E)$ algorithm described in [5] and [6]. Next, the split components of each biconnected component are found using the algorithm outlined above and presented in detail in the next two sections. This gives us the split components of the entire graph. The total size of the split components is $O(V + E)$, by Lemma 1. Next we identify the set of triple bonds $\mathscr{B}_3$ and the set of triangles $\mathscr{T}$. For each of these two sets, we construct an auxiliary graph S whose vertices are the elements of the set; two split components are joined by an edge in an auxiliary graph if they have a common virtual edge. The connected components of $S(\mathscr{B}_3)$ and $S(\mathscr{T})$ correspond to the bonds and polygons which are triconnected components of G. Finding these bonds and polygons requires $O(V + E)$ time. Below is an outline of the entire algorithm.

PROCEDURE 2.

```
procedure TRICONNECTIVITY ($G$); begin comment an outline of the tri-
        connected components algorithm;
    A: split off multiple edges of $G$ to form a set of triple bonds and a
       graph $G'$;
    B: find biconnected components of $G'$;
       for each biconnected component $C$ of $G'$ do
    C: find split components of $C$;
    D: combine triple bonds and triangles into bonds and polygons by
       finding connected components of corresponding auxiliary graphs;
end;
```

Steps A, B, and D all require $O(V + E)$ time if correctly implemented. Implementation of step B is described in [5]; implementation of steps A and D is left as an exercise. The hard step is step C, whose implementation is described in the next two sections. Based on the results of these sections, the entire triconnectivity algorithm has $O(V + E)$ time and space bounds.

3. Finding separation pairs. Let $G = (\mathscr{V}, \mathscr{E})$ be a biconnected multigraph with V vertices and E edges. The main problem in dividing G into its split components lies in finding its separation pairs. This section gives a simple criterion, based upon depth-first search, for identifying the separation pairs of a multigraph. Two depth-first searches and some auxiliary calculations must be carried out. These calculations form the first part of the split components algorithm, and are outlined below. The definitions for the quantities LOWPT1, ND, etc., used in the outline will be given subsequently.

Step 1. Perform a depth-first search on the multigraph G, converting G into a palm tree P. Number the vertices of G in the order they are reached during the search. Calculate LOWPT1 (v), LOWPT2 (v), ND (v), and FATHER (v) for each vertex v in P.

Step 2. Construct an acceptable adjacency structure A for P by ordering the edges in the adjacency structure according to the LOWPT1 and LOWPT2 values.

Step 3. Perform a depth-first search of P using the adjacency structure A. Renumber the vertices of A from V to 1 in the order they are *last* examined during the search. Partition the edges into disjoint simple paths. Recalculate LOWPT1 (v) and LOWPT2 (v) using the new vertex numbers. Calculate $A1$ (v), DEGREE (v), and HIGHPT (v) for each vertex v.

The details of these calculations appear below. From steps 1, 2 and 3, we get enough information to rapidly determine the separation pairs of G. Lemma 13 gives a condition for this purpose.

Suppose G is explored in a depth-first manner, giving a palm tree P. Let the vertices of P be numbered from 1 to V so that $v \xrightarrow{*} w$ in P implies $v < w$, if we identify vertices by their number. For any vertex v in P, let FATHER (v) be the father of v in the spanning tree of P. Let ND (v) be the number of descendants of v. Let LOWPT1 (v) $= \min(\{v\} \cup \{w | v \xrightarrow{*} - \rightarrow w\})$. That is, LOWPT1 ($v$) is the lowest vertex reachable from v by traversing zero or more tree arcs in P followed by at most one frond. Let LOWPT2 (v) $= \min[\{v\} \cup (\{w | v \xrightarrow{*} - \rightarrow w\} - \{\text{LOWPT1}(v)\})]$. That is, LOWPT2 ($v$) is the *second* lowest vertex reachable from v by traversing zero or more tree arcs followed by at most one frond of P, unless LOWPT1 (v) $= v$. In this case, LOWPT2 (v) $= v$.

LEMMA 5. LOWPT1 (v) $\xrightarrow{*} v$ *and* LOWPT2 (v) $\xrightarrow{*} v$ *in* P.

Proof. LOWPT1 (v) $\leqq v$ by definition. If LOWPT1 (v) $= v$, the result is immediate. If LOWPT1 (v) $< v$, there is a frond $u - \rightarrow$LOWPT1 (v) such that $v \xrightarrow{*} u$. Since $u - \rightarrow$LOWPT1 (v) is a frond, LOWPT1 (v) $\xrightarrow{*} u$. Since P is a tree, $v \xrightarrow{*} u$ and LOWPT1 (v) $\xrightarrow{*} u$, either $v \xrightarrow{*}$ LOWPT1 (v) or LOWPT1 (v) $\xrightarrow{*} v$. But LOWPT1 (v) $< v$. Thus it must be the case that LOWPT1 (v) $\xrightarrow{*} v \xrightarrow{*} u$, and the lemma holds for LOWPT1 (v). The proof is the same for LOWPT2 (v).

LEMMA 6. *Suppose* LOWPT1 (v) *and* LOWPT2 (v) *are defined relative to some numbering for which* $v \xrightarrow{*} w$ *in* P *implies* NUMBER (v) $<$ NUMBER (w). *Then* LOWPT1 (v) *and* LOWPT2 (v) *identify unique vertices independent of the numbering used.*

Proof. LOWPT1 (v) always identifies an ancestor of vertex v. Furthermore, LOWPTI (v) is the lowest numbered ancestor of v with a certain property relative to the palm tree P. Since the order of the ancestors of v corresponds to the order of their numbers, LOWPT1 (v) identifies a unique vertex independent of the numbering, i.e., the first ancestor of v along the path $1 \xrightarrow{*} v$ which has the desired property. (Any satisfactory numbering assigns 1 to the root of P.) The proof is the same for LOWPT2 (v).

The LOWPT values of a vertex v depend only on the LOWPT values of sons of v and on the fronds leaving v; it is easy to see that if vertices are identified by number, then

$$\text{LOWPT1}(v) = \min(\{v\} \cup \{\text{LOWPT1}(w) | v \rightarrow w\} \cup \{w | v - \rightarrow w\})$$

and

$$\text{LOWPT2}(v) = \min(\{v\} \cup ((\{\text{LOWPT1}(w) | v \rightarrow w\} \cup \{\text{LOWPT2}(w) | v \rightarrow w\}$$
$$\cup \{w | v - \rightarrow w\}) - \{\text{LOWPT1}(v)\})).$$

We also have ND (v) $= 1 + \sum_{v \rightarrow w}$ ND (w). We may calculate LOWPT values, ND, and FATHER for all vertices in $O(V + E)$ time by inserting the following statements for the dummy statements a, b, c in DFS. Numbering the vertices in the order they are reached during the search clearly guarantees that $v \xrightarrow{*} w$ implies $v < w$.

PROCEDURE 3.

```
comment additions to DFS for step 1;
a: LOWPT1 (v) := LOWPT2 (v) := NUMBER (v);
   ND (v) := 1;

b: if LOWPT1 (w) < LOWPT1 (v) then begin
     LOWPT2 (v) := min {LOWPT1 (v), LOWPT2 (w)};
     LOWPT1 (v) := LOWPT1 (w);
   end else if LOWPT1 (w) = LOWPT1 (v) then
               LOWPT2 (v) := min {LOWPT2 (v), LOWPT2 (w)};
   else LOWPT2 (v) := min {LOWPT2 (v), LOWPT1 (w)};
   ND (v) := ND (v) + ND (w);
   FATHER (w) := v;

c: if NUMBER (w) < LOWPT1 (v) then begin
      LOWPT2 (v) := LOWPT1 (v);
      LOWPT1 (v) := NUMBER (w);
   end else if NUMBER (w) > LOWPT1 (v) then
   LOWPT2 (v) := min {LOWPT2 (v), NUMBER (w)};
```

It is easy to verify that DFS as modified above will compute LOWPT1, LOWPT2, ND, and FATHER correctly in $O(V + E)$ time. (See [8], [17].) LOWPT1 may be used to test the biconnectivity of G, as described in [5]. The following lemma is important.

LEMMA 7. *If G is biconnected and $v \to w$,* LOWPT1 $(w) < v$ *unless* $v = 1$, *in which case* LOWPT1 $(w) = v = 1$. *Also,* LOWPT1 $(1) = 1$.

Proof. See [5].

Let ϕ be the mapping from the edges of P into $\{1, 2, \cdots, 2V + 1\}$ defined by:

(i) if $e = v - \to w$, $\phi(e) = 2w + 1$.

(ii) if $e = v \to w$ and LOWPT2 $(w) < v$, $\phi(e) =$ 2LOWPT1 (w).

(iii) if $e = v \to w$ and LOWPT2 $(w) \geqq v$, $\phi(e) =$ 2LOWPT1 $(w) + 1$.

Let A be an adjacency structure for P. A is called *acceptable* if the edges e in each adjacency list of A are ordered according to increasing value of $\phi(e)$.

LEMMA 8. *Let P be a palm tree of a biconnected graph G whose vertices are numbered so that $v \overset{*}{\to} w$ in P implies $v < w$. Then the acceptable adjacency structures of P are independent of the exact numbering scheme.*

Proof. If $v \to w$ in P, then by Lemma 5, LOWPT2 (w) is an ancestor of w. By Lemma 6, LOWPT2 (w) is a fixed vertex independent of the numbering. Since the order of the ancestors is independent of the numbering, the question as to whether LOWPT2 (w) is less than v is independent of the numbering. Since G is biconnected if $v \to w$ in P, then LOWPT1 $(w) \leqq v$ by Lemma 7. By Lemma 5, LOWPT1 (w) is an ancestor of w. Since LOWPT1 $(w) \leqq v$, LOWPT1 (w) must be an ancestor of v. By Lemma 6, the vertex corresponding to LOWPT1 (w) is independent of the numbering scheme. Similarly, if $v - \to w$, then by Lemma 3 and the definition of a palm tree, w is an ancestor of v. But the order of the ancestors of v is identical to the order of their numbers, and this order is independent of the numbering. Thus the acceptable adjacency structures A for P depend only on P and not on the exact numbering.

In general, a palm tree P has many acceptable adjacency structures. Given a satisfactory numbering of the vertices of P, we may easily construct an acceptable adjacency structure A by using a radix sort with $2V + 1$ buckets. The following procedure gives the sorting algorithm, which is step 2 of the calculations. All vertices are identified by number. It is obvious that the sorting procedure requires $O(V + E)$ time.

PROCEDURE 4.

```
**comment** construction of ordered adjacency lists;
**for** i := 1 **until** $2*V + 1$ **do** BUCKET ($i$) := the empty list;
**for** ($v$, $w$) an edge of $G$ **do begin**
    compute $\phi((v, w))$;
    add ($v$, $w$) to BUCKET ($\phi(v, w)$);
**end**;
**for** $i$ := 1 **until** $V$ **do** $A(i)$ := the empty list;
**for** $i$ := 1 **until** $2*V + 1$ **do**
    **for** $(v, w) \in$ BUCKET ($i$) **do** add $w$ to end of $A(v)$;
```

In step 3 of the calculations, we perform a depth-first search of P using the acceptable adjacency structure A given by step 2. This search generates a set of paths in the following manner: each time we traverse an edge we add it to the path being built. Each time we traverse a frond, the frond becomes the last edge of the current path. Thus each path consists of a sequence of tree arcs followed by a single frond. Because of the ordering imposed on A, each path terminates at the lowest possible vertex, the initial path is a cycle, and each path except the first is simple and has only its initial and terminal vertex in common with previously generated paths ([7], [8]).

If $p:s \overset{*}{\Rightarrow} f$ is a generated path, we may form a cycle by adding the tree path $f \overset{*}{\rightarrow} s$ to p. The cycles formed in this way are the cycles generated by recursive calls on the basic triconnectivity algorithm explained in the last section.

We need only minimal information about the paths. Let the vertices of P be numbered so that $v \overset{*}{\rightarrow} w$ implies $v \leqq w$. Let A1(v) be the first vertex in A(v). If $v - \rightarrow w$ is the *first* frond explored in step 3 which terminates at w, let HIGHPT (w) $= v$. Let DEGREE (v) be the number of edges incident to vertex v. Step 3 numbers the vertices from V to 1 in the order they are *last* examined during the search. It is clear that this numbering guarantees that $v < w$ if $v \overset{*}{\rightarrow} w$. Step 3 also computes LOWPT1 (v), HIGHPT (v), A1 (v), and DEGREE (v) with respect to new numbering. Procedure 5, based on DFS, will perform step 3 in $O(V + E)$ time.

PROCEDURE 5.

```
step 3:     **begin comment** routine to generate paths in a biconnected palm
            tree with specially ordered adjacency lists $A(v)$. Vertex $s$ is a
            global variable denoting the start vertex of the current path. $s$
            is initialized to 0. Variable $m$ denotes the last number assigned
            to a vertex;
            **procedure** PATHFINDER ($v$); **begin**
X:              NEWNUM ($v$) := $m$ − ND ($v$) + 1;
```

```
            for $w \in A(v)$ do
                if $s = 0$ then begin
                    $s := v$;
                    start new path;
                end;
                add $(v, w)$ to current path;
                if $v \to w$ then begin
                    PATHFINDER $(w)$;
Y:                  $m := m - 1$;
            end else begin comment $v - \to w$;
                if HIGHPT (NEWNUM $(w)$) = 0 then
                    HIGHPT (NEWNUM$(w)$) :=
                        NEWNUM $(v)$;
                output current path;
                $s := 0$;
            end;
                end;
                $s := 0$;
Z:              $m := V$;
                for $i := 1$ until $V$ do NEWNUM $(i)$ := HIGHPT $(i)$ := 0;
                comment vertex 1 is the start vertex of the search;
                PATHFINDER (1);
                for all vertices $V$ do
                    compute A1 $(v)$, DEGREE $(v)$, LOWPT1 $(v)$, and
                    LOWPT2 $(v)$ using the new numbering;
            end;
```

Step 3 numbers the vertices from V to 1 in the order they are *last* reached during the search. However, each vertex must actually be assigned a number the *first* time it is reached, in order for the calculation of HIGHPT to proceed correctly. In order to accomplish this, variable i is set equal to V when the search begins (statement Z). The value of i is decreased by one each time a new vertex is discovered (statement Y). Thus when a vertex v is first reached, i is equal to the number we want to assign to v *minus* the number of vertices to be examined before v is examined for the last time. But the vertices to be reached between the time v is first examined and the time v is last examined are just the proper descendants of v. Thus if we assign the number $i - \text{ND}(v) + 1$ to v when v is first examined (statement X), the numbering will be correct. The other calculations performed in step 3 are straightforward and easy to implement. The palm tree for the graph G of Fig. 1 is illustrated in Fig. 4 along with LOWPT values and the set of paths generated by step 3.

Let G be a biconnected multigraph on which steps 1, 2, and 3 have been performed, giving a palm tree P and the sets of values defined above. Let A with adjacency lists $A(v)$ be the acceptable adjacency structure constructed in step 2. Let the vertices of G be identified by the numbers assigned in step 3. We need one more definition. If $u \to v$ and v is the *first* entry in $A(u)$, then v is called the *first son* of u. (For each vertex v, $A1(v)$, the first son of v if one exists, is calculated in step 3.)

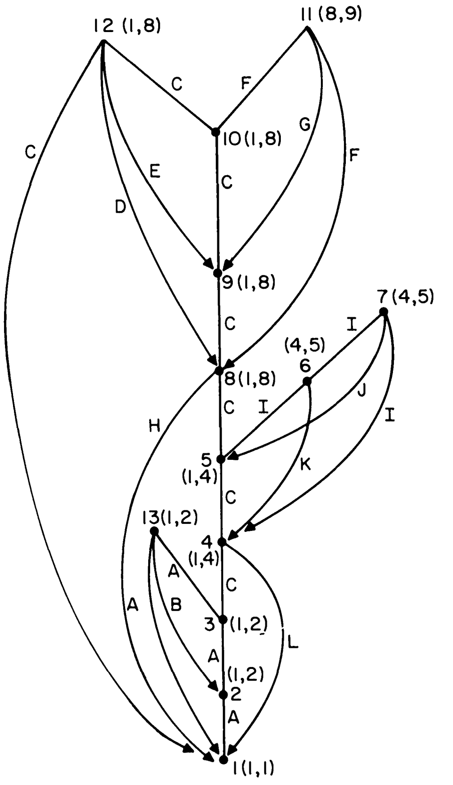


FIG. 4. *Ordered palm tree of graph* G *after pathfinding search with* LOWPT1 *and* LOWPT2 *values in parentheses*

Type 1 *pairs*: (1, 4), (1, 5), (4, 5), (1, 8), (1, 3)

Type 2 *pairs*: (4, 8), (8, 12).

Paths: A:(1, 2, 3, 13, 1) B:(13, 2) C:(3, 4, 5, 8, 9, 10, 12, 1) D:(12, 8) E:(12, 9) F:(10, 11, 8) G:(11, 9) H:(8, 1) I:(5, 6, 7, 4) J:(7, 5) K:(6, 4) L:(4, 1)

If $u_0 \to u_1 \to \cdots \to u_n$, and u_i is a first son of u_{i-1} for $1 \leqq i \leqq n$, then u_n is called a *first descendant* of u_0. The sequence of tree arcs $u_0 \to u_1 \to u_2 \to \cdots \to u_n$ is part of a path generated by step 3. The lemmas below give the properties we need to determine the separation pairs of G.

LEMMA 9. *Let $A(u)$ be the adjacency list of vertex u. Let $u \to v$ and $u \to w$ be tree arcs with v occurring before w in $A(u)$. Then $u < w < v$.*

Proof. Step 3 numbers the vertices from V to 1 in the order they are last examined in the search. If $u \to v$ is explored before $u \to w$, v will be examined last before w is examined last, and v will receive a higher number. Clearly u will be last examined after both v and w are last examined, so u receives the smallest number of the three vertices.

LEMMA 10. *A is acceptable with respect to the numbering given by step* 3.

Proof. The sorting in step 2 creates an acceptable adjacency structure for the original numbering. By Lemma 9, $u \to v$ implies $u < v$ and hence by Lemma 8, A is acceptable for the new numbering.

LEMMA 11. *If v is a vertex and $D(v)$ is the set of descendants of v, then* $D(v) = \{x|v \leqq x < v + \mathrm{ND}(v)\}$. *If w is a first descendant of v, then* $D(v) - D(w) = \{x|v \leqq x < w\}$.

Proof. Suppose we reverse all the adjacency lists $A(v)$ and use them to specify a depth-first search of P. Vertices will be examined for the first time in ascending order from 1 to V, if vertices are identified by their step 3 number. Thus descendants of v are assigned consecutive numbers from v to $v + \mathrm{ND}(v) - 1$. If w is a first descendant of v, vertices in $D(w)$ will be assigned numbers after all vertices in $D(v) - D(w)$. Thus $D(v) - D(w) = \{x|v \leqq x < w\}$.

LEMMA 12. *Let $\{a, b\}$ be a separation pair in G with $a < b$. Then $a \xrightarrow{*} b$ in the spanning tree T of P.*

Proof. Since $a < b$, a cannot be a descendant of b. Suppose b is not a descendant of a. Let E_i, for $1 \leqq i \leqq k$, be the separation classes with respect to $\{a, b\}$. Let $S = \mathcal{V} - D(a) - D(b)$. The vertices S define a subtree in T containing neither a nor b, so $E(S)$ must be contained in some separation class, say E_1. Let c be any son of a. $E(D(c))$ must be contained in some separation class. But since G is biconnected, and $a \neq 1$, LOWPT1 $(c) < a$, by Lemma 7. Thus some edge is incident to a vertex in S and to a vertex in $D(c)$. Thus $E(D(c)) \subseteq E_1$. A similar argument shows that edges incident to any descendant of b are in E_1. But this means that $E_1 = E$, and $\{a, b\}$ cannot be a separation pair.

LEMMA 13. *Suppose $a < b$. Then $\{a, b\}$ is a separation pair of G if and only if either* (i), (ii), or (iii) below holds.

(i) *There are distinct vertices $r \neq a, b$ and $s \neq a, b$ such that $b \rightarrow r$,* LOWPT1 $(r) = a$, LOWPT2 $(r) \geqq b$, *and s is not a descendant of r.* (*The pair $\{a, b\}$ is called a "type* 1" *separation pair. The type* 1 *pairs for the graph in Fig.* 4 *are* (1, 3), (1, 4), (1, 5), (4, 5) *and* (1, 8).)

(ii) *There is a vertex $r \neq b$ such that $a \rightarrow r \xrightarrow{*} b$; b is a first descendant of r (i.e., a, r, and b lie on a common generated path); $a \neq 1$; every frond $x -\!\!\rightarrow y$ with $r \leqq x < b$ has $a \leqq y$; and every frond $x -\!\!\rightarrow y$ with $a < y < b$ and $b \rightarrow w \xrightarrow{*} x$ has* LOWPT1 $(w) \geqq a$. (*$\{a, b\}$ is called a "type* 2" *separation pair. The type* 2 *pairs for the graph in Fig.* 4 *are* (4, 5) *and* (8, 12)).

(iii) *(a, b) is a multiple edge of G and G contains at least four edges.*

Proof. The converse part of the lemma is easiest to prove. Suppose pair $\{a, b\}$ satisfies (i), (ii), or (iii). Let E_i for $1 \leqq i \leqq k$ be the separation classes of G with respect to $\{a, b\}$. Suppose $\{a, b\}$ satisfies (i). Then the edge (b, r) is contained in some separation class, say E_1. Every tree arc with an endpoint in $D(r)$ has the other endpoint in $D(r) \cup \{a, b\}$. Also, since LOWPT1 $(r) = a$ and LOWPT2 $(r) \geqq b$, every frond with an endpoint in $D(r)$ has the other endpoint in $D(r) \cup \{a, b\}$. Thus E_1 consists of all edges with an endpoint in $D(r)$. No other edges are in E_1, and the edges incident to vertex s must be in some other class, say E_2. Since E_1 and E_2 each contain two or more edges, $\{a, b\}$ is a separation pair.

Suppose $\{a, b\}$ satisfies (ii). Let $S = D(r) - D(b)$. All edges incident to a vertex in S are in the same separation class, say E_1. Since b is a first descendant of r, $S = \{x|r \leqq x < b\}$ by Lemma 1. Let $b_1, b_2, \cdots, b_n$ be the sons of b in the order they occur in $A(b)$. Let $i_0 = \min \{i|\text{LOWPT1}(b_i) \geqq a\}$. By the ordering imposed on A, $i < i_0$ implies LOWPT1 $(b_i) < a$, and $i \geqq i_0$ implies LOWPT1 $(b_i) \geqq a$. By (ii), every frond with tail in S has its head in $S \cup \{a\}$. Also by (ii), every frond with head in S has its tail in $S \cup \{b\} \cup (\bigcup_{i \geqq i_0} D(b_i))$. Every edge with an endpoint in $D(b_i)$, $i \geqq i_0$, has its other endpoint in $S \cup \{a, b\} \cup D(b_i)$. Thus the class E_1 contains *at least* all edges with an endpoint in S, and *at most* all edges with an endpoint in $S \cup (\bigcup_{i \geqq i_0} D(b_i))$. Since $a \neq 1$, the edges incident to the root of P cannot be in E_1, and therefore $\{a, b\}$ is a separation pair.

Now we must prove the direct part of the lemma. Suppose that $\{a, b\}$ is a separation pair with $a < b$. If (a, b) is a multiple edge of G, then it is clear that $\{a, b\}$ satisfies (iii). Thus suppose that (a, b) is not a multiple edge of G. By Lemma 12, $a \xrightarrow{*} b$. Let E_i, for $1 \leqq i \leqq k$, be the separation classes of G with respect to $\{a, b\}$. Let v be the son of a such that $a \to v \xrightarrow{*} b$, $S = D(v) - D(b)$, and $X = V - D(a)$. (Either S or X or both may be empty.) $E(S)$ and $E(X)$ are each contained in a separation class, say $E(S) \subseteq E_1$ and $E(X) \subseteq E_2$.

Let $a_i \neq v$ be a son of a. If a has such a son, LOWPT1 $(a_i) < a$. This means that $E(D(a_i)) \subseteq E_2$. Let $Y = X \cup (\bigcup_i D(a_i))$. Let $b_1, b_2, \cdots, b_n$ be the sons of b in the order they occur on the adjacency list of b. Let $E(D(b_i))$ be the set of edges with an endpoint in $D(b_i)$. The separation classes must be unions of the sets $E(S)$, $E(Y)$, $\{(a, b)\}$, $E(D(b_1))$, $E(D(b_2))$, $\cdots$ $E(D(b_n))$.

If $E(D(b_i)) = E_j$ for some i and j, then LOWPT1 $(b_i) = a$ since G is biconnected, and this means LOWPT1 $(b_i) < b$ by Lemma 7. Also, LOWPT2 $(b_i) \geqq b$. Since $\{a, b\}$ is a separation pair, there must be a separation class other than E_j and $\{(a, b)\}$. Thus there is a vertex s such that $s \neq a, s \neq b$, and $s \notin D(b_i)$. This means that $\{a, b\}$ satisfies (i) where r is b_i.

Suppose now that no $E(D(b_i))$ is by itself a separation class. Let $i_0 = \min \{i | \text{LOWPT1}\ (b_i) \geqq a\}$. If $i \geqq i_0$, then since G is biconnected, it must be the case that LOWPT1 $(b_i) < b$, and the separation classes are $E_1 = E(S) \cup (\bigcup_{i \geqq i_0} E(D(b_i)))$, $E_2 = E(Y) \cup (\bigcup_{i < i_0} E(D(b_i)))$, $E_3 = \{(a, b)\}$. (E_3 may be empty.) We have $v \neq b$ since $\{a, b\}$ is not a type 1 pair and $a \neq 1$ since E_2 is nonempty. If $x - \to y$ is a frond with $v \leqq x < b$, then $x \in S$, $(x, y) \in E_1$, and $a \leqq y$. If $x - \to y$ is a frond with $a < y < b$ and $b \to b_i \xrightarrow{*} x$, then $y \in S$, $(x, y) \in E_1$, and $i \geqq i_0$, which means that LOWPT1 $(b_i) \geqq a$. We must verify one more condition to show that (ii) holds, i.e., that b is a first descendant of v. Since G is biconnected, LOWPT1 $(v) < a$. Thus some frond with tail in $D(v)$ has head less than a. By the ordering imposed on A and the definition of a first descendant, there exists some frond $x - \to y$ with $x \in D(v)$ and $y < a$ such that x is a *first* descendant of v. If b were not a first descendant of v, then x would be in S, and E_1 and E_2 could not be distinct separation classes. Thus b is a first descendant of v, and (ii) holds with $r = v$. This completes the proof of the direct part of the lemma.

Lemma 13 and its proof are worth pondering carefully. The lemma gives three easy-to-apply conditions for separation pairs. Conditions (i) and (ii) identify the nontrivial separation pairs of the multigraph. Condition (iii) handles multiple edges. Condition (i) requires that a simple test be performed on each tree arc of P. Thus testing for type 1 pairs requires $O(V)$ time. Testing for type 2 pairs is somewhat harder, but may be done in $O(V + E)$ time using another depth-first search. Let $\{a, b\}$ be a type 2 pair satisfying $a \to r \xrightarrow{*} b$, and $i_0 = \min \{i | \text{LOWPT2}\ (b_i) \geqq a\}$, where $b_1, b_2, \cdots, b_n$ are the sons of b in the order they occur in $A(b)$. Then one separation class with respect to $\{a, b\}$ is $E(\{x | r \leqq x < b_{i_0} + \text{ND}\ (b_{i_0})\} - \{b\})$. This follows from the proof of Lemma 13. The new numbering, which satisfies the somewhat strange condition in Lemma 9, thus makes it easy to determine the separation classes and to divide the graph once a separation pair is found. An algorithm for finding split components based on Lemma 13 is given in the next section.

4. Finding split components. We find split components by examining the generated paths in order and testing for separation pairs with Lemma 13. Separation pairs will be of several types. Multiple edges and type 1 pairs are easy to recognize. So are type 2 pairs $\{a, b\}$, where $a \rightarrow v \rightarrow b$ and v has degree two. Other type 2 pairs are somewhat harder to recognize. Let c be the first path generated (a cycle). The cycle consists of a set of tree arcs $1 \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n$ followed by a frond $v_n - \rightarrow 1$. The vertex numbering is such that $1 < v_1 < \cdots < v_n$. When c is removed, the graph falls into several connected pieces, called *segments*. Each segment consists either of a single frond (v_i, v_j), or of a tree arc (v_i, w) plus a subtree with root w plus all fronds leading from the subtree. The order of path generation is such that all paths in one segment are generated before paths in any other segment, and the segments are explored in decreasing order of v_i.

Suppose we repeat the pathfinding search, using it now to find split components. We shall keep a stack of edges, adding edges to this stack as we back up over them during the search. Each time we find a separation pair, we remove a set of edges from the stack corresponding to a split component. We add a virtual edge corresponding to the split both to the component and to the edge stack. We also need to update various pieces of information, since the fathers of vertices and the degrees of vertices may change when a graph is split. The complete pathfinding search will create a complete set of split components. Assembling the split components to give the triconnected components is then a simple matter.

To identify type 2 pairs, we keep a stack (called TSTACK) of triples (h, a, b). The pair $\{a, b\}$ is a possible type 2 pair and h denotes the largest numbered vertex in the corresponding split component. The pairs are in nested order on the stack; that is, if v_i is the current vertex being examined by the pathfinding search, and (h_1, a_1, b_1), (h_2, a_2, b_2), $\cdots$, (h_k, a_k, b_k) are on TSTACK, then $a_k \leqq a_{k-1} \leqq \cdots \leqq a_2 \leqq a_1 \leqq v_i \leqq b_1 \leqq b_2 \leqq \cdots \leqq b_k$. Furthermore, all the a_j and b_j are vertices on the cycle c.

We update TSTACK in the following ways.

1. Each time we traverse a new path $p: s \overset{*}{\Rightarrow} f$, we delete all triples (h_j, a_j, b_j) on top of the stack with $a_j > f$. If p has second vertex $v \neq f$, let $x = v + \mathrm{ND}(v) - 1$. Otherwise let $x = s$. Let $y = \max \{h_j |$ triple (h_j, a_j, b_j) was deleted from TSTACK$\}$. If (h_k, a_k, b_k) was the last triple deleted, we add $(\max(x, y), f, s)$ to the stack. If no triple was deleted, we add (x, f, s) to the stack.

2. When we back up over a tree arc $v_i \rightarrow v_{i+1}$ with $v_i \neq 1$, we delete all entries (h_j, a_j, b_j) on top of TSTACK satisfying HIGHPT $(v_i) > h_j$. This test is necessary to guarantee that entries not corresponding to type 2 pairs don't accumulate on TSTACK.

We use TSTACK to find separation pairs in the following way: whenever we back up along a tree arc $v_i \rightarrow v_{i+1}$ during the pathfinding search, we examine the top triple (h_1, a_1, b_1) on TSTACK. If $v_i \neq 1$, $a_1 = v_i$, and $a_i \neq$ FATHER (b_i), $\{a_1, b_1\}$ is a type 2 separation pair. If DEGREE $(v_{i+1}) = 2$ and v_{i+1} has a son, then v_i and the son of v_{i+1} form a type 2 separation pair. We split off components corresponding to type 2 pairs until these two conditions give us no more components. (Simultaneously, we test for components corresponding to multiple edges and split these off.) Then we apply Lemma 13 to test whether $\{v_i, \text{LOWPT1}(v_{i+1})\}$ is a type 1 pair, splitting off a component if necessary. (Again, we need to check for a multiple-edge component.)

We handle the recursive part of the algorithm in the following way: traversing a path $p:s \overset{*}{\Rightarrow} f$ which starts on c means the search is entering a new segment. Vertex f must be the lowest vertex in the segment by the ordering imposed on the pathfinding search. After we update TSTACK as described above, if p contains more than one edge we place an end-of-stack marker on TSTACK and continue finding paths. This corresponds to a recursive call of the basic triconnectivity algorithm. When we back up over the first edge of p, we delete entries from TSTACK all the way down to the end-of-stack marker. This corresponds to popping up from the recursion.

One more point needs explanation: the reason we use LOWPT2 as well as LOWPT1 to construct A, the acceptable adjacency structure which determines the pathfinding search order. This step is necessary so that all multiple edges are handled correctly. Suppose v is a vertex, and $w_1, w_2, \cdots, w_k$ are the sons of v such that $\text{LOWPT1}(w_i) = u$. Further suppose that $v - \to u$. Let the w_i be ordered as in $A(v)$. There is some i_0 such that $i \leqq i_0 \Rightarrow \text{LOWPT2}(w_i) < v$ and $i > i_0 \Rightarrow \text{LOWPT2}(w_i) \geqq v$. In $A(v)$, u will appear after all the w_i with $1 \leqq i \leqq i_0$. If $i > i_0$, then $\{u, w_i\}$ is a type 1 separation pair; splitting off the corresponding component produces a new (virtual) frond $v - \to u$. It is important that all the w_i with $i > i_0$ appear together in $A(v)$ so that these virtual fronds may be located and combined to give split components which are bonds.

Below is an ALGOL-like procedure to find split components based on the ideas outlined above. The procedure is applicable to any biconnected multigraph for which steps 1, 2, and 3 described in the previous section have been carried out.

PROCEDURE 6.

```
procedure SPLIT (G); begin
    comment procedure to determine split components of G, a biconnected
        multigraph on which steps 1, 2 and 3 have been carried out. G is repre-
        sented by a set of properly ordered adjacency lists A(v). TSTACK
        contains triples representing possible type 2 separation pairs. ESTACK
        contains edges backed up over during search. Other variables have been
        defined in the previous section;
    procedure PATHSEARCH (v); begin
        comment this recursive procedure repeats the pathfinding search, finding
            separation pairs and splitting off components as it proceeds. It is
            based on the material in this section and the last. Vertex v is the
            current vertex in the depth-first search;
        for w ∈ A(v) do
            if $v \to w$ then begin
A:              if $v \to w$ is first edge of a path then begin
                    $y := 0$;
            while (h, a, b) on TSTACK has $a >$ LOWPT1 (w) do begin
                y := max (y, h);
                delete (h, a, b) from TSTACK;
            end;
            if no triples deleted from TSTACK then add
                (w + ND (w) − 1, LOWPT1 (w), v) to TSTACK
            else if (h, a, b) last triple deleted then add
                (max {y, w + ND (w) − 1}, LOWPT1 (w), b) to
                TSTACK;
            add end-of-stack marker to TSTACK;
        end;
        PATHSEARCH (w);
        add (v, w) to ESTACK;
B:      while $v \neq 1$ and ((DEGREE (w) = 2) and (A1 (w) > w) or
                (h, a, b) on TSTACK satisfies (v = a)) do begin
                comment test for type 2 pairs;
            if (h, a, b) on TSTACK has (a = v) and
                (FATHER (b) = a)
                then delete (h, a, b) from TSTACK;
            else begin
                if (DEGREE (w) = 2) and (A1 (w) > w) do begin
                j = j + 1;
                    add top two edges (v, w) and (w, x) on ESTACK
                        to new component;
                    add (v, x, j) to new component;
                    if (y, z) on ESTACK has (y, z) = (x, v) then begin
                        FLAG := true;
                        delete (y, z) from ESTACK and save;
                    end;
E:          end else if (h, a, b) on TSTACK satisfies v = a and
                    $a \neq$ FATHER (b) then begin
                j = j + 1;
                delete (h, a, b) from TSTACK;
                while (x, y) on ESTACK has ($a \leqq x \leqq h$) and
                    ($a \leqq y \leqq h$) do
                    if (x, y) = (a, b) then begin
                    FLAG := TRUE;
                        delete (a, b) from TSTACK and save;
                end else begin
                    delete (x, y) from ESTACK and add to current
                        component;
                    decrement DEGREE (x), DEGREE (y);
                end
                add (a, b, j) to new component;
                x := b;
            end;
            if FLAG then begin
                FLAG := false;
                $j := j + 1$;
                add saved edge, $(x, v, j - 1)$, $(x, v, j)$ to new
                    component;
                decrement DEGREE $(x)$, DEGREE $(v)$;
            **end**;
            add $(v, x, j)$ to ESTACK;
            increment DEGREE $(x)$, DEGREE $(v)$;
            FATHER $(x) := v$;
            **if** A1 $(v) \overset{*}{\rightarrow} x$ **then** A1 $(v) = x$;
            $w := x$;
        **end**;
        **comment** test for a type 1 pair;
G:      **if** (LOWPT2 $(w) \geqq v$) **and** ((LOWPT1 $(w) \neq 1$) **or**
            (FATHER $(v) \neq 1$)
                **or** $(w > 3)$)
            **then begin**
                $j := j + 1$;
                **while** $(x, y)$ on top of ESTACK has
                    $(w \leqq x < w +$ ND $(w))$ **or**
                        $(w \leqq y < w +$ ND $(w))$
                    **then begin**
                        delete $(x, y)$ from ESTACK;
                        add $(x, y)$ to new component;
                        decrement DEGREE $(x)$, DEGREE $(y)$;
                    **end**;
                add $(v$, LOWPT1 $(w), j)$ to new component;
                **if** A1 $(v) = w$ **then** A1 $(v) :=$ LOWPT1 $(w)$;
                **comment** test for multiple edge;
                **if** $(x, y)$ on top of ESTACK has
                    $(x, y) = (v$, LOWPT1 $(w))$
                        **then begin**
                    $j := j + 1$;
                    add $(x, y)$, $(v$, LOWPT1 $(w), j - 1)$,
                        $(v$, LOWPT1 $(w), j)$ to new component;
                    decrement DEGREE $(v)$,
                        DEGREE (LOWPT1 $(w)$);
                **end**;
                **if** LOWPT1 $(w) \neq$ FATHER $(v)$ **then begin** add
                        $(v$, LOWPT1 $(w), j)$ to ESTACK;
                    increment DEGREE $(v)$,
                        DEGREE (LOWPT1 $(w)$);
                **end else begin**
                    $j := j + 1$;
                    add $(v$, LOWPT1 $(w), j - 1)$,
                        $(v$, LOWPT1 $(w), j)$, tree arc
                            (LOWPT1 $(w)$, $v$) to new component;
                    mark tree arc (LOWPT1 $(w)$, $v$) as virtual edge $j$;
```

```
                                **end**;
                            **end**;
C:                      **if** $v \to w$ is the first edge of a path *then* delete all entries on
                                TSTACK down to and including end-of-stack marker;
D:                      **while** $(h, a, b)$ on ESTACK has HIGHPT $(v) > h$, **do** delete
                                $(h, a, b)$ from TSTACK;
                    **end else begin comment** $v - \to w$;
F:                      **if** $v - \to w$ is first (and last) edge of a path **then begin**
                            $y := 0$;
                            **while** $(h, a, b)$ on TSTACK has $a > w$ **do begin**
                                $y := \max(y, h)$;
                                delete $(h, a, b)$ from TSTACK;
                            **end**;
                            **if** no triples deleted from TSTACK **then add** $(v, w, v)$ to
                                TSTACK;
                            **if** $(h, a, b)$ last triple deleted **then add** $(y, w, b)$ to TSTACK;
                        **end**;
                        **if** $w$ = FATHER $(v)$ **then begin**
                            $j := j + 1$;
                            add $(v, w)$, $(v, w, j)$, tree arc $(w, v)$ to new component;
                            decrement DEGREE $(v)$, DEGREE $(w)$;
                            mark tree arc $(w, v)$ as virtual edge $j$;
                        **end else** add $(v, w)$ to ESTACK;
        **end**; **end**;
        $j := 0$;
        FLAG := false;
        PATHSEARCH (1);
    **end**;
```

LEMMA 14. SPLIT *correctly divides a biconnected multigraph G into split components.*

Proof. We must prove two things: (i) if G is triconnected, SPLIT will not split it, and (ii) if G is not triconnected, the algorithm will split it. Once we have these two facts, we may prove the lemma by induction on the number of edges in the graph. The tests for multiple edges, for type 1 separation pairs, and for degree-two vertices are straightforward. (The type 1 test (G in PATHSEARCH) includes the condition (LOWPT1 $(w) \neq 1$) **or** (FATHER $(v) \neq 1$) **or** $(w > 3)$ to make sure that some vertex lies outside the corresponding split component.) These tests will discover a separation pair of the correct type if one exists, and they will not report a separation pair if one does not exist. Thus we must only show that the type 2 test works correctly on multigraphs with no degree-two vertices, multiple edges or type 1 separation pairs, and we will have verified (i) and (ii).

Suppose G is a biconnected multigraph with no degree-two vertices, multiple edges, or type 2 separation pairs. Let us consider the type 2 test and the changing contents of TSTACK as the search of G progresses. If (h_1, a_1, b), $\cdots$, (h_k, a_k, b_k) are the contents of TSTACK above the highest end-of-stack marker, and if v is the vertex currently being examined during the search, then $a_k \leqq a_{k-1} \leqq \cdots \leqq a_1 \leqq v \leqq b_1 \leqq \cdots \leqq b_k$. This follows by induction from an examination of the possible changes that can be made in TSTACK (statements A, B, C, D, E, F in PATHSEARCH). Furthermore, $a_k, a_{k-1} \cdots v, b_1 \cdots b_k$ all lie on the cycle corresponding to the current recursive call of the basic triconnectivity algorithm.

Suppose (h, a, b) on TSTACK is found to satisfy the type 2 test when the search returns along a tree arc $v \to w$. The test (B, E in PATHSEARCH) states that $a = v$, $v \neq 1$, and FATHER $(b) \neq a$. It follows that $r = A1\,(a) \neq b$ satisfies $a \to r \overset{*}{\to} b$ and that b is a first descendant of r (that is, a, r, and b lie on a common generated path). If some frond $x - \to y$ with $r \leqq x < b$ had $a > y$, the triple on TSTACK corresponding to (h, a, b) would have been deleted from TSTACK when the frond was explored (A or F in PATHSEARCH). Similarly, if some frond $x - \to y$ with $a < y < b$ and $b \to w \overset{*}{\to} x$ had LOWPT1 $(w) < a$, the triple on TSTACK corresponding to (h, a, b) would have been deleted by the HIGHPT test when vertex y was examined (D in PATHSEARCH). It follows that $\{a, b\}$ is a type 2 separation pair by Lemma 13.

Conversely, suppose G has a type 2 pair $\{a, b\}$. Let $b_1, \cdots, b_n$ be the sons of b in the order they occur in $A(b)$. Let $i_0 = \min\{i | \text{LOWPT1}\,(b_i) \geqq a\}$. If i_0 exists, then $(b_{i_0} + \text{ND}\,(b_{i_0}), \text{LOWPT1}\,(b_i), b)$ will be placed on TSTACK when tree arc $b \to b_i$ is explored. This triple may be deleted from TSTACK, but it will always be replaced by a triple of the form (h, x, b), with LOWPT1 $(b_i) \geqq x \geqq a$. Eventually such a triple will satisfy the type 2 test, unless some other type 2 pair is found first. If i_0 does not exist, let (i, j) be the first edge traversed after b is reached such that $a \leqq i$ and $j \leqq b$. If $i - \to j$, then (i, j, i) will be placed on TSTACK, possibly modified, and eventually selected as a type 2 pair, unless some other type 2 pair is found first. If $i \to j$, then $(j + \text{ND}\,(j), \text{LOWPT1}\,(j), i)$ will be placed on TSTACK, possibly modified, and eventually selected as a type 2 pair unless some other type 2 pair is found first. Thus if some type 2 pair exists, at least one type 2 pair will be found by the algorithm. It follows that the type 2 test works correctly, and the algorithm splits a multigraph if and only if a separation pair exists.

The lemma follows by induction on the number of edges in G. Suppose the lemma is true for graphs with fewer than k edges. Let G have k edges. If G cannot be split, the algorithm works correctly on G by the argument above. If G can be split, it will be split. Consider the first split performed by the algorithm, producing split graphs G_1 and G_2. The behavior of the algorithm on G is a composite of its behavior on G_1 and G_2. Since the algorithm splits G_1 and G_2 correctly by the induction hypothesis, it must split G correctly. The lemma follows by induction. Figure 5 gives the contents of ESTACK and TSTACK when the first separation pair (8, 12) in the graph of Fig. 1 is detected.

LEMMA 15. *The triconnected components algorithm processes a graph G with V vertices and E edges in $O(V + E)$ time.*

Proof. The number of edges in a set of split components of G is bounded by $3E - 6$, by Lemma 1. All steps except finding split components thus require $O(V + E)$ time, by the results of the last two sections. Consider execution of algorithm SPLIT. Each edge is placed on ESTACK once and deleted once. The depth-first search itself requires $O(V + E)$ time, including the various tests. The number of triples added to TSTACK is $O(V + E)$. Each triple may only be modified if it is on top of the stack. Thus the time necessary to maintain TSTACK is also $O(V + E)$ and SPLIT requires $O(V + E)$ time.

```
                  8,9  ┐
                  9,10 │
                 10,11 │   First component.
                  9,11 ├   Algorithm adds
                  8,11 │   virtual edge (8, 12).
                 10,12 │
                  9,12 ┘
(12,8,12)         8,12
(12,8,12)         1,12
   EOS            3,13
(13,1,13)         2,13
(13,1, 1)         1,13
 TSTACK          ESTACK
```

FIG. 5. *Contents of* TSTACK *and* ESTACK *when first separation pair* {8, 12} *is detected*

This completes our presentation of an $O(V + E)$ triconnected components algorithm. This algorithm may be used in the construction of an $O(V \log V)$ algorithm for testing isomorphism of planar graphs [3]. The algorithm is not only theoretically optimal (to within a constant factor) but practically useful. The split components algorithm has been implemented in ALGOL W and run on an IBM 360/65 computer. Experiments show that the algorithm can handle graphs with around 1000 edges in less than 10 seconds.

REFERENCES

[1] A. ARIYOSHI, I. SHIRIKAWA, AND O. HIROSHI, *Decomposition of a graph into compactly connected two-terminal subgraphs*, IEEE Trans. Circuit Theory., 18 (1971), pp. 430–435.

[2] J. BRUNO, K. STEIGLITZ, AND L. WEINBERG, *A new planarity test based on 3-connectivity*, Ibid., 17 (1970), pp. 197–206.

[3] J. HOPCROFT AND R. TARJAN, *Isomorphism of planar graphs*, Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 143–150.

[4] J. LEDERGERG, DENDRAL-64: *A system for computer construction, enumeration, and notation of organic molecules as tree structures and cyclic graphs II: Topology of cyclic graphs*, Interim Report on the National Aeronautics and Space Administration, Grants 681–60, NASA CR 68898, STAR N-66-14074, 1965.

[5] R. TARJAN, *Depth-first search and linear graph algorithms*, this Journal, 1 (1972), pp. 146–159.

[6] J. HOPCROFT AND R. TARJAN, *Efficient algorithms for graph manipulation*, Comm. ACM., to appear.

[7] R. TARJAN, *An efficient planarity algorithm*, Rep. STAN-CS-244-71, Computer Science Dept., Stanford Univ., Stanford, Calif., 1971.

[8] J. HOPCROFT AND R. TARJAN, *Efficient planarity testing*, Tech. Rep. 73–165, Dept. of Computer Science, Cornell University, Ithaca, New York, 1973.

[9] R. TARJAN, *Finding dominators in directed graphs*, Tech. Rep. 73–163, Dept. of Computer Science, Cornell Univ., Ithaca, New York, 1973.

[10] D. J. KLEITMAN, *Methods for investigating the connectivity of large graphs*, IEEE Trans. Circuit Theory., 16 (1969), pp. 232–233.

[11] S. A. COOK, *Linear-time simulation of deterministic two-way pushdown automata*, IFIP Congress 71: Foundations of Information Processing, Ljubljana, Yugoslavia, North Holland Pub. Co., Amsterdam, pp. 174–179.

[12] G. BUSACKER AND T. L. SAATY, *Finite Graphs and Networks: An Introduction with Applications*, McGraw-Hill, New York, 1965.

[13] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, Mass., 1969.

[14] S. MACLAINE, *A structural characterization of planar combinatorial graphs*, Duke Math. J., 3 (1937), pp. 460–472.

[15] W. T. TUTTE, *Connectivity in Graphs*, Univ. of Toronto Press, 1966.

[16] J. EDMONDS AND W. CUNNINGHAM, private communication, 1972.

[17] R. TARJAN AND J. HOPCROFT, *Finding the triconnected components of a graph*, Tech. Rep. 72–140, Dept. of Computer Science, Cornell University, Ithaca, New York, 1972.

[18] L. AUSLANDER AND S. V. PARTER, *On imbedding graphs in the plane*, J. Math. Mech., 10 (1961), pp. 517–523.

[19] A. J. GOLDSTEIN, *An efficient and constructive algorithm for testing whether a graph can be embedded in a plane*, Graph and Combinatorics Conference, Office of Naval Research Logistics Proj., Contract NONR 1858-(21), Dept. of Math., Princeton Univ., 1963, 2 unnumbered pp.

SIAM J. COMPUT.
Vol. 2, No. 3, September 1973

DUALITY APPLIED TO THE COMPLEXITY OF MATRIX MULTIPLICATION AND OTHER BILINEAR FORMS*

J. HOPCROFT† AND J. MUSINSKI†

Abstract. The paper considers the complexity of bilinear forms in a noncommutative ring. The dual of a computation is defined and applied to matrix multiplication and other bilinear forms. It is shown that the dual of an optimal computation gives an optimal computation for a dual problem. An $n \times m$ by $m \times p$ matrix product is shown to be the dual of an $n \times p$ by $p \times m$ or an $m \times n$ by $n \times p$ matrix product, implying that each of the matrix products requires the same number of multiplications to compute. Finally, an algorithm for computing a single bilinear form over a noncommutative ring with a minimum number of multiplications is derived by considering a dual problem.

Keywords. algorithms, bilinear forms, computational complexity, duality, matrix multiplication.

1. Introduction. This paper is concerned with determining the minimum number of multiplications necessary to compute certain bilinear forms over a noncommutative ring. We define the dual of a set of expressions and the dual of a computation in such a manner that the dual of the computation of a set of expressions is a computation for the dual of the expressions. Furthermore, a computation and its dual both use the same number of multiplications. This implies that the minimum number of multiplications necessary to compute a set of expressions is the same as that to compute its dual.

The concept of duality[1] is applied to matrix multiplication. The dual of a set of expressions representing the multiplication of two matrices is a set of expressions representing another matrix multiplication problem where the dimensions of the matrices have been permuted. Thus we are able to show that the minimum number of multiplications necessary to compute an $n \times m$ by $m \times p$ matrix product is the same as that required to compute an $n \times p$ by $p \times m$ or an $m \times n$ by $n \times p$ product. Optimal programs for computing $2 \times n$ by $n \times 2$ and 3×3 by 3×2 matrix products follow from previous results. Dual statements of several interesting theorems are presented. Finally, it is shown that Strassen's algorithm for 2×2 by 2×2 matrix multiplication is unique to within a linear transformation.

2. Definition of a computation. Let $\mathscr{C}$ be a commutative ring with a unit element and let ψ be a finite set of indeterminants. Let $\mathscr{R}$ be the noncommutative ring obtained by extending $\mathscr{C}$ by multinomial expressions of the elements of ψ. Throughout this section and the next, F will denote the set of bilinear forms

$$\left\{ \sum_{j=1}^{m} \sum_{k=1}^{n} c_{ijk} a_j x_k \middle| 1 \leqq i \leqq p, a_j, x_k \in \psi, c_{ijk} \in \mathscr{C} \right\}.$$

Similarly, a and x will denote the column vectors $(a_1, a_2, \cdots, a_m)^T$ and $(x_1, x_2, \cdots, x_n)^T$.

* Received by the editors November 14, 1972, and in revised form April 12, 1973.

† Department of Computer Science, Cornell University, Ithaca, New York 14850. This research was supported by the Office of Naval Research under Grant N00014-67-A-0077-0021.

[1] It has been pointed out that the concept of duality can be developed from Fiduccia [2] and is implicit in the tensor formulation of complexity in Strassen [7].

We consider the notion of a computation (see Ostrowski [5]) as a sequence of instructions $f_i = g_i \circ h_i$ where $\circ$ stands for one of the binary operations of multiplication, addition or subtraction. Each f_i is a new variable, and each g_i or h_i is either an element of $\mathscr{C} \cup \psi$ or a previously computed f_j. A multiplication of two elements of $\mathscr{R}$, neither of which is in $\mathscr{C}$, is assumed to take one unit of time. All other operations require no time to perform. The motivation for counting only multiplications between elements in $\mathscr{R} - \mathscr{C}$ is that in applications, the elements of ψ may be large matrices (Strassen [6]), and thus the scheme is not only mathematically tractable but also reflects the actual computation time within a constant factor. It is well known that without division, computations of bilinear forms can be reduced to computing linear combinations of products of pairs of linear forms. This motivates the following definition of a computation. Express[2] the set of expressions F as $(a^TX)^T$, where X is an $m \times p$ matrix with elements of the form $\sum_{i=1}^n c_ix_i$, $c_i \in \mathscr{C}$. A *computation* of F is an expression of the form $M(Pa \cdot Rx)$, where M, P and R are matrices of dimensions $p \times q$, $q \times m$ and $q \times n$, whose elements are from $\mathscr{C}$. The symbol $\cdot$ indicates element by element multiplication, and $M(Pa \cdot Rx) = (a^TX)^T$. Since the straightforward method of evaluating $M(Pa \cdot Rx)$ uses q multiplications between elements in $\mathscr{R} - \mathscr{C}$, the computation is said to have q multiplications.

3. Duality. This section defines the dual of a set of bilinear forms and the dual of a computation. It is then shown that the dual of any computation of F computes the dual of F.

Let b be the column vector $(b_1, b_2, \cdots, b_p)^T$, $b_i \in \psi$. The *left dual* of F is the system of expressions given by $(b^TX^T)^T$. Let $M(Pa \cdot Rx) = (a^TX)^T$ be a computation of F. The *P-dual* of the computation is the computation $P^T(M^Tb \cdot Rx)$.

LEMMA 1. *The P-dual of any computation of a system of expressions F computes the left dual of F.*

Proof. Let $M(Pa \cdot Rx) = (a^TX)^T$ be a computation of F. We must show that $P^T(M^Tb \cdot Rx)$ is a computation of $(b^TX^T)^T$. Let D be a diagonal matrix whose diagonal elements are the elements of the column vector Rx. Then $(M(Pa \cdot Rx))^T = (Pa)^TDM^T$. Since the elements of P commute with the elements of a, $(Pa)^T = a^TP^T$. Now $a^TP^TDM^T = a^TX$ for all a implies $P^TDM^T = X$, which in turn implies $b^TMDP = b^TX^T$ for all b. Thus $(M^Tb)^TDP = b^TX^T$, implying $P^T(M^Tb \cdot Rx) = (b^TX^T)^T$.

In a similar manner, the system of expressions F can be expressed as Ax where A is a $p \times n$ matrix with elements of the form $\sum_{i=1}^m c_ia_i$, $c_i \in \mathscr{C}$. The *right dual* of F is the system of expressions given by A^Tb. If $M(Pa \cdot Rx)$ is a computation of F, then the R-dual of the computation is the computation $R^T(Pa \cdot M^Tb)$. The R-dual of a computation of a system of expressions F computes the right dual of F.

LEMMA 2. *The R-dual of any computation of a system of expressions F computes the right dual of F.*

Proof. The proof is analogous to that of Lemma 1.

THEOREM 3. *There is a computation for the system of expressions computed by $M(Pa \cdot Rx)$ with q multiplications if and only if there is a computation with q multiplica-*

[2] By $(a^TX)^T$ we mean the matrix whose ijth element is the jith element of a^TX. Since the elements are from a noncommutative ring rather than a field, $(a^TX)^T \neq X^Ta$ in general.

tions for each of the systems of expressions computed by $P^T(M^Tb \cdot Rx)$, $R^T(Pa \cdot M^Tb)$, $R^T(M^Tb \cdot Pa)$, $P^T(Rx \cdot M^Tb)$ and *$M(Rx \cdot Pa)$.*

Proof. The result follows from the fact that a computation, its R-dual and its P-dual each have the same number of multiplications.

Let $M(Pa \cdot Rx)$ be a computation of F, and let c be a column vector such that $M(Pa \cdot Rx) = c$. Let T, U, V be $p \times p$, $m \times m$, $n \times n$ matrices, respectively, with elements from $\mathscr{C}$. A *transformation of a vector* c of bilinear forms is the result of replacing each element of a and x by the corresponding elements of Ua, Vx in Tc. A *transformation of the computation* $M(Pa \cdot Rx)$ is the computation $TM(PUa \cdot RVx)$.

LEMMA 4. *The transformation of a computation of c is a computation of the transformation of c.*

COROLLARY 5. *If c' is a transformation of c, then c' can be computed in q multiplications if c can be computed in q multiplications. If T, U and V are nonsingular, and c' can be computed in q multiplications, then c can be computed in q multiplications.*

Note that the concept of duality can be generalized. Up until now, $\mathscr{C}$ and $\mathscr{R}$ have been rings. However, the existence of an additive inverse is not needed.

Example 1. Let $\mathscr{C} = Z$, the integers, let $\psi = \{x_1, x_2, \cdots, x_n\}$ where the x_i are elements of Z, and replace addition and multiplication by minimum and addition, respectively.

Example 2. Let $\mathscr{C} = \{0, 1\}$, let $\psi = \{x_1, x_2, \cdots, x_n\}$ where the x_i are elements of $\mathscr{C}$, and replace addition and multiplication by the Boolean operations OR and AND.

Example 3. Let $\mathscr{C} = Z$, let $\psi = \{x_1, x_2, \cdots, x_n\}$ where the x_i are $k \times k$ matrices with elements from $\mathscr{C}$, and replace addition and multiplication by matrix addition and matrix multiplication.

4. Matrix multiplication.

Let A, B and C be $m \times n$, $n \times p$ and $m \times p$ matrices whose elements are from ψ. We will show that there is a computation of AB with q multiplications if and only if there are computations for

$$A^TC, B^TA^T, BC^T, C^TA, CB^T$$

with q multiplications. In other words, the number of multiplications needed to compute the product of an $m \times n$ matrix with an $n \times p$ matrix is the same as that required to compute an $n \times m$ by $m \times p$, $p \times n$ by $n \times m$, etc. If one uses the ordinary algorithms which require nmp multiplications, then the result is not surprising. However, the result claims that no matter what method is used, the minimum number of multiplications is the same.

Let a, b and c be column vectors whose elements are those of A, B and C, respectively, in row order (e.g., $a = (a_{11}, a_{12}, \cdots, a_{1n}, a_{21}, \cdots, a_{mn})^T$). The ijth element of AB is $\sum_{k=1}^{n} a_{ik}b_{kj}$. Therefore, there exist matrices M, P and R of dimensions $mp \times q$, $q \times mn$ and $q \times np$, whose elements are from $\mathscr{C}$, such that $M(Pa \cdot Rb)$ is a computation for AB.

THEOREM 6. *The following statements are equivalent.*

(a) $M(Pa \cdot Rb)$ *computes* AB *in row order using q multiplications.*
(b) $P^T(M^Tc \cdot Rb)$ *computes* CB^T *in row order using q multiplications.*
(c) $R^T(M^Tc \cdot Pa)$ *computes* $(C^TA)^T$ *in row order using q multiplications.*

(d) $M(Rb \cdot Pa)$ *computes* $(B^T A^T)^T$ *in row order using q multiplications.*
(e) $P^T(Rb \cdot M^T c)$ *computes* $(BC^T)^T$ *in row order using q multiplications.*
(f) $R^T(Pa \cdot M^T c)$ *computes* $A^T C$ *in row order using q multiplications.*

Proof. We will prove only that (a) $\Rightarrow$ (b). Let D_B be the $mn \times mp$ matrix $\begin{bmatrix} B0 & \cdot & 0 \\ 0B & \cdot & 0 \\ 00 & \cdot & B \end{bmatrix}$. Then $M(Pa \cdot Rb)$ computes AB in row order implies that $M(Pa \cdot Rb) = (a^T D_B)^T$ by definition of a computation. This in turn implies that $P^T(M^T c \cdot Rb) = (c^T D_B^T)^T$ by Lemma 1. Thus $P^T(M^T c \cdot Rb)$ computes CB^T in row order.

COROLLARY 7. *The minimum number of multiplications required to multiply $m \times n$ by $n \times p$ matrices without using commutativity is the same as to multiply $n \times m$ by $m \times p$, $n \times p$ by $p \times m$, $p \times m$ by $m \times n$, $p \times n$ by $n \times m$, or $m \times p$ by $p \times n$.*

Theorem 6 leads to new algorithms for multiplying various size matrices together. Some of the new algorithms are optimal, others are at least improvements over the best currently known. For example, in [4] it is shown that $\lceil (3pn + \max(n, p))/2 \rceil$ multiplications suffice for $p \times 2$ by $2 \times n$ matrix multiplication. It follows that $\lceil (3pn + \max(n, p))/2 \rceil$ multiplications suffice for $2 \times p$ by $p \times n$ matrix multiplication. Since $\lceil 7n/2 \rceil$ multiplications are necessary and sufficient for 2×2 by $2 \times n$ matrix multiplication [4], it follows that $\lceil 7n/2 \rceil$ multiplications are necessary and sufficient for $2 \times n$ by $n \times 2$ matrix multiplication. Similarly, since 15 multiplications are necessary and sufficient for 3×2 by 2×3 matrix multiplication, 15 multiplications are necessary and sufficient for 3×3 by 3×2 matrix multiplication.

The number of multiplications necessary to compute the product of two 3×3 matrices is an interesting open problem. If 21 or fewer multiplications are sufficient, then the asymptotic growth rate of Strassen's method [6] could be improved. An examination of 3×2 by 2×3 and 3×3 by 3×2 matrix multiplication algorithms may shed some insight on the development of an algorithm for 3×3 by 3×3 matrix multiplication.

Let A, X, C, and Y be 3×2, 2×3, 3×3 and 3×2 matrices whose elements are from ψ. Then

$$AX = \begin{bmatrix} m_1 + m_2 & -m_2 - m_3 + m_7 - m_8 & -m_1 - m_5 - m_{13} + m_{15} \\ -m_1 - m_4 + m_8 - m_9 & m_3 + m_4 & -m_3 - m_6 + m_{11} - m_{12} \\ -m_2 - m_6 + m_{13} - m_{14} & -m_4 - m_5 + m_{10} - m_{11} & m_5 + m_6 \end{bmatrix},$$

where

$$m_1 = (a_{11} - a_{12})x_{11}$$
$$m_2 = a_{12}(x_{11} + x_{21})$$
$$m_3 = a_{21}x_{12}$$
$$m_4 = a_{22}x_{22}$$
$$m_5 = a_{31}(x_{13} + x_{23})$$

$$\begin{aligned}
m_6 &= (-a_{31} + a_{32})x_{23} \\
m_7 &= (a_{11} + a_{21})(x_{11} + x_{12} + x_{21} + x_{22}) \\
m_8 &= (a_{11} - a_{12} + a_{21})(x_{11} + x_{21} + x_{22}) \\
m_9 &= (a_{11} - a_{12} + a_{21} - a_{22})(x_{21} + x_{22}) \\
m_{10} &= (a_{22} + a_{32})(x_{12} + x_{13} + x_{22} + x_{23}) \\
m_{11} &= (a_{22} - a_{31} + a_{32})(+x_{12} + x_{13} + x_{23}) \\
m_{12} &= (-a_{21} + a_{22} - a_{31} + a_{32})(+x_{12} + x_{13}) \\
m_{13} &= (a_{12} + a_{31})(x_{11} - x_{23}) \\
m_{14} &= (-a_{12} - a_{32})(x_{21} + x_{23}) \\
m_{15} &= (a_{11} + a_{31})(x_{11} + x_{13})
\end{aligned}$$

$$M = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & -1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\
-1 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$P = \begin{bmatrix}
1 & -1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 \\
1 & -1 & 1 & 0 & 0 & 0 \\
1 & -1 & 1 & -1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & -1 & 1 \\
0 & 0 & -1 & 1 & -1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\qquad
R = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}$$

then $M(Pa \cdot Rx)$ computes AX in row order with 15 multiplications. By Theorem 6, $P^T(M^Tc \cdot Ry)$ is an optimal algorithm for CY. Thus

$$BY = \begin{bmatrix} n_1 + n_7 + n_8 + n_9 + n_{15} & -n_1 + n_2 - n_8 - n_9 + n_{13} - n_{14} \\ n_3 + n_7 + n_8 + n_9 - n_{12} & n_4 - n_9 + n_{10} + n_{11} + n_{12} \\ n_5 - n_6 - n_{11} - n_{12} + n_{13} + n_{15} & n_6 + n_{10} + n_{11} + n_{12} - n_{14} \end{bmatrix},$$

where

$$\begin{aligned} n_1 &= (c_{11} - c_{13} - c_{21})y_{11} & n_9 &= -c_{21}(y_{12} + y_{22}) \\ n_2 &= (c_{11} - c_{12} - c_{31})(y_{11} + y_{12}) & n_{10} &= c_{32}(y_{21} + y_{22} + y_{31} + y_{32}) \\ n_3 &= (-c_{12} + c_{22} - c_{23})y_{21} & n_{11} &= (c_{23} - c_{32})(y_{21} + y_{31} + y_{32}) \\ n_4 &= (-c_{21} + c_{22} - c_{32})y_{22} & n_{12} &= -c_{23}(y_{21} + y_{31}) \\ n_5 &= (-c_{13} - c_{32} + c_{33})(y_{31} + y_{32}) & n_{13} &= (-c_{13} + c_{31})(y_{11} - y_{32}) \\ n_6 &= (-c_{23} - c_{31} + c_{33})y_{32} & n_{14} &= -c_{31}(y_{12} + y_{32}) \\ n_7 &= c_{12}(y_{11} + y_{12} + y_{21} + y_{22}) & n_{15} &= c_{13}(y_{11} + y_{31}) \\ n_8 &= (-c_{12} + c_{21})(y_{11} + y_{12} + y_{22}) & & \end{aligned}$$

The algorithm for AX is the union of three optimal algorithms that compute

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}, \begin{bmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{bmatrix}\begin{bmatrix} x_{11} & x_{13} \\ x_{21} & x_{23} \end{bmatrix} \text{ and } \begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}\begin{bmatrix} x_{12} & x_{13} \\ x_{22} & x_{23} \end{bmatrix},$$

respectively, such that each diagonal component of AX is computed with exactly two multiplications. Furthermore, both algorithms which compute a given diagonal component compute it with the same two multiplications. Each of the three algorithms uses seven multiplications, but each pair of algorithms has two multiplications in common. Thus only 15 multiplications are used in computing AX.

The algorithm for CY is the dual of the algorithm for AX. Thus, there is a dual construction for it. This construction is described briefly below and followed by an example. Let W be the 3×2 matrix such that $W = CY$. Construct optimal algorithms that compute

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}\begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} = \begin{bmatrix} w_{11} - c_{13}y_{31} & w_{12} - c_{13}y_{32} \\ w_{21} - c_{23}y_{31} & w_{22} - c_{23}y_{32} \end{bmatrix},$$

$$\begin{bmatrix} c_{11} & c_{13} \\ c_{31} & c_{33} \end{bmatrix}\begin{bmatrix} y_{11} & y_{12} \\ y_{31} & y_{32} \end{bmatrix} = \begin{bmatrix} w_{11} - c_{12}y_{21} & w_{12} - c_{12}y_{22} \\ w_{31} - c_{32}y_{21} & w_{32} - c_{32}y_{22} \end{bmatrix},$$

and

$$\begin{bmatrix} c_{22} & c_{23} \\ c_{32} & c_{33} \end{bmatrix}\begin{bmatrix} y_{21} & y_{22} \\ y_{31} & y_{32} \end{bmatrix} = \begin{bmatrix} w_{21} - c_{21}y_{11} & w_{22} - c_{21}y_{12} \\ w_{31} - c_{31}y_{11} & w_{32} - c_{31}y_{12} \end{bmatrix}$$

such that each c_{ii} appears in exactly two linear combinations which are right-hand sides of multiplications in each of the two algorithms involving c_{ii}. Furthermore, if

α and β are the left-hand sides of the two multiplications in one algorithm, then α and β are the left-hand sides in the other. Each pair of multiplications with the same left-hand sides whose right-hand sides contain c_{ii} are merged by the formula

$$\text{merge}\,((c_{ii} + l_1)\alpha, (c_{ii} + l_2)\alpha) = (c_{ii} + l_1 + l_2)\alpha,$$

where l_1 and l_2 are linear combinations of the components of C. Each of the three original algorithms contains seven multiplications, and between each pair of algorithms, two pairs of multiplications are merged. Thus the composite algorithm uses 15 multiplications in computing CY.

The following example should clarify the above description.

Example.

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} = \begin{bmatrix} n_1 - n_4 + n_6 - n_7 & n_1 + n_2 \\ -n_1 + n_4 + n_5 + n_7 & -n_1 + n_3 + n_5 + n_7 \end{bmatrix},$$

$$\begin{bmatrix} c_{11} & c_{13} \\ c_{31} & c_{33} \end{bmatrix} \begin{bmatrix} y_{11} & y_{12} \\ y_{31} & y_{32} \end{bmatrix} = \begin{bmatrix} n_8 + n_9 & n_8 - n_{11} + n_{13} - n_{14} \\ -n_8 + n_{10} + n_{12} + n_{14} & -n_8 + n_{11} + n_{12} + n_{14} \end{bmatrix},$$

$$\begin{bmatrix} c_{22} & c_{23} \\ c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} y_{21} & y_{22} \\ y_{31} & y_{32} \end{bmatrix} = \begin{bmatrix} n_{15} + n_{16} & n_{16} + n_{18} + n_{20} + n_{21} \\ -n_{15} + n_{17} + n_{19} + n_{21} & n_{17} - n_{18} \end{bmatrix},$$

where

$$\begin{aligned} n_1 &= c_{12}(y_{12} + y_{22}) & n_8 &= c_{13}(y_{11} + y_{31}) \\ n_2 &= (c_{11} - c_{12})y_{12} & n_9 &= (c_{11} - c_{13})y_{11} \\ n_3 &= (c_{21} - c_{22})(y_{21} - y_{22}) & n_{10} &= (c_{31} - c_{33})(-y_{31} + y_{32}) \\ n_4 &= c_{21}(y_{11} - y_{12} + y_{21} - y_{22}) & n_{11} &= c_{31}(-y_{11} + y_{12} - y_{31} + y_{32}) \\ n_5 &= (c_{12} + c_{22})y_{21} & n_{12} &= (c_{13} + c_{33})y_{32} \\ n_6 &= (c_{11} + c_{21})y_{11} & n_{13} &= (c_{11} + c_{31})y_{12} \\ n_7 &= (c_{12} + c_{21})(y_{12} - y_{21} + y_{22}) & n_{14} &= (c_{13} + c_{31})(y_{11} + y_{31} - y_{32}) \end{aligned}$$

$$\begin{aligned} n_{15} &= c_{23}(y_{21} + y_{31}) \\ n_{16} &= (c_{22} - c_{23})y_{21} \\ n_{17} &= (-c_{32} + c_{33})y_{32} \\ n_{18} &= c_{32}(-y_{22} - y_{32}) \\ n_{19} &= (-c_{23} - c_{33})(-y_{31} + y_{32}) \\ n_{20} &= (-c_{22} - c_{32})(y_{21} - y_{22}) \\ n_{21} &= (c_{23} + c_{32})(y_{21} + y_{32}) \end{aligned}$$

Then

$CY =$

$$\begin{bmatrix} m_1 - m_4 + m_6 - m_7 + m_8 & m_1 + m_2 + m_8 - m_{10} - m_{12} \\ -m_1 + m_4 + m_5 + m_7 + m_{13} & -m_1 + m_3 + m_5 + m_7 + m_{14} + m_{15} \\ -m_8 + m_9 + m_{11} + m_{12} - m_{13} + m_{15} & -m_8 + m_{10} + m_{11} + m_{12} - m_{14} \end{bmatrix},$$

where

$$\begin{aligned}
m_1 &= n_1 \\
m_2 &= \mathrm{merge}(n_2, n_{13}) = (c_{11} - c_{12} + c_{31})y_{12} \\
m_3 &= \mathrm{merge}(n_3, n_{20}) = (c_{21} - c_{22} - c_{32})(y_{21} - y_{22}) \\
m_4 &= n_4 \\
m_5 &= \mathrm{merge}(n_5, n_{16}) = (c_{12} + c_{22} - c_{23})y_{21} \\
m_6 &= \mathrm{merge}(n_6, n_9) = (c_{11} - c_{13} + c_{21})y_{11} \\
m_7 &= n_7 \\
m_8 &= n_8 \\
m_9 &= \mathrm{merge}(n_{10}, n_{19}) = (-c_{23} + c_{31} - c_{33})(-y_{31} + y_{32}) \\
m_{10} &= n_{11} \\
m_{11} &= \mathrm{merge}(n_{12}, n_{17}) = (c_{13} - c_{32} + c_{33})y_{32} \\
m_{12} &= n_{14} \\
m_{13} &= n_{15} \\
m_{14} &= n_{18} \\
m_{15} &= n_{21}
\end{aligned}$$

It is hoped that the techniques used above to construct algorithms for 3×2 by 2×3 and 3×3 by 3×2 matrix multiplication can be applied toward developing an optimal algorithm for 3×3 by 3×3 matrix multiplication. To date, no algorithm for the latter using less than 24 multiplications has been found. However, there is no indication that 24 multiplications is the minimum.

Let D be a 3×3 matrix with elements from ψ. Then CD can be computed with 24 multiplications by partitioning the problem into a 3×2 by 2×3 and a 3×1 by 1×3 matrix multiplication problem, or by partitioning the problem into a 3×3 by 3×2 and a 3×3 by 3×1 problem. These two partitions result in dual computations for 3×3 by 3×3 matrix multiplication. A third computation, also with 24 multiplications, can be obtained by using both of the above combining techniques. In this case, find optimal algorithms that compute

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}\begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix}, \begin{bmatrix} c_{11} & c_{13} \\ c_{31} & c_{33} \end{bmatrix}\begin{bmatrix} d_{11} & d_{13} \\ d_{31} & d_{33} \end{bmatrix} \text{ and } \begin{bmatrix} c_{22} & c_{23} \\ c_{32} & c_{33} \end{bmatrix}\begin{bmatrix} d_{22} & d_{23} \\ d_{32} & d_{33} \end{bmatrix}$$

so that in each pair of algorithms there are two multiplications that are either the same or can be merged into a single multiplication as demonstrated above. This eliminates three multiplications, leaving 18. To these are added the six multiplications $c_{13}d_{32}$, $c_{12}(d_{23} - d_{11})$, $(c_{23} + c_{13})d_{31}$, $(c_{21} + c_{31})d_{13}$, $c_{31}d_{12}$, and $c_{32}(d_{21} - d_{33})$. The computation of CD minus the above six multiplications is illustrated below.

$$\begin{aligned}
m_1 &= (c_{11} - c_{12})d_{11} \\
m_2 &= c_{12}(d_{21} + d_{11}) \\
m_3 &= c_{21}d_{12} \\
m_4 &= c_{22}d_{22} \\
m_5 &= (c_{11} + c_{21})(d_{11} + d_{12} + d_{21} + d_{22}) \\
m_6 &= (c_{11} - c_{12} + c_{21} - c_{22})(d_{21} + d_{22}) \\
m_7 &= (c_{11} - c_{12} + c_{21})(d_{11} + d_{21} + d_{22}) \\
m_8 &= (c_{11} - c_{13})d_{11} \\
m_9 &= c_{13}(d_{31} + d_{11}) \\
m_{10} &= c_{31}(d_{13} + d_{33}) \\
m_{11} &= (-c_{31} + c_{33})d_{33} \\
m_{12} &= (c_{11} + c_{31})(d_{11} + d_{13}) \\
m_{13} &= (-c_{13} - c_{33})(d_{31} + d_{33}) \\
m_{14} &= (c_{13} + c_{31})(d_{11} - d_{33}) \\
m_{15} &= c_{32}(d_{23} + d_{33}) \\
m_{16} &= (-c_{32} + c_{33})d_{33} \\
m_{17} &= c_{23}d_{32} \\
m_{18} &= c_{22}d_{22} \\
m_{19} &= (c_{23} + c_{33})(-d_{22} - d_{23} - d_{32} - d_{33}) \\
m_{20} &= (-c_{22} + c_{23} - c_{32} + c_{33})(-d_{22} - d_{23}) \\
m_{21} &= (c_{23} - c_{32} + c_{33})(-d_{22} - d_{23} - d_{33}) \\
y_{11} &= \text{merge}(m_1, m_8) + m_2 + m_9 \\
y_{12} &= -m_2 - m_3 + m_5 - m_7 \\
y_{13} &= -m_{10} + m_{12} - m_{14} - \text{merge}(m_1, m_8) \\
y_{21} &= -m_4 - m_6 + m_7 - m_9 - \text{merge}(m_1, m_8) \\
y_{22} &= m_3 + m_4 + m_{17} \\
y_{23} &= -m_{10} - m_{18} + m_{20} - m_{21} - \text{merge}(m_{11}, m_{16})
\end{aligned}$$

$$y_{31} = -m_9 - m_{13} + m_{14} - \mathrm{merge}(m_{11}, m_{16})$$
$$y_{32} = -m_{15} - m_{17} - m_{19} + m_{21}$$
$$y_{33} = \mathrm{merge}(m_{11}, m_{16}) + m_{10} + m_{15}$$

5. Dual theorems. In addition to helping find optimal (or better) algorithms for matrix multiplication, Theorem 6 or its more general form, Theorem 3, can be applied to previously published theorems to yield new results. Below we list several theorems with one dual theorem for each. Of course, in most cases, more than one dual exists. We have arbitrarily chosen one for demonstration purposes.

Although Lemmas 8, 10, 12, and 16 are established results, and Lemmas 9, 11, 14, and 18 follow from Theorems 3 and 6, we present a proof for Lemma 18 in order to underscore their dual nature. For the sake of simplicity, the lemmas are expressed for $\mathscr{C} = Z_2$, the integers modulo 2. Some are more general.

Before proceeding, we define a term used in many of the following lemmas. A set of vectors $v_1, v_2, \cdots, v_p$ with elements from $\mathscr{R}$ is *nondependent* if and only if it is linearly independent mod $\mathscr{C}$. That is, if and only if $\sum_{i=1}^{p} c_i v_i$ is a vector w with elements from $\mathscr{C}$, each c_i an element of $\mathscr{C}$, implies that each $c_i = 0$. Since an expression can be considered to be a one-dimensional vector, the notion of nondependence applies also to expressions.

Lemmas 8 and 9, due to Winograd [9] and Fiduccia [1], respectively, are duals.

LEMMA 8. (Winograd). *Let A be an $m \times n$ matrix whose elements are from $\mathscr{R}$, and let $x = (x_1, x_2, \cdots, x_n)^T$ where $x_i \in \psi$. If A has p nondependent columns, then and computation of Ax requires at least p multiplications.*

LEMMA 9. (Fiduccia). *Let A be an $m \times n$ matrix whose elements are from $\mathscr{R}$, and let x be an arbitrary vector. If A has p nondependent rows, then any algorithm computing Ax requires at least p multiplications.*

Lemmas 10, 12, and 16 are found in [4]. Lemmas 11, 14, and 18 are their duals.

LEMMA 10. *Let $\mathscr{C}$ be a field, and let $F = \{f_1, \cdots, f_k, \cdots, f_p\}$ be a set of expressions, where $f_1, \cdots, f_k$ are nondependent and each can be expressed as a single product. If F can be computed with q multiplications, then there exists an algorithm for F with q multiplications in which k of the multiplications are $f_1, \cdots, f_k$.*

LEMMA 11. *Let $\mathscr{C}$ be a field and F be the set of expressions*

$$\left\{ \sum_{j=1}^{m} c_{ij} a_j B_{ij} \middle| 1 \leqq i \leqq p; a_j \in \psi, c_{ij} \in \mathscr{C}; B_{ij} = \sum_{k=1}^{n} d_{ijk} x_k, d_{ijk} \in \mathscr{C}, x_k \in \psi \right\},$$

where $B_{1j} = \cdots = B_{pj}, 1 \leqq j \leqq t$. Let B be the $p \times t$ matrix whose ij-th element is $c_{ij}B_{ij}$. If F can be computed with q multiplications and B has t nondependent columns, then F can be computed with q multiplications in which $a_1, \cdots, a_t$ appear in exactly one multiplication each, and that multiplication has the form $(a_j + l_j)B_{1j}$, where $l_j = \sum_{i=t+1}^{m} l_{ij} a_i, l_{ij} \in \mathscr{C}, 1 \leqq j \leqq t$.

LEMMA 12. *Let A and X be 2×2 and $2 \times n$ matrices, respectively, whose elements are from ψ. If an algorithm for computing AX has k multiplications of forms $a_{11}\alpha, (a_{12} + a_{21})\beta$, and $(a_{11} + a_{12} + a_{21})\gamma$, then the algorithm requires at least $3n + k$ multiplications.*

COROLLARY 13. *If T is the group of transformations generated by the set of transformations which*:

(1) *interchange two rows of A, two columns of X, or two columns of A and the corresponding two rows of X,*
(2) *either add (subtract) row i of A to row j of A, column i of X to column j of X, or add (subtract) column i of A to column j of A and simultaneously subtract (add) row j of X to row i of X,*

then by applying transformations from T, we also have similar theorems for

(a) $(a_{11} + a_{21})\alpha, (a_{12} + a_{21} + a_{22})\beta, (a_{11} + a_{12} + a_{22})\gamma$
(b) $(a_{11} + a_{12})\alpha, (a_{12} + a_{21} + a_{22})\beta, (a_{11} + a_{21} + a_{22})\gamma$
(c) $(a_{11} + a_{12} + a_{21} + a_{22})\alpha, (a_{12} + a_{21})\beta, (a_{11} + a_{22})\gamma$
(d) $(a_{21})\alpha, (a_{11} + a_{22})\beta, (a_{11} + a_{21} + a_{22})\gamma$
(e) $(a_{21} + a_{22})\alpha, (a_{11} + a_{12} + a_{22})\beta, (a_{11} + a_{12} + a_{21})\gamma$
(f) $a_{12}\alpha, (a_{11} + a_{22})\beta, (a_{11} + a_{12} + a_{22})\gamma$
(g) $(a_{12} + a_{22})\alpha, (a_{11} + a_{21} + a_{22})\beta, (a_{11} + a_{12} + a_{21})\gamma$
(h) $a_{22}\alpha, (a_{12} + a_{21})\beta, (a_{12} + a_{21} + a_{22})\gamma$.

LEMMA 14. *Let A and X be $2 \times n$ and $n \times 2$ matrices, respectively, whose elements are from ψ. If an algorithm for computing $AX = Y$ has k multiplications that are used only in computing y_{11}, or only in computing y_{12} and y_{21}, or only in computing y_{11}, y_{12} and y_{21}, then the algorithm requires $3n + k$ multiplications.*

COROLLARY 15. *By applying transformations in T, we have similar theorems for*

(a) y_{11} *and* y_{21}; y_{12}, y_{21}, *and* y_{22}; y_{11}, y_{12}, *and* y_{22}
(b) y_{11} *and* y_{12}; y_{12}, y_{21}, *and* y_{22}; y_{11}, y_{21}, *and* y_{22}
(c) y_{11}, y_{12}, y_{21}, *and* y_{22}; y_{12} *and* y_{21}; y_{11} *and* y_{22}
(d) y_{21}; y_{11} *and* y_{22}; y_{11}, y_{21}, *and* y_{22}
(e) y_{21} *and* y_{22}; y_{11}, y_{12}, *and* y_{22}; y_{11}, y_{12}, *and* y_{21}
(f) y_{12}; y_{11} *and* y_{22}; y_{11}, y_{12}, *and* y_{22}
(g) y_{12} *and* y_{22}; y_{11}, y_{21}, *and* y_{22}; y_{11}, y_{12}, *and* y_{21}
(h) y_{22}; y_{12} *and* y_{21}; y_{12}, y_{21}, *and* y_{22}.

LEMMA 16. *Let A and X be 2×2 and $2 \times n$ matrices, respectively, whose elements are from ψ. Any algorithm for computing AX which has k multiplications of types $a_{11}\alpha, a_{12}\beta$, and $(a_{11} + a_{12})\gamma$ has at least $3n + k/2$ multiplications.*

COROLLARY 17. *By transformations, we have similar theorems for $a_{21}\alpha$, $a_{22}\beta$, $(a_{21} + a_{22})\gamma$ and for $(a_{11} + a_{21})\alpha, (a_{12} + a_{22})\beta, (a_{11} + a_{12} + a_{21} + a_{22})\gamma$.*

LEMMA 18. *Let A and X be $2 \times n$ and $n \times 2$ matrices, respectively, whose elements are from ψ. Any algorithm for computing $AX = Y$ which has k multiplications used only in computing y_{11}, y_{12} or both has at least $3n + k/2$ multiplications.*

Proof. We first state some results without proofs.

(1) Let $a_1, \cdots, a_p$ be n-vectors whose elements are of the form $\sum_{i=1}^n c_i x_i$, $c_i \in \mathscr{C}$ and $x_i \in \psi$. Then $a_1, \cdots, a_p$ are nondependent if and only if they are linearly independent.
(2) Let C and D be $m \times n$ matrices whose elements are from $\mathscr{R}$. If C and D have k_1 and k_2 nondependent columns, respectively, then $C + D$ has at most $k_1 + k_2$ nondependent columns.
(3) Let C be a $2 \times n$ matrix whose elements are from $\mathscr{R}$. If C has k nondependent columns, then row 1 or row 2 of C has at least $k/2$ nondependent elements.

Resuming the proof, we let $m_1, \dots, m_k$ be the k multiplications which are assumed to be used only in computing y_{11}, y_{12} or both. Then $y_{11} = M_1 + F_1$ and $y_{12} = M_2 + F_2$, where M_1 and M_2 are sums of the $m_1, \cdots, m_k$ and $F_1 = y_{11} - M_1$ and $F_2 = y_{12} - M_2$. Without loss of generality, we can assume $\begin{bmatrix} M_1 \\ M_2 \end{bmatrix} = Gx$ and $\begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = Hx$, where $x = [x_{11}, x_{12}, \cdots, x_{n1}, x_{n2}]^T$, and G and H are $2 \times 2n$ matrices whose elements are of the form $\sum_{i=1}^{2} \sum_{j=1}^{n} c_{ij}a_{ij}, c_{ij} \in \mathscr{C}$. Let G' and H' be the matrices resulting when we set $a_{21} = \cdots = a_{2n} = 0$. Matrix G' has at most k nondependent columns by Lemma 8. Since $\begin{bmatrix} y_{11} \\ y_{12} \end{bmatrix} = (G' + H')x$, $G' + H'$ must have $2n$ nondependent columns. Hence by (2), H' has at least $2n - k$ nondependent columns and by (3), row i, where i is 1 or 2, has at least $n - k/2$ nondependent elements. Therefore by (1), H has $n - k/2$ elements in row i of the form $\sum_{j=1}^{n} c_j a_{1j} + \sum_{k=1}^{n} d_k a_{2k}, c_j, d_k \in \mathscr{C}$, such that the $\sum_{j=1}^{n} c_j a_{1j}$ parts of each element are linearly independent.

We can remove $n + k/2$ multiplications from Q by

(1) removing $m_1, \cdots, m_k$,

(2) equating an appropriate choice of $n - k/2$ elements in row i of H to zero, and solving for $n - k/2$ a_{1j}'s.

The new algorithm computes y_{21} and y_{22}, which requires $2n$ multiplications. Hence Q must have had at least $3n + k/2$ multiplications.

COROLLARY 19. *By transformations we also have theorems for*

(a) y_{21}; y_{22}; y_{21} *and* y_{22}

(b) y_{11} *and* y_{21}; y_{12} *and* y_{22}; y_{11}, y_{12}, y_{21}, *and* y_{22}.

Theorems 3 and 6 and the preceding lemmas lead to the following lemmas for 2×3 by $3 \times n$ matrix multiplications (and hence for $2 \times n$ by $n \times 3$, 3×2 by $2 \times n$, $3 \times n$ by $n \times 2$, $n \times 2$ by 2×3, and $n \times 3$ by 3×2 matrix multiplications).

LEMMA 20. *Let A and X be 2×3 and $3 \times n$ matrices, respectively, whose elements are from ψ. Any algorithm for computing AX which has k multiplications of forms $a_{11}\alpha, a_{12}\beta, (a_{11} + a_{12})\gamma, a_{13}\delta, (a_{11} + a_{13})\xi, (a_{12} + a_{13})\theta$, and $(a_{11} + a_{12} + a_{13})\phi$ has at least $4n + 2k/3$ multiplications.*

Proof. The proof is similar to Lemma 16.

COROLLARY 21. *Extend the definition of T in Corollary 13 in the obvious way to 2×3 by $3 \times n$ matrix multiplication. Then by transformations in T, we have similar results for*

(a) $a_{21}\alpha, a_{22}\beta, (a_{21} + a_{22})\gamma, a_{23}\delta, (a_{21} + a_{23})\xi, (a_{22} + a_{23})\theta$, *and* $(a_{21} + a_{22} + a_{23})\phi$

(b) $(a_{11} + a_{21})\alpha, (a_{12} + a_{22})\beta, (a_{11} + a_{12} + a_{21} + a_{22})\gamma, (a_{13} + a_{23})\delta, (a_{11} + a_{13} + a_{21} + a_{23})\xi, (a_{12} + a_{13} + a_{22} + a_{23})\theta$, *and* $(a_{11} + a_{12} + a_{13} + a_{21} + a_{22} + a_{23})\phi$.

COROLLARY 22. *If $n = 3$ and Q is an optimal algorithm for computing AX, then $k \leqq 4$.*

COROLLARY 23. *Let $n = 3$ and let Q be an optimal algorithm for computing AX.*

Let S_A be the set of all multiplications in Q that begin with

$$a_{11}, a_{12}, a_{11}+a_{12}, a_{13}, a_{11}+a_{13}, a_{12}+a_{13}, a_{11}+a_{12}+a_{13}, a_{21},$$
$$a_{22}, a_{21}+a_{22}, a_{23}, a_{21}+a_{23}, a_{22}+a_{23}, a_{21}+a_{22}+a_{23}, a_{11}+a_{21}, a_{12}+a_{22},$$
$$a_{11}+a_{12}+a_{21}+a_{22}, a_{13}+a_{23}, a_{11}+a_{13}+a_{21}+a_{23}, a_{12}+a_{13}+a_{22}+a_{23},$$
$$a_{11}+a_{12}+a_{13}+a_{21}+a_{22}+a_{23}.$$

Then

(i) *at most* 12 *multiplications of Q are in S_A.*

(ii) *no two multiplications of Q that are in S_A have the same left-hand side.*

Proof.

(i) The result follows from Corollary 22.

(ii) Suppose some multiplication of Q is in S_A. By applying transformations from T, we can assume without loss of generality that multiplication has the form $a_{11}\alpha$. Set $a_{11} = 0$. This removes at least one multiplication from Q. Algorithm Q now computes

$$\begin{bmatrix} 0 & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}.$$

By Lemma 11, we can assume that setting $a_{21} = 0$ causes three multiplications to disappear. The resulting computation is a 2×2 by 2×3 matrix multiplication which requires 11 multiplications. Thus if setting $a_{11} = 0$ removed more than one multiplication, Q must originally have had 16 multiplications and hence was not optimal. Therefore Q had only one multiplication of form $a_{11}\alpha$.

We will conclude this section by showing that Strassen's algorithm for 2×2 by 2×2 matrix multiplication is unique to within a transformation of T (as defined in Corollary 13). That is, every optimal algorithm for 2×2 by 2×2 matrix multiplication can be obtained from any given optimal algorithm for 2×2 by 2×2 matrix multiplication by applying a transformation of T to the latter. Let A and X be 2×2 matrices whose elements are from ψ. Let $a = [a_{11}, a_{12}, a_{21}, a_{22}]^T$ and $x = [x_{11}, x_{12}, x_{21}, x_{22}]^T$. Let M, P, R be 4×7, 7×4, and 7×4 matrices, respectively, whose elements are from $\mathscr{C}$, such that $M(Pa \cdot Rx)$ computes AX in row order. The computation $M(Pa \cdot Rx)$ uses 7 multiplications, and hence is an optimal algorithm.

LEMMA 24. *For fixed P and R, M is unique.*

Proof. Assume $M(Pa \cdot Rx) = M'(Pa \cdot Rx)$ where M' is a 4×7 matrix whose elements are from $\mathscr{C}$ and $M \neq M'$. Then there exists an equation $m_1 + \cdots + m_k = 0$, $k \geqq 1$, where m_i is an entry of the column vector $Pa \cdot Rx$. Thus m_1 can be replaced by $(m_2 + m_3 + \cdots + m_k)$, implying that AX can be computed with 6 multiplications. In [4] it is shown that 7 multiplications are required. Therefore, M is unique.

THEOREM 25. *Any optimal algorithm Q for 2×2 by 2×2 matrix multiplication is unique to within a transformation of T.*

Proof. Divide the multiplications in Q into two disjoint sets A_A and S_B, where the multiplications in S_A have left-hand sides which can be mapped onto a_{11} by a transformation in T and the multiplications in S_B have left-hand sides which can be mapped onto $a_{11} + a_{22}$ by a transformation in T. In [4] it is shown that an optimal algorithm must have six multiplications from S_A and one from S_B. Since any element of S_B can be mapped to any other element of S_B by a transformation in T, we can assume without loss of generality that Q has a multiplication of the form $(a_{11} + a_{22})\alpha_1$. Lemmas 12 and 16 tell us that the remaining multiplications have forms $(a_{12} + a_{22})\alpha_2$, $(a_{11} + a_{21})\alpha_3$, $a_{22}\alpha_4$, $(a_{11} + a_{12})\alpha_5$, $(a_{21} + a_{22})\alpha_6$, $a_{11}\alpha_7$. Since the transformations of T preserve AX, any transformation that sends $a_{11} + a_{22}$ into itself will send the set of remaining left-hand sides into itself. Thus we can assume without loss of generality that

$$Pa = \begin{bmatrix} a_{11} + a_{22} \\ a_{12} + a_{22} \\ a_{11} + a_{21} \\ a_{22} \\ a_{11} + a_{12} \\ a_{21} + a_{22} \\ a_{11} \end{bmatrix}.$$

By the same reasoning and using duals of Lemmas 12 and 16, we can conclude that the right-hand sides of the multiplications of Q must be a transformation of $\{x_{11} + x_{22}, x_{21} + x_{22}, x_{11} + x_{12}, x_{22}, x_{11} + x_{21}, x_{12} + x_{22}, x_{11}\}$. Since for any two sets of possible right-hand sides there exists a transformation in T that sends one to the other without changing the set of left-hand sides corresponding to the former, we can assume without loss of generality that

$$WRx = \begin{bmatrix} x_{11} + x_{22} \\ x_{21} + x_{22} \\ x_{11} + x_{12} \\ x_{22} \\ x_{11} + x_{21} \\ x_{12} + x_{22} \\ x_{11} \end{bmatrix},$$

where W is a 7×7 permutation matrix. We need only show that R is unique.

Somehow we must form the product $a_{12}x_{21}$. Hence one of the four multiplications $(a_{12} + a_{22})(x_{21} + x_{22})$, $(a_{12} + a_{22})(x_{11} + x_{21})$, $(a_{11} + a_{12})(x_{21} + x_{22})$ and $(a_{11} + a_{12})(x_{11} + x_{21})$ must be present. Assume $(a_{12} + a_{22})(x_{11} + x_{21})$ is in Q. Then $(a_{11} + a_{12})(x_{11} + x_{22})$ must also be in Q since this is the only way to cancel the product $a_{12}x_{11}$ from $(a_{12} + a_{22})(x_{11} + x_{21})$ and to introduce the term $a_{12}x_{22}$. However, we cannot obtain $a_{12}x_{21}$ and $a_{12}x_{22}$ in separate expressions. Thus $(a_{12} + a_{22})(x_{11} + x_{21})$ is not in Q. Similar arguments eliminate $(a_{11} + a_{12})(x_{21} + x_{22})$ and $(a_{11} + a_{12})(x_{11} + x_{21})$, leaving $(a_{12} + a_{22})(x_{21} + x_{22})$.

Considering products involving $a_{12}, a_{21}, x_{12}, x_{21}$, we find that $(a_{12} + a_{22})(x_{21} + x_{22})$, $(a_{11} + a_{21})(x_{11} + x_{12})$, $a_{22}(x_{11} + x_{21})$, $(a_{11} + a_{12})x_{22}$, $(a_{21} + a_{22})x_{11}$, $a_{11}(x_{12} + x_{22})$ are in Q. This leaves $(a_{11} + a_{22})$ to match with $(x_{11} + x_{22})$.

Since the left and right-hand sides can match up only one way, R is unique. Thus by Lemma 24, M is unique and thus the algorithm is unique to within a transformation of T.

6. Single bilinear forms. Let $a_1, \cdots, a_m, x_1, \cdots, x_n, d$ be in ψ and let $c_{11}, c_{12}, \cdots, c_{mn}$ be in $\mathscr{C}$. Let $a = [a_1, \cdots, a_m]^T$, $x = [x_1, \cdots, x_m]^T$ and $d = [d]$. In this section we develop an effective procedure which will yield an optimal algorithm for computing a single expression $\sum_{j=1}^n \sum_{i=1}^m c_{ij}a_ix_j$. Vari [8] accomplishes the above, provided that $\sum_{j=1}^n \sum_{i=1}^m c_{ij}a_ix_i = 0$ if and only if $a_i = x_j = 0$ for all i, j. Vari has subsequently removed this condition. Using Theorem 3, we give a second proof.

THEOREM 26. *There exists an effective procedure which yields an optimal algorithm for computing the expression* $\sum_{j=1}^n \sum_{i=1}^m c_{ij}a_ix_j$.

Proof. Theorem 3 tells us that an optimal algorithm for computing $\sum_{j=1}^n \sum_{i=1}^m c_{ij}a_ix_j$ is the P-dual of an optimal algorithm for computing the set of expressions $S = \{\sum_{j=1}^n c_{ij}\, dx_j | i = 1, \cdots, m\}$. The minimum number of multiplications needed to compute this set of expressions equals the maximum number of nondependent expressions in the set $\{\sum_{j=1}^n c_{ij}x_j | i = 1, \cdots, m\}$. Clearly, then, we can find matrices M, P, R of appropriate dimensions such that $M(Pd \cdot Rx)$ computes S with the minimum number of multiplications. Then $P^T(M^Ta \cdot Rx)$ computes $\sum_{i=1}^m \sum_{j=1}^n c_{ij}a_ix_j$ with the minimum number of multiplications.

REFERENCES

[1] C. M. FIDUCCIA, *Fast matrix multiplication*, Proc. of Third Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio, 1971, pp. 45–49.

[2] ———, *On obtaining upper bounds on the complexity of matrix multiplication*, Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 3–40.

[3] N. GASTINEL, *Sur le calcul des produits de matrices*, Numer. Math., 17 (1971), pp. 222–229.

[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. 30–35.

[5] A. M. OSTROWSKI, *On two problems in abstract algebra connected with Horner's rule*, Studies in Mathematics and Mechanics, Academic Press, New York, 1954, pp. 40–48.

[6] V. STRASSEN, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[7] ———, *Evaluation of rational functions*, Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 1–10.

[8] T. M. VARI, *On the number of multiplications required to compute quadratic functions*, TR 72–120, Computer Science Dept., Cornell Univ., Ithaca, N.Y., 1972.

[9] S. WINOGRAD, *On the number of multiplications required to compute certain functions*, Proc. Nat. Acad. Sci. U.S.A., 58 (1967), pp. 1840–1842.

SIAM J. Comput.
Vol. 2, No. 3, September 1973

HOW TO MAKE ARBITRARY GRAMMARS LOOK LIKE CONTEXT-FREE GRAMMARS*

WALTER J. SAVITCH†

Abstract. Normal form theorems which factor arbitrary phrase-structure grammars into context-free grammars and "a little more" are presented. Every phrase-structure grammar is proven equivalent to one in which each production is either context-free or pure erasing. A pda-like machine is shown to characterize the phrase-structure languages.

Key words. phrase-structure grammar, normal form theorem, recursively enumerable set, context-free grammar, pushdown automata, bracketed grammar.

Introduction. Three normal form theorems for arbitrary phrase-structure grammars are presented: one in terms of grammars, one in terms of machines, and the third algebraic in nature. In each case the goal is to make the grammars look as much like context-free grammars as possible. By phrase-structure grammars we mean the type-0 grammars of Chomsky. So, we will be analyzing parsing algorithms for arbitrary recursively enumerable sets.

In order to establish the notation used, we will give an informal definition of phrase-structure language. Formal definitions of this and those other terms, used without definition, may be found in Hopcroft and Ullman [6]. In any event, it is always the commonly used definitions that we will have in mind. A *phrase structure grammar* (psg) is a four-tuple $G = (V, \Sigma, P, S)$ such that V and Σ are finite sets of symbols called syntactic variables and terminal symbols, respectively. $S \in V$ is called the start symbol. P is a set of productions (rewrite rules) of the form $\alpha \to \beta$, where α and β are in $(V \cup \Sigma)^*$ and α is nonempty. The language generated by G, denoted $L(G)$, is the set of all strings over Σ which can be obtained from S by finitely many applications of the rewrite rules. Two grammars are equivalent if they generate the same language.

1. Grammars.

Definition. A psg $G = (V, \Sigma, P, S)$ is said to be in *normal form* if every production is in one of the following forms.

(i) (context-free) $A \to \alpha$ for $A \in V$ and $\alpha \in (V \cup \Sigma)^*$

(ii) (pure erasing[1]) $AB \to \varepsilon$ for $A, B \in V$.

We wish to show that every psg can be put in normal form. We will actually prove that every psg can be put into a stronger normal form.

Definition. A psg $G = (V, \Sigma, P, S)$ is said to be in *strong normal form* if there is a partition of V into three sets, V_{cf}, $\vec{V}_\varepsilon$ and $\overleftarrow{V}_\varepsilon$, such that every production is in one of the following forms.

* Received by the editors June 29, 1972, and in revised form March 19, 1973.

† Department of Applied Physics and Information Science, University of California at San Diego, La Jolla, California 92037. This research was supported by the National Science Foundation under Grant GJ-27278.

[1] The empty string is denoted by ε.

(i) $A \to \alpha$ for $A \in V_{cf}$ and $\alpha \in (V \cup \Sigma)^*$
(ii) $AB \to \varepsilon$ for $A \in \vec{V}_\varepsilon$ and $B \in \overleftarrow{V}_\varepsilon$. Elements of $\vec{V}_\varepsilon \cup \overleftarrow{V}_\varepsilon$ are called *cancellation variables.*

If $AB \to \varepsilon$, then A and B may be thought of as inverses that cancel each other out. If the grammar is in strong normal form, then the cancellation is one-sided in a very strong sense. If B is a right inverse for A, then B is never a left inverse for any variable.

THEOREM 1. *Every phrase-structure grammar is equivalent to a grammar in strong normal form.*

In order to prove Theorem 1, we will need the following lemma. This is a standard alternative characterization of psg.

LEMMA 1. *Every psg is equivalent to a grammar in which every production is in one of the following forms, where A, B are syntactic variables, and α is any string of allowable symbols.*

(i) $BA \to \alpha A$ (*right context only*)
(ii) $AB \to A\alpha$ (*left context only*)
(iii) $B \to \alpha$ (*context-free*).

Proof of Theorem 1. Let $G' = (V, \Sigma, P, S)$ be an arbitrary psg. We will construct an equivalent psg, G, which is in strong normal form. The construction is similar to one in Peters and Ritchie [7].

We assume G is in the form guaranteed by Lemma 1. Define the new psg $G' = (V', \Sigma, P', S)$ as follows. $V' = V \cup \vec{V}_1 \cup \overleftarrow{V}_1 \cup \vec{V}_2 \cup \overleftarrow{V}_2$, where $\vec{V}_1, \overleftarrow{V}_1, \vec{V}_2$ and $\overleftarrow{V}_2$ are mutually disjoint sets of new symbols. Each of these four sets will be indexed by V. So, in effect, we have five distinct copies of V. Notationally, this is indicated as follows. If $A \in V$, then $\vec{A}_1$ will denote the element of $\vec{V}_1$ which is associated with A. G' has the same start symbol and terminal alphabet as G. P' consists of all productions of the following form.

(1) $\vec{A}_1\overleftarrow{A}_2 \to \varepsilon$ and $\vec{A}_2\overleftarrow{A}_1 \to \varepsilon$ for all $A \in V$.
(2) $A \to \overleftarrow{A}_1 A$ and $A \to A\vec{A}_1$ for all $A \in V$.
(3) $B \to \overleftarrow{A}_2\alpha$, for all A, $B \in V$ and $\alpha \in (V \cup \Sigma)^*$ such that $AB \to A\alpha$ is in P.
(4) $B \to \alpha\vec{A}_2$, for all A, $B \in V$ and $\alpha \in (V \cup \Sigma)^*$ such that $BA \to \alpha A$ is in P.
(5) $B \to \alpha$ if $B \in V$ and $B \to \alpha$ is in P.

Clearly, G' is in strong normal form. It remains to show that G and G' are equivalent. A straightforward induction on the number of steps in a derivation shows that $L(G)$ is included in $L(G')$. For example, a derivation step $AB \Rightarrow A\alpha$ in G is accomplished in G' by the three steps $AB \Rightarrow A\vec{A}_1B \Rightarrow A\vec{A}_1\overleftarrow{A}_2\alpha \Rightarrow A\alpha$. Thus it remains to show that $L(G')$ is included in $L(G)$. To show this we will need one more lemma. The notation will be as follows.

DEFINITION. Let α and γ be arbitrary strings of terminal and nonterminal symbols from G'. A derivation $\alpha = \beta_1 \Rightarrow \beta_2 \Rightarrow \cdots \Rightarrow \beta_n = \gamma$ in G' is said to be a *quick-cancel* derivation provided that any occurrence of a cancellation variable that appears in more than two β_1's also occurs in $\beta_n = \gamma$. In other words, cancellation variables never wait to cancel out.

LEMMA 2. *Let α and γ be strings of terminal and nonterminal symbols of G' such that α contains no occurrences of cancellation variables. If $\alpha \overset{*}{\Rightarrow} \gamma$ in k steps in G', then $\alpha \overset{*}{\Rightarrow} \gamma$ in G' by a quick-cancel derivation of k steps.*

We defer the proof of Lemma 2 for a moment and use Lemma 2 to show that any G' derivation can be replaced by a G derivation. Specifically, to show $L(G')$ is included in $L(G)$, we prove the following result by induction on the number of steps in the derivation.

Result. For any string γ of terminal and nonterminal symbols of G, if $S \overset{*}{\Rightarrow} \gamma$, in G', then $S \overset{*}{\Rightarrow} \gamma$ in G.

The base step of the induction is trivial. Suppose $S \overset{*}{\Rightarrow} \gamma$ in G' and assume inductively that the claim is true for all shorter derivations. By Lemma 2, we may assume that $S \overset{*}{\Rightarrow} \gamma$ is a quick-cancel derivation. If the last step in the G' derivation $S \overset{*}{\Rightarrow} \gamma$ is not an erasing rule, then a simple application of the induction hypothesis shows that $S \overset{*}{\Rightarrow} \gamma$ in G. So we may assume that the G' derivation decomposes as follows:

$$S \overset{*}{\Rightarrow} \beta AB\delta \Rightarrow \beta\alpha\vec{B}_2B\delta \Rightarrow \beta\alpha\vec{B}_2\overleftarrow{B}_1B\delta \Rightarrow \beta\alpha B\delta = \gamma,$$

or some minor variation on this. A and B are single sentential variables. Since the derivation is quick-cancel, α, β and δ contain only symbols of the original grammar G. By the induction hypothesis, $S \overset{*}{\Rightarrow} \beta AB\delta$ in G. Also, $AB \to \alpha B$ is a production of G. So, $S \overset{*}{\Rightarrow} \beta AB\delta \overset{*}{\Rightarrow} \beta\alpha B\delta = \gamma$ in G, and the induction is complete. All that remains is to prove Lemma 2.

Proof of Lemma 2. The proof is by induction on k, the number of steps in the derivation $\alpha \overset{*}{\Rightarrow} \gamma$. The basis of the induction is immediate, since any one-step derivation is already a quick-cancel derivation. Suppose $\alpha \overset{*}{\Rightarrow} \gamma$ in G' in k steps and assume, inductively, that Lemma 2 is true for all shorter derivations. The derivation $\alpha \overset{*}{\Rightarrow} \gamma$ decomposes to $\alpha = \alpha_1A\alpha_2 \Rightarrow \alpha_1\beta\alpha_2 \overset{*}{\Rightarrow} \gamma$, where $A \to \beta$ is the first rule used. If β contains no cancellation variables, then a simple application of the induction hypothesis to $\alpha_1\beta\alpha_2 \overset{*}{\Rightarrow} \gamma$ gives the desired result. So, assume $\beta = \delta\vec{C}_j$, $j = 1$ or 2. The case $\alpha = \overleftarrow{C}_j\delta$ is similar. If this occurrence of $\vec{C}_j$ appears in γ, then $\alpha \Rightarrow \alpha_1\delta\vec{C}_j\alpha_2$ $\overset{*}{\Rightarrow} \xi_1\vec{C}_j\xi_2 = \gamma$, where $\alpha_1\delta \overset{*}{\Rightarrow} \xi_1$ and $\alpha_2 \overset{*}{\Rightarrow} \xi_2$. In this case, the result follows by applying the induction hypothesis to $\alpha_1\delta \overset{*}{\Rightarrow} \xi_1$ and $\alpha_2 \overset{*}{\Rightarrow} \xi_2$. So we are left with the case $\alpha = \alpha_1A\alpha_2 \Rightarrow \alpha_1\delta\vec{C}_j\alpha_2 \overset{*}{\Rightarrow} \gamma$, and this C_j does not occur in γ.

In this case,

$$\alpha = \alpha_1A\alpha_2 \Rightarrow \alpha_1\delta\vec{C}_j\alpha_2 \overset{*}{\Rightarrow} \alpha'_1\vec{C}_j\overleftarrow{C}_i\alpha'_2 \Rightarrow \alpha'_1\alpha'_2 \overset{*}{\Rightarrow} \gamma,$$

where $\alpha_1\delta \overset{*}{\Rightarrow} \alpha'_1$ and $\alpha_2 \overset{*}{\Rightarrow} \overleftarrow{C}_i\alpha'_2$. Since the cancellation is one-sided, we may postpone the application of the rule $\vec{C}_j\overleftarrow{C}_i \to \varepsilon$ to near the end of the derivation. The only rules it must precede are certain other erasing rules. Specifically, we may rearrange the derivation $\alpha \overset{*}{\Rightarrow} \gamma$ to get it in the form

$$\alpha = \nu\eta \overset{*}{\Rightarrow} \nu'A_nA_{n-1}\cdots A_1B_1B_2\cdots B_n\eta' \overset{*}{\Rightarrow} \nu'\eta' = \gamma,$$

where

$$\nu \overset{*}{\Rightarrow} \nu'A_nA_{n-1}\cdots A_1, \quad \eta \overset{*}{\Rightarrow} B_1B_2\cdots B_n\eta' \quad \text{and} \quad A_iB_i \to \varepsilon,$$

for $i = 1, 2 \cdots, n$. (Take $\nu = \alpha_1A$, $\eta = \alpha_2$, $A_1 = \vec{C}_j$ and $B_1 = \overleftarrow{C}_i$.) Now, by the induction hypothesis, $\nu \overset{*}{\Rightarrow} \nu'A_nA_{n-1}\cdots A_1$ and $\eta \overset{*}{\Rightarrow} B_1B_2\cdots B_n\eta'$ by quick-cancel derivations. It is easy to combine these two quick-cancel derivations together to get a quick-cancel derivation of $\alpha = \nu\eta \overset{*}{\Rightarrow} \nu'\eta' = \gamma$ once the following claim is established.

Claim. The quick-cancel derivation $\nu \overset{*}{\Rightarrow} \nu' A_n A_{n-1} \cdots A_1$ may be rearranged to get a quick-cancel derivation of the form

$$\nu \overset{*}{\Rightarrow} \nu_1 A_1 \overset{*}{\Rightarrow} \nu_2 A_2 A_1 \overset{*}{\Rightarrow} \cdots \overset{*}{\Rightarrow} \nu_{n-1} A_{n-1} A_{n-2} \cdots A_1 \overset{*}{\Rightarrow} \nu_n A_n A_{n-1} \cdots A_1$$
$$= \nu' A_n A_{n-1} \cdots A_1 .$$

Also, $\eta \overset{*}{\Rightarrow} B_1 B_2 \cdots B_n \eta'$ may be similarly rearranged to produce the B's in the order $B_1, B_2 \cdots, B_n$.

Consider $\nu \overset{*}{\Rightarrow} \nu' A_n A_{n-1} \cdots A_1$. The other derivation is treated similarly. The claim follows from the fact that all the A_i's are one-sided inverses canceling in the same direction. More precisely, suppose that A_k, for some $k > 1$, is produced before A_1. Then $\nu \overset{*}{\Rightarrow} \mu_1 C \mu_2 \Rightarrow \mu_1 \mu_3 A_k \mu_2$, and $\mu_2 \overset{*}{\Rightarrow} A_{k-1} A_{k-2} \cdots A_1$. In this case, we can get the same final string by

$$\nu \overset{*}{\Rightarrow} \mu_1 C \mu_2 \overset{*}{\Rightarrow} \mu_1 C A_{k-1} \cdots A_1 \Rightarrow \mu_1 \mu_3 A_k A_{k-1} \cdots A_1 \overset{*}{\Rightarrow} \nu' A_n A_{n-1} \cdots A_1$$

and this is still a quick-cancel derivation. So, we can insure that A_1 is produced first. Similarly, we can insure that A_2 is produced second, and so forth. This establishes the claim.

The final quick-cancel derivation of $\alpha \overset{*}{\Rightarrow} \gamma$ looks like

$$\alpha = \nu\eta \overset{*}{\Rightarrow} \nu_1 A_1 B_1 \eta_1 \Rightarrow \nu_1 \eta_1 \overset{*}{\Rightarrow} \nu_2 A_2 B_2 \eta_2 \Rightarrow \nu_2 \eta_2 \overset{*}{\Rightarrow} \cdots \overset{*}{\Rightarrow} \nu_n A_n B_n \eta_n \Rightarrow \nu_n \eta_n = \gamma .$$

This completes the proof of Lemma 2 and of Theorem 1.

In the process of proving Theorem 1, we actually established the following slightly stronger normal form result.

COROLLARY 1. *Every psg is equivalent to a grammar in strong normal form in which each context-free production is of one of the following forms:*

$$A \to \alpha, \quad A \to \overleftarrow{B}\alpha \quad \text{or} \quad A \to \alpha\overrightarrow{B},$$

where α contains no cancellation variables, $\overleftarrow{B} \in \overleftarrow{V}_\varepsilon$ and $\overrightarrow{B} \in \overrightarrow{V}_\varepsilon$. ($\overleftarrow{V}_\varepsilon$, $\overrightarrow{V}_\varepsilon$ are the same as in the definition of strong normal form.)

Grammars in strong normal form are very much like the bracket grammars of Harrison and Schkolnick [5]. Our cancellation variables correspond to the brackets of Harrison and Schkolnick. Santos [8] has shown that every recursively enumerable language can be generated by a general two-type bracket grammar. Corollary 1 says that every recursively enumerable language can be generated by what could naturally be called one-type bracket grammars. Thus the corollary may be viewed as a strengthening of Santos' result. A simple alternate proof of Santos' result can be constructed based on this corollary.

2. Machines. A grammar in normal form is structurally very similar to a context-free language. As such it can be parsed in much the same way a context-free language can be parsed. In particular, there is a device, analogous to a pda, which does the parsing in a natural way. The device is called a cancellation pushdown automata (cpda) and can be described, informally, as follows. It consists of a nondeterministic pda with a second pushdown store. The second pushdown store is called the auxiliary pushdown store. The machine may write in the auxiliary pushdown store but may not read in it. The device operates just like a pda, except that at each step it is allowed to write a symbol on top of the auxiliary pushdown store. Thus, an alternate way to describe a cpda is to say that it is a nondeterministic pda with an auxiliary write-only output stack. The finite control can neither read nor erase in the auxiliary stack. However, a finite set of pairs of auxiliary symbols are specified as canceling. Whenever such a pair occurs in the auxiliary stack, the two symbols spontaneously disappear. The device accepts just like a pda accepting by empty store. Both pushdown stores must be emptied. Formally, the definition is as follows.

DEFINITION. A *cancellation pushdown automaton* (cpda) is a six-tuple $M = (K, \Sigma, \Gamma, \delta, E, q_0, Z_0)$ such that: K is a finite set (of states), Σ and Γ are finite sets (of input and stack symbols, respectively), $q_0 \in K$ (q_0 is the start state), $Z_0 \in \Gamma$ (Z_0 is the initial stack symbol), δ is a (next move) function from $K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ into finite subsets of $K \times \Gamma^* \times \Gamma^*$, and, finally, E (the cancellation relation) is a subset of $\Gamma \times \Gamma$.

Intuitively, the meaning of δ is as follows. If $(p, \alpha, \beta) \in \delta(q, a, X)$ and M is in state q, with X on top of the ordinary pushdown stack, and if M is scanning a on its input tape, then one possible move is to go to state p, replace X by α, write β on top of the auxiliary stack and, finally, advance the input head past a.

Intuitively, $(A, B) \in E$ means the following. Any time A and B are adjacent in the auxiliary stack with A on top of B, A and B are both spontaneously erased. Note that E need not be symmetric. In actual computations, all cancellations will occur at the top of the auxiliary stack.

DEFINITION. An *instantaneous description* (ID) of the cpda M is a triple (w_1qw_2, β, α) where $q \in K$, $w_1, w_2 \in \Sigma^*$ and $\alpha, \beta \in \Gamma^*$. The interpretation is that M is in state q with input w_1w_2; the input head is scanning the beginning of w_2; β and α are respectively the contents of the auxiliary and ordinary pushdown stores. The left-most symbols are considered the "top" symbols.

DEFINITION. The yield relation $\vdash$ between ID's is defined as follows.

(i) $(w_1qw_2, AB\beta, \alpha) \vdash (w_1qw_2, \beta, \alpha)$ provided $(A, B) \in E$

(ii) $(w_1qaw_2, \beta, X\alpha) \vdash (w_1apw_2, \xi\beta, \gamma\alpha)$ provided $(p, \gamma, \xi) \in \delta(q, a, X)$ and β is not of the form $AB\beta'$ for $(A, B) \in E$. $\vdash^*$ denotes the *transitive closure of* $\vdash$.

DEFINITION. $N(M) = \{w \in \Sigma^* | (q_0w, \varepsilon, Z_0) \vdash^* (wp, \varepsilon, \varepsilon) \text{ for some } p \in K\}$. If $L = N(M)$ we say that M *accepts* L (by empty store).

THEOREM 2. *A language L is a phrase-structure language if and only if it is accepted by some cpda.*

Proof. If L is accepted by some cpda, it clearly is a recursively enumerable set and hence is a phrase-structure language. Conversely, suppose L is a phrase-structure language. Then L is generated by a grammar $G = (V, \Sigma, P, S)$ in strong normal form. Let V_{cf}, $\vec{V}_\varepsilon$ and $\overleftarrow{V}_\varepsilon$ be as in the definition of strong normal form. Define a cpda $M = (K, \Sigma, \Gamma, \delta, E, q_0, Z_0)$ as follows. There are two states, i.e., $K = \{q_0, q_1\}$. The stack alphabet $\Gamma = V \cup \Sigma \cup \{\vec{X}, \overleftarrow{X}, Z_0\}$, where $\vec{X}$, $\overleftarrow{X}$ and Z_0 are new symbols. $E = \{(A, B) | BA \to \varepsilon\} \cup \{(\vec{X}, \overleftarrow{X})\}$. δ is defined as follows: $\delta(q_0, \varepsilon, Z_0) = \{(q_1, SZ_0, \overleftarrow{X})\}$. If $A \in V_{cf}$, then $\delta(q_1, \varepsilon, A) = \{(q_1, \gamma, \varepsilon) | A \to \gamma \text{ is in } P\}$. If $A \in \vec{V}_\varepsilon \cup \overleftarrow{V}_\varepsilon$, then $\delta(q_1, \varepsilon, A) = \{(q_1, \varepsilon, A)\}$. If $a \in \Sigma$, then $\delta(q_1, a, a) = \{(q_1, \varepsilon, \overleftarrow{X}\vec{X})\}$. Finally, $\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon, \vec{X})\}$. Intuitively, M constructs a derivation from S in the ordinary stack, omitting all erasing rules. The erasing rules are performed in the auxiliary stack. The pair of symbols $\overleftarrow{X}\vec{X}$ is used to prevent unintentional canceling in the auxiliary stack. The generated string is compared with the input to see if they match. Thus $L = N(M)$. The formal details are similar to the proof that every context-free language is accepted by empty store on some pda. This concludes the proof of Theorem 2.

The analogy between pda's and cpda's is close enough so that many definitions and proofs can be carried over with no real change. The analogy is not complete, however. For example, it would be a mistake to carry over the notion of acceptance by final state from pda's to cpda's without significant changes in the definition. If state acceptance is defined in the usual way, that is, the cpda accepts if it is in one of a designated set of accepting states at the end of the input, then the cpda can never get any use out of the auxiliary stack, and so it accepts exactly the context-free languages. Clearly, this is not a good notion of acceptance. If we instead define acceptance to mean that, after scanning the input, the cpda is in one of a designated set of accepting states and the auxiliary stack is empty, then we get results analogous to the classical pda results. This kind of mixed mode acceptance may seem a bit strange. However, if one is to have a notion of acceptance by final state, it appears that it must be this mixed mode of acceptance.

DEFINITION. The cpda $M = (K, \Sigma, \Gamma, \delta, E, q_0, Z_0)$ is said to *accept* the language L by *final states* F provided $L = \{w \in \Sigma^* | (q_0 w, \varepsilon, Z_0) \vdash^* (wp, \varepsilon, \alpha)$ for some $p \in F$ and some $\alpha \in \Gamma^*\}$.

THEOREM 3. *For any language L, the following are equivalent.*

(i) *There is a cpda M and a set of states F such that M accepts L by final states F.*

(ii) *There is a cpda which accepts L by empty store.*

Theorem 3 is proven in the same way as the corresponding result for pda's, so the proof will be omitted here.

The notion of determinism carries over from pda's to cpda's with no real change.

DEFINITION. A cpda is called *deterministic* provided that

(i) for each state q and pushdown symbol Z, if $\delta(q, \varepsilon, Z)$ is nonempty, then $\delta(q, a, Z)$ is empty for each input symbol a;

(ii) for each state q, pushdown symbol Z and each input a (including $a = \varepsilon$), $\delta(q, a, Z)$ contains at most one element. As before, δ denotes the next move function.

For deterministic cpda's, acceptance by final states and acceptance by empty store are not equivalent. Every language accepted by empty store on a deterministic cpda is also accepted by final states on some deterministic cpda. The converse is not true, however. With either definition of acceptance, there are languages accepted by cpda's which are not accepted by any deterministic cpda. As the next theorem indicates, any recursively enumerable set which is not recursive is an example of such a language.

THEOREM 4. *If L is accepted by some deterministic cpda* (*either by empty store or by final states*), *then L is recursive.*

In order to prove Theorem 4, we will need the following lemma.

LEMMA 3. *For every deterministic cpda M, there is an integer k such that for every computation of M, the following is true. If the ordinary stack ever increases in length by k or more symbols, without M advancing its input head, then: no additional* *input will be read; the ordinary stack will grow arbitrarily large; the finite control of M will continually loop through a finite sequence of states; and each time M cycles through these states, it will place the same string of symbols on top of the auxiliary stack.*

Proof. Assume M adds at most one symbol per move to the ordinary stack. M can always be modified to make this true. Set k equal to the number of states of M multiplied by the number of different stack symbols of M. If the ordinary stack ever grows in length by k or more symbols without advancing its input head, then M must have moved from some ID $(uqv, \beta, A\alpha)$ to some ID of the form $(uqv, \xi, A\gamma A\alpha)$, where A is a single stack symbol and the ordinary stack symbols $A\alpha$ are never altered during the intervening computation. Since both the state and the top symbol of the ordinary stack are repeated, M is in a type of infinite loop. No more input will be read, and the finite control will cycle through a fixed pattern of states. Every time M goes through such a cycle, the net change in the ordinary stack will be to add the string $A\gamma$ to the top of the stack. Since the symbols placed on the auxiliary stack depend only on the state and the top symbol of the ordinary stack, M will place the same string of symbols on top of the auxiliary stack during each cycle. The above proof is similar to that of related results on deterministic pda's. A more detailed analysis of these techniques can be found in Chapter 12 of [6].

Proof of Theorem 4. The proof, though fairly lengthy, is based on standard techniques and so will be rather informal. First, notice the following. If the auxiliary stack is removed from a deterministic cpda M, the result is a well-defined deterministic pda. Call the pda so obtained from M *the pda associated with M*. If M accepts a string w by empty store (respectively, final states F), then the pda associated with M also accepts w by empty store (respectively, final states F).

Given a determinsitic cpda, M, and a string w, we can easily decide whether or not M accepts w by empty store. To do this, we proceed as follows. First decide if the pda associated with M accepts w by empty store. It is well known that this is decidable. If the associated pda does not accept w, then M does not accept w. If the associated pda does accept w by empty store, then the unique computation of M on w is a halting computation. So, in order to determine if M accepts w, we need only simulate M on w.

Given a deterministic cpda M, a string w and a set F of final states, it is more difficult to decide if M accepts w by final states F. The additional complication is due to the fact that, after accepting a string w by final state, a pda can continue to compute. Accepting computations need not be halting computations. The problem is still decidable, however. The procedure is as follows. First decide if the associated pda accepts w by final states F. If not, then M does not accept w. If the associated pda accepts w, then we simulate M on w. Since the associated pda accepts w, we know that M reads the entire input string w. It will suffice to show that, in this case, we can eventually determine whether or not M accepts w.

Consider M after it has scanned the entire input string. If M eventually halts, then we can certainly determine whether M accepts the input. So, assume M does not halt. The ordinary stack will either remain bounded in length or else grow arbitrarily large. If the ordinary stack is bounded, then we can detect this fact. For M will eventually start to loop through a finite sequence of state-ordinary stack configuration. By Lemma 3, we can also tell if the ordinary stack grows arbitrarily large. In either case, the finite control is cycling through a finite sequence of states. Furthermore, a fixed pattern of stack symbols is being placed on the auxiliary stack during each such cycle. It remains to decide if the auxiliary stack is empty at one of the configurations in which M is a final state.

At this point, the portion of the ordinary stack that M can access is cycling through a finite number of configurations. So, the ordinary stack is providing M with only a finite amount of information. So, M is very much like a pda with the auxiliary stack corresponding to the pda stack. Although the finite control does not read or erase in the auxiliary stack, all changes in the auxiliary stack do occur at the top of the stack and could be simulated by a deterministic pda. So, just as in Lemma 3, we can calculate an integer k' such that if the auxiliary stack ever grows by more than k' symbols, then the stack will grow arbitrarily large, and it will never be empty.

To determine if the auxiliary stack is empty when M enters a final state, proceed as follows. If the auxiliary stack increases its length by more than k' symbols, then M will never empty the auxiliary stack. So, in this case, we know that M does not accept the input w. If the auxiliary stack does not increase its length by more than k' symbols, then M is cycling through a finite pattern of state-auxiliary stack configurations. To determine if M accepts w, we need only check this finite number of state-auxiliary stack configurations. So, no matter how the computation proceeds, we can always determine whether or not M accepts w.

3. Algebraic form. Implicit in the proof of Theorem 2 is the following algebraic characterization of phrase-structure languages.

THEOREM 5. *Every phrase-structure language is expressible in the form $h(h^{-1}(D) \cap L)$, where L is a context-free language, D a Dyck set and h a homomorphism.*

This result is similar, though not identical, to various other results which express recursively enumerable sets in terms of context-free languages and homomorphisms ([1], [2], [4]). A proof of Theorem 5 based on Theorem 3.1 of [4], for example, is not too difficult. We will instead give a proof based on Theorems 1 and 2.

Proof. Given a psg G, put it in strong normal form. Construct a cpda, M, as in the proof of Theorem 2. Thus $N(M) = L(G)$. Let D be the set of all strings, over the stack alphabet, which would cancel if placed in the auxiliary stack. D is a Dyck set because G is in strong normal form. Let L be all strings of the form $w_0A_1w_1A_2w_2 \cdots A_nw_n$ for which the following is true: $w_0w_1w_2 \cdots w_n$ is a string over the input alphabet of M. $A_1A_2 \cdots A_n$ are stack symbols of M. If M were given the input $w_0w_1w_2 \cdots w_n$, it would behave as follows. M would place no symbols in the auxiliary stack until it had read all of w_0. Then M would place A_1 in the auxiliary stack. M would place nothing more in the auxiliary stack until it had read w_1. Then M would place A_2 in the auxiliary stack, and so forth. So $A_1, A_2, \cdots A_n$ are exactly the symbols M puts into the auxiliary stack. Furthermore, after processing this input, the ordinary stack of M is empty. The auxiliary stack may or may not be empty, depending on whether or not $A_1A_2 \cdots A_n$ cancels or not. Let h be the homomorphism which is the identity map on input symbols and which sends all stack symbols onto the empty word. The string $w \in N(M)$ if and only if there is some string $w_0A_1w_1A_2w_2 \cdots A_nw_n \in L$ such that $w = w_0w_1w_2 \cdots w_n = h(w_0A_1w_1A_2w_2 \cdots A_nw_n)$ and $A_1A_2 \cdots A_n \in D$. Thus $L(G) = N(M) = h(h^{-1}(D) \cap L)$. All that remains is to show that L is a context-free language. But L is accepted by empty store by a pda, and so it is a context-free language. A pda to accept L is obtained from M by removing the auxiliary stack from M and modifying the finite control of M as follows. Instead of placing a symbol, A_i, in the auxiliary stack, it checks to see if A_i is the next input symbol. This completes the proof.

REFERENCES

[1] B. S. Baker and R. V. Book, *Reversal-bounded multi-pushdown machines*, J. Comput. System Sci., to appear.

[2] S. Greibach and S. Ginsburg, *Multitape AFA*, J. Assoc. Comput. Mach., 19 (1972), pp. 193–221.

[3] S. Ginsburg, *The Mathematical Theory of Context-free Languages*, McGraw-Hill, New York, 1966.

[4] S. Ginsburg, S. A. Greibach and M. A. Harrison, *One-way stack automata*, J. Assoc. Comput. Mach., 14 (1967), pp. 389–418.

[5] M. Harrison and M. Schkolnick, *A grammatical characterization of one-way nondeterministic stack languages*, Ibid., 18 (1971), pp. 148–172.

[6] J. E. Hopcroft and J. D. Ullman, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, Mass., 1969.

[7] P. S. Peters and R. W. Ritchie, *Context-sensitive immediate constituent analysis–Context-free languages revisited*, conference record of ACM Symposium on Theory of Computing, Marina del Rey, Calif., May, 1969, pp. 1–8.

[8] E. S. Santos, *A note on bracketed grammars*, J. Assoc. Comput. Mach., 19 (1972), pp. 222–224.

SIAM J. COMPUT.
Vol. 2, No. 3, September 1973

APPROXIMATE MODELS FOR PROCESSOR UTILIZATION IN MULTIPROGRAMMED COMPUTER SYSTEMS*

D. P. GAVER† AND G. S. SHEDLER‡

Abstract. This paper presents results of an approximation study of cyclic queueing phenomena that occur in multiprogrammed computer systems. Based on Wald's identity and using ideas of diffusion, the objective is to develop convenient and nearly explicit formulas relating processor utilization in such systems to simple program parameters and the level of multiprogramming. Some numerical results to indicate the quality of the proposed approximation are given.

Key words. multiprogramming, queueing, diffusion approximations.

1. Introduction. In a previous paper [2], we have initiated an approximation study of cyclic queueing phenomena that occur in multiprogrammed computer systems. Particular attention was focused upon processor utilization estimation, as it depends upon the statistical properties of programs. The basis for the approximation was the observation that under "heavy traffic" conditions, it is plausible to approximate the flow of programs in a multiprogrammed computer system by means of a diffusion or Wiener process (cf. [1, pp. 332–345], [6, pp. 101–109]) with appropriate infinitesimal parameters and boundary conditions. The results were seen to be usefully accurate, as judged numerically, and to be of an extremely simple analytical form. They can thus be put to use for at least preliminary design purposes, with follow-up refined analysis or simulation furnishing further corrections if needed.

One deficiency of the results of [2] is that they tend to misestimate CPU utilization (i.e., the long-run fraction of time that the CPU is busy) when CPU service or processing times come from distributions of greater positive skewness than the exponential. In the present paper, we wish to alter our approximation so as to render it more accurate in the case of such hyperexponential-appearing CPU service times. This change is important, since currently available data indicates that greater-than-exponential skewness is common.

2. The model. We suppose, as we did in [2], that J programs are in the central processing unit (CPU)–data transfer unit (DTU) cycle. Each program is (i) in the process of awaiting, or receiving, service at the CPU, at the termination of which (ii) it repairs to the DTU, again queueing as if at a single server. Having received the requisite information at the DTU stage, it then returns to the CPU stage. This process continues indefinitely. When programs are completed and thus removed from the system, new programs are immediately reintroduced. A diagram indicating the situation appears in Fig. 1.

* Received by the editors October 25, 1972.

† Naval Postgraduate School, Monterey, California 93940. This author is also a consultant to IBM research.

‡ IBM Research Laboratory, San Jose, California 95193.

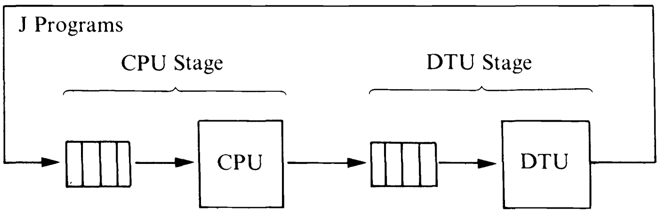


FIG. 1

The assumptions made concerning program behavior are the following.

(a) The sequence of CPU service or processing times is one of independent identically distributed random variables (i.i.d.r.v.) $\{C_i, i = 1, 2 \cdots\}$.

(b) The sequence of DTU service or auxiliary memory access and data transfer times is also one of i.i.d.r.v., $\{D_i, i = 1, 2, \cdots\}$.

(c) CPU and DTU processing times are mutually independent. Furthermore, we must assume the following.

(d) The Laplace transform, $E[e^{-sC}]$, of a generic CPU service time converges for $-s_0 < s$, for some $s_0 > 0$. This latter is truly a mathematical restriction, but is probably not a serious one; all gamma densities and convex combinations of exponentials (hyperexponentials) are covered, for example.

3. Analysis of the model. In summary, our present approximate analysis of the multiprogramming model proceeds by first attempting to find an appropriate set of parameters μ and σ in the diffusion equation

$$\partial F/\partial t = -\mu(\partial F/\partial x) + (\sigma^2/2)(\partial^2 F/\partial x^2). \tag{3.1}$$

Here $F(x; t)$ is the approximate distribution of the number of jobs in the CPU stage at time t. A method for obtaining parameters μ and σ^2 which was based on asymptotic renewal theory appears in [2] and is sketched in Appendix B. Then we truncate the stationary distribution to allow for the boundary at $x = J$, and compare several methods for determining a crucial constant (denoted by B in [2]) that allows us to deal with the boundary at $x = 0$; the latter is important, for it is directly related to CPU utilization, which it is our intention to estimate.

Consider the waiting time, W_n, of the nth customer to arrive at an ordinary single-server, one in which there is no restriction placed upon the number waiting. This model would approximate the behavior of a cyclic queue or multiprogramming system in which the number of programs J is unlimited. We shall assume, as is realistic, that the CPU service rate outstrips that of the DTU, i.e., $E[C] < E[\mathrm{D}]$. Now Feller [1, pp. 194–198] shows that W_n has the same distribution as the maximum of the partial sums of the unrestricted random walk

$$W_n \overset{(d)}{=} M_n = \max[0, S_1, S_2, \cdots, S_n], \tag{3.2}$$

where

$$S_n = X_1 + X_2 + \cdots + X_n \tag{3.3}$$

and

(3.4) $$X_k = C_k - D_k.$$

To study M_n, invoke Wald's identity (see Feller [1, p. 603] or Kingman [4]):

(3.5) $$E\{e^{sS_N}/[\psi(s)]^N\} = 1,$$

N being the random time at which a boundary is reached, and

(3.6) $$\psi(s) = E[e^{sX}] = E[e^{sC}]E[e^{-sD}].$$

Now place a boundary at $x > 0$, and another at $-b$, $b > 0$. Then since by definition of N, $P\{-b \leqq S_N \leqq x\} = 0$,

(3.7) $$E\{e^{sS_N}/\psi^N|S_N > x\}P\{S_N > x\} + E\{e^{sS_N}/\psi^N|S_N < -b\}P\{S_N < -b\} = 1$$

If $E[C] < E[D]$, it may be shown that the equation

(3.8) $$\psi(s) = 1$$

has a solution at $s = 0$, and one at $\underset{\sim}{s} > 0$. Put $s = \underset{\sim}{s}$, let $b \to \infty$, and observe that then

(3.9) $$P\{S_N > x\} = 1/E\{e^{\underset{\sim}{s}S_N}|S_N > x\}.$$

This is the probability that the unrestricted random walk S ever exceeds the boundary at x, and is, by (3.2), equal to the probability that the waiting time exceeds x. We write this as

(3.10) $$P\{W > x\} = e^{-\underset{\sim}{s}x}/E\{e^{\underset{\sim}{s}(S_N - x)}|S_N > x\}$$

where $S_N - x > 0$ represents the *excess*; if we neglect this excess we obtain the estimate $P\{W > x\} \leqq e^{-\underset{\sim}{s}x}$; if x is large the excess is comparatively small, which suggests the approximation $P\{W > x\} \cong K\,e^{-\underset{\sim}{s}x}$, where K is a constant that must be determined.

By the result of Haji and Newell [3], the number, Q, of customers in the queue is the number that arrive during the waiting time of an arbitrary customer; reference is to the stationary distributions of both W and Q. Conditionally,

(3.11) $$P\{Q \geqq n|W = x\} = G^{n*}(x),$$

where G is the distribution function of D, and $*$ represents Stieltjes convolution. Then, by (3.11) above,

(3.12) $$P\{Q \geqq n\} \cong K\int_0^\infty G^{n*}(x)\,e^{-\underset{\sim}{s}x}\underset{\sim}{s}\,dx = K[\hat{G}(\underset{\sim}{s})]^n$$

where $\hat{G}(\underset{\sim}{s})$ is the Laplace–Stieltjes transform of G, evaluated at $\underset{\sim}{s}$. This effectively states that, at least under heavy traffic conditions ($\rho = (E[C]/E[D])$ barely < 1), the stationary distribution of the number in the system is exponential, but with parameter somewhat different from that of the diffusion approximation:

(3.13a) Diffusion: $P\{Q \geqq x\} \cong K'\,e^{(2\mu/\sigma^2)x}$, where $\mu = 1/E[D] - 1/E[C]$ and $\sigma^2 = \text{var}\,[D]/(E[D])^3 + \text{var}\,[C]/(E[C])^3$

(3.13b) $$\text{Wald:}\quad P\{Q \geqq x\} \cong K\,e^{[\ln \hat{G}(\underset{\sim}{s})]x} = K[\hat{G}(\underset{\sim}{s})]^x;$$

see Gaver and Shedler [2]. For a new approximation, we then merely replace the ratio $2\mu/\sigma^2$ by $\ln \hat{G}(\underset{\sim}{s})$ and fit constants as was done in [2]. The relation between the parameters in the diffusion approximation expressed by (3.13a) and that in the approximation resulting from Wald's identity (3.13b) is considered in Appendix A.

Given the values of μ and σ^2, the stationary diffusion approximation for the distribution F of Q satisfies

(3.14) $$0 = -dF/dx + (\sigma^2/2\mu)(d^2F/dx^2),$$

in which we now propose to replace $2\mu/\sigma^2$ by $\ln \hat{G}(\underset{\sim}{s})$. We also must determine the constant B in the solution to (3.14):

(3.15) $$F(x;J) = (1 - B\,e^{-\alpha x})/(1 - B\,e^{-\alpha J}),$$

where the latter expression satisfies the boundary condition at $x = J : F(J;J) = 1$. Here we have introduced the notation $F(x;J)$ to emphasize the dependence upon the parameter J. The constant $\alpha > 0$ can be determined either by an argument based on asymptotic normality in conflicting renewal processes (see [2]), or as we have argued, using Wald–Haji–Newell results.

4. Fitting the constant B: approximations for CPU utilization. We suggest and investigate two ways in which the constant B in (3.15) can readily be determined.

Method 1. If $J = \infty$, then it is well known (see Takacs [7, p. 142]) that server (CPU) utilization is

(4.1) $$1 - F(0+;\infty) = \rho \equiv E[C]/E[D].$$

Hence it follows that to achieve this approximately for large J, we should put $B = B_1 = \rho$, from which

(4.2) $$F(x;J) = (1 - \rho\,e^{-\alpha x})/(1 - \rho\,e^{-\alpha J}), \qquad 0 \leqq x \leqq J;\alpha > 0.$$

This approach was taken in [2] with good results for exponential CPU service.

Method 2. With probability $F(J-1;J)$, there is at least one program in residence at the DTU. Hence the long-run input to the CPU should be $1 \cdot F(J-1;J)$, assuming that $E[D] = 1$. Now the long-run output rate from the CPU must equal the input rate, and the output rate approximates $[1 - F(0+;J)] \cdot (1/E[C])$. By this conservation principle, then,

$$[1 - B\,e^{-\alpha(J-1)}]/(1 - B\,e^{-\alpha J}) = (B/E[C])[(1 - e^{-\alpha J})/(1 - B\,e^{-\alpha J})],$$

from which we find that $B_2 = \rho/(1 + \rho\,e^{-\alpha(J-1)} - e^{-\alpha J})$. Of course, $B_2 \to B_1$ as $J \to \infty$.

We shall shortly provide some numerical comparisons that illustrate the behavior of the two methods when they are applied to actual measurement data.

5. Special cases. We now describe the manner in which our approximations may be applied when certain specific distributions are in force.

Case 1. CPU service exponential, $E[C] = \lambda^{-1}$; DTU service Erlang $- k$, $E[D] = 1$.

In this case, equation (3.8) has the form

$$\psi(s) \equiv [\lambda/(\lambda - s)][1/(1 + s/k)]^k = 1. \tag{5.1}$$

It must be solved numerically for $\underset{\sim}{s}$, a task that can be carried out by Newton–Raphson iteration.

Case 2. CPU services exponential; DTU service constant, $E[D] = 1$.

For this limiting case of (5.1), let $k \to \infty$ to obtain the equation

$$[\lambda/(\lambda - s)]\, e^{-s} = 1. \tag{5.2}$$

Case 3. CPU services Erlang $- k$, $E[C] = \lambda^{-1}$; DTU service constant, $E[D] = 1$.

Here we must solve

$$[1/(1 - s/\lambda k)]^k e^{-s} = 1. \tag{5.3}$$

Case 4. CPU services hyperexponential; DTU services constant, $E[D] = 1$.

Representation of CPU services by means of a convex combination of exponentials (the hyperexponential distribution) suggests itself according to actual program trace data. This model leads to the equation

$$\left(p\frac{\lambda_1}{\lambda_1 - s} + (1 - p)\frac{\lambda_2}{\lambda_2 - s}\right) e^{-s} = 1, \tag{5.4}$$

where

$$E[C] \equiv \lambda^{-1} = p(\lambda_1)^{-1} + (1 - p)(\lambda_2)^{-1},$$

and p takes on an appropriate value between zero and unity. In practice it is convenient (if not statistically efficient) to fit the parameters of Cases 3 and 4 by the matching of low moments from model and data. Supposing that $(\lambda_2)^{-1} < E[C] < (\lambda_1)^{-1}$, it can be shown that, given $E[C]$ and var $[C]$ such that $(\text{var}\,[C])^{1/2}/E[C] > 1$, along with $(\lambda_2)^{-1}$, p and λ_1 are uniquely determined.

Unfortunately, all of the above models require the numerical solution of a transcendental equation in order to generate actual numerical estimates of CPU utilization. This disadvantage is not possessed by the diffusion approximation of [2].

It is of interest that our procedure gives results in complete accord with an exact analysis in one particular case.

Case 5. CPU and DTU services exponential.

This case can easily be analyzed by simple birth-and-death process methods, for which see [2]. Our procedure demands that we first solve

$$[\lambda/(\lambda - s)][1 + s]^{-1} = 1, \tag{5.5}$$

which in this case has the explicit solution $\underset{\sim}{s} = \lambda - 1$; consequently $\hat{G}(\underset{\sim}{s}) = 1/\lambda \equiv \rho$. Then the approximation yields

$$F(0+\,; J) = (1 - B_i)/(1 - B_i\rho^J), \qquad i = 1, 2.$$

Here B_i refers to the constant B as determined by Method i ($i = 1$ or 2). But for the present model we have

$$B_2 = \rho/(1 + \rho[\rho^{J-1}] - \rho^J) = \rho = B_1, \tag{5.6}$$

and use of $B_i = \rho$ yields

$$F(0+;J) = (1 - \rho)/(1 - \rho^{J+1}), \tag{5.7}$$

so our approximation is in this case equal to the birth-and-death result. For our other cases, exact equality will not hold.

6. Numerical results. We now present numerical results to indicate the quality of the proposed approximation. Our examples are in the context of a single processor system with two-level memory, multiprogrammed and operated in a demand paging environment. A discussion of cyclic queueing phenomena in such systems is given in Lewis and Shedler [5]. Accordingly, we interpret the CPU service times in our model as execution intervals, i.e., times between page exceptions as programs execute in (constrained) memory of given capacity. We concentrate on Case 4 of § 5 (CPU services hyperexponential, DTU services constant) on the basis of our experience that execution intervals often fit well to a hyperexponential model. The assumption of constant DTU service times arises from the consideration of average access time along with the time to transfer a page of information.

In all cases that we shall consider, values for p, λ_1 and λ_2 in the hyperexponential were obtained by matching first and second moments of the empirical distribution obtained from actual program data.

Tables 1 and 2 contain numerical results for CPU utilization obtained by the approximation technique (for both methods of fitting the constant B) along with results of exact analysis based on semi-Markov (S–M) methods as given in [5].

Finally, we present some results of CPU utilization obtained by trace-driven simulation of the cyclic queueing system. By this we mean that CPU service times in the model were taken to be the actual sequence of execution

TABLE 1

CPU *utilization comparisons*

J	S–M	Approx. B_1	Approx. B_2	J	S–M	Approx. B_1	Approx. B_2
2	0.3903	0.1909	0.3972	2	0.2216	0.1455	0.2216
3	0.4054	0.2486	0.4274	3	0.2286	0.1789	0.2313
4	0.4178	0.2924	0.4440	4	0.2333	0.2003	0.2361
5	0.4280	0.3264	0.4545	5	0.2366	0.2144	0.2388
6	0.4367	0.3534	0.4616	6	0.2388	0.2238	0.2404
7	0.4439	0.3751	0.4668	7	0.2403	0.2301	0.2415
8	0.4501	0.3927	0.4708	8	0.2413	0.2344	0.2422
9	0.4553	0.4072	0.4736	9	0.2420	0.2373	0.2426
10	0.4598	0.4193	0.4759	10	0.2425	0.2393	0.2429

$E[C] = 4871$, var $[C] = 0.26492 \times 10^9$, $(\lambda_2)^{-1} = 1929$, $E[D] = 10{,}000$

$E[C] = 4871$, var $[C] = 0.26492 \times 10^9$, $(\lambda_2)^{-1} = 1929$, $E[D] = 20{,}000$

TABLE 2

CPU *utilization comparisons*

J	S–M	Approx. B_1	Approx. B_2	J	S–M	Approx. B_1	Approx. B_2
2	0.4076	0.0770	0.4249	2	0.5316	0.2148	0.5993
3	0.4281	0.1094	0.4579	3	0.5548	0.2884	0.6667
4	0.4449	0.1385	0.4764	4	0.5752	0.3481	0.7064
5	0.4587	0.1649	0.4882	5	0.5935	0.3974	0.7326
6	0.4702	0.1887	0.4964	6	0.6098	0.4388	0.7511
7	0.4798	0.2105	0.5024	7	0.6245	0.4741	0.7650
8	0.4879	0.2304	0.5070	8	0.6379	0.5045	0.7757
9	0.4948	0.2485	0.5106	9	0.6500	0.5309	0.7842
10	0.5006	0.2654	0.5136	10	0.6611	0.5542	0.7911
$E[C] = 10{,}735$, var $[C] = 0.12313 \times 10^{10}, (\lambda_2)^{-1} = 2953$, $E[D] = 20{,}000$				$E[C] = 17{,}026$, var $[C] = 0.39780 \times 10^{10}, (\lambda_2)^{-1} = 3682$, $E[D] = 20{,}000$			

intervals derived from a program trace, J copies of this sequence being multiprogrammed. In Table 3, these trace-driven results are displayed, along with values of CPU utilization obtained by the approximation technique.

7. Summary and conclusions. This paper presents the results of approximating processor utilization in multiprogrammed computer systems using ideas of diffusion. In particular, the objective is to develop convenient and nearly explicit formulas relating CPU utilization to simple program parameters and to the level of multiprogramming.

Numerical comparisons indicate that a reasonably effective approximation has been obtained when the constant B_2 is utilized. Examples show that for the actual program traces studied our present approximation is superior to that of [2], which assumed exponentially distributed CPU service times. Data from our trace material is far more skewed (long-tailed) than that yielded by the exponential. Research continues in an attempt to improve the approximate procedures obtained to date. A promising approach is the iteration of our approximate solutions. Of course, an eventual goal is that of obtaining simple but adequate approximations to properties of somewhat more complex and truly realistic networks of servers.

Appendix A. The relation between the parameter in the diffusion approximation of [2], as expressed in (3.13a), and that in the approximation resulting from Wald's identity, (3.13b), will now be considered. Application of Wald's identity requires that we find the positive root, $\underset{\sim}{s}$, of (3.8). Let us expand $\psi(s)$ by a Taylor series:

$$\psi(s) = 1 + s\mu_x + (s^2/2)\sigma_x^2 + R(s) \tag{A.1}$$

where the remainder is $o(s^2)$, provided that required moments exist. Here

$$\begin{aligned} \mu_x &= E[X] = E[C - D] < 0 \\ \sigma_x^2 &= \text{var}\,[X] = \text{var}\,[C] + \text{var}\,[D]. \end{aligned} \tag{A.2}$$

TABLE 3

Trace-driven simulation CPU *utilization comparisons*

J	Trace	Approx. B_1	Approx. B_2	J	Trace	Approx. B_1	Approx. B_2	J	Trace	Approx. B_1	Approx. B_2
3	0.227	0.1789	0.2313	3	0.419	0.1094	0.4579	3	0.538	0.2884	0.6667
6	0.229	0.2238	0.2404	6	0.425	0.1887	0.4964	6	0.546	0.4388	0.7511
$E[C] = 4871$, var $[C] = 0.26492 \times 10^9$, $E[D] = 20{,}000$				$E[C] = 10{,}735$, var $[C] = 0.12313 \times 10^{10}$, $E[D] = 20{,}000$				$E[C] = 17{,}026$, var $[C] = 0.39780 \times 10^{10}$, $E[D] = 20{,}000$			

At $\underset{\sim}{s}$ we have from (A.1) and (3.13b), after dispensing with the root at $s = 0$,

$$\mu_x + \underset{\sim}{s}(\sigma_x^2/2) + r(\underset{\sim}{s}) = 0, \tag{A.3}$$

or

$$2\mu_x/\underset{\sim}{s}\sigma_x^2 + 1 + (2/\sigma_x^2)[(1/\underset{\sim}{s})r(\underset{\sim}{s})] = 0. \tag{A.4}$$

Therefore, if we consider a sequence of queueing situations in which $\underset{\sim}{s} \to 0$ and σ_x^2 does not approach zero, the remainder term approaches zero, since $r(s) = o(s)$. We see then that as $\underset{\sim}{s} \to 0$,

$$(2\mu_x/\underset{\sim}{s}\sigma_x^2) \to -1, \tag{A.5}$$

or

$$\underset{\sim}{s} \sim -2\mu_x/\sigma_x^2. \tag{A.6}$$

In the event that $\underset{\sim}{s} \to 0$, our Wald approximation and the approximation of [2] coincide, as will now be shown. For $\underset{\sim}{s}$ approaching zero, as will be true in heavy traffic,

$$-\ln \hat{G}(\underset{\sim}{s}) = \underset{\sim}{s}E[D] + o(\underset{\sim}{s}). \tag{A.7}$$

Consequently the parameter in the Wald–Haji–Newell approximation becomes, in heavy traffic,

$$\begin{aligned} -\underset{\sim}{s}E[D] &= 2\frac{\mu_x E[D]}{\sigma_x^2} = -2\frac{(E[D] - E[C])E[D]}{\operatorname{var}[D] + \operatorname{var}[C]} \\ &= -2\frac{1/E[C] - 1/E[D]}{\operatorname{var}[D]/(E[D])^2E[C] + \operatorname{var}[C]/(E[D])^2E[C]} \\ &\sim 2\frac{1/E[D] - 1/E[C]}{\operatorname{var}[D]/(E[D])^3 + \operatorname{var}[C]/(E[C])^3} = 2\frac{\mu}{\sigma^2}. \end{aligned} \tag{A.8}$$

For the specific models introduced earlier in § 5, it is clearly sufficient to allow the mean CPU service time to approach unity from below in order to force $\underset{\sim}{s}$ to zero. Consider, for example, Case 3: if $1/\lambda = E[C]$ is allowed to increase, it becomes apparent that for every fixed s, $\psi(s)$, the left-hand side of (5.3), increases, and $\underset{\sim}{s}$ moves continuously towards the origin; when $1/\lambda = 1$, there is a (double) root at $s = 0$. A similar effect occurs when, say, $\lambda_1 \to 0$ in (5.4), a maneuver that allows $E[C]$ to approach unity. Again, $\psi(s)$ is increased for every s, and in the limit there is a double root at $s = 0$. Recall that the region of convergence of the transform $\psi(s)$ is $s < \min(\lambda_1, \lambda_2) = \bar{s}$, and since $\bar{s} > \underset{\sim}{s}$, a decrease in either λ_1 or λ_2 eventually sends $\underset{\sim}{s}$ to zero. Examination of the denominator of (3.10) suggests also that if $\underset{\sim}{s}$ is near zero, the expectation is near unity, thus further justifying the use of our approximation.

Appendix B. Let $N_C(t)$ denote the number of programs present at the CPU at time t; this includes those queued in addition to the program currently in service. Then if $N_C(0) = 0$,

$$N_C(t) = A(t) - D(t), \tag{B.1}$$

where $A(t)$ represents the number of arrivals at the CPU in $(0, t)$, and $D(t)$ is the number of CPU departures in $(0, t)$. If we neglect boundary effects at 0 and J, $A(t)$ and $D(t)$ are independent renewal processes, so as t becomes large, $A(t)$ and $D(t)$ are approximately normally distributed with mean $t/E[D]$ and $t/E[C]$, and variances $t\{\mathrm{var}\,[D]/(E[D])^3\}$ and $t\{\mathrm{var}\,[C]/(E[C])^3\}$, respectively. It follows that $N_C(t)$ is approximately normally distributed with mean $\mu t = [1/E[D] - 1/E[C]]t$ and variance $\sigma^2 t = \{\mathrm{var}\,[D]/(E[D])^3 + \mathrm{var}\,[C]/(E[C])^3\}t$. We now approximate by replacing the difference of renewal processes by a diffusion (Wiener process) with drift μ and infinitesimal variance σ^2. Thus, $F(x, t)$, the distribution of $N_C(t)$, satisfies the diffusion equation

$$\partial F/\partial t = -\mu(\partial F/\partial x) + (\sigma^2/2)(\partial^2 F/\partial x^2), \tag{B.2}$$

again approximately. A reflecting boundary condition at $x = 0$ must be imposed, for $N_C(t) \geqq 0$, and another such boundary condition at $x = J$ constrains $N_C(t)$ to be $\leqq J$. We require the solution to (B.2), subject to an initial distribution, e.g.,

$$F(x; 0) = \begin{cases} 1, & x \geqq x_0 > 0 \\ 0, & x < x_0, \end{cases} \tag{B.3}$$

and boundary conditions

$$F(0+; t) \geqq 0, \qquad F(J, t) = 1. \tag{B.4}$$

The stationary or long-run distribution associated with our problem is $F(x) = \lim_{t\to\infty} F(x; t)$ and satisfies $(\sigma^2/2)(d^2F/dx^2) - \mu(dF/dx) = 0$ for $x > 0$. Routine integrations lead to the solution

$$F(x) = A[1 - B\, e^{(2\mu/\sigma^2)x}]. \tag{B.5}$$

Invocation of the upper boundary condition provides that

$$1 = A[1 - B\, e^{(2\mu/\sigma^2)J}]. \tag{B.6}$$

According to (B.5) and (B.6), we have

$$F(x) = [1 - B\, e^{(2\mu/\sigma^2)x}]/[1 - B\, e^{(2\mu/\sigma^2)J}]. \tag{B.7}$$

It remains to determine the constant B; see § 4 above.

REFERENCES

[1] W. FELLER, *An Introduction to Probability Theory*, vol. II, 2nd Edition, John Wiley, New York, 1971.

[2] D. P. GAVER AND G. S. SHEDLER, *Processor utilization in multiprogramming systems via diffusion approximations*, Operations Res., 21 (1973), pp. 569–576.

[3] R. HAJI AND G. F. NEWELL, *A relation between stationary queue and waiting time distributions*, J. Appl. Probability, 8 (1971), pp. 617–620.

[4] J. F. C. KINGMAN, *A Martingale inequality in the theory of queues*, Proc. Cambridge Philos. Soc., 60 (1964), pp. 359–361.

[5] P. A. W. LEWIS AND G. S. SHEDLER, *A cyclic-queue model of system overhead in multiprogrammed computer systems*, J. Assoc. Comput. Mach., 18 (1971), pp. 199–220.

[6] G. F. NEWELL, *Applications of Queueing Theory*, Chapman and Hall, London, 1971.

[7] TAKACS, L., *Introduction to the Theory of Queues*, Oxford Univ. Press, New York, 1962.

SIAM J. COMPUT.
Vol. 2, No. 3, September 1973

FINITE STATE REPRESENTATIONS OF DISCRETE OPTIMIZATION PROBLEMS*

TOSHIHIDE IBARAKI†

Abstract. This paper is concerned with the representation of a discrete optimization problem given in the form of a *ddp* (discrete decision process) by a *G-sdp* (*G*-sequential decision process). A *G-sdp* is a finite state model of discrete optimization problem, consisting of a finite number of states and a rule specifying the transition from one state to another corresponding to each decision applied to it. A cost function, taken from a given family of functions *G*, is associated with each transition. A necessary and sufficient condition for a given *ddp* to be represented by a *G-sdp*, which is valid for most important *G*'s, is obtained; it turns out that various representation theorems obtained in the earlier paper [3] are special cases of this theorem. Furthermore, a case in which the existence of the unique minimal representation is guaranteed to exist receives special attention, and some sufficient conditions are discussed.

Key words. Discrete optimization, *G*-sequential decision process, representation theorem, finite automata, unique minimal representation.

1. Introduction.

In papers by Karp and Held [7], and by Ibaraki [3], [4], the representation of a discrete optimization problem given in the form of a *ddp* (discrete decision process; the formal definition will be given in § 2) by a finite state model, i.e., an *sdp* (sequential decision process), was considered. An *sdp* consists of a finite number of states and a rule specifying the transition from one state to another corresponding to each decision applied to it. A cost function is associated with each transition. An optimal policy of an *sdp* is a sequence of decisions which transforms the initial state into one of the designated final states with the minimum cost. (For the formal definition see § 2.)

An *sdp* is called a *monotone sdp* (*msdp*) if its cost functions satisfy a certain monotonicity property. The *msdp* has received special attention [7] in conjunction with the theory of dynamic programming. Various subclasses of the class of *msdp*'s with additional restrictions on cost functions were introduced by Ibaraki [3] from the viewpoint of algorithm efficiency for obtaining optimal policies. These subclasses have a common feature in that the set of optimal policies is always regular (in the sense of the automata theory).

In view of this line of development of the theory, the following problem appears quite interesting: given a set of functions *G* eligible for cost functions, what is a necessary and sufficient condition for a discrete optimization problem to be represented by a *G-sdp* (an *sdp* with cost functions taken from *G*)? In this paper, we will establish a theorem for this problem. It is valid for many meaningful *G*'s. In particular, the representation theorems for *sdp*, *msdp* and some subclasses of *msdp* obtained in [3] and [4] are derived as special cases.

Furthermore, it will be shown that there is a case in which the existence of the unique minimal representation (i.e., with the fewest states) is guaranteed. Some sufficient conditions for the property to hold will also be discussed.

* Received by the editors July 5, 1972.

† Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, Kyoto, Japan.

2. Definitions. We assume that a discrete optimization problem under consideration is written in the form of a *ddp* defined below.

DEFINITION 2.1. A *discrete decision process* (*ddp*) Υ is a system (Σ, S, f), where

Σ is a finite nonempty set of primitive decisions;

Σ^* is the set of all policies, where a policy is a sequence of finite primitive decisions; ε denotes the null policy (i.e., the policy consisting of 0 primitive decisions); xy for $x, y \in \Sigma^*$ denotes the policy obtained by concatenating x and y;

$S \subset \Sigma^*$ is a set of feasible policies;

$f: S \to A$, where $A \subset E$ and E is the set of real numbers.

A policy $x \in \Sigma^*$ is *optimal* if $x \in S \wedge (\forall y \in S)(f(x) \leqq f(y))$ holds. In the rest of this paper, Υ always stands for a *ddp* (Σ, S, f) even if it is not explicitly stated.

DEFINITION 2.2. A *finite automaton* (*fa*) M is a system $(Q, \Sigma, q_0, \lambda, Q_F)$, where Σ is the same as above and

Q is a finite nonempty set of states;

$q_0 \in Q$ is an initial state;

$\lambda: Q \times \Sigma \to Q$ is a state transition function;

$Q_F \subset Q$ is a set of final states.

λ can be extended to $Q \times \Sigma^* \to Q$ inductively by

$$(\forall q \in Q)(\forall x \in \Sigma^*)(\forall a \in \Sigma)(\lambda(q, \varepsilon) = q \wedge \lambda(q, xa) = \lambda(\lambda(q, x), a)).$$

$\bar{\lambda}(x) \equiv \lambda(q_0, x)$ is used for convenience. $F(M) \equiv \{x | \bar{\lambda}(x) \in Q_F\}$ is the set of policies *accepted* by M. $B \subset \Sigma^*$ is said to be *regular* if there exists an *fa* M such that $F(M) = B$.

DEFINITION 2.3. A *sequential decision process* (*sdp*) Π is a system (M, h, ξ_0), where

M is an *fa*;

h is $A \times Q \times \Sigma \to A$, the cost function of Π;

$\xi_0 \in A$ is an initial cost value of state q_0.

h can be extended to $A \times Q \times \Sigma^* \to A$ by

$$(\forall \xi \in A)(\forall q \in Q)(\forall x \in \Sigma^*)(\forall a \in \Sigma)(h(\xi, q, \varepsilon) = \xi \wedge h(\xi, q, xa) = h(h(\xi, q, x), \lambda(q, x), a)).$$

The notation $\bar{h}(x) \equiv h(\xi_0, q_0, x)$ is used for convenience. The set *of feasible policies* of Π is given by $F(\Pi) = F(M)$. The policy $x \in \Sigma^*$ is *optimal* in Π if $x \in F(\Pi) \wedge (\forall y \in F(\Pi))(\bar{h}(x) \leqq \bar{h}(y))$ holds.

Many interesting examples of *ddp*'s and *sdp*'s in practical applications are found in [7].

DEFINITION 2.4. Let G be a family of functions: $A \to A$. An *sdp* $\Pi = (M, h, \xi_0)$ is a *G-sdp* if $h_{qa}: A \to A$ defined by

$$(\forall \xi \in A)(h_{qa}(\xi) = h(\xi, q, a))$$

belongs to G for all $q \in Q$ and $a \in \Sigma$. Throughout this paper, it is assumed that G forms a *monoid* under composition (i.e., the identity function g_I belongs to G and $g, h \in G$ implies $gh \in G$).

Example 2.5. Let $A = E$, where E is the set of real numbers.

(i) $G_0 = \{g : E \to E\}$. Then G_0-*sdp* is the same as the *sdp* of Definition 2.3. The *sdp* was discussed in [7] and [3].

(ii) $G_m = \{g \in G_0 | (\forall \xi, \zeta \in E)(\xi \leqq \zeta \Rightarrow g(\xi) \leqq g(\zeta))\}$. Then G_m-*sdp* is the *msdp* (monotone *sdp*) discussed in [7] and [3].

(iii) $G_s = \{g \in G_0 | (\forall \xi, \zeta \in E)(\xi < \zeta \Rightarrow g(\xi) < g(\zeta))\}$. Then G_s-*sdp* is the *smsdp* (strictly monotone *sdp*) discussed in [3].

(iv) $G_p = \{g \in G_m | (\forall \xi \in E)(g(\xi) \geqq \xi)\}$. Then G_p-*sdp* is the *pmsdp* (positively monotone *sdp*) discussed in [3].

(v) $G_a = \{g \in G_0 | (\exists k \in E)(\forall \xi \in E)(g(\xi) = \xi + k)\}$. Then G_a-*sdp* is the *ap* (additive process) discussed in [7] and [3].

These models will be carried along throughout this paper, and it will be shown that the representation theorem obtained in this paper is valid for all these models except for G_p.

Example 2.6. Let $A = Z$, where Z is the set of integers.

(i) $G_r = \{g : Z \to Z | \ g \text{ is a recursive function}^1 \text{ on } Z\}$. Then G_r-*sdp* is the *tr-sdp* (totally recursive *sdp*).

(ii) $G_{rm} = \{q \in G_r | (\forall \xi, \zeta \in Z)(\xi \leqq \zeta \Rightarrow g(\xi) \leqq g(\zeta))\}$. Then G_{rm}-*sdp* is the *tr-msdp* (totally recursive *msdp*).

Various decision problems (in the sense of the theory of computation) of these models were extensively discussed in [4]. However, it appears difficult to deal with these models in the framework of the theory developed in this paper.

Now we present the precise meaning of the representation.

DEFINITION 2.7. Let $\Upsilon = (\Sigma, S, f)$ be a *ddp* and let $\Pi = (M, h, \xi_0)$ be a *G-sdp*. Then Π *strongly represents*[2] (*s-represents*) Υ if

$$F(\Pi) = S \wedge (\forall x \in S)(\bar{h}(x) = f(x))$$

holds. Π is a minimal s-representation of Υ if Π s-represents Υ, and there exists no *G-sdp* which s-represents Υ and has fewer states than Π.

The rest of this paper will be devoted to the derivation of a representation theorem which is a necessary and sufficient condition for a *ddp* to be s-represented by a *G-sdp*. For that, a few more definitions are necessary.

DEFINITION 2.8. Let R, T be equivalence relations on Σ^*. T *refines* R if $(\forall x, y \in \Sigma^*)(xTy \Rightarrow xRy)$. This is denoted $T \leqq R$. Let $B \subset \Sigma^*$. R *refines* B if $(\forall x, y \in \Sigma^*)(xRy \Rightarrow (x \in B \Leftrightarrow y \in B))$. B/R stands for the set of equivalence classes of B under R. $|B/R|$ denotes the number of equivalence classes in B/R. Obviously, $T \leqq R$ implies $|\Sigma^*/T| \geqq |\Sigma^*/R|$. Let R be a binary relation (not necessarily an equivalence relation). R is *right invariant* if $(\forall x, y \in \Sigma^*)(xRy \Rightarrow (\forall z \in \Sigma^*)(xzRyz))$ holds. $\Lambda(B)$ denotes the set of right invariant equivalence relations which refine $B \subset \Sigma^*$. $\Lambda_F(B)$ is the set of all $R \in \Lambda(B)$ with $|\Sigma^*/R| < \infty$.

Remark 2.9. Define R_B for $B \subset \Sigma^*$ by

$$(\forall x, y \in \Sigma^*)(xR_By \Leftrightarrow (\forall z \in \Sigma^*)(xz \in B \Leftrightarrow yz \in B)).$$

[1] For the definition, see [1] and [4].

[2] The weak representation was defined in [3], but it will not be discussed in this paper.

In other words, $(\forall x, y \in \Sigma^*)(xR_By \Leftrightarrow (x\backslash B = y\backslash B))$, where $x\backslash B = \{z|xz \in B\}$. As proved in [8], $R_B \in \Lambda(B)$ holds, and furthermore any $T \in \Lambda(B)$ satisfies $T \leqq R_B$. Thus $\Lambda_F(B)$ is nonempty if and only if $R_B \in \Lambda_F(B)$. It is also known that $R_B \in \Lambda_F(B)$ holds if and only if B is regular [8].

DEFINITION 2.10. Let $\Lambda_F(B)$ be nonempty and let $T \in \Lambda_F(B)$. Then *fa* $M = (Q, \Sigma, q_0, \lambda, Q_F)$ defined by

$$Q = \{[B_i] | B_i \in \Sigma^*/T\}$$

$$\lambda([x], a) = [xa] \quad \text{for } x \in \Sigma^*, a \in \Sigma$$

$$q_0 = [\varepsilon]$$

$$Q_F = \{[B_i] | B_i \in B/T\}$$

is the *standard construction* of T, where $[x]$ denotes $[B_i]$ such that $x \in B_i \wedge B_i \in \Sigma^*/T$. By definition, $|Q| = |\Sigma^*/T|$ follows.

Remark 2.11. It is known [8] that the standard construction M is well-defined for any $T \in \Lambda_F(B)$ and that $F(M) = B$ holds. Conversely, define T for an *fa* $M = (Q, \Sigma, q_0, \lambda, Q_F)$ by $(\forall x, y \in \Sigma^*)(xTy \Leftrightarrow \bar{\lambda}(x) = \bar{\lambda}(y))$, then $T \in \Lambda_F(F(M))$ follows and M is the standard construction of T.

Remark 2.12. If a *ddp* $\Upsilon = (\Sigma, S, f)$ is s-represented by a *G-sdp* $\Pi = (M, h, \xi_0)$, then S is regular since $S = F(M)$ holds by definition. However, the regularity of S does not always guarantee the existence of a *G-sdp* s-representing Υ since cost functions h_{qa} of Π must be taken from G.

3. Representation theorem. In this section, we first introduce a binary relation Φ defined on Σ^* relative to given Υ and G. This relation plays an important role in the representation theorem (Theorem 3.17).

DEFINITION 3.1. For a *ddp* $\Upsilon = (\Sigma, S, f)$, define H_Υ by

$$H_\Upsilon = \{\gamma : \Sigma^* \to A | (\forall x \in S)(\gamma(x) = f(x))\},$$

i.e., $\gamma \in H_\Upsilon$ is an extension of $f : S \to A$.

DEFINITION 3.2. For a family of functions: $A \to A$, G and a *ddp* $\Upsilon = (\Sigma, S, f)$, define $\Gamma_\Upsilon^G(x, y) \subset H_\Upsilon$ as follows for $x, y \in \Sigma^*$ satisfying xR_Sy, where R_S was defined in Remark 2.9.

$$\Gamma_\Upsilon^G(x, y) = \{\gamma \in H_\Upsilon | (\forall w \in \Sigma^*)(\forall a \in \Sigma)(\exists g_{wa} \in G)(g_{wa}(\gamma(xw)) = \gamma(xwa) \wedge g_{wa}(\gamma(yw)) = \gamma(ywa))\}.$$

The binary relation Φ_Υ^G is then defined by

$$(\forall x, y \in \Sigma^*)(x\Phi_\Upsilon^G y \Leftrightarrow xR_Sy \wedge \Gamma_\Upsilon^G(x, y) \neq \varnothing).$$

The subscript Υ and the superscript G of Γ and Φ will be usually omitted throughout this paper, since there will be no confusion.

Remark 3.3. $x\Phi y$ implies that xR_Sy holds, and, moreover, it is possible to assign cost values $\gamma(z)$ to all $z \in \Sigma^*$ so that

(i) $\gamma(z) = f(z)$ may hold for all $z \in S$, and

(ii) for each $w \in \Sigma^*$ and $a \in \Sigma$, cost values $\gamma(xw)$, $\gamma(yw)$ and $\gamma(xwa)$, $\gamma(ywa)$ can be linked by a function g_{wa} taken from G.

The symbol $\Gamma(x, y)$ denotes the set of all assignments of cost values satisfying these conditions. By definition, Φ is symmetric, i.e., $x\Phi y \Leftrightarrow y\Phi x$, but in general neither reflexive nor transitive. The order of x and y in $\Gamma(x, y)$ is also immaterial since $\Gamma(x, y) = \Gamma(y, x)$ always holds. For most G, the reflexivity of Φ can be easily proved. For example, Φ is reflexive for G_0, G_m, G_s and G_a regardless of the *ddp* Υ under consideration. However, $\Phi_\Upsilon^{G_p}$ may not be reflexive, depending upon the Υ under consideration. Finally it is noted that Φ is right invariant, as proved next. First, R_S is right invariant (see Remark 2.9). Next, it holds that $\gamma \in \Gamma(x, y) \Rightarrow (\forall z \in \Sigma^*)(\gamma \in \Gamma(xz, yz))$ by definition, implying that $\Gamma(x, y) \neq \varnothing \Rightarrow (\forall z \in \Sigma^*)(\Gamma(xz, yz) \neq \varnothing)$.

Remark 3.4. Since G is assumed to be a monoid, a slightly simplified definition of Φ is also possible:

$$(\forall x, y \in \Sigma^*)(x\Phi y \Leftrightarrow xR_Sy \wedge \Gamma'(x, y) \neq \varnothing)$$

$$\Gamma'(x, y) \equiv \{\gamma \in H_\Upsilon | (\forall w \in x\backslash U)(\forall u \in xw\backslash U)(\exists g_{wu} \in G)$$
$$(g_{wu}(\gamma(xw)) = \gamma(xwu) \wedge g_{wu}(\gamma(yw)) = \gamma(ywu))\},$$

where U is given by

$$U = \begin{cases} S \cup \{x, y\} \text{ if neither of } x \text{ and } y \text{ is a prefix}^3 \text{ of the other}, \\ S \cup \{x, xv, xv^2, \cdots\} \quad \text{if } y = xv \quad \text{for } v \in \Sigma^*, \\ S \cup \{y, yv, yv^2, \cdots\} \quad \text{if } x = yv \quad \text{for } v \in \Sigma^*. \end{cases}$$

(The simplification consists in that $\gamma(x)$ is essential only for $x \in U$ in this definition.)

Proof. First we prove that $\Gamma(x, y) \neq \varnothing \Rightarrow \Gamma'(x, y) \neq \varnothing$. For $\gamma \in \Gamma(x, y)$, define g_{wu} for $w \in x\backslash U$ and $u = a_1a_2 \cdots a_k \in xw\backslash U$ by

$$g_{wu} = g_{w_{k-1}a_k}g_{w_{k-2}a_{k-1}} \cdots g_{w_0a_1},$$

where $w_i = wa_1a_2 \cdots a_i$ for $i = 1, 2, \cdots, k$ and $w_0 = w$, and $g_{w_ia_j} \in G$ is the one used in the definition of $\Gamma(x, y)$. Then $g_{wu} \in G$ since $g_{w_ia_j} \in G$ by definition and G is a monoid. In case $u = \varepsilon$, let $g_{wu} = g_I$ (the identity function) $\in G$. In any case, it is straightforward to prove that $g_{wu}(\gamma(xw)) = \gamma(xwu) \wedge g_{wu}(\gamma(yw)) = \gamma(ywu)$ holds. This proves $\Gamma'(x, y) \neq \varnothing$. To prove that $\Gamma'(x, y) \neq \varnothing \Rightarrow \Gamma(x, y) \neq \varnothing$, first note that $xR_Sy \Rightarrow xR_Uy$ holds since

$$x\backslash U = x\backslash S \cup \{\varepsilon\} = y\backslash S \cup \{\varepsilon\} = y\backslash U$$

if $U = S \cup \{x, y\}$, and

$$x\backslash U = x\backslash S \cup \{\varepsilon, v, v^2, \cdots\} = y\backslash S \cup \{\varepsilon, v, v^2, \cdots\} = y\backslash U,$$

otherwise. Now define $g_{wa} \in G$ for $w \in \Sigma^*$ and $a \in \Sigma$ by

$$g_{wa} = \begin{cases} g_I & \text{if } xwa \notin U \\ g_{u_1u_2} & \text{if } xwa \in U, \text{ where } wa = u_1u_2 \text{ and } u_1 \text{ is the longest} \\ & \text{proper prefix of } wa \text{ such that } xu_1 \in U \text{ and } g_{u_1u_2} \\ & \text{is the one used in the definition of } \Gamma'(x, y). \end{cases}$$

---

[3] x is a *prefix* of y if there exists $v \in \Sigma^*$ such that $y = xv$. x is a *proper prefix* of y if $v \neq \varepsilon$.

Then γ, defined for $\gamma' \in \Gamma'(x, y)$ by

$$\gamma(z) = \begin{cases} \gamma'(u), & \text{where } u \text{ is the longest prefix of } z \text{ satisfying } u \in U \\ & \text{if there exists such } u, \\ \gamma'(\varepsilon), & \text{otherwise,} \end{cases}$$

belongs to $\Gamma(x, y)$, since for any $w \in \Sigma^*$ and $a \in \Sigma$,

$$g_{wa}(\gamma(xw)) = g_{wa}(\gamma'(xu_1)) = g_{u_1u_2}(\gamma'(xu_1)) = \gamma'(xu_1u_2) = \gamma(xwa)$$

if $xwa \in U$, and

$$g_{wa}(\gamma(xw)) = g_{wa}(\gamma'(xu_1)) = g_I(\gamma'(xu_1)) = \gamma'(xu_1) = \gamma(xwa)$$

if $xwa \notin U$, where u_1 is the longest proper prefix of wa such that $xu_1 \in U$. Similarly it is possible to show that $g_{wa}(\gamma(yw)) = \gamma(ywa)$ holds. Thus $\Gamma'(x, y) \neq \varnothing \Rightarrow \Gamma(x, y) \neq \varnothing$. Q.E.D.

Remark 3.5. If G is a group (i.e., g^{-1} exists for all $g \in G$), the definition of Φ is further simplified:

$$(\forall x, y \in \Sigma^*)(x\Phi y \Leftrightarrow xR_Sy \wedge \Gamma''(x, y) \neq \varnothing)$$

$$\Gamma''(x, y) \equiv \{\gamma \in H_\Upsilon | (\forall w \in x \backslash U)(\exists g_w \in G)(g_w(\gamma(x)) = \gamma(xw) \wedge g_w(\gamma(y)) = \gamma(yw))\},$$

where U was defined in Remark 3.4.

Proof. It is trivial to prove that $xR_Sy \wedge \Gamma'(x, y) \neq \varnothing \Rightarrow xR_Sy \wedge \Gamma''(x, y) \neq \varnothing$ holds. To prove the converse, assume that $xR_Sy \wedge \Gamma''(x, y) \neq \varnothing$ holds. Then for $\gamma \in \Gamma''(x, y)$ and $w \in x \backslash U$, $u \in xw \backslash U$, there exist $g_w, g_v \in G$ (where v stands for wu) such that

$$g_w(\gamma(x)) = \gamma(xw) \wedge g_w(\gamma(y)) = \gamma(yw),$$
$$g_v(\gamma(x)) = \gamma(xwu) \wedge g_v(\gamma(y)) = \gamma(ywu).$$

Define g_{wu} by $g_{wu} = g_v g_w^{-1}$. Then $g_{wu} \in G$, since G is a group and

$$g_{wu}(\gamma(xw)) = g_v g_w^{-1}(\gamma(xw)) = g_v(\gamma(x)) = \gamma(xwu),$$
$$g_{wu}(\gamma(yw)) = g_v g_w^{-1}(\gamma(yw)) = g_v(\gamma(y)) = \gamma(ywu).$$

Thus $\gamma \in \Gamma'(x, y)$, and hence we have $xR_Sy \wedge \Gamma''(x, y) \neq \varnothing \Rightarrow xR_Sy \wedge \Gamma'(x, y) \neq \varnothing$. Q.E.D.

Example 3.6. Let $G = G_0$ (see Example 2.5(i)). G_0 is a monoid but not a group. Now define a binary relation R_Υ on Σ^* for a *ddp* $\Upsilon = (\Sigma, S, f)$ by

$$(\forall x, y \in \Sigma^*)(xR_\Upsilon y \Leftrightarrow xR_Sy \wedge (\forall w \in x \backslash S)(f(xw) = f(yw))).$$

R_Υ is an equivalence relation and right invariant [3]. For $x, y \in \Sigma^*$ with xR_Sy, $\Gamma_\Upsilon^{G_0}$ (abbreviated by Γ_0) is given as follows.

$$\Gamma_0(x, y) = \{\gamma \in H_\Upsilon | (\forall w \in \Sigma^*)(\forall a \in \Sigma)(\exists g_{wa} \in G_0)$$
$$(g_{wa}(\gamma(xw)) = \gamma(xwa) \wedge g_{wa}(\gamma(yw)) = \gamma(ywa))\}$$
$$= \{\gamma \in H_\Upsilon | (\forall w \in \Sigma^*)(\forall a \in \Sigma)(\gamma(xw) = \gamma(yw) \Rightarrow \gamma(xwa) = \gamma(ywa))\}.$$

Thus we have ($\Phi_\Upsilon^{G_0}$ is denoted Φ_0) that for $x, y \in \Sigma^*$,

$$x\Phi_0 y (\Leftrightarrow xR_S y \wedge \Gamma_0(x,y) \neq \phi) \Leftrightarrow xR_S y \wedge (\forall w \in x\backslash S)(f(xw) = f(yw) \Rightarrow xwR_\Upsilon yw).$$

The $\Rightarrow$ part is obvious since $(\forall w \in \Sigma^*)(\forall a \in \Sigma)(\gamma(xw) = \gamma(yw) \Rightarrow \gamma(xwa) = \gamma(ywa))$ $\Rightarrow (\forall w \in x\backslash S)(\forall u \in xw\backslash S)(f(xw) = f(yw) \Rightarrow f(xwu) = f(ywu)) \Rightarrow (\forall w \in x\backslash S)$ $(f(xw) = f(yw) \Rightarrow xwR_\Upsilon yw)$ holds. To prove the $\Leftarrow$ part, consider $\gamma: \Sigma^* \to E$ satisfying $\gamma \in H_\Upsilon \wedge (\forall w \in \Sigma^*)(\gamma(xw) = \gamma(yw) \Leftrightarrow xwR_\Upsilon yw)$. Obviously $\gamma \in \Gamma_0(x,y)$, and such γ exists if $xR_S y \wedge (\forall w \in x\backslash S)(f(xw) = f(yw) \Rightarrow xwR_\Upsilon yw)$ holds. (Note that $(\forall w \in x\backslash S)(f(xw) = f(yw) \Leftrightarrow xwR_\Upsilon yw)$ holds, and it is possible to define γ so that $(\forall w \notin x\backslash S)(\gamma(xw) = \gamma(yw) \Leftrightarrow xwR_\Upsilon yw)$ may hold since $\gamma \in H_\Upsilon$ does not pose any restriction on the values $\gamma(xw)$ and $\gamma(yw)$.) It is true that Φ_0 is reflexive but not always transitive.

Example 3.7. Let $G = G_m$ (see Example 2.5(ii)). G is a monoid but not a group. We first define a binary relation $\preccurlyeq_\Upsilon$ on Σ^* for a *ddp* $\Upsilon = (\Sigma, S, f)$ by

$$(\forall x, y \in \Sigma^*)(x \preccurlyeq_\Upsilon y \Leftrightarrow xR_S y \wedge (\forall w \in x\backslash S)(f(xw) \leqq f(yw))).$$

$\preccurlyeq_\Upsilon$ is a pseudo-ordering on Σ^* and right invariant [3]. Note that $xR_\Upsilon y \Leftrightarrow x \preccurlyeq_\Upsilon y$ $\wedge\ y \preccurlyeq_\Upsilon x$ holds. Now for $x, y \in \Sigma^*$ with $xR_S y$, it follows that ($\Gamma_\Upsilon^{G_m}$ is denoted Γ_m)

$$\Gamma_m(x,y) = \{\gamma \in H_\Upsilon | (\forall w \in \Sigma^*)(\forall a \in \Sigma)((\gamma(xw) \leqq \gamma(yw) \Rightarrow$$
$$\gamma(xwa) \leqq \gamma(ywa)) \wedge (\gamma(xw) = \gamma(yw) \Rightarrow \gamma(xwa) = \gamma(ywa)))\}$$

from the definition of G_m. Thus we have ($\Phi_\Upsilon^{G_m}$ is denoted Φ_m) that for $x, y \in \Sigma^*$,

$$x\Phi_m y (\Leftrightarrow xR_S y \wedge \Gamma(x,y) \neq \varnothing) \Leftrightarrow xR_S y \wedge (\forall w \in x\backslash S)((f(xw) = f(yw)$$
$$\Rightarrow xwR_\Upsilon yw) \wedge (x \preccurlyeq_\Upsilon y \vee y \preccurlyeq_\Upsilon x))(\text{i.e., } x\Phi_0 y \wedge (x \preccurlyeq_\Upsilon y \vee y \preccurlyeq_\Upsilon x)).$$

(The $\Rightarrow$ part is obvious. To prove the $\Leftarrow$ part, consider $\gamma: \Sigma^* \to E$ satisfying

$$\gamma \in H_\Upsilon \wedge (\forall w \in \Sigma^*)((xw \preccurlyeq_\Upsilon yw \Leftrightarrow \gamma(xw) \leqq \gamma(yw)) \wedge (yw \preccurlyeq_\Upsilon xw \Leftrightarrow \gamma(yw) \leqq \gamma(xw))).$$

Obviously $\gamma \in \Gamma_m(x,y)$, and such γ exists if $x\Phi_0 y \wedge (x \preccurlyeq_\Upsilon y \vee y \preccurlyeq_\Upsilon x)$ holds.) Again, Φ_m is reflexive but not always transitive.

Example 3.8. Let $G = G_s$ (see Example 2.5 (iii)). G_s is a monoid but not a group. Define a binary relation $\trianglelefteq_\Upsilon$ on Σ^* for a *ddp* $\Upsilon = (\Sigma, S, f)$ by

$$(\forall x, y \in \Sigma^*)(x \trianglelefteq_\Upsilon y \Leftrightarrow xR_S y \wedge (xR_\Upsilon y \vee (\forall w \in x\backslash S)(f(xw) < f(yw)))).$$

Then by an argument similar to Example 3.7, we have ($\Phi_\Upsilon^{G_s}$ is denoted Φ_s) that for $x, y \in \Sigma^*$,

$$x\Phi_s y \Leftrightarrow x\Phi_0 y \wedge (x \trianglelefteq_\Upsilon y \vee y \trianglelefteq_\Upsilon x).$$

As previously, Φ_s is reflexive but not always transitive.

Example 3.9. Let $G = G_p$ (see Example 2.5 (iv)). From the definition of G_p, it follows that for $x, y \in \Sigma^*$ satisfying $xR_S y$ ($\Gamma_\Upsilon^{G_p}$ is denoted Γ_p),

$$\Gamma_p(x,y) = \{\gamma \in H_\Upsilon | (\forall w \in \Sigma^*)(\forall a \in \Sigma)((\gamma(xw) \leqq \gamma(yw) \Rightarrow$$
$$\gamma(xwa) \leqq \gamma(ywa)) \wedge (\gamma(xw) = \gamma(yw) \Rightarrow \gamma(xwa) = \gamma(ywa))$$
$$\wedge (\gamma(xw) \leqq \gamma(xwa)) \wedge (\gamma(yw) \leqq \gamma(ywa)))\}.$$

The additional restriction on γ, $\gamma(xw) \leqq \gamma(xwa) \wedge \gamma(yw) \leqq \gamma(ywa)$, makes it difficult to define $\Phi_{\Upsilon}^{G_p}$ in a simple manner. Namely, a somewhat involved discussion is required to see whether $\Gamma_p(x, y) \neq \emptyset$ holds or not, though a necessary and sufficient condition for $\Gamma_p(x, y) \neq \emptyset$ can be directly obtained from the discussion given in [3]. In general, Φ_p is neither reflexive nor transitive.

Example 3.10. Let $G = G_a$ (see Example 2.5 (v)). G_a is a group since g^{-1} is given for $g(\xi) = \xi + k$ by $g^{-1}(\xi) = \xi - k$. Define a binary relation D_{Υ} on Σ^* for a *ddp* $\Upsilon = (\Sigma, S, f)$ by

$$(\forall x, y \in \Sigma^*)(xD_{\Upsilon}y \Leftrightarrow xR_Sy \wedge (\forall w, u \in x\backslash S)(f(xw) - f(yw) = f(xu) - f(yu))).$$

D_{Υ} is an equivalence relation and right invariant [7], [3]. $\Gamma_a(x, y)(\equiv \Gamma_{\Upsilon}^{G_a}(x, y))$ for $x, y \in \Sigma^*$ satisfying xR_Sy is given by

$$\Gamma_a(x, y) = \{\gamma \in H_{\Upsilon} | (\forall w \in \Sigma^*)(\forall a \in \Sigma)(\gamma(xw) - \gamma(yw) = \gamma(xwa) - \gamma(ywa))\}.$$

Obviously it holds that $\Gamma_a(x, y) \neq \emptyset \Leftrightarrow xD_{\Upsilon}y$. Thus we have that for $x, y \in \Sigma^*$ ($\Phi_{\Upsilon}^{G_a}$ is denoted Φ_a), $x\Phi_a y \Leftrightarrow xD_{\Upsilon}y$. Since D_{Υ} is an equivalence relation, Φ_a is also an equivalence relation (i.e., Φ_a is reflexive and transitive as well as symmetric).

DEFINITION 3.11. For a set $B \subset \Sigma^*$, let $\Gamma(B)$ be defined by

$$\Gamma(B) = \{\gamma \in H_{\Upsilon} | (\forall w \in \Sigma^*)(\forall a \in \Sigma)(\exists g_{wa} \in G)(\forall x \in B)(g_{wa}(\gamma(xw)) = \gamma(xwa))\}.$$

(Note that g_{wa} is independent of x.) Similarly, for $B_1, B_2, \cdots, B_n \subset \Sigma^*$, let $\Gamma(B_1, B_2, \cdots, B_n)$ be defined by

$$\Gamma(B_1, B_2, \cdots, B_n) = \{\gamma \in H_{\Upsilon} | (\forall i \in N)(\forall w \in \Sigma^*) \\ (\forall a \in \Sigma)(\exists g_{wa}^i \in G)(\forall x \in B_i)(g_{wa}^i(\gamma(xw)) = \gamma(xwa))\},$$

where $N = \{1, 2, \cdots, n\}$. For $T \in \Lambda_F(S)$ with

$$\Sigma^*/T = \{B_1, B_2, \cdots, B_n\},$$

$\Gamma(B_1, B_2, \cdots, B_n)$ is also denoted by $\Gamma(T)$. A set $B \subset \Sigma^*$ is *Φ-consistent* if $(\forall x, y \in B)(x\Phi y)$ holds. An equivalence relation $T \in \Lambda_F(S)$ is *Φ-consistent* if all $B_i \in \Sigma^*/T$ is Φ-consistent.

Remark 3.12. It is obvious that $\Gamma(B) \subset \Gamma(x, y)$ holds for any $x, y \in B$. Thus for $B_i \in \Sigma^*/T$ where $T \in \Lambda_F(S)$, $\Gamma(B_i) \neq \emptyset \Rightarrow (\forall x, y \in B_i)(xR_Sy \wedge \Gamma(x, y) \neq \emptyset)$ $\Leftrightarrow (\forall x, y \in B_i)(x\Phi y) \Leftrightarrow B_i$ is Φ-consistent. The converse, however, does not always hold. (See also Assumption 3.15 and Remark 3.16.)

PROPOSITION 3.13. *Let $T \in \Lambda_F(S)$ and $\Sigma^*/T = \{B_1, B_2, \cdots, B_n\}$. Define $\bar{\Gamma}(T)$ by*

$$\bar{\Gamma}(T) = \{\gamma \in H_{\Upsilon} | (\forall i \in N)(\forall a \in \Sigma)(\exists g_a^i \in G)(\forall x \in B_i)(g_a^i(\gamma(x)) = \gamma(xa))\}.$$

Then $\Gamma(T) = \bar{\Gamma}(T)$ holds.

Proof. $\Gamma(T) \subset \bar{\Gamma}(T)$ is obvious. To prove $\bar{\Gamma}(T) \subset \Gamma(T)$, let $\gamma \in \bar{\Gamma}(T)$. Since T is right invariant, it holds that $(\forall i \in N)(\forall w \in \Sigma^*)(\exists j \in N)(B_iw \subset B_j)$, where $B_iw = \{xw | x \in B_i\}$. Define g_{wa}^i for $i \in N$, $w \in \Sigma^*$ and $a \in \Sigma$ by $g_{wa}^i = g_a^j$, where $B_iw \subset B_j$. Then $g_{wa}^i \in G$ since $g_a^j \in G$ by definition, and g_{wa}^i satisfies

$$(\forall x \in B_i)(g_{wa}^i(\gamma(xw)) = g_a^j(\gamma(xw)) = \gamma(xwa)).$$

This proves $\gamma \in \Gamma(T)$. Q.E.D.

LEMMA 3.14. *A ddp $\Upsilon = (\Sigma, S, f)$ is s-representable by a G-sdp if and only if there exists $T \in \Lambda_F(S)$ such that $\Gamma(T) \neq \varnothing$.*

Proof. Necessity. Let *G-sdp* $\Pi = (M(Q, \Sigma, q_0, \lambda, Q_F), h, \xi_0)$ s-represent Υ. Define $T \in \Lambda_F(S)$ by $(\forall x, y \in \Sigma^*)(xTy \Leftrightarrow \bar{\lambda}(x) = \bar{\lambda}(y))$ (see Remark 2.11), and $\gamma \in H_\Upsilon$ by $(\forall x \in \Sigma^*)(\gamma(x) = \bar{h}(x))$. Let $\Sigma^*/T = \{B_1, B_2, \cdots, B_n\}$. Then $\gamma \in \bar{\Gamma}(T)$ holds, since

$$(\forall i \in N)(\forall a \in \Sigma)(\forall x \in B_i)(g_a^i(\gamma(x)) = \gamma(xa))$$

is satisfied by $g_a^i \in G$ defined by

$$(\forall x \in B_i)(g_a^i(\gamma(x)) = h(\gamma(x), \bar{\lambda}(x), a)).$$

($g_a^i \in G$ holds since Π is a *G-sdp*.) Thus by Proposition 3.13, we have $\Gamma(T) \neq \varnothing$.

Sufficiency. Let M be the standard construction of T (see Definition 2.10), and let $\gamma \in \Gamma(T)$. Define $h: A \times Q \times \Sigma \to A$ as follows:

For each $q_i \equiv [B_i]$, where $B_i \in \Sigma^*/T$, and $a \in \Sigma$, define $h_{q_i a}(\xi) (\equiv h(\xi, q_i, a))$ by

$$h_{q_i a}(\xi) = g_a^i(\xi) \quad \text{for } \xi \in A,$$

where g_a^i satisfies

$$g_a^i \in G \wedge (\forall a \in \Sigma)(\forall x \in B_i)(g_a^i(\gamma(x)) = \gamma(xa)),$$

for $\gamma \in \Gamma(T)$. (Such g_a^i exists by definition.) Finally let $\xi_0 = \gamma(\varepsilon)$. Then

$$sdp\, \Pi = (M, h, \xi_0)$$

is a *G-sdp* s-representing Υ. Q.E.D.

A main result of this paper, s-representation theorem, will be obtained by adding the following assumption to Lemma 3.14.

Assumption 3.15. For G and *ddp* $\Upsilon = (\Sigma, S, f)$ under consideration, any Φ-consistent $T \in \Lambda_F(S)$ satisfies $\Gamma(T) \neq \varnothing$.

Remark 3.16. As obvious from Remark 3.12, the converse of Assumption 3.15 is always true. Since the role of Assumption 3.15 in the s-representation theorem is crucial, its validity will be extensively studied in § 4. For example, it will be shown that Assumption 3.15 holds true for G_0, G_m, G_s and G_a irrelevant of the *ddp* Υ under consideration.

THEOREM 3.17 (s-representation). *Under Assumption* 3.15, *a ddp $\Upsilon = (\Sigma, S, f)$ is s-representable by a G-sdp if and only if there exists a Φ-consistent equivalence relation $T \in \Lambda_F(S)$.*

Proof. Necessity. By Remark 3.12, $T \in \Lambda_F(S)$ defined in the proof of the necessity of Lemma 3.14 is Φ-consistent.

Sufficiency. This is obvious from Lemma 3.14 and Assumption 3.15. Q.E.D.

By applying Theorem 3.17 to G_0, G_m, G_s and G_a (for which Assumption 3.15 holds true), we have the following corollaries which are those s-representation theorems obtained in [3].

COROLLARY 3.18. *A ddp Υ is s-representable by a G_0-sdp if and only if there exists $T \in \Lambda_F(S)$ such that*

$$(\forall x, y \in \Sigma^*)(xTy \wedge x, y \in S \Rightarrow (f(x) = f(y) \Rightarrow xR_\Upsilon y)).$$

(Note that $xTy \Rightarrow xR_Sy \Rightarrow (x \in S \Leftrightarrow y \in S)$ by definition.)

Proof. $(\forall x, y \in \Sigma^*)(xTy \wedge x, y \in S \Rightarrow (f(x) = f(y) \Rightarrow xR_\Upsilon y)) \Leftrightarrow (\forall x, y \in \Sigma^*)$ $(xTy \Rightarrow (\forall w \in x \setminus S)(f(xw) = f(yw) \Rightarrow xwR_\Upsilon yw))$ (since $xTy \Rightarrow (\forall w \in \Sigma^*)(xwTyw)$ holds by definition) $\Leftrightarrow (\forall x, y \in \Sigma^*)(xTy \Rightarrow x\Phi_0 y)$ (see Example 3.6). Then by Theorem 3.17 and Remark 3.16, the corollary is proved. Q.E.D.

COROLLARY 3.19. *A ddp Υ is s-representable by a G_m-sdp if and only if there exists $T \in \Lambda_F(S)$ such that*

$$(\forall x, y \in \Sigma^*)(xTy \Rightarrow (x \preccurlyeq_\Upsilon y \vee y \preccurlyeq_\Upsilon x) \wedge (x, y \in S \Rightarrow (f(x) = f(y) \Rightarrow xR_\Upsilon y))).$$

Proof. $(\forall x, y \in \Sigma^*)(xTy \Rightarrow (x \preccurlyeq_\Upsilon y \vee y \preccurlyeq_\Upsilon x) \wedge (x, y \in S \Rightarrow (f(x) = f(y) \Rightarrow xR_\Upsilon y))) \Leftrightarrow (\forall x, y \in \Sigma^*)(xTy \Rightarrow (x \preccurlyeq_\Upsilon y \vee y \preccurlyeq_\Upsilon x) \wedge x\Phi_0 y) \Leftrightarrow (\forall x, y \in \Sigma^*)(xTy \Rightarrow x\Phi_m y)$ (see Example 3.7). Q.E.D.

COROLLARY 3.20. *A ddp Υ is s-representable by a G_s-sdp if and only if there exists $T \in \Lambda_F(S)$ such that*

$$(\forall x, y \in \Sigma^*)(xTy \Rightarrow (x \trianglelefteq_\Upsilon y \vee y \trianglelefteq_\Upsilon x) \wedge (x, y \in S \Rightarrow (f(x) = f(y) \Rightarrow xR_\Upsilon y))).$$

Proof. This is obvious from Theorem 3.17, Remark 3.16 and Example 3.8. Q.E.D.

COROLLARY 3.21. *A ddp Υ is s-representable by a G_a-sdp if and only if there exists $T \in \Lambda_F(S)$ such that*

$$(\forall x, y \in \Sigma^*)(xTy \Rightarrow xD_\Upsilon y).$$

Proof. The proof is obvious from Theorem 3.17, Remark 3.16 and Example 3.10. Q.E.D.

4. Properties of $\Gamma(T)$. This section discusses various properties of $\Gamma(T)$ for Φ-consistent $T \in \Lambda_F(S)$, and shows that $\Gamma(T) \neq \emptyset$ is assured for any *ddp* Υ (i.e., Assumption 3.15), if G satisfies certain conditions.

DEFINITION 4.1. Let G be a family of functions: $A \to A$, and let $A_0 \subset A$. Assume that $g_{\xi\eta} \in G$ is defined for each pair (ξ, η) such that $\xi, \eta \in A_0$ (the order of subscripts of $g_{\xi\eta}$ is not important, i.e., $g_{\xi\eta} = g_{\eta\xi}$ is assumed), and they satisfy

$$(\forall \xi \in A_0)(\forall \eta_1, \eta_2 \in A_0)(g_{\xi\eta_1}(\xi) = g_{\xi\eta_2}(\xi)).$$

Let $g^*: A_0 \to A$ be defined by

$$(\forall \xi, \eta \in A_0)(g^*(\xi) = g_{\xi\eta}(\xi)).$$

Then G is said *pairwise extendible* if, for any $A_0 \subset A$ and a set of $g_{\xi\eta} \in G$, $\xi, \eta \in A_0$, satisfying the above condition, $g \in G$ exists such that g is an extension of g^*.

Example 4.2. Let $G = G_m$ (see Example 2.5 (ii)). For a given $A_0 \subset A(=E)$, the existence of $g_{\xi\eta}$, ξ, $\eta \in A_0$, satisfying the condition in Definition 4.1 implies that

$$(\forall \xi, \eta \in A_0)(\xi \leqq \eta \Rightarrow g^*(\xi) \leqq g^*(\eta)).$$

It is then obvious that there exists $g \in G$ which is an extension of $g^*: A_0 \to A$ defined in Definition 4.1. Thus G_m is pairwise extendible. In a similar manner, it is possible to prove that G_0, G_s, G_p and G_a are all pairwise extendible.

PROPOSITION 4.3. *Let G be pairwise extendible. Then*
(i) for any Φ-consistent $B \subset \Sigma^*$, it holds that

$$\Gamma(B) = \bigcap\{\Gamma(x, y) | x, y \in B\},$$

(ii) for any Φ-consistent $B_1, B_2, \cdots, B_n \subset \Sigma^*$, it holds that

$$\Gamma(B_1, B_2, \cdots, B_n) = \bigcap_{i=1}^{n} \Gamma(B_i).$$

Proof. (i) Let $\tilde{\Gamma}(B) \equiv \bigcap\{\Gamma(x, y) | x, y \in B\}$. That $\Gamma(B) \subset \tilde{\Gamma}(B)$ is obvious by definition. To prove $\Gamma(B) \supset \tilde{\Gamma}(B)$, let $\gamma \in \tilde{\Gamma}(B)$. Then $(\forall w \in \Sigma^*)(\forall a \in \Sigma)(\forall x, y \in B)$ $(\exists g_{wa}^{xy} \in G)(g_{wa}^{xy}(\gamma(xw)) = \gamma(xwa) \wedge g_{wa}^{xy}(\gamma(yw)) = \gamma(ywa))$. Since G is pairwise extendible, this implies the existence of $g_{wa} \in G$ such that

$$(\forall w \in \Sigma^*)(\forall a \in \Sigma)(\forall x \in B)(g_{wa}(\gamma(xw)) = \gamma(xwa)).$$

(For given $w \in \Sigma^*$ and $a \in \Sigma$, consider $\{\gamma(xw) | x \in B\}$ as A_0 of Definition 4.1, and for $\xi = \gamma(xw) \in A_0$ and $\eta = \gamma(yw) \in A_0$, consider g_{wa}^{xy} as $g_{\xi\eta}$ of Definition 4.1.) Thus we have $\gamma \in \Gamma(B)$. Part (ii) is obvious by definition (see Definition 3.11). Q.E.D.

Now we show the next theorem which will be used as a basis for proving the validity of Assumption 3.15.

THEOREM 4.4. *Let G be pairwise extendible. For each pair $x, y \in \Sigma^*$ with $x\Phi y$, assume that there exists a nonempty $\hat{\Gamma}(x, y)$ satisfying*
(i) $\hat{\Gamma}(x, y) \subset \Gamma(x, y)$;
(ii) $\hat{\Gamma}(x, y)$ *can be written as follows*:

$$\hat{\Gamma}(x, y) = \{\gamma : \Sigma^* \to A | (\forall w \in \Sigma^*)((\gamma(xw), \gamma(yw)) \in \Delta(x, y; w))\},$$

where $\Delta(x, y; w) \subset A \times A$;
(iii) $\Delta(x, y; w) \supset \Delta(xw, yw; \varepsilon)$ holds for any x, y, $w \in \Sigma^*$ satisfying $x\Phi y$.
(Condition (ii) assumes that the restriction posed on $\gamma \in \hat{\Gamma}(x, y)$ can be decomposed into a set of restrictions posed on pairs $(\gamma(xw), \gamma(yw))$.)
Furthermore, for Φ-consistent $T \in \Lambda_F(S)$ with $\Sigma^*/T = \{B_1, B_2, \cdots, B_n\}$, let

$$\Gamma^*(B_i) \equiv \{\gamma : \Sigma^* \to A | (\forall x, y \in B_i)((\gamma(x), \gamma(y)) \in \Delta(x, y; \varepsilon))\}.$$

Then $\Gamma^*(B_i) \neq \varnothing$ for $i = 1, 2, \cdots, n$ implies $\Gamma(T) \neq \varnothing$.

Proof. First note that the restriction posed on $\gamma \in \Gamma^*(B_i)$ is limited to the values $\gamma(x), x \in B_i$, and no restriction is posed on $\gamma(x)$, $x \notin B_i$. Therefore $B_i \cap B_j = \varnothing$ for $i \neq j$ implies that $(\forall i \in N)(\Gamma^*(B_i) \neq \varnothing) \Rightarrow \bigcap_{i=1}^{n} \Gamma^*(B_i) \neq \varnothing$. Let $\gamma \in \bigcap \Gamma^*(B_i)$. Then for any $x, y \in B_i$, $B_i \in \Sigma^*/T$, we have $(\forall w \in \Sigma^*)(\exists B_j \in \Sigma^*/T)$ $\cdot (xw, yw \in B_j) \Rightarrow (\forall w \in \Sigma^*)((\gamma(xw), \gamma(yw)) \in \Delta(xw, yw; \varepsilon)) \Rightarrow (\forall w \in \Sigma^*)((\gamma(xw),$ $\gamma(yw)) \in \Delta(x, y; w))$ (by condition (iii)) $\Rightarrow \gamma \in \hat{\Gamma}(x, y)$. Thus $\gamma \in \bigcap\{\Gamma(x, y) | x, y \in B_i\}$ by condition (i). Since G is pairwise extendible, this implies $\gamma \in \Gamma(B_i)$ for $i = 1, 2, \cdots, n$ by Proposition 4.3 (i). Hence $\gamma \in \Gamma(T)$ follows by Proposition 4.3 (ii). Q.E.D.

Example 4.5. Let $G = G_0$. For $x, y \in \Sigma^*$ with $x\Phi_0 y$, let

$$\hat{\Gamma}_0(x, y) = \{\gamma \in H_\Upsilon | (\forall w \in \Sigma^*)(\gamma(xw) = \gamma(yw) \Leftrightarrow xwR_\Upsilon yw)\}.$$

From the discussion given in Example 3.6, it is obvious that $\hat{\Gamma}_0(x, y) \subset \Gamma_0(x, y)$.

$\hat{\Gamma}_0(x, y)$ is alternatively defined by

$$\hat{\Gamma}_0(x, y) = \{\gamma : \Sigma^* \to A | (\forall w \in \Sigma^*)((\gamma(xw), \gamma(yw)) \in \Delta(x, y; w))\},$$

where

$$\Delta(x, y; w) = \begin{cases} \{(f(xw), f(yw))\} & \text{if } xw, yw \in S, \\ \{(\xi, \xi) | \xi \in A\} & \text{if } xw, yw \notin S \wedge xwR_\Upsilon yw, \\ \{(\xi, \eta) | \xi \neq \eta \wedge \xi, \eta \in A\} & \text{if } xw, yw \notin S \wedge \sim xwR_\Upsilon yw. \end{cases}$$

Thus it holds that for any $w \in \Sigma^*$,

$$\Delta(x, y; w) = \Delta(xw, yw; \varepsilon),$$

and condition (iii) of Theorem 4.4 is satisfied. It is also easy to prove that $\Gamma^*(B_i) \neq \emptyset$ for any Φ_0-consistent $B_i \subset \Sigma^*$ (see Example 3.6). Consequently, by Theorem 4.4, we are assured that $\Gamma_0(T) \neq \emptyset$ for any Φ_0-consistent $T \in \Lambda_F(S)$.

Example 4.6. Let $G = G_m$. For $x, y \in \Sigma^*$ with $x\Phi_m y$, let

$$\begin{aligned} \hat{\Gamma}_m(x, y) &= (\gamma \in H_\Upsilon | (\forall w \in \Sigma^*)((\gamma(xw) \leqq \gamma(yw) \Leftrightarrow xw \preccurlyeq_\Upsilon yw) \\ &\qquad \wedge (\gamma(yw) \leqq \gamma(xw) \Leftrightarrow yw \preccurlyeq_\Upsilon xw))\} \\ &= \{\gamma : \Sigma^* \to A | (\forall w \in \Sigma^*)((\gamma(xw), \gamma(yw)) \in \Delta(x, y; w))\}, \end{aligned}$$

where

$$\Delta(x, y; w) = \begin{cases} \{(f(xw), f(yw))\} & \text{if } xw, yw \in S, \\ \{(\xi, \xi) | \xi \in A\} & \text{if } xw, yw \notin S \wedge xwR_\Upsilon yw, \\ \{(\xi, \eta) | \xi, \eta \in A \wedge \xi < \eta\} & \text{if } xw, yw \notin S \wedge xw \preccurlyeq_\Upsilon yw \wedge \sim xwR_\Upsilon yw, \\ \{(\xi, \eta) | \xi, \eta \in A \wedge \eta < \xi\} & \text{if } xw, yw \notin S \wedge yw \preccurlyeq_\Upsilon xw \wedge \sim xwR_\Upsilon yw. \end{cases}$$

It is obvious that $\hat{\Gamma}_m(x, y) \subset \Gamma_m(x, y)$ (see Example 3.7), and $\Delta(x, y; w) = \Delta(xw, yw; \varepsilon)$. Furthermore $\Gamma^*(B_i) \neq \emptyset$ obviously holds for any Φ_m-consistent $B_i \subset \Sigma^*$. Hence $\Gamma_m(T) \neq \emptyset$ holds for any Φ_m-consistent $T \in \Lambda_F(S)$.

Example 4.7. The case of $G = G_s$ can also be treated in a manner similar to G_m. We are also assured that $\Gamma_s(T) \neq \emptyset$ for any Φ_s-consistent $T \in \Lambda_F(S)$.

Example 4.8. Let $G = G_p$. It seems difficult to define $\hat{\Gamma}_p(x, y)$ satisfying conditions of Theorem 4.4, because of the restriction $(\forall w \in \Sigma^*)(\gamma(x) \leqq \gamma(xw))$ required by G_p. (Note that the above restriction is not on the pair $(\gamma(xw), \gamma(yw))$, but on the pair $(\gamma(x), \gamma(xw))$.)

It is also possible to apply Theorem 4.4 to G_a, though the construction of $\hat{\Gamma}_a(x, y)$ is slightly more complicated. However, it can also be derived as a corollary of the following theorem.

THEOREM 4.9. *Let G be pairwise extendible and, moreover, let G be a group. Then* $\Gamma(T) \neq \emptyset$ *holds for any* Φ*-consistent* $T \in \Lambda_F(S)$.

Proof. Let $\Sigma^*/T = \{B_1, B_2, \cdots, B_n\}$. For each $B_i \subset W$, where

$$W = \{y \in \Sigma^* | y \backslash S \neq \emptyset\},$$

define w_i satisfying $B_i w_i \subset S$. If $B_i \subset S$, w_i is assumed to be $\varepsilon \in \Sigma^*$. Let $\gamma : \Sigma^* \to A$ be defined by

$$\gamma(x) = \begin{cases} f(xw_i) & \text{if } x \in B_i \subset W \\ \gamma(u) & \text{if } B_i \not\subset W, \text{ where } u \text{ is the longest prefix} \\ & \text{of } x \text{ such that } u \in W. \end{cases}$$

(Note that B_i is included in one of W and $\Sigma^* - W$ since $T \in \Lambda_F(S)$.) Now recall that for each $x, y \in B_i$, $\gamma^{xy} : \Sigma^* \to A$ exists such that

$$\gamma^{xy} \in H_\Upsilon \wedge (\forall a \in \Sigma)(\exists g_a^{xy} \in G)(g_a^{xy}(\gamma^{xy}(x)) = \gamma^{xy}(xa) \wedge g_a^{xy}(\gamma^{xy}(y)) = \gamma^{xy}(ya)),$$

since $x\Phi y$. Next define $h_i^{xy} \in G$ for each $x, y \in B_i \subset W$ such that

$$h_i^{xy}(\gamma^{xy}(x)) = \gamma^{xy}(xw_i)(= f(xw_i) = \gamma(x)).$$

For example, h_i^{xy} is given by

$$h_i^{xy} = g_{a_k}^{x_{k-1}y_{k-1}} \cdots g_{a_2}^{x_1 y_1} g_{a_1}^{xy},$$

where $w_i = a_1 a_2 \cdots a_k$ and $x_j = xa_1 \cdots a_j \wedge y_j = ya_1 \cdots a_j$, and $g_{a'}^{x'y'}$ is the one defined above. Then it follows that

$$(\forall i \in N)(\forall a \in \Sigma)(\forall x, y \in B_i)(\exists \bar{g}_a^{xy} \in G)(\bar{g}_a^{xy}(\gamma(x)) = \gamma(xa) \wedge \bar{g}_a^{xy}(\gamma(y)) = \gamma(ya)),$$

because $\bar{g}_a^{xy}$ can be given, for example, as follows:

$$\bar{g}_a^{xy} = \begin{cases} h_j^{x'y'} g_a^{xy} (h_i^{xy})^{-1} & \text{if } B_i \subset W \wedge B_i a \subset B_j \subset W, \text{ where } x' = xa \wedge y' = ya \\ g_I & \text{if } B_i a \subset B_j \not\subset W. \end{cases}$$

(See Fig. 1.) Since G is pairwise extendible, this implies that

$$(\forall i \in N)(\forall a \in \Sigma)(\exists \bar{g}_a^i \in G)(\forall x \in B_i)(\bar{g}_a^i(\gamma(x)) = \gamma(xa)).$$

Therefore $\gamma \in \Gamma(T)$ by Proposition 3.13. Q.E.D.

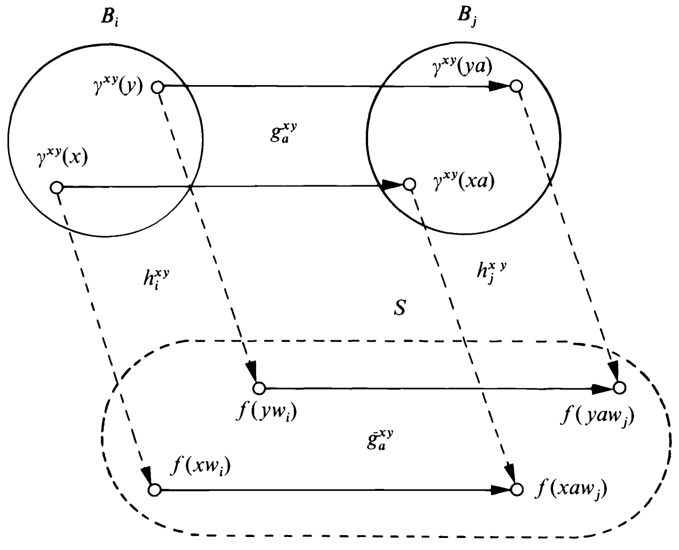


FIG. 1. *Illustration of $\bar{g}_a^{xy}$ defined in the proof of Theorem* 4.9

Example 4.10. Let $G = G_a$. Since G_a is pairwise extendible and a group, it follows from Theorem 4.9 that $\Gamma(T) \neq \emptyset$ holds for any Φ_a-consistent $T \in \Lambda_F(S)$.

5. *G-sdp* with unique minimal representation. In this section an important special class of *G-sdp*'s, for which the unique minimal representation of a *ddp* Υ is guaranteed to exist, is introduced. After giving such a theorem, some sufficient conditions for the theorem to hold will be discussed.

THEOREM 5.1. *Assume that* Φ *is an equivalence relation on* Σ^*. *Then under Assumption* 3.15, *a ddp* Υ *is s-representable by a G-sdp if and only if* $|\Sigma^*/\Phi| < \infty$ *holds. Furthermore, if* $|\Sigma^*/\Phi| < \infty$, *there exists the unique minimal s-representation of* Υ *with* $|\Sigma^*/\Phi|$ *states, in the sense that all minimal s-representations* $\Pi = (M, h, \xi_0)$ *have the same fa M.*

Proof. Necessity. Let $\Pi = (M(Q, \Sigma, q_0, \lambda, Q_F), h, \xi_0)$ s-represent Υ. Then T defined by $xTy \Leftrightarrow \bar{\lambda}(x) = \bar{\lambda}(y)$ is an equivalence relation on Σ^* satisfying $T \in \Lambda_F(S)$. Note that $\Phi \in \Lambda(S)$ also holds since Φ is an equivalence relation by assumption, Φ is right invariant (as shown in Remark 3.3) and Φ refines S (recall that $\Phi \leqq R_S$ by definition). As shown in Theorem 3.17, T is Φ-consistent and hence $T \leqq \Phi$ follows (see Definition 2.8). Thus we have $|\Sigma^*/\Phi| \leqq |\Sigma^*/T| < \infty$.

Sufficiency. Since Φ is Φ-consistent by definition, and $\Phi \in \Lambda_F(S)$ by $|\Sigma^*/\Phi| < \infty$, Theorem 3.17 can be immediately applied.

Uniqueness. Let $\Pi = (M(Q, \Sigma, q_0, \lambda, Q_F), h, \xi_0)$ be a minimal s-representation of Υ by a *G-sdp*. Let T be defined by $xTy \Leftrightarrow \bar{\lambda}(x) = \bar{\lambda}(y)$. Then $T \leqq \Phi$ as proved in the proof of necessity. However, since Π is minimal, it holds that $|\Sigma^*/T| \leqq |\Sigma^*/\Phi|$. Thus we have $T = \Phi$, and M is the standard construction of Φ (see Remark 2.11).

Q.E.D.

COROLLARY 5.2. (See [7].) *A ddp* Υ *is s-representable by a* G_a*-sdp if and only if* $|\Sigma^*/\Phi_a| < \infty$ (see Example 3.10). *Furthermore, if* $|\Sigma^*/\Phi_a| < \infty$, *there exists the unique minimal representation of* Υ *by a* G_a*-sdp.*

Proof. Φ_a is an equivalence relation by Example 3.10, and Assumption 3.15 holds for G_a. Q.E.D.

The uniqueness of the minimal representation of the above special class of *G-sdp*'s cannot be extended to other classes of *G-sdp*'s such as G_0, G_m, G_s and G_p-*sdp* as demonstrated in [6].

Now we move to the investigation of properties which make Φ an equivalence relation.

DEFINITION 5.3. G is *transitive* if

$$(\forall \xi_1, \xi_2, \xi_3 \in A)(\forall \eta_1, \eta_2, \eta_3 \in A)((\exists g_{12} \in G)(\exists g_{23} \in G)(g_{12}(\xi_1) = \eta_1$$
$$\wedge\ g_{12}(\xi_2) = g_{23}(\xi_2) = \eta_2 \wedge g_{23}(\xi_3) = \eta_3) \Rightarrow (\exists g_{13} \in G)(g_{13}(\xi_1) = \eta_1$$
$$\wedge\ g_{13}(\xi_3) = \eta_3))$$

holds.

Example 5.4. G_0 is not transitive. To prove this, consider the case in which $\xi_1 = \xi_3 \neq \xi_2$ holds and η_1, η_2, η_3 are all distinct. Then $g_{13} \in G$ of Definition 5.3 does not exist, though $g_{12}, g_{23} \in G$ do exist. Similarly, it is possible to prove that G_m, G_s and G_p are not transitive.

Example 5.5. G_a is transitive since $(\exists g_{12}\in G_a)(\exists g_{23}\in G_a)(g_{12}(\xi_1)=\eta_1 \wedge g_{12}(\xi_2) = \eta_2 \wedge g_{23}(\xi_2)=\eta_2 \wedge g_{23}(\xi_3)=\eta_3) \Rightarrow (\xi_1-\xi_2=\eta_1-\eta_2)\wedge(\xi_2-\xi_3=\eta_2-\eta_3) \Rightarrow (\xi_1-\xi_3=\eta_1-\eta_3) \Rightarrow (\exists g_{13}\in G_a)(g_{13}(\xi_1)=\eta_1 \wedge g_{13}(\xi_3)=\eta_3)$.

THEOREM 5.6. *Let Φ be reflexive, and let G be transitive and satisfy*

$$(\forall x, y, z \in \Sigma^*)(x\Phi y \wedge y\Phi z \Rightarrow \Gamma(x,y)\cap\Gamma(y,z)\neq\emptyset).$$

Then Φ is an equivalence relation.

Proof. It is sufficient to prove the transitivity of Φ. If $x\Phi y \wedge y\Phi z$ holds, then there exists $\gamma\in\Gamma(x,y)\cap\Gamma(y,z)$ by assumption, such that

$$(\forall w\in\Sigma^*)(\forall a\in\Sigma)(\exists g_{wa}^{xy}\in G)(g_{wa}^{xy}(\gamma(xw))=\gamma(xwa) \wedge g_{wa}^{xy}(\gamma(yw))=\gamma(ywa)),$$

$$(\forall w\in\Sigma^*)(\forall a\in\Sigma)(\exists g_{wa}^{yz}\in G)(g_{wa}^{yz}(\gamma(yw))=\gamma(ywa) \wedge g_{wa}^{yz}(\gamma(zw))=\gamma(zwa)).$$

Since G is transitive, this implies that

$$(\forall w\in\Sigma^*)(\forall a\in\Sigma)(\exists g_{wa}^{xz}\in G)(g_{wz}^{xz}(\gamma(xw))=\gamma(xwa) \wedge g_{wa}^{xz}(\gamma(zw))=\gamma(zwa)),$$

and hence $x\Phi z$ follows. Q.E.D.

Remark 5.7. The assumption in Theorem 5.6 that

$$(\forall x, y, z\in\Sigma^*)(x\Phi y \wedge y\Phi z \Rightarrow \Gamma(x,y)\cap\Gamma(y,z)\neq\emptyset)$$

holds for most of important G. In particular, if G is a group, this is always guaranteed as proved below.

Proof. Let $x, y, z\in\Sigma^*$ satisfy $x\Phi y \wedge y\Phi z$, i.e.,

$$(\exists\gamma^{xy}\in H_\Upsilon)(\forall w\in\Sigma^*)(\forall a\in\Sigma)(\exists g_{wa}^{xy}\in G)(g_{wa}^{xy}(\gamma^{xy}(xw)) = \gamma^{xy}(xwa) \wedge g_{wa}^{xy}(\gamma^{xy}(yw))=\gamma^{xy}(ywa))$$

$$(\exists\gamma^{yz}\in H_\Upsilon)(\forall w\in\Sigma^*)(\forall a\in\Sigma)(\exists g_{wa}^{yz}\in G)(g_{wa}^{yz}(\gamma^{yz}(yw)) = \gamma^{yz}(ywa) \wedge g_{wa}^{yz}(\gamma^{yz}(zw))=\gamma^{yz}(zwa)),$$

and $xR_S y \wedge yR_S z$ (hence $x\backslash S = y\backslash S = z\backslash S$). First consider the case of $x\backslash S=\emptyset$. Then we can let $g_{wa}^{xy}=g_{wa}^{yz}=g_I$ for any $w\in\Sigma^*$ and $a\in\Sigma$, and $\gamma^{xy}=\gamma^{yz}=\gamma$, where $\gamma(u)=k\in A$ (constant) for all $u\in\Sigma^*$. It is direct to show $\gamma\in\Gamma(x,y)\cap\Gamma(y,z)$. Next consider the case of $x\backslash S\neq\emptyset$. For each $v\in\Sigma^*$, let $u_v\in xv\backslash S$ $(=yv\backslash S = zv\backslash S)$ be defined if $xv\backslash S\neq\emptyset$. If $xv\in S$, u_v is taken to be ε. Furthermore if $xv=yv'$ (or zv'), u_v and $u_{v'}$ satisfy $u_v=u_{v'}$. Define $\gamma:\Sigma^*\to A$ as follows:

$$\gamma(xv)=f(xvu_v)\wedge\gamma(yv)=f(yvu_v)\wedge\gamma(zv)=f(zvu_v)$$

for v satisfying $xv\backslash S(=yv\backslash S=zv\backslash S)\neq\emptyset$, and

$$\gamma(xv)=\gamma(xp)\wedge\gamma(yv)=\gamma(yp)\wedge\gamma(zv)=\gamma(zp)$$

for v satisfying $xv\backslash S=\emptyset$, where p is the longest prefix of v such that $xp\backslash S\neq\emptyset$ holds (p is independent of x, y and z because $x\backslash S = y\backslash S = z\backslash S$). One can arbitrarily define $\gamma(u)$ for $u\in\Sigma^*$, such that none of x, y and z is a prefix of u. Based on this γ, define $h_v^{xy}\in G$ satisfying

$$h_v^{xy}(\gamma^{xy}(xv))=\gamma(xv)\wedge h_v^{xy}(\gamma^{xy}(yv))=\gamma(yv)$$

for each $v \in \Sigma^*$. h_v^{xy} is, for example, given by

$$h_v^{xy} = \begin{cases} g_{vu_v}^{xy} & \text{if } xv \setminus S \neq \varnothing \\ g_{pu_p}^{xy}(g_{pq}^{xy})^{-1} & \text{if } xv \setminus S = \varnothing, \end{cases}$$

where $v = pq$, p, $q \in \Sigma^*$, and p is the one defined above. Here g_{rs}^{xy}, with r, $s \in \Sigma^*$, is given by

$$g_{rs}^{xy} = g_{r_{k-1}a_k}^{xy} \cdots g_{r_1a_2}^{xy} g_{ra_1}^{xy}$$

where $s = a_1a_2 \cdots a_k \wedge r_j = ra_1 \cdots a_j$ for $j = 1, 2, \cdots, k-1$, and $g_{r_ja_i}^{xy}$ is the one given in the definition of γ^{xy} above. (In other words, $g_{rs}^{xy}(\gamma^{xy}(xr)) = \gamma^{xy}(xrs)$ holds.) Since G is a group, $h_v^{xy} \in G$. Now let

$$\bar{g}_{wa}^{xy} = h_{w'}^{xy} g_{wa}^{xy} (h_w^{xy})^{-1}$$

for $w \in \Sigma^*$ and $a \in \Sigma$, where $w' = wa$. Since G is a group, $\bar{g}_{wa}^{xy} \in G$. Then we have that for any $w \in \Sigma^*$ and $a \in \Sigma$,

$$\begin{aligned} \bar{g}_{wa}^{xy}(\gamma(xw)) &= h_{w'}^{xy} g_{wa}^{xy} (h_w^{xy})^{-1}(\gamma(xw)) \\ &= h_{w'}^{xy} g_{wa}^{xy}(\gamma^{xy}(xw)) \\ &= h_{w'}^{xy}(\gamma^{xy}(xwa)) \\ &= \gamma(xwa), \\ \bar{g}_{wa}^{xy}(\gamma(yw)) &= \gamma(ywa). \end{aligned}$$

Thus $\gamma \in \Gamma(x, y)$. In a similar manner, it is possible to prove $\gamma \in \Gamma(y, z)$. Hence $\gamma \in \Gamma(x, y) \cap \Gamma(y, z)$. Q.E.D.

Example 5.8. G_a is a group and transitive (see Example 3.10 and Example 5.5). Since Φ_a is obviously reflexive, Φ_a is an equivalence relation by Theorem 5.6 and Remark 5.7.

Next, we will introduce a class of G whose transitivity can be easily proved. Thus Φ defined for such G is an equivalence relation, and hence Theorem 5.1 may be immediately applied.

DEFINITION 5.9. G is *definable by single point* if

$$(\forall g, h \in G)((\exists \xi \in A)(g(\xi) = h(\xi)) \Rightarrow g = h)$$

holds.

Example 5.10. G_a is definable by single point since $(\forall g, h \in G)((\exists \xi \in A)$ $(g(\xi) = h(\xi)) \Rightarrow (\exists k \in A)(g(\xi) = \xi + k \wedge h(\xi) = \xi + k) \Rightarrow g = h)$.

Example 5.11. Let $G_{\text{mul}} = \{g : E_+ \to E_+ | (\exists k \in E_+)(\forall \xi \in E_+)(g(\xi) = \xi \cdot k)\}$, where $E_+ = \{\xi \in E | \xi > 0\}$. G_{mul} is a group since for any $g(\xi) = \xi \cdot k$, $h(\xi) = \xi \cdot k^{-1}$ satisfies $(\forall \xi \in E_+)(h(g(\xi)) = \xi)$ (i.e., $h = g^{-1}$). Moreover, G_{mul} is definable by single point since $(\forall g, h \in G_{\text{mul}})((\exists \xi \in E_+)(g(\xi) = h(\xi)) \Rightarrow (\exists k \in E_+)(g(\xi) = \xi \cdot k \wedge h(\xi) = \xi \cdot k) \Rightarrow g = h)$ holds.

Example 5.12. Let $G_{\text{pow}} = \{g : E'_+ \to E'_+ | (\exists k \in E_+)(\forall \xi \in E'_+)(g(\xi) = \xi^k)\}$, where $E'_+ = \{\xi \in E | \xi > 0 \wedge \xi \neq 1\}$. G_{pow} is a group since for any $g(\xi) = \xi^k$, $h(\xi) = \xi^{(1/k)}$ satisfies $f(g(\xi)) = \xi$. Moieover, G_{pow} is definable by single point since $(\forall \xi \in E'_+)$ $(\forall k_1, k_2 \in E)(\xi^{k_1} = \xi^{k_2} \Rightarrow k_1 = k_2)$ holds.

Example 5.13. Let $G_{\exp} = \{g : E_+ \to E_+ | (\exists k \in E_+)(\forall \xi \in E_+)(g(\xi) = k^{\xi})\}$. Although $G_{\exp}$ is not a group, $G_{\exp}$ is definable by single point as easily proved.

PROPOSITION 5.14. *If G is definable by single point, then G is pairwise extendible.*

Proof. Let $A_0 \subset A$. Then $(\forall \xi_1, \xi_2, \eta_1, \eta_2 \in A_0)((\exists g_{\xi_1\xi_2} \in G)(\exists g_{\eta_1\eta_2} \in G)(\exists g_{\xi_2\eta_1} \in G)(g_{\xi_2\eta_1}(\xi_2) = g_{\xi_1\xi_2}(\xi_2) \wedge g_{\xi_2\eta_1}(\eta_1) = g_{\eta_1\eta_2}(\eta_1)) \Rightarrow g_{\xi_1\xi_2} = g_{\xi_2\eta_1} \wedge g_{\xi_2\eta_1} = g_{\eta_1\eta_2}$ (since G is definable by single point) $\Rightarrow g_{\xi_1\xi_2} = g_{\eta_1\eta_2})$. Thus g defined by $g = g_{\xi\eta}$, with $\xi, \eta \in A_0$ ($g_{\xi\eta}$ is independent of ξ, η as proved above) belongs to G, and it is an extension of the g^* given in Definition 4.1. Q.E.D.

PROPOSITION 5.15. *If G is definable by single point, then G is transitive.*

Proof. $(\forall \xi_1, \xi_2, \xi_3 \in A)(\forall \eta_1, \eta_2, \eta_3 \in A)((\exists g_{12} \in G)(\exists g_{23} \in G)(g_{12}(\xi_1) = \eta_1 \wedge g_{12}(\xi_2) = \eta_2 \wedge g_{23}(\xi_2) = \eta_2 \wedge g_{23}(\xi_3) = \eta_3) \Rightarrow g_{12} = g_{23}(\equiv g)$ (since G is definable by single point) $\Rightarrow g(\xi_1) = \eta_1 \wedge g(\xi_3) = \eta_3)$. Q.E.D.

THEOREM 5.16. *Let G be definable by single point and G be a group. In addition, let Φ be reflexive. Then Φ is an equivalence relation. Therefore, a ddp Υ is s-representable by a G-sdp if and only if $|\Sigma^*/\Phi| < \infty$. In case of $|\Sigma^*/\Phi| < \infty$, there exists the unique minimal s-representation in the same sense as that of Theorem* 5.1.

Proof. G is transitive by Proposition 5.15. By assumption, G is a group, and furthermore, Φ is reflexive. Thus Φ is an equivalence relation by Theorem 5.6 and Remark 5.7. Since G is pairwise extendible by Proposition 5.14, $\Gamma(T) \neq \emptyset$ holds for any Φ-consistent $T \in \Lambda_F(S)$ by Theorem 4.9. Thus Theorem 5.1 can be directly applied. Q.E.D.

Example 5.17. Theorem 5.16 can be applied to G_a, G_{mul} and G_{pow} (see Examples 2.5, 5.11 and 5.12), since they are definable by single point and groups, respectively.

Example 5.18. $G_{\exp}$ of Example 5.13 is transitive by Proposition 5.15. Although $G_{\exp}$ is not a group, it is possible to prove $(\forall x, y, z \in \Sigma^*)(x\Phi y \wedge y\Phi z \Rightarrow \Gamma(x, y) \cap \Gamma(y, z) \neq \emptyset)$ holds for $G_{\exp}$ (the proof is omitted). Thus Φ is an equivalence relation by Theorem 5.6, and hence Theorem 5.1 is applicable to $G_{\exp}$.

6. Discussion. A comment is given here that the binary relation $\bar{\Phi}$ (abbreviation of $\bar{\Phi}^G_\Upsilon$) defined below may be used instead of Φ in most of the preceding discussion:

$$(\forall x, y \in \Sigma^*)(x\bar{\Phi}y \Leftrightarrow xR_Sy \wedge \bar{\Gamma}(x, y) \neq \emptyset)$$

$$\bar{\Gamma}(x, y) \equiv \{\gamma \in H_\Upsilon | (\forall a \in \Sigma)(\exists g_a \in G)(g_a(\gamma(x)) = \gamma(xa) \wedge g_a(\gamma(y)) = \gamma(ya))\}.$$

This binary relation has the feature that it can be easily checked for each $x, y \in \Sigma^*$, since only values $\gamma(x)$, $\gamma(y)$, $\gamma(xa)$ and $\gamma(ya)$ for $a \in \Sigma$ are relevant. By using an argument similar to the proof of Proposition 3.13, it can be shown that $\bar{\Phi} \wedge T = \Phi \wedge T$ holds for any $\bar{\Phi}$-consistent $T \in \Lambda_F(S)$. ($P \wedge T$ is the binary relation defined by $(\forall x, y \in \Sigma^*)(x(P \wedge T)y \Leftrightarrow xPy \wedge xTy)$.) Based on this, it is possible to prove that the representation theorem, Theorem 3.17, holds even if Φ is replaced by $\bar{\Phi}$. The argument of § 4 may also be applied to $\bar{\Phi}$ with slight modifications.

However, we did not adopt this definition in the preceding sections because then the search for $T \in \Lambda_F(S)$ used in the representation theorem might become more difficult, since the restriction incurred by the relation $x\bar{\Phi}y$ is rather weak compared with that incurred by $x\Phi y$. In other words, if we used $\bar{\Phi}$ instead of Φ, the equivalence relation $T \in \Lambda_F(S)$ satisfying the conditions of the representation theorem must take care of most of properties which make the representation possible, while otherwise the binary relation Φ takes care of the essential part of representation.

Furthermore, the discussion in § 5 cannot be directly applied to $\overline{\Phi}$ since $\overline{\Phi}$ is no longer right invariant.

Finally, it should be noted that although most of representation theorems obtained in the earlier papers [3] and [7] are special cases of Theorem 3.17, there exist many others to which our theory is not readily applicable. Such examples are G_p-*sdp* defined in Example 2.5, and various *G-sdp*'s for which $g \in G$ is a function from Z to Z (integers), as defined in Example 2.6 and in [4]. It is desired to develop a more general theory which can include all these *G-sdp*'s.

Acknowledgment. The author wishes to appreciate Professor H. Mine of Kyoto University for his encouragement and valuable discussion.

REFERENCES

[1] M. DAVIS, *Computability and Unsolvability*, McGraw-Hill, New York, 1958.
[2] J. E. HOPCROFT AND J. D. ULLMAN, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, Mass., 1969.
[3] T. IBARAKI, *Representation theorems for equivalent optimization problems*, Information and Control, 21 (1972), pp. 397–435.
[4] ———, *Classes of discrete optimization problems and their decision problems*, working paper, Dept. of Applied Math. and Physics, Kyoto Univ., 1971.
[5] ———, *Solvable classes of discrete dynamic programming*, J. Math. Anal. Appl., 43 (1973).
[6] ———, *Minimal representations of some classes of dynamic programming*, working paper, Dept. of Applied Math. and Physics, Kyoto Univ., 1972.
[7] R. M. KARP AND M. HELD, *Finite-state processes and dynamic programming*, SIAM J. Appl. Math., 15 (1967), pp. 693–718.
[8] M. RABIN AND D. SCOTT, *Finite automata and their decision problems*, IBM J. Res. Develop., 3 (1959), pp. 115–125.

SIAM J. COMPUT.
Vol. 2, No. 3, September 1973

ENUMERATION OF THE ELEMENTARY CIRCUITS OF A DIRECTED GRAPH*

ROBERT TARJAN†

Abstract. An algorithm to enumerate all the elementary circuits of a directed graph is presented. The algorithm is based on a backtracking procedure of Tiernan, but uses a lookahead and labeling technique to avoid unnecessary work. It has a time bound of $O((V \cdot E)(C + 1))$ when applied to a graph with V vertices, E edges, and C elementary circuits.

Key words. algorithm, backtracking, circuit, cycle, digraph, graph.

There is a class of problems in which the goal is the enumeration of a set of objects associated with a given graph. Examples include enumeration of the elementary circuits, enumeration of the spanning trees, and enumeration of the cliques of a given graph. For each of these three problems, graphs with V vertices may be constructed which have 2^V or more objects to be enumerated. Thus, in general, algorithms to solve these problems must have a running time exponential in the size of the graph. However, in practical applications, the number of objects to be enumerated is usually a much smaller function of the size of the graph. Thus it would be useful to find enumeration algorithms with running times polynomial in the number of objects generated. Presented here is an algorithm for enumerating elementary circuits whose running time is a small-degree polynomial function of the size of its input and output.

A (*directed*) *graph* $G = (\mathscr{V}, \mathscr{E})$ consists of a set of *vertices* $\mathscr{V}$ and a set of ordered pairs of vertices $\mathscr{E}$, called the *edges* of G. If (v, w) is an edge of G, vertices v and w are said to be *adjacent*. A *path* in a graph is a sequence of edges (v_1, v_2), $(v_2, v_3), \cdots, (v_{n-1}, v_n)$, such that the terminal vertex of an edge in the sequence is the initial vertex of the next edge. A path may be denoted by the sequence of vertices on it. An *elementary path* contains no vertex twice. An *elementary circuit* is an elementary path with the exception that its first and last vertices are identical. Two elementary circuits whose edge sequences are cyclic permutations of each other are regarded as identical. For simplicity we shall assume that a graph contains no self-loops (edges of the form (v, v)) and no multiple edges.

We wish to enumerate all the elementary circuits of a given graph. Tiernan [2] presents an algorithm for accomplishing this. His algorithm uses an essentially unconstrained backtracking procedure which explores elementary paths of the graph and checks to see if they are cycles. If the vertices of the graph are numbered from 1 to V, the algorithm will generate all elementary paths $p = (v_1, v_2, \cdots, v_k)$ with $v_1 < v_i$ for all $2 \leqq i \leqq k$, by starting from some vertex v_1, choosing an edge to traverse to some vertex $v_2 > v_1$, and continuing in this way. Whenever no new vertex can be reached, the procedure backs up one vertex and chooses a different edge to traverse. If v_1 is adjacent to v_k, the algorithm lists an elementary cycle $(v_1, v_2, \cdots, v_k, v_1)$. The algorithm enumerates each elementary cycle exactly once, since each such cycle contains a unique smallest vertex v_1 and thus corresponds

* Received by the editors October 17, 1972, and in revised form March 28, 1973.

† Department of Computer Science, Cornell University, Ithaca, New York 14850.

to a unique elementary path with starting vertex v_1. However, the algorithm has a worst-case running time exponential in the size of its output; it may explore many more elementary paths than are necessary. Consider the graph G in Fig. 1. It contains $3n + 1$ vertices, $5n$ edges and $2n$ elementary circuits. However, G contains 2^n elementary paths from vertex 1 to vertex $3n + 1$, all of which will be generated by Tiernan's algorithm. Thus the worst-case time bound of the algorithm is exponential in the number of elementary circuits, as well as exponential in the size of the graph.

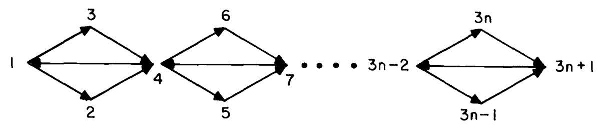


FIG. 1. *An example showing the inefficiency of Tiernan's algorithm*

Weinblatt [3] gives an algorithm for finding elementary circuits which is related to Tiernan's, but which requires substantially more bookkeeping. The algorithm has the property that it examines each edge of the graph exactly once. Given a graph G, we start from some vertex and choose an edge to traverse. We continue the search at each step by selecting an *unexplored* edge leading from the vertex most recently reached which still has unexplored edges. Eventually each edge of the graph will be traversed. Such a search is easy to program, because the set of old vertices with possibly unexplored edges may be stored on a stack. (See [1], for instance.) This sequence of vertices is an elementary path from the initial vertex to the vertex currently being examined. (Weinblatt calls it the TT, or "trial thread".) Whenever we traverse an edge leading to a vertex already on the stack, we have found a new elementary circuit, corresponding to a sequence of vertices on top of the stack. Whenever we traverse an edge leading to an old vertex which is *not* currently on the stack, some portion of the stack plus a sequence of subpaths from circuits already found may form a new elementary circuit. Weinblatt uses a recursive backtracking procedure to test combinations of subpaths from old circuits to see if they give new circuits in this way.

Although Weinblatt's algorithm is often much more efficient than Tiernan's, the recursive backtracking procedure requires exponential time in the worst case. For example, consider the graph in Fig. 2. It contains $3n + 2$ vertices, $5n + 3$ edges and $2n + 2$ elementary circuits. Suppose we start Weinblatt's algorithm by exploring the edge $(0, 1)$. Then the algorithm will generate all circuits of the form $(3i - 2, 3i + 1, 3i, 3i - 2)$ and $(3i - 2, 3i + 1, 3i - 1, 3i - 2)$ rapidly. Eventually the algorithm will traverse edge $(0, 3n + 1)$. Then Weinblatt's recursive procedure will attempt to find an elementary path back to vertex 0 by combining parts of old circuits. The recursive backtracking will require an exponential amount of time but will produce only one new circuit, i.e., $(0, 3n + 1, 0)$. Thus Weinblatt's algorithm does not have a running time polynomial in the number of circuits.

However, it *is* possible to construct a polynomial-time algorithm for the circuit enumeration problem. Such an algorithm uses Tiernan's backtracking

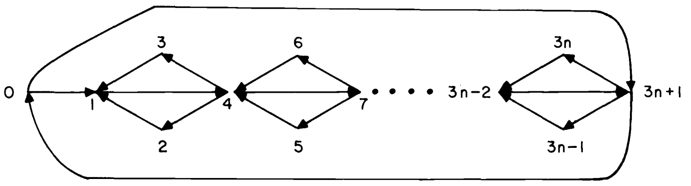


Fig. 2. *An example showing the inefficiency of Weinblatt's algorithm*

procedure restricted so that only fruitful paths are explored. The circuit enumeration algorithm is presented below in ALGOL-like notation. The algorithm assumes that the vertices of the graph are numbered from 1 to V, and that the graph is represented by a set of adjacency lists, one for each vertex. The adjacency list $A(v)$ of vertex v contains all vertices w such that (v, w) is an edge of the graph. The point stack used in the algorithm denotes the elementary path p currently being considered; the elementary path has start vertex s. Every vertex v on such a path must satisfy $v \geqq s$. A vertex v becomes *marked* when it lies on the current elementary path p. As long as v lies on the current elementary path or it is known that every path leading from v to s intersects p at a point other than s, v stays marked.

For each vertex s, the algorithm generates elementary paths which start at s and contain no vertex smaller than s. Once a vertex v has been used on a path, it can only be used to extend a new path when it has been deleted from the point stack *and* when it becomes unmarked. A vertex v becomes unmarked when it might lie on a simple circuit which is an extension of the current elementary path. Whenever the last vertex on an elementary path is adjacent to the start vertex s, the elementary path corresponds to an elementary circuit which is enumerated. The marking procedure is the key to avoiding the unnecessary searches which may occur when using Weinblatt's algorithm or Tiernan's original algorithm.

ALGORITHM.

```
procedure circuit enumeration;
    begin
        procedure BACKTRACK (integer value $v$, logical result $f$);
            begin
                logical $g$;
                $f$:= false;
                place $v$ on point stack;
                mark ($v$) := true:
                place $v$ on marked stack;
                for $w \in A(v)$ do
                    if $w < s$ then delete $w$ from $A(v)$
                    else if $w = s$ then
                        begin
                            output circuit from $s$ to $v$ to $s$ given by point
                            stack;
                            $f$ := true;
                        end
                    else if ¬ mark ($w$) then
                        begin
                            BACKTRACK ($w$, $g$);
                            $f := f \vee g$;
                        end;
                comment $f$ = true if an elementary circuit containing the
                    partial path on the stack has been found;
                if $f$ = true then
                    begin
                    a: while top of marked stack $\neq v$ do
                            begin
                                $u$ := top of marked stack;
                                delete $u$ from marked stack;
                                mark ($u$) := false;
                            end;
                        delete $v$ from marked stack;
                        mark ($v$) := false;
                    end;
                delete $v$ from point stack;
            end;
        integer $n$;
        for $i$ := 1 until $V$ do mark ($i$) := false;
        for $s$ := 1 until $V$ do
            begin
            b: BACKTRACK ($s$, flag);
                while marked stack not empty do
                    begin
                        $u$ := top of marked stack
                        mark ($u$) := false;
                        delete $u$ from marked stack;
                    end;
            end;
    end;
```

LEMMA 1. *Let $c = (v_1, v_2, \cdots, v_n, v_1)$ be an elementary circuit in a graph G, satisfying $v_i > v_1$ for $2 \leqq i \leqq n$. Let the circuit enumeration procedure be applied to G. Consider the execution of statement b:* BACKTRACK (s, flag), *with $s = v_1$. During the execution of this statement (which may involve recursive calls on* BACKTRACK), *we have: for all $1 \leqq k \leqq n$, if v_k is marked, then for some $j \geqq k$, v_j is on the point stack.*

Proof. We prove the lemma by induction on k, with k decreasing. Let $k = n$. Suppose v_n becomes marked. Then it is added to the point stack at the same time. Before v_n is removed from the point stack, variable f becomes true, since (v_n, v_1) is an edge of the graph. Thus v_n will become unmarked when it is deleted from the stack, and the lemma is true for $k = n$.

Let the lemma be true for v_i with $i > k$. We prove the lemma for v_k. Suppose v_k becomes marked. Then v_k is placed on the point stack. There are two cases:

(a) Vertex v_k is placed on top of some v_j on the point stack with $j > k$. Then v_j is also underneath v_k on the marked stack. Thus if v_k is marked, v_j is also marked. But if v_j is marked, there is some v_l with $l \geqq j$ on the point stack by the induction hypothesis, and since $l \geqq j > k$, the lemma holds for v_k.

(b) Vertex v_k is not placed on top of any v_j on the point stack with $j > k$. Then when the edge (v_k, v_{k+1}) is examined, v_{k+1} is unmarked by the induction hypothesis, and will be added to the stack. Subsequently $v_{k+2}, v_{k+3}, \cdots, v_n$ will be added to the stack, and an elementary circuit containing v_k will be found. A flag is set true to note this discovery. The flag filters up through recursive returns from BACKTRACK, and v_k will be unmarked by step *a* when v_k is removed from the point stack. Thus the lemma holds for v_k.

By induction, the lemma holds in general.

LEMMA 2. *The circuit enumeration algorithm lists each elementary circuit of a given graph exactly once.*

Proof. The starting vertex of any elementary path *p* generated by the algorithm is the lowest numbered vertex on the path *p*. Since the algorithm generates an elementary path at most once, and since an elementary circuit has only one lowest numbered vertex, each elementary circuit is generated at most once.

Let $c = (v_1, v_2, \cdots, v_n, v_1)$ be an elementary circuit. Let the circuit enumeration procedure be applied to G. Consider the execution of step *b*: BACKTRACK (s, flag) with $s = v_1$. If $(v_1, \cdots, v_k)$ is on the point stack for any $1 \leqq k \leqq n$, then by Lemma 1, v_{k+1} must be unmarked, and v_{k+1} will be added to the point stack on top of v_k when edge (v_k, v_{k+1}) is examined. By induction, $(v_1, v_2, \cdots, v_n)$ will eventually be on the point stack, and the algorithm will enumerate the circuit c. Thus each elementary circuit is generated at least once.

LEMMA 3. *If G is a graph with V vertices and E edges, applying the circuit enumeration algorithm to G requires $O(V + E + S)$ space, where S is the sum of the lengths of all the elementary circuits, and $O(VE(C + 1))$ time, where C is the number of elementary circuits.*

Proof. The space bound is obvious; storage of the graph's adjacency lists requires $O(V + E)$ space, storage for the algorithm's data structures requires $O(V)$ space, and storage for the output requires $O(S)$ space. If we do not want to store all the elementary circuits after they are generated the algorithm requires only $O(V + E)$ space.

The time bound follows from the following observation: after a circuit c is enumerated, but before another circuit with the same start vertex is enumerated, any vertex can become unmarked at most V times (at most once for each vertex on the point stack when c is enumerated). Thus after c is enumerated but before another circuit with the same start vertex is enumerated, any edge can be explored at most V times. After the start vertex has changed but before any new circuits are enumerated, any edge can be explored only once, since no vertices become unmarked during this time. Thus the algorithm requires $O(E)$ time for each start vertex plus $O(VE)$ time for each circuit, for a total time of $O(VE(C + 1))$. In terms of V, E and S, the time bound is $O(E(S + V))$.

The variation of Tiernan's algorithm presented here has a running time polynomial in the size of the input and output, a fact not true in general for either Tiernan's original circuit generation algorithm or for Weinblatt's algorithm. However, the number of circuits may be exponential in the number of vertices. On graphs with very large numbers of circuits, all three algorithms run in time exponential in the size of the problem graph. On graphs with simple looping structure, Weinblatt's algorithm may run faster than the one presented here by a factor of at most V, but the new algorithm may be speeded up in this case if a little preprocessing is done. Tiernan's original algorithm is slightly simpler to program than the version presented here, but by avoiding the labeling process it may perform many unnecessary searches. The new version has a provably reasonable time bound on all types of graphs.

The new algorithm was implemented in ALGOL W, the Stanford University version of ALGOL (see [4]), and run on a variety of sample graphs using an IBM 360/65. The program counted 125,664 circuits in a nine-vertex complete graph in 101.2 seconds. The running time of the new algorithm was faster than that claimed by either Tiernan or Weinblatt, although they used slower computers (B-5500 and an IBM 7094, respectively). The new algorithm is certainly competitive in practice. It is still an open question whether a circuit enumeration algorithm exists whose time bound is *linear* in the size of its input and output.

Acknowledgments. The author wishes to thank the referees and Professors Lloyd Fosdick and Leon Osterweil of the University of Colorado for their discovery of errors in the original formulation of the algorithm, and Professor John Hopcroft of Cornell University, who helped verify the corrections.

REFERENCES

[1] R. TARJAN, *Depth-first search and linear graph algorithms*, this Journal, 1 (1972), pp. 146–160.

[2] J. C. TIERNAN, *An efficient search algorithm to find the elementary circuits of a graph*, Comm. ACM, 13 (1970), pp. 722–726.

[3] H. WEINBLATT, *A new search algorithm for finding the simple cycles of a finite directed graph*, J. Assoc. Comput. Mach., 19 (1972), pp. 43–56.

[4] R. L. SITES, *Algol W Reference Manual*, Tech. Rep. STAN-CS-71-230, Computer Science Dept., Stanford University, Stanford, California, August, 1971.

SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

TOWARD CHARACTERIZATION OF PERFECT ELIMINATION DIGRAPHS*

LOREN HASKINS† AND DONALD J. ROSE‡

Abstract. Perfect elimination digraphs arise in the study of Gaussian elimination on sparse linear systems. With a view toward numerical computational complexity we show four conditions (C1–C4) to be necessary for the perfect elimination property. The sufficiency of C1 is shown in general and the sufficiency of C2–C4 is shown in the symmetric case. The equivalence of C1–C3 is conjectured.

Key words. sparse linear systems, Gaussian elimination, perfect elimination matrices, directed graphs, paths, separators

1. Introduction. Perfect elimination digraphs arise in the graph-theoretic study of the numerical solution of sparse linear systems by Gaussian elimination. Our investigation is an extension and generalization of earlier work by Rose [4], [5] who considers symmetric linear systems and, hence, undirected graphs. In addition to providing insight into the combinatorial nature of the algebraic process of Gaussian elimination on sparse systems, such a graph-theoretic approach appears useful in providing bounds on the numerical computational complexity of solving such systems (see Hoffman et al.[3]).

In § 2 we define perfect elimination digraphs and relate them to the algebraic process of elimination. This brief discussion is analogous to the more extensive presentation given in [5] for symmetric systems. A further exposition of elimination methods for numerically solving linear systems is available in Forsythe and Moler [2].

Section 3 contains some connectivity considerations which we use in § 4. In particular, we develop and examine the notions of minimal x, y paths and vertex separators. When the digraph can be regarded as an undirected graph, this notion of separation becomes the usual notion; however, our development avoids any generalized notion of components of a graph with respect to a separator.

In § 4 we examine four conditions (C1–C4) on digraphs and relate them to the perfect elimination condition. These conditions are shown (or previously known) to be equivalent when the graph is undirected. The perfect elimination property and C1 are conditions requiring the existence of special orderings of the vertex set, while conditions C2–C4 are global conditions on paths and separators. We show that the perfect elimination condition and C1 are equivalent and that C2–C4 are necessary (but C4 is not sufficient) for perfect elimination. Finally we conjecture that the perfect elimination property and C1–C3 are equivalent when G is strongly connected.

2. Motivation and preliminaries. We consider the numerical solution of the linear system

$$Mx = b, \tag{2.1}$$

* Received by the editors September 25, 1972, and in revised form May 14, 1973. This research was supported by the Office of Naval Research under Contract N00014-67-A-0394-0004.

† Department of Mathematics, University of Denver, Denver, Colorado 80210.

‡ Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138.

where M is $n \times n$ and sparse, and PMP^T has a stable (with respect to rounding error) LU factorization for any $n \times n$ permutation matrix P. That is, for any fixed P, $PMP^T = LU$ where $L = (l_{ij})$ and $U = (u_{ij})$ are respectively *unique* unit lower triangular and upper triangular $n \times n$ matrices. Let $C = (c_{ij})$ be the $n \times n$ matrix (depending on P) defined by

$$\begin{aligned} c_{ij} &= l_{ij}, \qquad i > j, \\ c_{ij} &= u_{ij}, \qquad i \leqq j. \end{aligned} \tag{2.2}$$

Comparing the zero-nonzero structure of C with that of PMP^T gives the extent of *fill in* caused by the factorization. We wish to study combinatorially those matrices M for which there exists a P such that $B = PMP^T = (b_{ij})$ has

$$b_{ij} = 0 \Leftrightarrow c_{ij} = 0 \tag{2.3}$$

(disregarding accidentally created zeros).

Algebraically, the $M = LU$ factorization proceeds as follows. Let M be written as

$$M = \begin{bmatrix} a & r^T \\ c & \overline{M} \end{bmatrix}$$

where a is scalar, r and c are $(n-1) \times 1$ and $\overline{M}$ is $(n-1) \times (n-1)$. Then the first "elimination" step of the LU factorization of M may be written as

$$M = M^{(1)} = \begin{bmatrix} 1 & 0 \\ c/a & I \end{bmatrix} \begin{bmatrix} a & r^T \\ 0 & \overline{M} - cr^T/a \end{bmatrix} = L_1 U_1. \tag{2.4}$$

One then proceeds recursively, and supposing the LU factorization of $M^{(2)} = \overline{M} - (cr^T/a)$ is $M^{(2)} = L_2 U_2$, we obtain the LU factorization of M as

$$M = L_1 \begin{bmatrix} 1 & 0 \\ 0 & L_2 \end{bmatrix} \begin{bmatrix} a & r^T \\ 0 & U_2 \end{bmatrix} .$$

Given M as in (2.1) we define the directed graph of M as the pair $G(M) = (X, A)$, where vertex $x_i \in X$ is associated with row i of M and arc $a = (x_i, x_j) \in A$ if and only if $m_{ij} \neq 0$ and $i \neq j$. Here the vertices of X are regarded as ordered; i.e., $X = \{x_i\}_{i=1}^n$. $G(M)$ with unordered vertices represents the equivalence class PMP^T.

To interpret (2.4) graph-theoretically, we proceed as follows. Given $G(M)$, the elimination graph G_y is obtained from G by deleting y and its incident arcs and adding an arc (x, z) whenever there exists a directed x, z path of length 2 containing y. That is, formally,

$$G_y = (X - y, A(X - y) \cup \tau_y), \tag{2.5}$$

where for $V \subset X$,

$$A(V) = \{(x, y) \in A | x, y \in V\} \tag{2.6}$$

and

$$\tau_y = \{(x, z) | (x, y), (y, z) \in A, (x, z) \notin A\}. \tag{2.7}$$

The sequence of elimination graphs, $G_1, G_2, \cdots, G_{n-1}$, corresponding to the ordering implicit in (2.4), is defined recursively by $G_1 = G_{x_1}$ and $G_i = (G_{i-1})_{x_i}$, $2 \leqq i \leqq n - 1$, the corresponding sets τ_i of (2.7) being defined in terms of the adjacency relation in G_{i-1}. Disregarding accidentally created zeros due to exact cancellation, we see that G_1 is the digraph of $M^{(2)}$, G_{i-1} is the digraph of $M^{(i)}$ as the elimination proceeds, and τ_i in G_{i-1} is the set of fill in arcs at the ith step of elimination. The digraph

$$G' = \left(X, A \cup \left(\bigcup_{i=1}^{n-1} \tau_i\right)\right) \tag{2.8}$$

is the graph of C of (2.2).

We will call M a *perfect elimination matrix* if there exists a permutation matrix P such that $B = PMP^T$ and

$$b_{ij} \neq 0 \quad \text{and} \quad b_{ki} \neq 0 \Rightarrow b_{kj} \neq 0, \tag{2.9}$$

for $1 \leqq i < j \leqq n$ and $1 \leqq i < k \leqq n$. It is easy to see that (2.9) holds if and only if (2.3) holds, and in this case the sets τ_i of (2.8) have $\tau_i = \varnothing$. We then call $G(M)$ a *perfect elimination digraph.* We note that the digraph G' of (2.8) is always a perfect elimination digraph, hence studying such graphs provides information on the numerical computational complexity of the elimination process.

Finally we mention that as a consequence of algebraic and complexity considerations, it is appropriate to restrict attention to matrices M such that $G(M)$ is strongly connected (Bunch and Rose [1]). Figure 3b (§ 4) indicates that such a restriction arises combinatorially.

3. Paths and separators. Let $G = (X, A)$ be a finite directed graph (digraph) with vertex set X and arc set A of ordered pairs of distinct vertices. An x, y *path*, p, in G of length $n - 1$ is a sequence of vertices $p = (x = z_1, z_2, \cdots, z_n = y)$ such that $(z_i, z_{i+1}) \in A$ and $z_i \neq z_j$ for $i \neq j$, except possibly for $x = y$, and then p is a *cycle.* A *trivial path* is a path containing only two distinct vertices, hence both (x, y) and (x, y, x) are trivial. A *minimal* x, y path is an x, y path containing no proper subpath, i.e., no proper x, y subsequence which is a path.

Let $x, y \in X$ be not necessarily distinct. A subset $S \subset X$ is an x, y *separator* if $x, y \notin S$ and there exists a nontrivial minimal x, y path in G (implying $(x, y) \notin A$), but no such path exists in $G(X - S) = (X - S, A(X - S))$, $A(X - S)$ defined in (2.6). A *minimal* x, y separator contains no proper subset which is an x, y separator. A vertex z *separates* x, y if z is contained in a minimal x, y or y, x separator.

Consider the following example (Fig. 1).

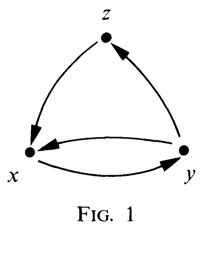


FIG. 1

The path (z, x, y, z) is a nontrivial minimal path, but (x, y, z, x) is not a minimal path since it contains the trivial subpath (x, y, x). The set $\{x, y\}$ is a z, z separator and each of $\{x\}$, $\{y\}$ are minimal z, z separators. There can be no y, y separator since there is no *nontrivial* minimal path from y to y in G. The set $\{y\}$ is a minimal x, z separator, but the set $\{z\}$ is not a y, x separator.

If p is a nontrivial minimal x, y path and S is any minimal x, y separator, then some vertex $u \in S$ must be in p; otherwise p exists in $G(X - S)$. Also, for any $u \in S$ there exists a nontrivial minimal x, y path p containing u but no other $v \in S$; otherwise S is not minimal, $S - u$ being a separator. Hence, summarizing our definitions, we have the following proposition.

PROPOSITION 1. *Let S be a minimal x, y separator and $u \in S$. Every nontrivial minimal x, y path contains at least one element of S, and at least one such path contains u but no other $v \in S$.*

Note that if $p = (x = z_1, z_2, \cdots, z_n = y)$ is any particular x, y path, there may be other x, y paths on the vertex set $\{z_i\}$, and there may be several minimal x, y paths on subsets of $\{z_i\}$. However, when p itself is a minimal x, y path we have the next proposition.

PROPOSITION 2. *If $p = (x = z_1, z_2, \cdots, z_n = y)$ is a minimal x, y path, p is the only x, y path on a subset of $\{z_i\}$.*

Proof. Suppose $p' = (x = z_{i_1}, z_{i_2}, \cdots, z_{i_m} = y)$, $m \leqq n$, is another path on a subset of $\{z_i\}$. If $z_{i_{m-1}} \neq z_{n-1}$, p could not be minimal, there being an arc (z_j, z_n), $j < n$, in A. The result now follows by induction on n since $q = (x = z_1, z_2, \cdots, z_{n-1})$ is a minimal x, z_{n-1} path and the case $n = 2$ is immediate. ■

PROPOSITION 3. *A vertex z is contained in a nontrivial minimal x, y path $(z \neq x, y)$ if and only if z is contained in some minimal x, y separator.*

Proof. If $z \in S$, a minimal x, y separator, the result follows from Proposition 1.

Let $z \in p_0$, a nontrivial minimal x, y path. If $G(X - z)$ contains no such x, y path, z itself is a minimal x, y separator, so let $P = \{p_i\}$ be the set of such paths in $G(X - z)$. For any path p_i there must be a vertex $v_i \in p_i$ such that $v_i \notin p_0$ by Proposition 2. The set $S = \{z, \bigcup_i v_i\}$ certainly separates x and y in G. Let $S_0 \subseteq S$ be a minimal x, y separator. Then $z \in S_0$, since z is the only vertex of p_0 in S_0. ■

Recall that we are not requiring x, y to be distinct. In the case $x = y$ we have the following result.

COROLLARY. *If y separates x, x, then there exists a z such that* (i) *y separates x, z;* (ii) *z separates x, y; and* (iii) *z separates x, x.*

Proof. If y separates x, x, then y is contained in some minimal x, x separator, and, by Proposition 3, y must be contained in a *nontrivial* minimal path (cycle) containing at least three vertices. Hence a z exists on the path and the result follows by reapplying Proposition 3. ■

Let $G = (X, A)$ be a digraph and $x \in X$. We define

$$H_1 = G(X - x),$$

$$H_2 = G_x, \text{ the } x\text{-elimination graph of (2.5)},$$

$$H_3 = (X, A \cup \tau_x), \ \tau_x \text{ as in (2.7)}.$$

We relate separators in H_1, H_2, H_3 to those in G in the following proposition.

PROPOSITION 4. *If w separates y, z in any H_i, $1 \leqq i \leqq 3$, then w separates y, z in G.*

Proof. If w separates y, z in any digraph G', there exists (Proposition 3) a nontrivial minimal y, z path $p = (y = u_1, u_2, \cdots, w = u_k, \cdots, u_n = z)$. If $G' = H_1$, the result is obvious.

Taking $G' = H_3$, we note that there may be arcs (u_i, u_{i+1}) of p which are not in G since such arcs are in τ_x. There may, however, be only one such arc, since if (u_i, u_{i+1}) and (u_j, u_{j+1}), $j \geqq i + 1$, are both in τ_x, then $(u_i, u_{j+1}) \in A \cup \tau_x$, contradicting the minimality of p. So if p is not a path in G the sequence

$$p' = (u_1, u_2, \cdots, u_i, x, u_{i+1}, \cdots, u_n)$$

is a path in G, (u_i, u_{i+1}) being in τ_x.

Path p' is nontrivial in G since p is nontrivial in G'. Furthermore, no arc (u_j, u_l), $l > j + 1$, is in A since p is minimal and such an arc would be in G'. Finally, no arcs (u_j, x), $j < i$, or (x, u_k), $k > i + 1$, are in A; otherwise (u_j, u_{i+1}) or (u_i, u_k) are in G', and p could not be minimal. Thus p' is minimal and contains w, which separates y, z in G (Proposition 3).

Noting that $H_2 = H_3(X - x)$, we have from above that if w separates y, z in $G' = H_3(X - x)$, it separates y, z in H_3 and hence in G. ■

We conclude this section with some remarks.

If $(x, y) \in A$, there are clearly no nontrivial minimal paths from x to y. If $(x, y) \notin A$ and x, y are *distinct*, then any path from x to y contains a nontrivial minimal subpath from x to y, and in this case, if S is a minimal x, y separator, then no paths exist from x to y in $G(X - S)$. However, the digraph of Fig. 2 shows that S may be a minimal x, x separator, and yet paths exist from x to x in $G(X - S)$.

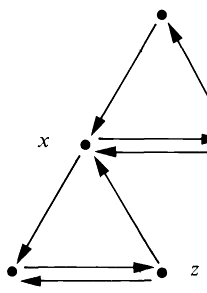


FIG. 2. *The set $\{z\}$ is a minimal x, x separator, but trivial minimal paths exist in $G(X - z)$*

If G is undirected and we regard each edge xy as a pair of arcs (x, y) and (y, x), there never exist any z which separate x, x (since there exist minimal trivial subpaths). Our notion of separation then reduces to the usual notion for undirected graphs.

4. Perfect elimination digraphs. Recall that a digraph $G = (X, A)$ is a *perfect elimination digraph* if and only if X can be ordered, $X = \{x_i\}$, such that the set τ_i of (2.7) in the elimination graph G_{i-1} has $\tau_i = \varnothing$ for $1 \leqq i \leqq n - 1$. We study the following conditions on G where vertices x, y are not necessarily distinct and vertices u, v are not necessarily distinct.

Condition C1 (ordering, separation condition). There exists a bijection $\alpha : X \leftrightarrow \{1, 2, \cdots, |X|\}$ such that for all $x, y \in X$ at least one of x, y, say x, is such that $\alpha(x) < \alpha(z)$ for all $z \in X$ which separate x and y.

Condition C2 (antisymmetric separation condition). For all $x, y \in X$, at least one of x, y, say x, is such that for all u, v which separate x, y, x does not separate u, v.

Condition C3 (chorded path condition). For all $x, y \in X$, at least one of x, y, say x, is such that for all u, v which separate x, y every set V of $n > 2$ vertices contains a subset W of $n - 1$ vertices such that any path from u to v through x whose elements are exactly those of V has a subpath from u to v whose elements are exactly those of W.

Condition C4 (chorded cycle condition). For any set V of $n > 2$ vertices there exists a subset W of $n - 1$ vertices such that any cycle on V has a subcycle on W.

Condition C3 includes the apparently weaker statement that every nontrivial path from u to v through x has a subpath from u to v on all but one of its vertices. Condition C3 says that a fixed vertex is avoided by some subpath of every path on the vertex of any given path from u to v through x.

Consider, momentarily, G to be an undirected graph or, equivalently, a directed graph such that $(x, y) \in A$ if and only if $(y, x) \in A$. An undirected graph is *triangulated* if every cycle on n vertices (note there are two directed cycles) contains a chord joining nonconsecutive vertices. Here a chord from p to q of the directed cycle pair is an arc pair (p, q) and (q, p). As shown in Rose [4], a triangulated graph is a perfect elimination graph and a perfect elimination graph is triangulated. Such graphs are also characterized by the property that every minimal x, y separator is a clique (Rose [4]). In this case we have the following theorem.

THEOREM 1. *Let $G = (X, A)$ be an undirected graph. Then the following are equivalent.*

(i) *G is triangulated.*
(ii) *G satisfies* C2.
(iii) *G satisfies* C3.

Proof. Since (in this case) G has the perfect elimination property, and we shall show below, generally, that this property implies C2 and C3, we need only show that C2 implies (i) and C3 implies (i). We will show that every minimal x, y separator, S, is a clique. To prove this we could, since G is undirected, make use of the x and y connected components of $G(X - S)$, but we prefer to proceed somewhat differently.

Let distinct $x, y \in X$ and distinct $u, v \in S$ be chosen, where S is a minimal x, y separator. Since G is undirected, S is also a minimal y, x separator, and by Propositions 1 and 3 there exist minimal paths

$$p_1 = [w_1 = y, w_2, \cdots, w_k = u, \cdots, w_n = x],$$

containing only $u \in S$, and

$$p_2 = [z_1 = x, z_2, \cdots, z_l = v, \cdots, z_m = y],$$

containing only $v \in S$. Notice that no w_i is adjacent or equal to a z_j for $i < k$ and

$j < l$ or for $i > k$ and $j > l$, for then the sets $\{w_i\}$ and $\{z_j\}$ must contain another vertex of S, contradicting the construction of p_1 and p_2. Let

$$r = \min\{i : i \geqq k \text{ and } w_i \text{ is adjacent to } z_j \text{ for } j \leqq l\}.$$

Let

$$t = \max\{i : i \leqq l \text{ and } w_r \text{ is adjacent to } z_i\}.$$

If u, v are not adjacent, then at least one of w_r, $z_t = x'$ which separates u, v. Let

$$c = \min\{i : i \geqq l \text{ and } z_i \text{ is adjacent to } w_j \text{ for } j \leqq k\}.$$

Let

$$d = \max\{i : i \leqq k \text{ and } z_c \text{ is adjacent to } w_i\}.$$

Again, at least one of w_d, z_c is equal to y', which separates u, v whenever u, v are not adjacent. But u, v separate x', y' by construction, which would contradict both C2 and C3 if u, v were not adjacent. Therefore S is a clique. ■

We now establish some general relations between conditions C1–C4.

THEOREM 2. *$G = (X, A)$ is a perfect elimination digraph if and only if G satisfies* C1.

Proof. The "if" part of the proof proceeds by induction on $|X|$, the case $|X| = 1$ being trivial. Let G satisfy C1 and $w \in X$ have $\alpha(w) = 1$. Note that $w \notin S$, S being any minimal x, y separator, otherwise $\alpha(x), \alpha(y) < 1$. Hence if (x, w) and (w, y) are in A, then $(x, y) \in A$, w not separating x, y.

The subgraph $G(X - w)$ satisfies C1, since if u separates x, y in $G(X - w)$, it separates x, y in G (Proposition 4). Hence by induction, $G(X - w)$ is a perfect elimination digraph and so is G.

Conversely, suppose G is a perfect elimination digraph, and let α assign to each vertex its elimination order. Let z separate x and y; then z is contained in some minimal x, y path p. Hence either $\alpha(x) < \alpha(z)$ or $\alpha(y) < \alpha(z)$, otherwise p could not be minimal. ■

THEOREM 3. *The following relations are valid for a digraph $G = (X, A)$*:

(i) C1 *implies* C3.

(ii) C3 *implies* C2.

(iii) C3 *implies* C4.

Proof. C1 *implies* C3. Let α be the ordering, and u and v separate x and y with $\alpha(x) < \alpha(y)$. Let V be the set of vertices in $p = (u = z_1, z_2, \cdots, z_k = x, \cdots, z_n = v)$, a nontrivial u, v path, and let $z_l \in p$ be such that $\alpha(z_l) \leqq \alpha(z)$ for all $z \in p$. Note that $z_l \neq u, v$ since $\alpha(x) < \alpha(u), \alpha(v)$.

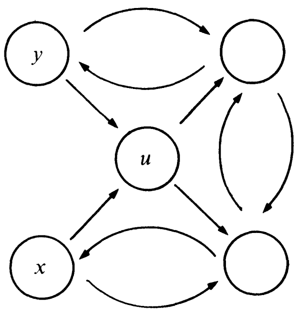
Let p' be any path from u to v whose vertices are exactly those of V. Say $p' = [u = z_1, z_{m_2}, \cdots, z_{m_t} = z_l, \cdots, z_n = v]$. Now since $\alpha(z_l) < \alpha(z_{m_{t-1}}), \alpha(z_{m_{t+1}})$, z_l cannot separate $z_{m_{t-1}}, z_{m_{t+1}}$ and $W = V - \{z_l\}$ is the desired set.

C3 *implies* C2. Let u, v separate x, y. Then one of x, y, say x, is such that every nontrivial u, v path through x has a proper subpath; furthermore we may assume this subpath does not contain x. Hence x does not separate u, v, being on no minimal u, v path.

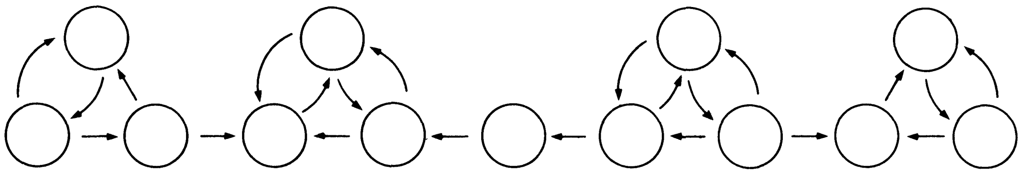
C3 *implies* C4. Let V be a set on $n > 2$ vertices. Suppose there exists u, x, $y \in V$ such that u separates x, y, x, y not necessarily distinct. Now any cycle on V

is a u, u path through both x, y and our conclusion is immediate from C3. Suppose there exists a cycle on V, but no u, x, y in V are such that u separates x, y. At least one path p exists from x to y for all x, y in V such that the vertices of p are all in V. Then $(x, y) \in A$ for all x, y, and a complete undirected subgraph exists on V; the conclusion follows. ■

The digraph in Fig. 3a shows that C4 is not sufficient for the perfect elimination property. Note that u separates x and y, but both x and y separate u, u. Figure 3b shows that neither C2 nor C3 is sufficient for the elimination property if the digraph fails to be strongly connected.



a



b

FIG. 3

We conclude with the following conjecture.

Conjecture. In a strongly connected digraph $G = (X, A)$ the perfect elimination condition, and conditions C1, C2 and C3 are equivalent.

REFERENCES

[1] J. R. BUNCH AND D. J. ROSE, *The role of partitioning in the numerical solution of sparse systems*, Sparse Matrices and Their Applications, Plenum Press, N.Y., 1972.

[2] G. E. FORSYTHE AND C. B. MOLER, *Computer Solution of Linear Algebraic Systems*, Prentice-Hall, N.Y., 1967.

[3] A. J. HOFFMAN, M. S. MARTIN AND D. J. ROSE, *Complexity bounds for regular finite difference and finite element grids*, *SIAM J. Numer. Anal.*, 10 (1973), pp. 364–369.

[4] D. R. ROSE, *Triangulated graphs and the elimination process*, J. Math. Anal. Appl., 32 (1970), pp. 597–609.

[5] ———, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, Graph Theory and Computing, Academic Press, N.Y., 1972.

SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

AN $n^{5/2}$ ALGORITHM FOR MAXIMUM MATCHINGS IN BIPARTITE GRAPHS*

JOHN E. HOPCROFT† AND RICHARD M. KARP‡

Abstract. The present paper shows how to construct a maximum matching in a bipartite graph with n vertices and m edges in a number of computation steps proportional to $(m + n)\sqrt{n}$.

Key words. algorithm, algorithmic analysis, bipartite graphs, computational complexity, graphs, matching

1. Introduction. Suppose we are given a rectangular array in which each cell is designated as "occupied" or "unoccupied". A set of cells is *independent* if no two of the cells lie in the same row or column. Our object is to construct an independent set of occupied cells having maximum cardinality.

In one interpretation, the rows of the array represent boys, and the columns represent girls. Cell i, j is occupied if boy i and girl j are compatible, and we wish to match a maximum number of compatible couples.

An alternate statement of the problem is obtained by representing the rows and columns of the array as the vertices of a bipartite graph. The vertices corresponding to row i and column j are joined by an edge if and only if cell i, j is occupied. We then seek a maximum matching; i.e., a maximum number of edges, no two of which meet at a common vertex.

This problem has a wide variety of applications ([3], [4], [5]). These include the determination of chain decompositions in partially ordered sets, of coset representatives in groups, of systems of distinct representatives, and of block-triangular decompositions of sparse matrices. The problem also occurs as a subroutine in the solution of the Hitchcock transportation problem, and in the determination of whether one given tree is isomorphic to a subtree of another.

In view of this variety of applications, the computational complexity of the problem of finding a maximum matching in a bipartite graph is of interest. The best previous methods ([1], [3], [4], [5]) seem to require $O(mn)$ steps, where m is the number of edges, and n the number of vertices. The present method requires only $O((m + n)\sqrt{n})$ steps.

We hope to extend our results to the nonbipartite case (cf. [2]). With this in mind, all the results in § 2 are derived for general graphs. The specialization to the bipartite case occurs in § 3.

2. Matchings and augmenting paths. Let $G = (V, E)$ be a finite undirected graph (without loops, multiple edges, or isolated vertices) having the vertex set V and the edge set E. An edge incident with vertices v and w is written $\{v, w\}$. A set $M \subseteq E$ is a *matching* if no vertex $v \in V$ is incident with more than one edge in M. A matching of maximum cardinality is called a *maximum matching*.

We make the following definitions relative to a matching M. A vertex v is *free* if it is incident with no edge in M.

* Received by the editors November 14, 1972, and in revised form April 23, 1973. This research was supported in part by the National Science Foundation under Grants NSF GJ96, GP-25081 and GJ474.

† Department of Computer Science, Cornell University, Ithaca, New York, 14850.

‡ Computer Science Department, University of California, Berkeley, California 94720.

A path (without repeated vertices)

$$P = (v_1, v_2), (v_2, v_3), \cdots, (v_{2k-1}, v_{2k})$$

is called an *augmenting path* if its endpoints v_1 and v_{2k} are both free, and its edges are alternatively in $E - M$ and in M; i.e.,

$$P \cap M = \{(v_2, v_3), (v_4, v_5), (v_6, v_7), \cdots, (v_{2k-2}, v_{2k-1})\}.$$

When no ambiguity is possible, we let P denote the set of edges in an augmenting path P as well as the sequence of edges which is the path itself. If S and T are sets, then $S \oplus T$ denotes the symmetric difference of S and T, and $S - T$ denotes the set of elements in S which are not in T. If S is a finite set, then $|S|$ denotes the cardinality of S.

LEMMA 1. *If M is a matching and P is an augmenting path relative to M, then $M \oplus P$ is a matching, and $|M \oplus P| = |M| + 1$.*

Figure 1 denotes a graph G with a matching M and augmenting path P along with the matching $M \oplus P$.

THEOREM 1. *Let M and N be matchings. If $|M| = r$, $|N| = s$ and $s > r$, then $M \oplus N$ contains at least $s - r$ vertex-disjoint augmenting paths relative to M.*

Proof. Consider the graph $\bar{G} = (V, M \oplus N)$ with vertex set V and edge set $M \oplus N$. Since M and N are matchings, each vertex is indicent with at most one edge from $N - M$ and at most one edge from $M - N$; hence each (connected) component of $\bar{G}$ is either

(i) an isolated vertex,

(ii) a cycle of even length, with edges alternatively in $M - N$ and in $N - M$, or

(iii) a path whose edges are alternatively in $M - N$ and in $N - M$.

Let the components of $\bar{G}$ be $C_1, C_2, \cdots, C_g$, where $C_i = (V_i, E_i)$. Let $\delta(C_i) = |E_i \cap N| - |E_i \cap M|$. Then $\delta(C_i) \in \{-1, 0, 1\}$, and $\delta(C_i) = 1$ if and only if C_i is an augmenting path relative to M.

$$\sum_i \delta(C_i) = |N - M| - |M - N| = |N| - |M| = s - r.$$

Hence there are at least $s - r$ components C_i of $\bar{G}$ such that $\delta(C_i) = 1$. These components are vertex-disjoint, and each is an augmenting path relative to M.

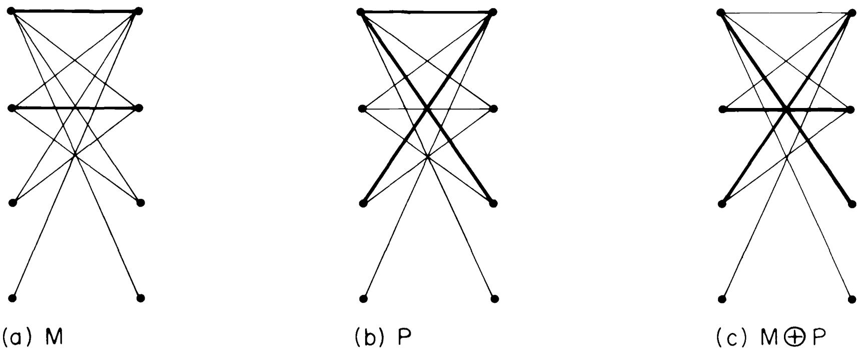


FIG. 1. *Graph G with* (a) *matching M,* (b) *augmenting path P and* (c) *new matching $M \oplus P$ in dark edges*

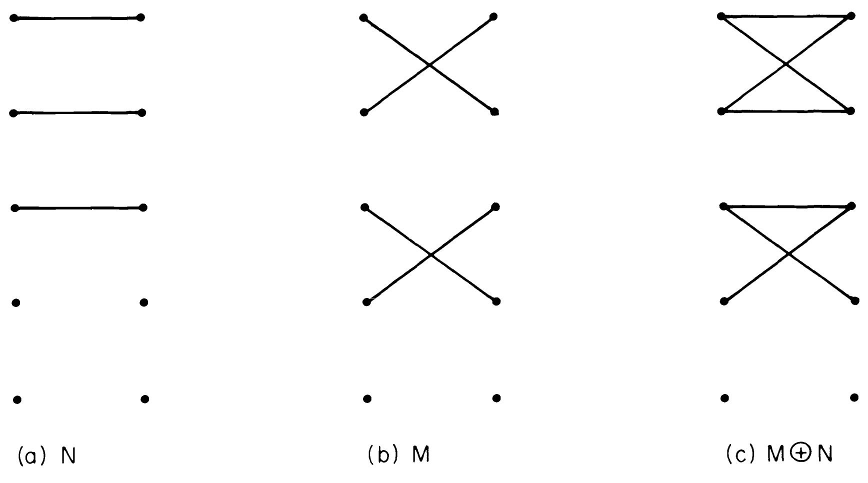


FIG. 2. *Matchings. N, M and $M \oplus N$ in a graph G. Edges of G are not shown.*

An example of matchings N and M in a graph $G = (V, E)$ along with the graph $(V, N \oplus M)$ is given in Fig. 2.

COROLLARY 1 (Berge). *M is a maximum matching if and only if there is no augmenting path relative to M.*

COROLLARY 2. *Let M be a matching. Suppose $|M| = r$, and suppose that the cardinality of a maximum matching is s, $s > r$. Then there exists an augmenting path relative to M of length $\leqq 2\lfloor r/(s - r)\rfloor + 1$.*[1]

Proof. Let N be a maximum matching. Then $M \oplus N$ contains $s - r$ vertex-disjoint (and hence edge-disjoint) augmenting paths relative to M. Altogether these contain at most r edges from M, so one of them must contain at most $\lfloor r/(s - r)\rfloor$ edges from M, and hence at most $2\lfloor r/(s - r)\rfloor + 1$ edges altogether.

Let M be a matching. The augmenting path P is called *shortest* relative to M if P is of least cardinality among augmenting paths relative to M.

THEOREM 2. *Let M be a matching, P a shortest augmenting path relative to M, and P' an augmenting path relative to $M \oplus P$. Then*

$$|P'| \geqq |P| + |P \cap P'|.$$

Proof. Let $N = M \oplus P \oplus P'$. Then N is a matching and $|N| = |M| + 2$, so $M \oplus N$ contains two vertex-disjoint augmenting paths relative to M; call them P_1 and P_2. Since $M \oplus N = P \oplus P'$, $|P \oplus P'| \geqq |P_1| + |P_2|$. But $|P_1| \geqq |P|$ and $|P_2| \geqq |P|$, since P is a shortest augmenting path. So $|P \oplus P'| \geqq |P_1| + |P_2| \geqq 2|P|$, and also we have the identity $|P \oplus P'| = |P| + |P'| - |P \cap P'|$. Hence $|P'| \geqq |P| + |P \cap P'|$.

We envisage the following scheme of computation: starting with a matching $M_0 = \emptyset$, compute a sequence $M_0, M_1, M_2, \cdots, M_i, \cdots$, where P_i is a shortest augmenting path relative to M_i, and $M_{i+1} = M_i \oplus P_i$.

[1] The symbol $\lfloor x\rfloor$ denotes the greatest integer less than or equal to x, and $\lceil x\rceil$ denotes the least integer greater than or equal to x.

COROLLARY 3. $|P_i| \leqq |P_{i+1}|$.

COROLLARY 4. *For all i and j such that $|P_i| = |P_j|$, P_i and P_j are vertex-disjoint.*

Proof. Suppose, for contradiction, that $|P_i| = |P_j|$, $i < j$, and P_i and P_j are not vertex-disjoint. Then there exist k and l such that $i \leqq k < l \leqq j$, P_k and P_l are not vertex-disjoint, and for each m, $k < m < l$, P_m is vertex-disjoint from P_k and P_l. Then P_l is an augmenting path relative to $M_k \oplus P_k$, so $|P_l| \geqq |P_k| + |P_k \cap P_l|$. But $|P_l| = |P_k|$, so $|P_k \cap P_l| = 0$. Thus P_k and P_l have no edges in common. But if P_k and P_l had a vertex v in common, they would have in common that edge incident with v which is in $M_k \oplus P_k$. Hence P_k and P_l are vertex-disjoint, and a contradiction is obtained.

THEOREM 3. *Let s be the cardinality of a maximum matching. The number of distinct integers in the sequence*

$$|P_0|, |P_1|, \cdots, |P_i|, \cdots$$

is less than or equal to $2\lfloor\sqrt{s}\rfloor + 2$.

Proof. Let $r = \lfloor s - \sqrt{s}\rfloor$. Then $|M_r| = r$ and, by Corollary 2,

$$|P_r| \leqq 2\lfloor s - \sqrt{s}\rfloor/(s - \lfloor s - \sqrt{s}\rfloor) + 1 \leqq 2\lfloor\sqrt{s}\rfloor + 1.$$

Thus, for each $i < r$, $|P_i|$ is one of the $\lfloor\sqrt{s}\rfloor + 1$ positive odd integers less than or equal to $2\lfloor\sqrt{s}\rfloor + 1$. Also $|P_{r+1}|, \cdots, |P_s|$ contribute at most $s - r = \lceil\sqrt{s}\rceil$ distinct integers, and the total number of distinct integers is less than or equal to $\lfloor\sqrt{s}\rfloor + 1 + \lceil\sqrt{s}\rceil \leqq 2\lfloor\sqrt{s}\rfloor + 2$, and the proof is complete.

In view of Corollaries 3 and 4 and Theorem 3, the computation of the sequence $\{M_i\}$ breaks into at most $2\lfloor\sqrt{s}\rfloor + 2$ phases, within each of which all the augmenting paths found are vertex-disjoint and of the same length. Since these paths are vertex-disjoint, they are all augmenting paths relative to the matching with which the phase is begun. This gives us an alternative way of describing the computation of a maximum matching.

ALGORITHM A (Maximum matching algorithm).

```
Step 0. $M \leftarrow \varnothing$.
Step 1. Let $l(M)$ be the length of a shortest augmenting path relative to $M$.
        Find a maximal[2] set of paths $\{Q_1, Q_2, \cdots, Q_t\}$ with the properties
        that
        (a) for each $i$, $Q_i$ is an augmenting path relative to $M$ and $|Q_i| = l(M)$;
        (b) the $Q_i$ are vertex-disjoint.
        Halt if no such paths exist.
Step 2. $M \leftarrow M \oplus Q_1 \oplus Q_2 \oplus \cdots \oplus Q_t$; go to 1.
```

COROLLARY 5. *If the cardinality of a maximum matching is s, then Algorithm* A *constructs a maximum matching within $2\lfloor\sqrt{s}\rfloor + 2$ executions of Step* 1.

This way of describing the construction of a maximum matching suggests that we should not regard successive augmentation steps as independent computations, but should concentrate instead on the efficient implementation of an entire phase (i.e., the execution of Step 1 in Algorithm A). The next section shows the advantage of this approach in the case where G is a bipartite graph.

[2] A set is *maximal* with a given property if it has the property and is not properly contained in any set that has the property.

3. The bipartite case. The graph $G = (V, E)$ is *bipartite* if the set of vertices V can be partitioned into two sets, X and Y, such that each edge of G joins a vertex in X with a vertex in Y. An element of X will be called a *boy*, and an element of Y, a *girl*.

Let M be a matching in a bipartite graph G. We discuss the implementation of Step 1 of Algorithm A, in which a maximal vertex-disjoint set of shortest augmenting paths relative to M is found. First we assign directions to the edges of G in such a way that augmenting paths relative to M become directed paths. This is done by directing each edge in $E - M$ so that it runs from a girl to a boy, and each edge in M so that it runs from a boy to a girl. The resulting directed graph is $\bar{G} = (V, \bar{E})$, where

$$\bar{E} = \{(y,x) | \{x,y\} \in E - M, x \in X, y \in Y\} \cup \{(x,y) | \{x,y\} \in M, x \in X, y \in Y\}.$$

Next we extract a subgraph $\hat{G}$ of $\bar{G}$, with the property that the directed paths of $\hat{G}$ running from a free girl to a free boy correspond one-to-one to the shortest augmenting paths in G relative to M. This is done as follows.

Let L_0 be the set of free boys, and let

$$E_i = \{(u,v) | (u,v) \in \bar{E}, v \in L_i, u \notin L_0 \cup L_1 \cup \cdots \cup L_i\}, \qquad i = 0,1,2,\cdots,$$

$$L_{i+1} = \{u | \text{ for some } v, (u,v) \in E_i\}, \qquad i = 0,1,2,\cdots.$$

Let $i^* = \min\{i | L_i \cap \{\text{free girls}\} \neq \emptyset\}$.

Then $\hat{G} = (\hat{V}, \hat{E})$, where

$$\hat{V} = L_0 \cup L_1 \cup \cdots \cup L_{i^*-1} \cup (L_{i^*} \cap \{\text{free girls}\}),$$

$$\hat{E} = E_0 \cup E_1 \cup \cdots \cup E_{i^*-2} \cup \{(u,v) | v \in L_{i^*-1} \text{ and } u \in \{\text{free girls}\}\}.$$

An example of a graph $\hat{G}$ is given in Fig. 3.

The following properties of $\hat{G}$ are immediate.

(i) The vertices at even levels ($L_0, L_2, \cdots$) are boys, and those at odd levels are girls.

(ii) If $(u,v) \in \hat{E}$, then for some i, $u \in L_{i+1}$ and $v \in L_i$.

(iii) $\hat{G}$ is acyclic.

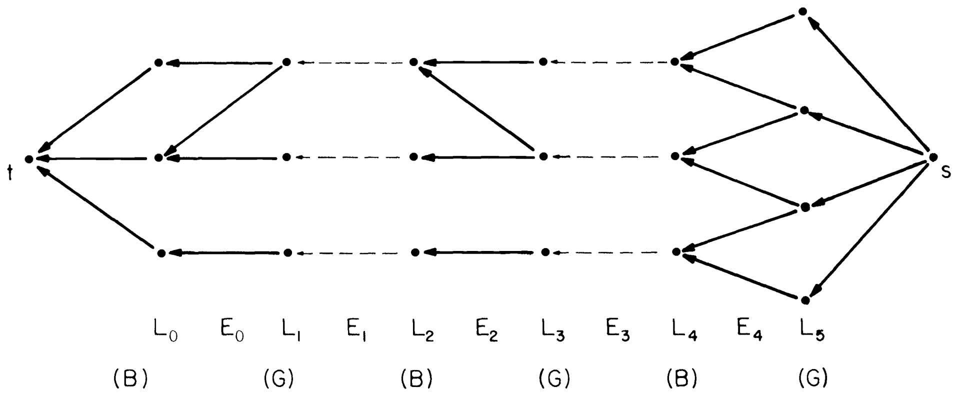


FIG. 3. *The graph $\hat{G}$ used by Algorithm* B. *Dotted arrows indicate edges in current matching.*

(iv) The shortest augmenting paths relative to M are in one-to-one correspondence with the paths of $\hat{G}$ which begin at a free girl and end at a free boy. These paths are all of length i^*.

For convenience, adjoin to $\hat{G}$ two new vertices, s (the source) and t (the sink); adjoin an edge from s to every free girl in $\hat{V}$, and an edge to t from every free boy in $\hat{V}$. Then we seek a maximal set of paths from s to t, subject to the restriction that the paths are vertex-disjoint except for their endpoints.

We give an algorithm to find a maximal vertex-disjoint (except for endpoints) set of paths from s to t in an arbitrary acyclic directed graph H. The mechanism for finding a maximal set of paths is a straightforward depth-first search. Each edge processed either becomes part of the path being constructed from s to t, or else there is no s-t path using that edge. In either case, the edge need never be examined again and so is deleted.

We assume that the graph is represented as follows: for each vertex u, a read-only linear list LIST(u) is given containing, in an arbitrary order, the vertices v such that (u, v) is an edge. The algorithm also uses an auxiliary last-in first-out list called STACK, which is initially empty, and a set B of vertices which is initially the empty set. The following primitives occur in the algorithm.

Variables	
TOP	top element of STACK
FIRST	
Operations	
PUSH x	push element x onto STACK
POP	pop an element from STACK
DELETE	delete the first element from LIST(TOP)
PRINT	POP until STACK is empty and print the successive elements
Predicates	
EMPTY	STACK is empty
NULL	LIST(TOP) is empty

ALGORITHM B (maximal set of s-t paths).

```
PUSH s
while STACK ¬EMPTY do
begin
* while LIST(TOP) ¬NULL do
  begin
    FIRST = first element of LIST(TOP)
    if FIRST ∉ B then
    begin
      PUSH FIRST
      if TOP ¬= t then B ← B ∪ {TOP}
        else PRINT, PUSH s
    end
  end
  POP
end
```

We make the following inductive assertions: every time instruction * is executed,

(i) STACK contains the vertex sequence of a path from s to TOP;

(ii) for every vertex $u \neq s, t$, one of the following holds:

(a) $u \in B$ and u is on a s-t path already printed;

(b) $u \in B$ and u is on the stack;

(c) $u \in B$ and u does not occur in any s-t path which is disjoint from the s-t paths already printed;

(d) $u \notin B$, u does not occur on the stack or in any s-t path previously printed, and for every v, $u \in \text{LIST}(v)$ if and only if (v, u) is an edge of the graph H (i.e., u has not been DELETEd from any LIST).

The algorithm terminates when s is POPed from STACK after it is found that LIST(s) is EMPTY. The inductive assertions given above then imply that upon termination, no s-t path exists disjoint from those already PRINTed.

Each while block in the algorithm contains either a POP or a DELETE operation. Since no vertex is POPed more than once, or DELETEd from any LIST more than once, the running time of the algorithm is bounded by a constant times (number of vertices + number of edges).

Hence the execution of Step 1 of Algorithm A requires at most $O(m + n)$ operations, and the execution of the entire maximum matching algorithm requires at most $O((m + n)\sqrt{s}) = O(n^{5/2})$ steps.

REFERENCES

[1] C. BERGE, *Two theorems in graph theory*, Proc. Nat. Acad. Sci. U.S.A., 43 (1957), pp. 842–844.

[2] J. EDMONDS, *Paths, trees and flowers*, Canad. J. Math., 17 (1965), pp. 449–467.

[3] L. R. FORD AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, N.J., 1965.

[4] M. HALL, *Distinct representatives of subsets*, Bull. Amer. Math. Soc., 54 (1948), pp. 922–926.

[5] H. W. KUHN, *The Hungarian method for the assignment problem*, Naval Res. Logist. Quart., 2 (1955), pp. 83–97.

SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

QUEUEING ANALYSIS OF A MULTIPROGRAMMED COMPUTER SYSTEM HAVING A MULTILEVEL STORAGE HIERARCHY*

STEPHEN S. LAVENBERG†

Abstract. We formulate a class of closed queueing network models which can be used to represent certain features of multiprogrammed computer systems having multilevel storage hierarchies. The resources which comprise the system are described by a network of interconnected multiserver stages where each stage can provide more than one type of service. The sequence of services required by a program executing in the system is described by a finite Markov chain over the service types. This description permits an explicit representation in the model of the data transfers which occur as determined by the data transfer rules and data paths in the hierarchy. The queueing discipline at each stage is nonpreemptive priority among the types of service provided by the stage, and first-come first-served within a service type.

We derive simple expressions relating the work rates for different stages and obtain simple upper bounds on the work rates. These results are valid for general service time distributions. We then apply a model in this class to the analysis of a multiprogrammed three level staging hierarchy. Under the assumption that all service time distributions are exponential, we numerically investigate the effects on system performance of different service priorities and of varying the program load parameters and level of multiprogramming.

Key words. computer system modeling and analysis, multiprogrammed computers, paging, queueing networks, storage hierarchies

1. Introduction. Consider a multiprogrammed computer system having a multilevel storage hierarchy. Under multiprogramming, programs contend for the services of the central processing unit (CPU) and the storage devices which comprise the hierarchy, and queueing delays result. In addition, CPU processing for one program is overlapped with data transfer activities for other programs. System performance measures such as CPU utilization and the average access times to each level of the hierarchy depend on the program load and the system design in a complicated way. Each set of programs and each system design requires a detailed simulation in order to determine system performance accurately. Alternatively, mathematical models suggest themselves. One approach to the mathematical modeling of such a system is found in [9]. A mathematical model of such a system might incorporate representations of the program load on the system, the storage devices which comprise the hierarchy, the data transfers which occur as determined by the data transfer rules and data paths in the hierarchy, the queueing which occurs due to temporal contention for the CPU and storage devices, the dynamic sharing of storage space among programs and the CPU overhead activities.

In this paper we formulate a class of queueing models which can be used to represent certain features of multiprogrammed computer systems having multilevel storage hierarchies. A model in this class consists of a Markov chain description of the sequence of services required by a single program executing in the system as determined by the data transfer rules and data paths in the hierarchy, augmented by a closed queueing network description of the resources (CPU and

* Received by the editors November 9, 1972, and in revised form May 8, 1973.

† IBM Corporation, San Jose, California 95193.

storage devices) which comprise the system. By choosing some of the Markov chain transition probabilities equal to unity, we explicitly represent deterministic sequences of data transfers which occur in the hierarchy. In order to keep the model analytically tractable, it incorporates simple representations of the program load and storage devices. CPU overhead activities and dynamic space sharing are not represented. Numerical studies of such a model are of themselves interesting in that they may provide insight into the effect of load and system parameters on system performance. Such a model may also be used as a control variable in a Monte Carlo simulation incorporating more accurate representations of program load, storage devices and other system features such as CPU overhead. An application of control variable techniques to queueing models of computer systems is found in [3].

In the following section we formally describe the class of queueing models to be considered and derive some general relationships which pertain to such models. These models are related to ones independently introduced by Muntz and Baskett [6]. In § 3 we describe a specific system to be modeled, a multiprogrammed computer system having a three level staging hierarchy [7], and present the resulting queueing model of this system. In § 4 the analysis of this model is discussed, and in § 5 numerical results are presented and interpreted. In a final section we discuss some extensions and limitations of this model.

2. A class of queueing models. Consider a network of S interconnected service stages named by the integers $1, 2, \cdots, S$ and serving N customers. Customers neither enter nor leave the network, but pass repeatedly through the stages. For each i, stage i consists of a queue and m_i identical parallel servers. There are $L \geqq S$ distinct types of service named by the integers $1, 2, \cdots, L$ provided by the network, with service l provided by stage $s(l)$; $s(l)$ is a function which maps the integers $1, 2, \cdots, L$ onto the integers $1, 2, \cdots, S$. Upon receiving service l, the customer just served proceeds instantaneously to stage $s(j)$ in order to receive service j with probability $p_{lj} \geqq 0$, $\sum_{j=1}^{L} p_{lj} = 1$. Thus, the sequence of services required by a customer is described by a finite Markov chain over the service names with transition matrix $P = (p_{lj})$. This Markov chain is assumed to be irreducible. Service times for service l are independent identically distributed (i.i.d.) nonnegative random variables, each having nonzero finite mean μ_l. Service times corresponding to different types of service are mutually independent. For each service stage, there is a total priority ordering among the types of service provided by the stage. The queueing discipline at each stage is nonpreemptive priority among the types of service, and first-come first-served within a service type.

A related class of queueing models was independently introduced by Muntz and Baskett [6]. In our terminology, they allow the Markov chain over the service types to be decomposable into irreducible subchains, thereby allowing, for example, different types of customers. In our formulation, all customers are statistically identical. They consider four different types of service stages according to the queueing discipline employed. The disciplines considered are first-come first-served, processor sharing, no queueing and last-come first-served preemptive-resume. They only allow different service time distributions for different service types at stages of the types other than first-come first-served. This restriction does not permit a convenient service stage representation of the storage devices comprising a level of a storage hierarchy with different page sizes at different levels (see § 3). They obtain a relatively simple form for the equilibrium state probabilities of their models when service time distributions are exponential at first-come first-served stages and have rational Laplace–Stieltjes transforms elsewhere. Unfortunately, this result does not extend to queueing models of storage hierarchies as introduced in this paper.

We now obtain some preliminary results for our class of models. These results, which hold under arbitrary service time distributional assumptions, are extensions of results obtained by Chang and Lavenberg [2] for the special case of models for which each stage provides only one type of service, i.e., $L = S$.

We assume that for a model in our class there is zero service time rendered at time zero. Thus, the initial state v_0 of the model is specified by the initial service required by each customer. We define the following quantities for each v_0, l and time $t > 0$:

$M_l(t, v_0)$ = number of starts of service l in $[0, t]$;

$N_l(t, v_0)$ = number of completions of service l in $[0, t]$;

$W_l(t, v_0)$ = amount of service time of service l rendered in $[0, t]$.

Note that

$$M_l(t, v_0) - m_{s(l)} \leqq N_l(t, v_0) \leqq M_l(t, v_0), \tag{1}$$

where $m_{s(l)}$ is the number of servers at the stage which provides service l.

Let $\pi = (\pi_1, \pi_2, \cdots, \pi_L)$ denote the unique probability vector satisfying $\pi P = \pi$, where $\pi_l > 0$, $1 \leqq l \leqq L$, since P describes an irreducible Markov chain.

The following lemmas are proved in the Appendix.

LEMMA 1. *For all* v_0 *and* l,

$$N_l(t, v_0) < \infty \quad \textit{with probability one (abbreviated P1) for all } t < \infty, \tag{2}$$

$$\lim_{t \to \infty} N_l(t, v_0) = \infty \qquad \text{P1}, \tag{3}$$

$$\lim_{t \to \infty} M_l(t, v_0)/N_l(t, v_0) = 1 \qquad \text{P1}. \tag{4}$$

LEMMA 2. *For all* v_0 *and* l,

$$\lim_{t \to \infty} N_l(t, v_0)/N_i(t, v_0) = \pi_l/\pi_i \qquad \text{P1} \quad \textit{for all } i. \tag{5}$$

LEMMA 3. *For all* v_0, *the existence* P1 *of either of the limits*, $\lim_{t\to\infty} W_l(t, v_0)/t$, $\lim_{t\to\infty} N_l(t, v_0)/t$, *implies the existence* P1 *of the other. When these limits exist, the following relationship holds*:

$$\lim_{t \to \infty} W_l(t, v_0)/t = \mu_l \lim_{t \to \infty} N_l(t, v_0)/t. \tag{6}$$

We now assume that for each l, $\lim_{t\to\infty} W_l(t, v_0)/t$ exists with probability one and is a degenerate random variable W_l, i.e., a constant, whose value is independent of the initial state v_0. W_l is called the *work rate for service* l. It then follows from (5) and (6) that for all l and j,

$$W_l/W_j = \pi_l\mu_l/\pi_j\mu_j. \tag{7}$$

The work rate for a stage is defined as the sum of the work rates for all services provided by the stage. Thus, letting U_q denote the *work rate for stage* q, we have

$$U_q = \sum_{l:s(l)=q} W_l.$$

Using (7), we see that for all stages q and r,

$$U_q/U_r = \sum_{l:s(l)=q} \pi_l \mu_l \Big/ \sum_{l:s(l)=r} \pi_l \mu_l. \tag{8}$$

U_q is the long-run time-average amount of service time of all types of service rendered by stage q. U_q can also be interpreted as the long-run time-average number of servers busy at stage q. If $m_q = 1$, then U_q is simply the utilization for stage q. Substituting the inequality $U_r \leqq m_r$ into (8) for each r yields the inequality

$$U_q \leqq \sum_{l:s(l)=q} \pi_l \mu_l \Big/ \frac{1}{m_{q^*}} \sum_{l:s(l)=q^*} \pi_l \mu_l \quad \text{for all } q, \tag{9}$$

where q^* is chosen so that

$$\frac{1}{m_{q^*}} \sum_{l:s(l)=q^*} \pi_l \mu_l = \max_{1 \leqq q \leqq S} \frac{1}{m_q} \sum_{l:s(l)=q} \pi_l \mu_l. \tag{10}$$

Furthermore, equality holds in (9) if and only if all the servers at stage q^* are essentially always busy, i.e., $U_{q^*} = m_{q^*}$. For any stage q not achieving the maximum in (10), $U_q/m_q < U_{q^*}/m_{q^*}$. Let $U_q(N)$ denote the dependence of U_q on N, the number of customers in the network. We conjecture that $\lim_{N\to\infty} U_{q^*}(N) = m_{q^*}$. This conjecture was proved for the special case of models for which $L = S$ [2], but the method of proof does not extend to $L > S$.

We now define two terms which are useful in discussing some performance aspects of our class of models. We will use them in interpreting numerical results for the queueing model application presented in the next section.

DEFINITION 1. A stage is a *limiting stage* if it achieves the maximum in (10).

DEFINITION 2. Two stages q and r are in *balance* if

$$\frac{1}{m_q} \sum_{l:s(l)=q} \pi_l \mu_l = \frac{1}{m_r} \sum_{l:s(l)=r} \pi_l \mu_l.$$

Note that from the above, the work rate per server for any limiting stage is greater than the work rate per server for any nonlimiting stage, and two stages which are in balance have the same work rate per server.

3. Three-level staging hierarchy; description and resulting model. By a staging hierarchy we mean a storage hierarchy in which data paths exist only between adjacent levels of the hierarchy and in which larger quantities of information are transferred between the lower levels of the hierarchy than between the higher levels [7]. We now describe a three level staging hierarchy with a particular physical implementation and set of data transfer rules. This hierarchy is shown in Fig. 1. The top level of the hierarchy is physically implemented by a random access memory, and the second and third levels are physically implemented by asynchronous rotating storage facilities (for example, a drum storage facility at level 2 and a disk storage facility at level 3). The top two levels have limited capacities,

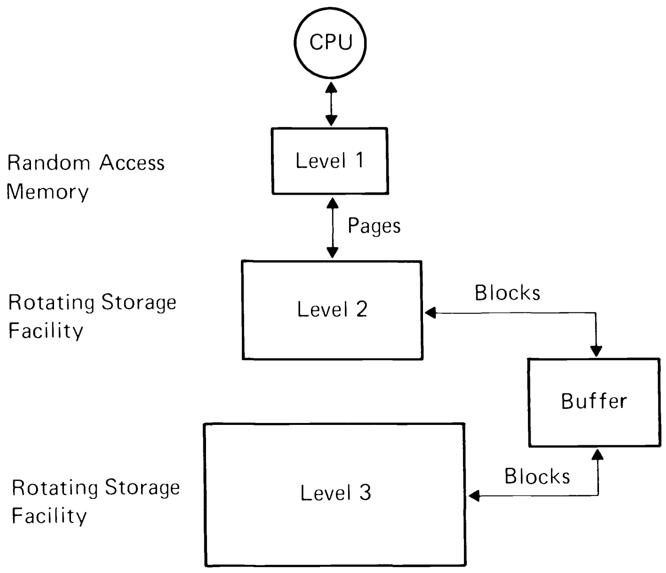


FIG. 1. *Three level staging hierarchy*

while the bottom level has sufficient capacity to contain all the information which can be referenced by the CPU. The CPU can only process information residing in the top level. Since levels 2 and 3 of the hierarchy operate asynchronously, information transfer between these levels occurs through an intermediate buffer. This buffer is of sufficient capacity so that there is always space available for the movement of information between levels 2 and 3.

The information which can be referenced by the CPU is logically divided into equal-sized quantities called *pages*, and into larger equal-sized quantities called *blocks*. A block consists of an integral number of pages. Pages are moved between levels 1 and 2 of the hierarchy and blocks between levels 2 and 3. When information is referenced which resides in a level $i > 1$ of the hierarchy, but does not reside in any higher level (i.e., any level j for which $j < i$), this information must be moved up the hierarchy to the top level for processing. For example, if information residing in level 3, but in no higher level, is referenced, the block containing this information is moved to level 2, and then the page containing this information is moved to level 1. If any of the higher levels are filled to capacity, information is first chosen for replacement at these levels. The hierarchy is designed and managed so that every page residing in the top level has its containing block in the second level. One way of accomplishing this is described in [7]. If the page to be replaced at the top level has been modified, the containing block must be updated. Thus, the page chosen for replacement at the top level is written into its containing block at the second level. This downward movement is assumed to occur even if the page being replaced was not modified. (An extension of the model presented in this section which addresses this point is discussed in § 6.) Similarly, the block chosen for replacement at the second level is always moved down to the third level.

TABLE 1
Description of services required by a single program executing in a three level staging hierarchy

event	service	service type	server
hit to level 1	process	1	CPU
hit to level 2	page: level 1 → level 2	2	level 2
	page: level 2 → level 1	3	level 2
	process	1	CPU
hit to level 3	page: level 1 → level 2	4	level 2
	block: level 2 → buffer	5	level 2
	block: buffer → level 3	6	level 3
	block: level 3 → buffer	7	level 3
	block: buffer → level 2	8	level 2
	page: level 2 → level 1	9	level 2
	process	1	CPU

In Table 1, we describe the services required by a single program executing in a computer system having the staging hierarchy just described. On each CPU reference, one of the three events in column 1 can occur, where a hit to level i is a reference to information residing in level i but not residing in any higher level. Each event results in a sequence of services being performed. It is assumed that the top two levels of the hierarchy are filled to capacity and that when information is to be replaced at levels 1 or 2, it is first moved down the hierarchy. The services are performed sequentially in the order shown in column 2. We distinguish $L = 9$ types of service in column 2 and these are named by the integers 1 to 9 in column 3. We model the sequence of services required by a single program by a Markov chain over the integers 1 to 9 with transition probabilities,

$$(11)\qquad \begin{aligned} p_{23} &= p_{31} = p_{45} = p_{56} = p_{67} = p_{78} = p_{89} = p_{91} = 1,\\ p_{12} &= \alpha,\\ p_{14} &= 1 - \alpha. \end{aligned}$$

All other transition probabilities are zero. The deterministic sequence of services resulting from each event is represented explicitly. Note that we do not distinguish successive services of type 1, i.e., $p_{11} = 0$. Thus α is the conditional probability that a fault out of level 1 (i.e., a hit to level 2 or 3) is a hit to level 2. The quantity α could be estimated from the address trace of a given program by dividing the number of hits to level 2 by the number of faults out of level 1 [7]. It is convenient to represent pictorially the above Markov chain with the service transition diagram shown in Fig. 2.

Under multiprogramming, with a fixed number N of programs executing in the system, we assume that the storage capacity at each level of the hierarchy is statically allocated among the N programs and that all replacement algorithms operate locally. All programs have identical page sizes and identical block sizes.

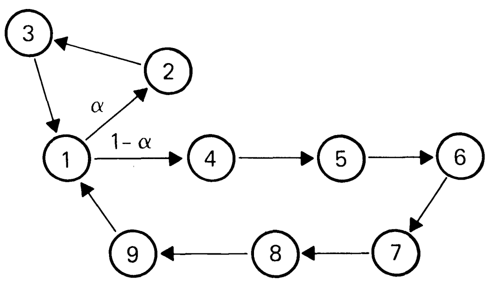


FIG. 2. *Service transition diagram for a single program*

A given program has sole access to the CPU until it references information not contained in level 1, in which case the next program ready for processing is given access to the CPU. A program is ready for processing when its information transfer activities have been completed. These programs require service from the resources which comprise the system giving rise to contention for these resources. The only resources explicitly modeled are the CPU, level 2 and level 3. The random access memory (level 1) and the buffer are assumed to be fast enough to handle the load imposed on them without significant congestion. The CPU is modeled as a single server stage while levels 2 and 3 are modeled as multiserver stages to reflect the parallelism in such storage facilities (see Fig. 3). These resources form a network of $S = 3$ interconnected stages serving N customers (programs). The possible paths between these stages are indicated in Fig. 4. The sequence of services required by each customer is modeled by the Markov chain in (11). Service 1 is provided by the CPU, services 2, 3, 4, 5, 8 and 9 by level 2, and services 6 and 7 by level 3 as indicated in column 4 of Table 1. Thus, the function $s(l)$ of § 2 is given by

$$s(l) = \begin{cases} 1, & l = 1, \\ 2, & l = 2,3,4,5,8,9, \\ 3, & l = 6,7. \end{cases}$$

Service times of a given type are assumed to be i.i.d. exponential random variables, with mean μ_l for service type l, and service times of different types are mutually independent. All programs are statistically identical. A CPU service time (type 1 service time) is the duration of an interval during which a given program references and processes information residing in level 1 of the hierarchy. $\mu_1 \equiv \mu_{\text{CPU}}$ could be obtained from the address trace of a given program by multiplying the average number of references between faults out of level 1 by the average CPU processing time per address reference. A type l service time, $l = 2, 3, 4$ or 9, is the time to read or write a page from or to level 2 of the hierarchy, and we assume $\mu_2 = \mu_3 = \mu_4 = \mu_9 \equiv \mu_{2P}$. A type 5 or 8 service time is the time to read or write a block from or to level 2, and we assume $\mu_5 = \mu_8 \equiv \mu_{2B}$, where $\mu_{2B} > \mu_{2P}$. A type 6 or 7 service time is the time to read or write a block from or to level 3 and we assume $\mu_6 = \mu_7 \equiv \mu_{3B}$, where $\mu_{3B} > \mu_{2B}$.

The queueing discipline at each stage of the model is of the nonpreemptive priority type described in § 2. Since the CPU renders only one type of service, the service discipline at the CPU is first-come first-served. The following priority

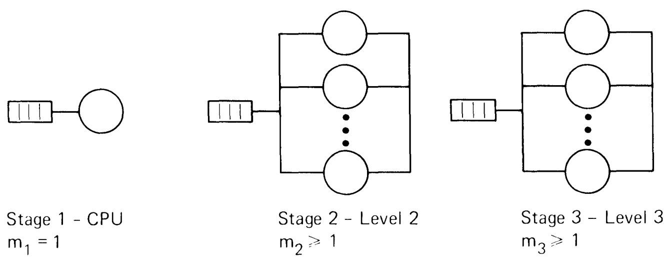


FIG. 3. *Service stages which comprise the queueing network*

orderings among the services provided by stage 2 will be considered, where $l > j$ means service l has nonpreemptive priority over service j:

(i) $3 > 2 > 9 > 8 > 5 > 4$;
(ii) $2 > 3 > 4 > 5 > 8 > 9$;
(iii) $9 > 8 > 5 > 4 > 3 > 2$;
(iv) $4 > 5 > 8 > 9 > 2 > 3$.

In disciplines (i) and (ii), services 2 and 3 due to hits to level 2 are given priority over services 4, 5, 8 and 9 due to hits to level 3. In addition, in (i), the order of services 2 and 3 and of services 4, 5, 8 and 9 corresponds to giving highest priority to those services for which the fewest services remain until the program is ready for processing by the CPU. In (ii), highest priority is given to those services for which the most services remain. In (iii) and (iv), services due to hits to level 3 are given priority over services due to hits to level 2. In (iii), the order of services 4, 5, 8 and 9 and of services 2 and 3 is fewest remaining services, and (iv), most remaining services. The priority orderings considered for stage 3 are

(i) $7 > 6$,
(ii) $6 > 7$,

corresponding to fewest remaining services and most remaining services, respectively.

We summarize the model parameters in Table 2.

4. Analysis of the model. All service time distributions for the model in § 3 were assumed to be exponential. Thus, the analysis of this model involves computing the steady state probabilities of a finite state Markov process and using these to compute the response variables of interest. The state of the process at any time is determined by the service required by each program at that time and the

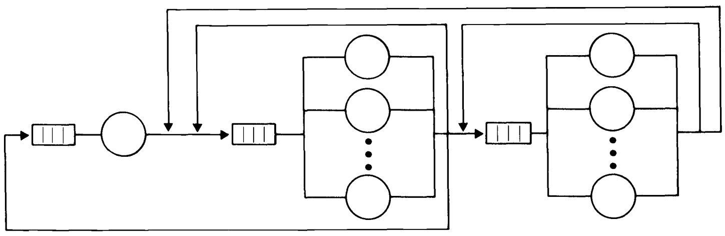


FIG. 4. *Possible paths in the queueing network*

TABLE 2
Parameters for model of multiprogrammed three level staging hierarchy

parameter	value
S	3
m_1	1
m_2, m_3	$\geqq 1$
L	9
$s(1)$	1
$s(l)$, $l = 2, 3, 4, 5, 8, 9$	2
$s(l)$, $l = 6, 7$	3
p_{12}	α
p_{14}	$1 - \alpha$
$p_{23}, p_{31}, p_{45}, p_{56}, p_{67}, p_{78}, p_{89}, p_{91}$	1
μ_1	μ_{CPU}
$\mu_2, \mu_3, \mu_4, \mu_9$	μ_{2P}
μ_5, μ_8	μ_{2B}
μ_6, μ_7	μ_{3B}
Stage 2 priorities	(i) $3 > 2 > 9 > 8 > 5 > 4$
	(ii) $2 > 3 > 4 > 5 > 8 > 9$
	(iii) $9 > 8 > 5 > 4 > 3 > 2$
	(iv) $4 > 5 > 8 > 9 > 2 > 3$
Stage 3 priorities	(i) $7 > 6$
	(ii) $6 > 7$

order in which each stage will provide these services as determined by the queueing disciplines. Formally, the state of the process at any time can be represented as an 18-tuple $(r_1, r_2, \cdots, r_9, w_1, w_2, \cdots, w_9)$, where r_l is the number of programs receiving service l and w_l is the number of the program waiting in queue to receive service l. Clearly,

$$\sum_{l=1}^{L} (r_l + w_l) = N. \tag{12}$$

Not all 18-tuples of nonnegative integers satisfying (12) are possible states. Additional constraints must be satisfied; for example, $r_6 + r_7 \leqq m_3$, and $w_6 + w_7 > 0$ only if $r_6 + r_7 = m_3$.

It can be shown that for the queueing disciplines we consider, the state corresponding to all programs at the CPU, i.e., the state with $r_1 = 1, w_1 = N - 1$ and $r_l = w_l = 0$ for all $l \neq 1$, is reachable from any other state. Thus, the set of states which communicate with this state comprise the single ergodic class of the process. All other states are transient. The steady state probabilities exist and are independent of the initial state.

The response variables we consider are the work rates for the stages, the average access times to levels 2 and 3 and the overall average access time. The work rate U_q for stage q was defined in § 2. The work rates exist for our model and work rates for different stages are related by (8). Using parameter values for the model given in Table 2, it follows from (8) that

$$\begin{aligned} U_2/U_1 &= [2\mu_{2P} + 2(1 - \alpha)\mu_{2B}]/\mu_{\text{CPU}}, \\ U_3/U_1 &= 2(1 - \alpha)\mu_{3B}/\mu_{\text{CPU}}. \end{aligned} \tag{13}$$

We will present numerical values only for U_1, the CPU work rate. Values for U_2 and U_3 can be obtained from (13). It follows from (9) and (10) that

$$(14) \qquad U_1 \leqq \bar{U}_1 \equiv \mu_{\mathrm{CPU}} \Big/ \max\left(\mu_{\mathrm{CPU}}, \frac{2\mu_{2P} + 2(1-\alpha)\mu_{2B}}{m_2}, \frac{2(1-\alpha)\mu_{3B}}{m_3}\right).$$

Stage 1, 2 or 3 is a limiting stage if μ_{CPU}, $[2\mu_{2P} + 2(1-\alpha)\mu_{2B}]/m_2$ or $2(1-\alpha)\mu_{3B}/m_3$ respectively achieves the maximum in the denominator on the right-hand side of (14). Suppose stage q^* is a limiting stage, where $q^* = 2$ or 3. In § 2 we conjectured that $\lim_{N\to\infty} U_1(N) = \bar{U}_1$. For infinite N, stage q^* is a *bottleneck* relative to the CPU work rate in the following sense: if we speed up stage q^* by decreasing the mean service times at the stage or by increasing the number of servers at the stage, then the CPU work rate increases. If $q^* = 2$ and we speed up stage 3, or if $q^* = 3$ and we speed up stage 2, then the CPU work rate does not change.

The access time to a level is the time necessary to complete all data transfers resulting from a hit to that level. In the model, this is the time elapsed between the moment a program leaves the CPU stage due to a hit to that level and the moment it next enters the CPU stage. Let a_i^k denote the kth access time to level i. Conditions hold which are sufficient for Little's formula [4] to be valid. Thus, $\lim_{n\to\infty} (1/n) \sum_{k=1}^{n} a_i^k$ exists with probability one and is a constant A_i, the average access time to level i, given by

$$A_i = Q_i/\lambda_i, \qquad i = 2, 3,$$

where Q_i is the average number of programs requiring services 2 or 3 if $i = 2$, services 4, 5, 6, 7, 8 or 9 if $i = 3$, and λ_i is the rate at which hits to level i occur, i.e.,

$$\lambda_i = \lim_{t\to\infty} \frac{N_1(t)}{t} \cdot \begin{cases} \alpha, & i = 2, \\ 1 - \alpha, & i = 3, \end{cases}$$

where $N_1(t)$ is the number of CPU completions in $[0, t]$. (We have omitted any notational dependence on the initial state since, in the limit, this dependence disappears.) Using (6), we see

$$\lambda_i = \frac{U_1}{\mu_{\mathrm{CPU}}} \cdot \begin{cases} \alpha, & i = 2, \\ 1 - \alpha, & i = 3. \end{cases}$$

The overall access time to the hierarchy is the time necessary to complete all data transfers resulting from a fault out of level 1 (memory), i.e., the time elapsed between the moment a program leaves the CPU stage and the moment it next enters the CPU stage. The overall average access time, denoted by A, is given by

$$A = \alpha A_2 + (1 - \alpha) A_3.$$

A PL/I program was written which computed the steady state probabilities and used these to compute the response variables. Numerical methods similar to those in [8] were used. For example, the transition intensity matrix of the process was stored in sparse form, and an iterative technique was used to compute the steady state probabilities. In addition, the transient states were first determined from the structure of the process, and the initial iterate was chosen with all components corresponding to transient states set equal to zero. At successive iterations, only

TABLE 3
Cardinality of the state space

m_2	m_3	N	cardinality
1	1	2	61
		4	1,065
		6	8,755
1	2	2	60
		4	1,023
		6	8,310
2	1	2	46
		4	1,065
		6	11,569
2	2	2	45
		4	1,038
		6	11,019

those components corresponding to ergodic states were updated, thus avoiding wasteful multiplications by zero. This was found to increase the speed of computation by up to a factor of two. This program was run on an IBM 360/195. The increase in cardinality of the state space as N increases (see Table 3) imposed a limit on the maximum level of multiprogramming for which it was reasonable to obtain numerical results. Both the amount of storage required and the CPU time required increase as N increases. For $N = 6$, $m_2 = 1$ and $m_3 = 2$, a typical run required 300,000 bytes of storage and 2.5 minutes of CPU time.

5. Numerical results. We present some numerical results in Tables 4–8. In Table 4, we demonstrate the effects of different queueing disciplines on system performance. The numbers in parentheses refer to the priority orderings in Table 2. The orderings corresponding to fewest remaining services ((i) and (iii) at stage 2, (i) at stage 3) result in higher CPU work rate and shorter overall average access time than the orderings corresponding to most remaining services ((ii) and (iv) at stage 2, (ii) at stage 3). The CPU work rate and overall average access time are not significantly affected whether or not services due to hits to level 2 have priority at stage 2 over services due to hits to level 3 (compare (i) with (iii) and (ii) with (iv) at stage 2). However, by giving priority to services due to hits to level 2 (respectively, 3), the average access time to level 2 (respectively, 3) decreases, and the average access time to level 3 (respectively, 2) increases. In what follows, we consider only priority ordering (i) at stage 2 and (i) at stage 3.

In Tables 5–8 we fix parameters m_2, m_3, μ_{2P}, μ_{2B} and μ_{3B} associated with levels 2 and 3 of the hierarchy and vary the program load parameters μ_{CPU} and α and the level of multiprogramming N. In these tables, $\bar{U}_1$ is the upper bound on CPU work rate defined in (14).

Stage 1, 2 or 3 is a limiting stage for these parameter values if μ_{CPU}, $1 + 6(1 - \alpha)$, or $36(1 - \alpha)$, respectively, achieves the maximum of these three expressions. In

Tables 5 and 6, stage 2 is the limiting stage for $\alpha = .99$, stages 2 and 3 are in balance and are limiting stages for $\alpha = .967$, and stage 3 is the limiting stage for $\alpha = .95$ and $\alpha = .9$. In Table 7, stage 1 is the limiting stage for $\alpha = .99$, all stages are in balance for $\alpha = .967$, and stage 3 is the limiting stage for $\alpha = .95$ and $\alpha = .9$. In Table 8, stage 1 is the limiting stage for $\alpha = .99$, stages 1 and 3 are in balance for $\alpha = .958$, and stage 3 is the limiting stage for $\alpha = .95$ and $\alpha = .9$. We make several observations from these numerical results.

U_1 increases as N increases. When the stages are far out of balance, e.g., $\alpha = .99$ or $\alpha = .9$ in Table 7, all the servers at the limiting stage rapidly become essentially always busy as N increases, so that U_1 rapidly approaches $\bar{U}_1$. When the stages are close to or in balance, e.g., $\alpha = .967$ in Table 7, U_1 approaches $\bar{U}_1$ more slowly. This is shown in Fig. 5. As α decreases, $\bar{U}_1$ remains equal to unity as long as stage 1 is the limiting stage and decreases when stage 2 or 3 becomes the limiting stage. This behavior is less pronounced for finite values of N as shown in Fig. 6. $\bar{U}_1$ increases as μ_{CPU} increases when stage 2 or 3 is the limiting stage and remains equal to unity when stage 1 is the limiting stage. This behavior is less pronounced for finite values of N, as shown in Fig. 7.

TABLE 4
CPU *work rate and average access times*
$m_2 = 1, m_3 = 2, \alpha = .99, \mu_{CPU} = 1.0, \mu_{2P} = .5, \mu_{2B} = 3.0, \mu_{3B} = 36.0$

stage 2 priorities	stage 3 priorities	$N = 1$	2	3	4	5	6
		U_1					
(i)	(i)	.360	.575	.695	.764	.808	.837
(ii)	(ii)	.360	.562	.668	.728	.767	.794
(iii)	(i)	.360	.575	.693	.761	.802	.830
(iv)	(ii)	.360	.562	.666	.724	.761	.787
		A_2					
(i)	(i)	1.00	1.35	1.77	2.21	2.67	3.12
(ii)	(ii)	1.00	1.39	1.85	2.33	2.81	3.29
(iii)	(i)	1.00	1.37	1.81	2.31	2.84	3.39
(iv)	(ii)	1.00	1.40	1.89	2.41	2.95	3.49
		A_3					
(i)	(i)	79.0	81.4	86.7	94.2	104.	116.
(ii)	(ii)	79.0	81.5	86.1	91.7	98.	105.
(iii)	(i)	79.0	80.4	82.7	84.7	86.4	87.7
(iv)	(ii)	79.0	80.5	82.8	84.9	86.6	88.1
		A					
(i)	(i)	1.78	2.15	2.62	3.13	3.68	4.25
(ii)	(ii)	1.78	2.19	2.70	3.23	3.76	4.31
(iii)	(i)	1.78	2.16	2.62	3.13	3.68	4.24
(iv)	(ii)	1.78	2.20	2.70	3.24	3.78	4.33

TABLE 5
CPU *work rate and average access times*
$\mu_{CPU} = .7$, $m_2 = 1$, $m_3 = 2$, $\mu_{2P} = .5$, $\mu_{2B} = 3.0$, $\mu_{3B} = 36.0$, $3 > 2 > 9 > 8 > 5 > 4$, $7 > 6$

α	$N = 1$	2	3	4	5	6	
	U_1						$\bar{U}_1$
.99	.282	.456	.549	.599	.627	.642	.660
.967	.163	.296	.380	.436	.471	.492	.583
.95	.125	.233	.299	.340	.361	.372	.389
.90	.074	.142	.174	.189	.194	.194	.194
	A_2						
.99	1.00	1.40	1.92	2.49	3.08	3.62	
.967	1.00	1.28	1.61	1.94	2.25	2.50	
.95	1.00	1.24	1.51	1.74	1.91	2.02	
.90	1.00	1.20	1.36	1.46	1.51	1.53	
	A_3						
.99	79.0	81.6	88.0	102.	123.	156.	
.967	79.0	81.3	92.8	108.	127.	152.	
.95	79.0	80.8	94.8	114.	139.	167.	
.90	79.0	80.2	100.	127.	158.	192.	
	A						
.99	1.78	2.21	2.79	3.48	4.28	5.15	
.967	3.61	3.93	4.62	5.43	6.36	7.43	
.95	4.90	5.23	6.17	7.33	8.74	10.3	
.90	8.80	9.09	11.3	14.0	17.2	20.6	

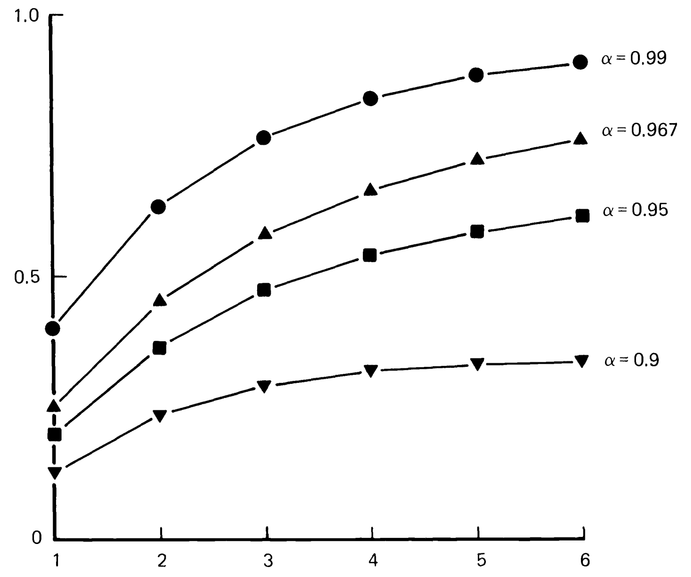


FIG. 5. U_1 *vs.* N *for Table* 7

TABLE 6

CPU *work rate and average access times*

$\mu_{CPU} = 1.0$, $m_2 = 1$, $m_3 = 2$, $\mu_{2P} = .5$, $\mu_{2B} = 3.0$, $\mu_{3B} = 36.0$, $3 > 2 > 9 > 8 > 5 > 4$, $7 > 6$

α	$N = 1$	2	3	4	5	6	
	U_1						$\overline{U}_1$
.99	.360	.575	.695	.764	.808	.837	.943
.967	.217	.392	.505	.578	.630	.668	.833
.95	.170	.314	.406	.467	.502	.524	.556
.90	.102	.196	.244	.268	.277	.278	.278
	A_2						
.99	1.00	1.35	1.77	2.21	2.67	3.12	
.967	1.00	1.26	1.56	1.86	2.16	2.44	
.95	1.00	1.23	1.48	1.72	1.91	2.06	
.90	1.00	1.19	1.36	1.48	1.54	1.56	
	A_3						
.99	79.0	81.4	86.7	94.2	104.	116.	
.967	79.0	81.0	90.7	105.	120.	137.	
.95	79.0	80.8	93.0	109.	130.	155.	
.90	79.0	80.1	98.8	124.	154.	187.	
	A						
.99	1.78	2.15	2.62	3.13	3.68	4.25	
.967	3.61	3.89	4.51	5.25	6.05	6.88	
.95	4.90	5.21	6.06	7.08	8.33	9.72	
.90	8.80	9.09	11.1	13.7	16.8	20.1	

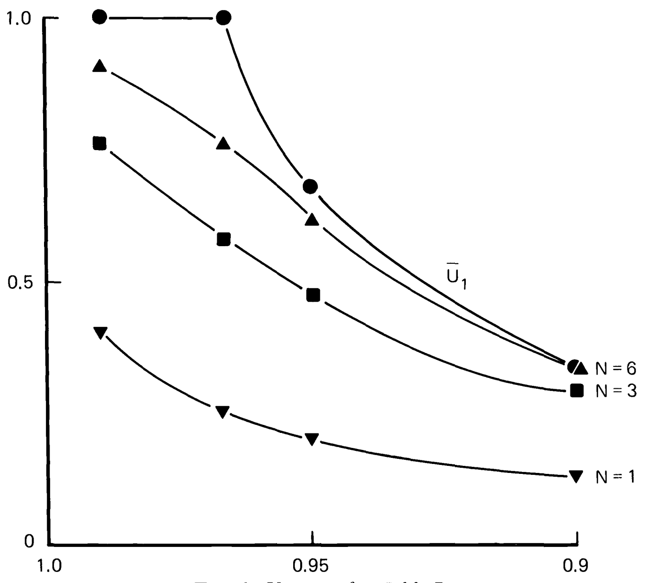


FIG. 6. U_1 *vs.* α *for Table* 7

TABLE 7
CPU work rate and average access times
$\mu_{CPU} = 1.2, m_2 = 1, m_3 = 2, \mu_{2P} = .5, \mu_{2B} = 3.0, \mu_{3B} = 36.0, 3 > 2 > 9 > 8 > 5 > 4, 7 > 6$

α	$N = 1$	2	3	4	5	6	
	U_1						$\bar{U}_1$
.99	.403	.636	.763	.836	.880	.910	1.
.967	.250	.448	.575	.658	.715	.757	1.
.95	.196	.362	.469	.541	.586	.616	.667
.90	.120	.230	.288	.319	.331	.333	.333
	A_2						
.99	1.00	1.33	1.69	2.05	2.39	2.72	
.967	1.00	1.25	1.53	1.81	2.08	2.33	
.95	1.00	1.23	1.47	1.69	1.89	2.05	
.90	1.00	1.19	1.36	1.48	1.55	1.58	
	A_3						
.99	79.0	81.4	85.8	91.5	97.7	104.	
.967	79.0	80.7	89.6	102.	116.	131.	
.95	79.0	80.8	92.1	106.	125.	147.	
.90	79.0	80.1	97.9	122.	151.	184.	
	A						
.99	1.78	2.13	2.53	2.94	3.34	3.74	
.967	3.61	3.87	4.44	5.10	5.82	6.57	
.95	4.90	5.20	6.00	6.93	8.06	9.32	
.90	8.80	9.08	11.0	13.5	16.5	19.8	

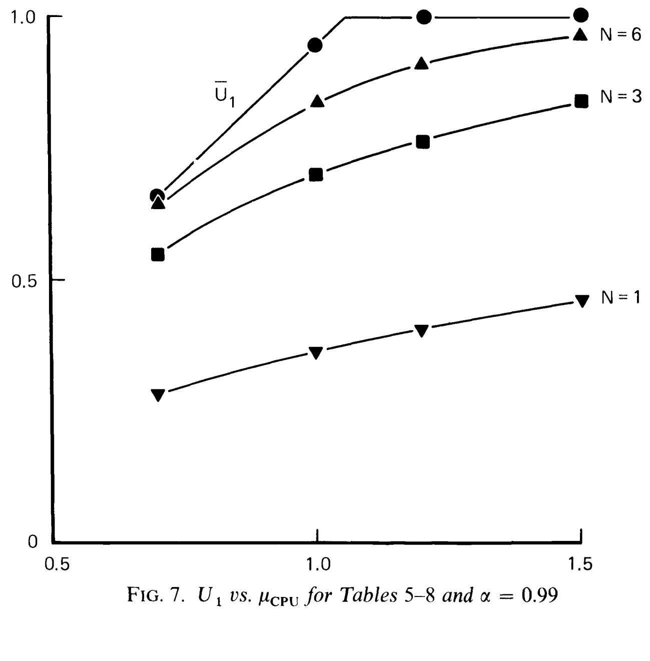


FIG. 7. U_1 *vs.* μ_{CPU} *for Tables 5–8 and* $\alpha = 0.99$

TABLE 8

CPU work rate and average access times

$\mu_{CPU} = 1.5$, $m_2 = 1$, $m_3 = 2$, $\mu_{2P} = .5$, $\mu_{2B} = 3.0$, $\mu_{3B} = 36.0$, $3 > 2 > 9 > 8 > 5 > 4$, $7 > 6$

α	$N = 1$	2	3	4	5	6	
	U_1						$\bar{U}_1$
.99	.457	.707	.835	.902	.940	.962	1.
.958	.261	.466	.599	.685	.745	.789	1.
.95	.234	.426	.550	.635	.693	.732	.833
.90	.146	.278	.351	.392	.410	.417	.417
	A_2						
.99	1.00	1.29	1.59	1.85	2.07	2.25	
.958	1.00	1.23	1.46	1.69	1.89	2.07	
.95	1.00	1.21	1.44	1.65	1.84	2.00	
.90	1.00	1.19	1.36	1.49	1.56	1.60	
	A_3						
.99	79.0	81.4	84.9	88.9	92.5	95.7	
.958	79.0	80.7	90.0	102.	116.	131.	
.95	79.0	80.8	91.3	103.	118.	136.	
.90	79.0	80.1	96.7	118.	146.	178.	
	A						
.99	1.78	2.09	2.42	2.72	2.98	3.19	
.958	4.28	4.56	5.18	5.92	6.69	7.47	
.95	4.90	5.19	5.93	6.74	7.67	8.70	
.90	8.80	9.08	10.9	13.2	16.1	19.2	

When stage 2 or 3 is the limiting stage, A increases more rapidly as N increases than when stage 1 is the limiting stage. This is shown in Fig. 8. Finally, the increase of A with decreasing α is more pronounced for larger values of N (see Fig. 9), and the decrease of A with increasing μ_{CPU} is more pronounced for larger values of N (see Fig. 10).

6. Extensions and limitations of the model. The model in § 3 can be extended to incorporate a simple representation of CPU overhead. We introduce a new service type 1′ corresponding to CPU overhead activities with $s(1') = 1$. The service transition diagram describing the sequence of services required by a single program is shown in Fig. 11 (compare with Fig. 2). The priority ordering among CPU services is $1' > 1$.

In developing the model in § 3, we assumed that when a page or block is to be replaced, it is always moved down the hierarchy to the next level. However, in managing the hierarchy, a bit associated with the page or block could be examined to determine if the page or block had been modified. The page or block is moved down to the next level only if it had been modified. This can be represented in the

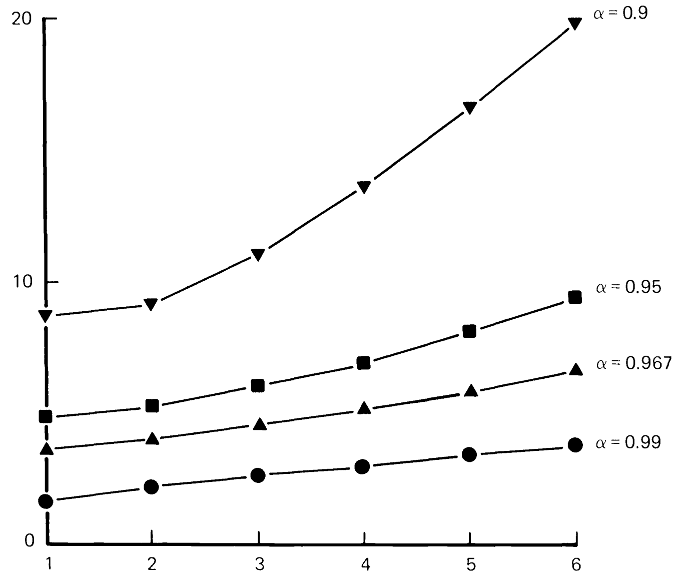


FIG. 8. *A vs. N for Table* 7

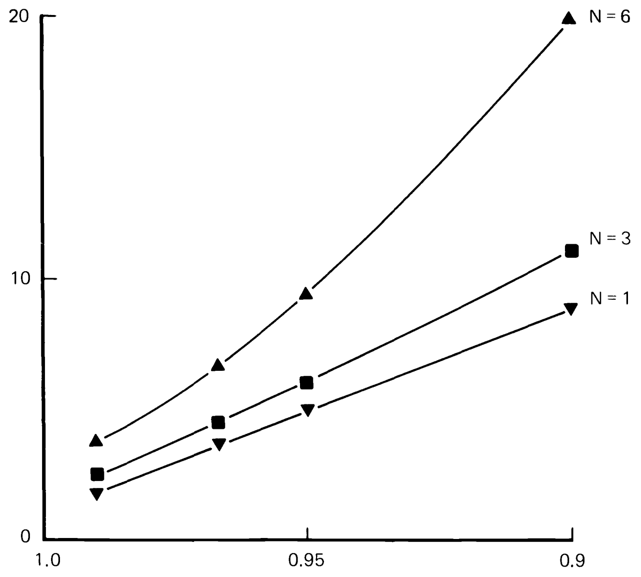


FIG. 9. *A vs. α for Table* 7

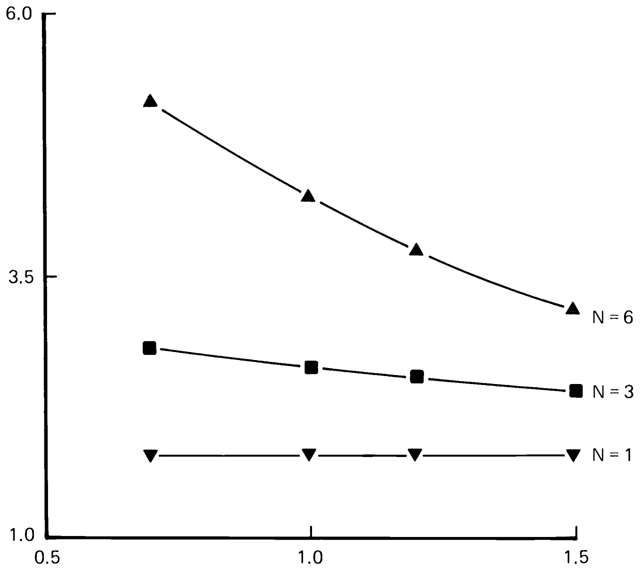


FIG. 10. *A vs.* μ_{CPU} *for Tables* 5–8 *and* $\alpha = 0.99$

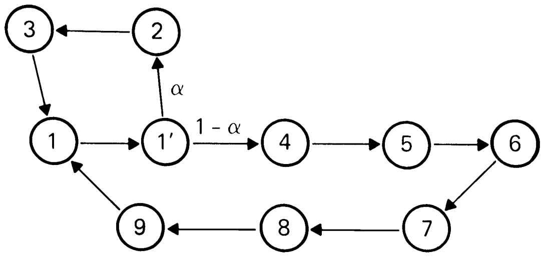


FIG. 11. *Service transition diagram incorporating* CPU *overhead*

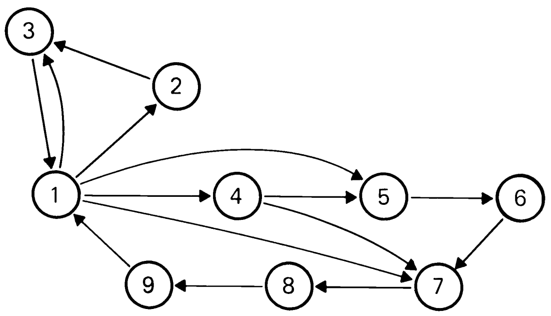


FIG. 12. *Service transition diagram incorporating the effect of not always updating*

service transition diagram of Fig. 12, where the arrows indicate transitions of nonzero probability, and suitable probabilities are chosen for these transitions.

Both the extensions just discussed increase the computational complexity of the model, the former by increasing the cardinality of the state space and the latter by destroying some of the sparseness in the transition intensity matrix of the Markov process. Note, however, that by using (9) and (10) we could still obtain a simple upper bound on the CPU work rate in terms of the model parameters, and we could identify the limiting stages in terms of these parameters.

Two principal limitations of the model are the representations of the program load and of the rotating storage facilities. Recent statistical analyses of address traces suggest that successive CPU service times for a single program might be more adequately modeled as a semi-Markov process than as i.i.d. random variables [5]. In addition, the empirical CPU service time distributions obtained from these traces are more skewed than the exponential distribution. There is no evidence to suggest that a sequence of Bernoulli trials determines whether successive faults out of level 1 are hits to level 2 or hits to level 3. The rotating storage facilities comprising a level of the hierarchy form a complicated queueing structure [1] which we approximate by a multiserver exponential stage in the model. More accurate representations in either of these two areas pose extreme analytic and/or computational difficulties. This suggests Monte Carlo simulation of models incorporating more accurate representations of program load and storage facilities. The simpler models may then be used in a control variable method of improving simulation efficiency [3].

Appendix. Consider a model in our class as it evolves in time. Note that the kth service of type l to start is not necessarily the kth service of type l to complete, since we consider multiserver stages. Let T_l^k denote duration of the kth service of type l to start. The *service time sequence* for service l, $\{T_l^k : k \geqq 1\}$, is a sequence of i.i.d. nonnegative random variables, each with mean μ_l.

We define integer-valued director random variables ψ_l^k, where $\psi_l^k = j$, $1 \leqq j \leqq L$, if and only if at the kth service completion of type l, the customer just served next requires service j. The sequence $\{\psi_l^k : k \geqq 1\}$, called the *director sequence* for service l, is a sequence of i.i.d. random variables with Prob $\{\psi_l = j\} = p_{lj}$ for all $k \geqq 1$. The L service time sequences and the L director sequences form a collection of $2L$ mutually independent sequences.

The service time sequence for service l is composed of $m_{s(l)}$ disjoint subsequences, where $\{T_l^{r_j} : j \geqq 1\}$ is the subsequence of service l service times provided by the rth server at stage $s(l)$, $1 \leqq r \leqq m_{s(l)}$. It is straightforward to show that $\{T_l^{r_j} : j \geqq 1\}$ is a sequence of i.i.d. random variables, each distributed according to the service time distribution for service l. Note that the kth service of type l to start at server r of stage $s(l)$ is also the kth service of type l to complete at this server. We also define random variables ϕ_{lj}^k by

$$\phi_{lj}^k = \begin{cases} 1 & \text{if} \quad \psi_l^k = j, \\ 0 & \text{if} \quad \psi_l^k \neq j. \end{cases}$$

The sequence $\{\phi_{lj}^k : k \geqq 1\}$ is a sequence of i.i.d. binary-valued random variables with Prob $\{\phi_{lj}^k = 1\} = p_{lj}$. We now prove Lemmas 1–3.

Proof of Lemma 1. For each l,

$$\operatorname{Prob}\{N_l(t,v_0) \geqq Km_{s(l)}\} \leqq \operatorname{Prob}\left\{\bigcup_{r=1}^{m_{s(l)}} \text{server } r \text{ completes at least } K \text{ services in } [0,t]\right\}$$

$$\leqq \sum_{r=1}^{m_{s(l)}} \operatorname{Prob}\{\text{server } r \text{ completes at least } K \text{ services in } [0,t]\}$$

$$\leqq \sum_{r=1}^{m_{s(l)}} \operatorname{Prob}\{T_l^{r_1} + T_l^{r_2} + \cdots + T_l^{r_K} \leqq t\},$$

which tends to zero in the limit as $K \to \infty$ from the strong law of large numbers, since $\mu_l > 0$. Thus (2) holds.

Let $M(t,v_0) = \sum_{l=1}^{L} M_l(t,v_0)$. We now show $\lim_{t\to\infty} M(t,v_0) = \infty$ P1. For all t,

$$\sum_{l=1}^{L}\sum_{k=1}^{M_l(t,v_0)} T_l^k \geqq \sum_{l=1}^{L} W_l(t,v_0) \geqq t.$$

Assume that for some $K > 0$,

$$\operatorname{Prob}\{\lim_{t\to\infty} M(t,v_0) < K\} = \delta > 0.$$

But $\lim_{t\to\infty} M(t,v_0) < K$ implies $\sum_{l=1}^{L}\sum_{k=1}^{K} T_l^K \geqq t$ for all t, so that

$$\operatorname{Prob}\left\{\sum_{l=1}^{L}\sum_{k=1}^{K} T_l^k = \infty\right\} \geqq \delta.$$

This is impossible, since the service times are finite-valued P1. Thus, $\lim_{t\to\infty} M(t,v_0) = \infty$ P1.

It follows that for some l, $\lim_{t\to\infty} M_l(t,v_0) = \infty$ P1, and from (1), $\lim_{t\to\infty} N_l(t,v_0) = \infty$ P1. We now show this holds for all $j \neq l$. Since the Markov chain over $1, 2, \cdots, L$ is irreducible, there exists a finite sequence $i_0, i_1, \cdots, i_I$ such that $i_0 = l$, $i_I = j$ and $p_{i_{n-1}i_n} > 0$, $1 \leqq n \leqq I$. Clearly,

$$N_{i_1}(t,v_0) \geqq \sum_{k=1}^{N_{i_0}(t,v_0)} \phi_{i_0i_1}^k - N,$$

where $\operatorname{Prob}\{\phi_{i_0i_1}^k = 1\} = p_{i_0i_1} > 0$. Thus, $\lim_{t\to\infty} N_{i_0}(t,v_0) = \infty$ P1 implies $\lim_{t\to\infty} N_{i_1}(t,v_0) = \infty$ P1. Continuing in this way, we see that $\lim_{t\to\infty} N_{i_I}(t,v_0) = \infty$ P1, which completes the proof of (3). Equation (4) follows from (3) and (1).

Proof of Lemma 2. For each j,

$$N_j(t,v_0) = \left(\sum_{l=1}^{L}\sum_{k=1}^{N_l(t,v_0)} \phi_{lj}^k\right) + b(t),$$

where $b(t)$ is uniformly bounded for all t. Thus

$$\frac{N_j(t,v_0)}{N_i(t,v_0)} = \left(\sum_{l=1}^{L} \frac{N_l(t,v_0)}{N_i(t,v_0)} \frac{1}{N_l(t,v_0)} \sum_{k=1}^{N_l(t,v_0)} \phi_{lj}^k\right) + \frac{b(t)}{N_i(t,v_0)}$$

(from Lemma 1, for each l, $N_l(t, v_0) < \infty$ P1 for t finite, and $N_l(t, v_0) > 0$ for t sufficiently large). Using (3) and the strong law of large numbers, we see that

$$\lim_{t\to\infty} \frac{N_j(t, v_0)}{N_i(t, v_0)} = \sum_{l=1}^{L} p_{lj} \lim_{t\to\infty} \frac{N_l(t, v_0)}{N_i(t, v_0)}.$$

Thus

$$\lim_{t\to\infty} N_j(t, v_0)/N_i(t, v_0) = \text{constant} \cdot \pi_j,$$

and setting $j = i$, we conclude that $1/\pi_i$ = constant. This completes the proof of Lemma 2.

Proof of Lemma 3. Let $N_{l,r}(t, v_0)$ be the number of completions of service l by the rth server at stage $s(l)$ in $[0, t]$. Then,

$$\sum_{r=1}^{m_{s(l)}} N_{l,r}(t, v_0) = N_l(t, v_0).$$

Clearly, for each l,

$$\frac{N_l(t, v_0)}{t} \sum_{r=1}^{m_{s(l)}} \frac{N_{l,r}(t, v_0)}{N_l(t, v_0)} \frac{1}{N_{l,r}(t, v_0)} \sum_{k=1}^{N_{l,r}(t,v_0)} T_l^{r_k} \leqq \frac{W_l(t, v_0)}{t}$$

$$\leqq \frac{N_l(t, v_0)}{t} \frac{M_l(t, v_0)}{N_l(t, v_0)} \frac{1}{M_l(t, v_0)} \sum_{k=1}^{M_l(t,v_0)} T_l^k.$$

Using (3), (4) and the strong law of large numbers, we conclude that $\lim_{t\to\infty} W_l(t, v_0)/t$ exists P1 if and only if $\lim_{t\to\infty} N_l(t, v_0)/t$ exists P1, in which case (6) holds, and the proof is complete.

Acknowledgments. I wish to thank G. S. Shedler for discussions leading to the development of the staging hierarchy model. This work also benefited from discussions with R. L. Mattson and I. L. Traiger.

REFERENCES

[1] J. ABATE, H. DUBNER AND S. B. WEINBERG, *Queueing analysis of the IBM* 2314 *disk storage facility*, J. Assoc. Comput. Mach., 15 (1968), pp. 577–589.

[2] A. CHANG AND S. S. LAVENBERG, *Work-rates in closed queueing networks with general independent servers*, IBM Research Rep. RJ 989, San Jose, Calif., 1972.

[3] D. P. GAVER AND G. S. SHEDLER, *Control variable methods in the simulation of a model of a multiprogrammed computer system*, Naval Res. Logist. Quart., 18 (1971), pp. 435–450.

[4] W. S. JEWELL, *A simple proof of*: $L = \lambda W$, Operations Res., 15 (1967), pp. 1109–1116.

[5] P. A. W. LEWIS AND G. S. SHEDLER, *Empirically derived models for sequences of page references and for sequences of page exceptions*, IBM Research Rep. RJ 1138, San Jose, Calif., 1972.

[6] R. R. MUNTZ AND F. BASKETT, *Open, closed and mixed networks of queues with different classes of customers*, Tech. Rep. 33, Digital Systems Lab., Stanford Univ., Stanford, Calif., 1972.

[7] D. R. SLUTZ AND I. L. TRAIGER, *Determination of hit ratios for a class of staging hierarchies*, IBM Research Rep. RJ 1044, San Jose, Calif., 1972.

[8] V. L. WALLACE AND R. S. ROSENBERG, *RQA*-1, *the recursive queue analyzer*, Systems Engineering Lab. Tech. Rep. 2, Univ. of Mich., Ann Arbor, 1966.

[9] A. M. WOOLF, *Analysis and optimization of multiprogrammed computer systems using storage hierarchies*, Systems Engineering Lab. Tech. Rep. 53, Univ. of Mich., Ann Arbor, 1971.

SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

CONTINUOUS GROUP AVERAGING AND PATTERN CLASSIFICATION PROBLEMS*

J. C. DUNN†

Abstract. The relationship between pattern classification problems and the elementary theory of group invariants is considered. A general procedure for obtaining quantitative invariants by averaging functionals over the manifold of a continuous group is examined in some detail, and then applied to the classification of plane figures. Specifically, the Fourier transform of a plane figure is averaged over the one-parameter continuous group of dilatations to obtain a pair of interesting scale invariant transforms which tend to pick out corners and flat spots in the figure's boundary.

Key words. patterns, features, group invariants, averaging, plane figure, dilatation, Fourier transform, corners, sides

1. Introduction. The classical theory of groups contains the following elementary but far-reaching principle for constructing quantitative invariants under a transformation group, $\mathcal{G}$: to each object, s, in the domain[1] of $\mathcal{G}$, assign the average, $f^*(s)$, of the values assumed by an arbitrary functional on all the images of s under the transformations in $\mathcal{G}$. If $\mathcal{G}$ is a finite group and if "average" is interpreted in the ordinary way, then the invariance under $\mathcal{G}$ of the averaged functional, f^*, is an immediate consequence of the group axioms and the permutation symmetry of finite sums. However, even if $\mathcal{G}$ is a nondenumerably infinite group, it is frequently still possible to extend the averaging principle through the construction of integrals taken with respect to a certain class of measures on the manifold of $\mathcal{G}$. According to Weyl [1], an extension of this kind was first formulated by Hurwitz in 1897, and plays a part in the classical theory of compact Lie groups.

In recent years, the group averaging principle has appeared in the literature on pattern classification problems, which is not surprising if one considers (a) that "classifying a pattern" generally means detecting an equivalence, relative to a specified group of transformations, between the pattern in question and some member of a set of "canonical" patterns (i.e., representatives from the group-induced equivalence classes); and (b) that this kind of equivalence is generally established by computing and comparing a sufficient number of transformation-invariant properties or "features" (i.e., group invariants). In 1947, Pitts and McCulloch [2] applied finite group averaging to the design of artificial neuronal networks mimicking the function, and apparently to some extent, the structure of auditory and visual systems in the human brain. More recently, the same finite group averaging principle appears again in a book on perceptrons by Minsky and Pappert [3]. So far, however, little or no use has been made of continuous group averaging within the context of pattern classification. This paper indicates the possibilities in such an application by averaging Fourier transforms of plane

* Received by the editors April 18, 1972, and in revised form April 3, 1973.

† Department of Theoretical and Applied Mathematics, Cornell University, Ithaca, New York 14850.

[1] I.e., in the common domain of the transformations in $\mathcal{G}$.

figures over a one-parameter continuous group of dilatations (i.e., homothetic scale changes) to produce a pair of interesting scale invariant transforms.

It turns out that exact computations of the scale-averaged Fourier transforms are feasible on the class of polygonal figures, and the results show that the transform values are different from zero only if (a) the transform argument, ω, is orthogonal to a side of the polygon, or (b), ω is orthogonal to a line passing through the centroid of the polygon and one of its vertices. This behavior is explained in terms of the averaged transform's selective response to chordal width irregularities produced by corners and sides. Unfortunately, the method of computation employed for polygons does not carry over in a straightforward manner to figures with curved boundaries, and while certain pieces of evidence suggest that a selectivity for corners and flat spots probably occurs in the averaged Fourier transform of a general figure, the issue is not completely resolved here.

The results of this paper are presented without an evaluation of their technological or biological significance. It may or may not be practical to build a device which detects the presence of corners and flat spots in the boundary of plane figures by constructing approximations to scale-averaged Fourier transforms. Similarly, it may or may not be the case that wave-like signal patterns interacting in an animal nervous system can form something approximating a Fourier-like integral transform of sensory imputs. Still, it is interesting that a simple averaging process applied to a simple frequency spectrum representation tends to pick out precisely those features of a plane polygon which are also conspicuous to the human perceptual apparatus.

2. General considerations. The idea of classifying objects according to whether or not they are "connected" by some member of a given class of transformations, and the correlated idea of detecting the existence of such a connection through an examination of transformation invariant properties, both run through all of mathematics (especially geometry and topology).[2] In a fundamental way, these ideas are direct abstractions from primitive characteristics of the interaction between human perceptual mechanisms and the external world. Consequently, they are also of central importance in any theoretical study of pattern classification.

For example, each portion of Fig. 1 generates a different state of internal nervous activity in the brain, i.e., a different perception. Yet, what differences do exist can be completely canceled by suitable external manipulations of the things which are "causing" the perceptions, e.g., by tearing the page along the dashed lines and bringing each separate portion to the appropriate spatial location relative to the eye. Our perceptions of the objects in Fig. 1 have the common name, "square", precisely because they are obtainable one from another by a delimited class of actions called similarity transformations (composites of translations, rotations, and dilations).

Any square can be obtained from any other square by a similarity transformation, and to this extent, all squares are equivalent. Similarly, any equilateral triangle can be obtained from any other equilateral triangle by a similarity transformation, hence all equilateral triangles are equivalent in the same way. But

[2] Klein's "Erlanger program"; cf. [1, pp. 14–18] and [4, p. 125].

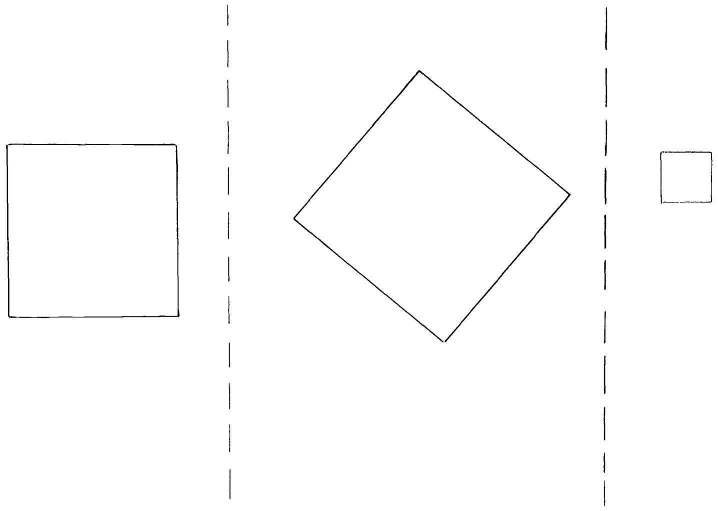


FIG. 1

no square can be obtained from any equilateral triangle by a similarity transformation; consequently squares and equilateral triangles are not (similarity) equivalent. In general, the group of similarity transformations divides the totality of plane figures into disjoint equivalence classes, of which "square" and "equilateral triangle" are but two examples.

A figure is recognizable as a square because it displays certain similarity-invariant properties which all squares have in common and which, in total, no other figure possesses. For instance, "number of corners in the boundary" is a figural property that survives any similarity transformation; evidently, this property is sufficient to distinguish between circles, squares, and equilateral triangles, up to a similarity transformation; however, it is not sufficient to distinguish between squares and rectangles, etc. Finer discrimination requires additional "elementary" invariant properties; total discrimination requires a *complete* set of invariant properties, capable of separating all the similarity-equivalence classes of figures in the plane.

The abstract concepts of group equivalence and group invariant are straightforward generalizations of the foregoing considerations. An account of these ideas can be found in basic texts on algebra and group theory, e.g., [1] and [4]; however, a brief discussion is included here for convenience.

Let S denote an arbitrary set of objects s, let $\mathcal{G}$ denote an arbitrary group of transformations $T: S \to S$, and let $=$ always signify equality (i.e., identity). $\mathcal{G}$ induces a relation $\equiv$ on S in the following way:

$$s_1 \equiv s_2 \Leftrightarrow s_2 = Ts_1 \quad \text{for some } T \in \mathcal{G}.$$

This relation is reflexive ($s \equiv s$ for all $s \in S$) because $\mathcal{G}$ must contain the identity transformation; it is symmetric ($s_1 \equiv s_2 \Rightarrow s_2 \equiv s_1$) because $\mathcal{G}$ must contain with every T its inverse transformation; and finally, it is transitive ($s_1 \equiv s_2$ and

$s_2 \equiv s_3 \Rightarrow s_1 \equiv s_3$) because $\mathscr{G}$ must be closed under its operation, viz., composition of transformations. Thus, $\equiv$ has all the basic abstract properties of an equivalence relation; like any equivalence relation, it divides S into disjoint equivalence classes, $E_s = \{s' \in S | s' \equiv s\}$. Evidently, $s \in E_s$ for all s in S, and $s_1 \not\equiv s_2 \Rightarrow E_{s_1} \cap E_{s_2} = \varnothing$ (empty set).

Let Y denote an arbitrary set (different from S, in general) and let f^* map S into Y. The function f^* is an *invariant* under $\mathscr{G}$ if and only if f^* is constant on each equivalence class E_s, or equivalently, if and only if

$$f^*(Ts) = f^*(s) \quad \text{for all } s \text{ in } S \text{ and all } T \text{ in } \mathscr{G}.$$

The function f^* is a *complete invariant* under $\mathscr{G}$ if and only if f^* is an invariant and f^* has different values on different equivalence classes, or, more precisely, if and only if $f^*(s_1) = f^*(s_2) \Leftrightarrow s_1 \equiv s_2$ ($\Leftrightarrow s_2 = Ts_1$ for some $T \in \mathscr{G}$); in this sense, a complete invariant "separates" the equivalence classes induced by $\mathscr{G}$.

Each group of transformations on S poses the problem of finding a complete invariant which characterizes equivalence under that group. A consideration of specific instances of this problem shows that complete invariants typically are built upon a substructure of many "simpler" but individually incomplete components which are, in effect, elementary invariant "features" of the objects in S. Thus, if Z is some arbitrary set and if $\alpha^*(\omega, \cdot): S \to Z$ is an invariant under $\mathscr{G}$ for each ω in some (not necessarily denumerable) index set, Ω, then

$$f^*(s) = \alpha^*(\cdot, s): \Omega \to Z$$

defines a function, $f^*: S \to Y = \{\text{set of functions}: \Omega \to Z\}$, which is also invariant under $\mathscr{G}$. Furthermore, if Ω is "sufficiently large", f^* may be a complete invariant even though the objects in Z are in some sense much simpler than the objects in S.

When Z is the real or complex field, $\alpha^*: S \to Z$ is called a *numerical* or *quantitative invariant* (e.g., "number of corners in the boundary" is an integer-valued similarity-invariant). Depending on the nature of S and $\mathscr{G}$, numerical invariants may provide a "natural" basis for a complete invariant.[3] In any event, the next section shows how significant numerical invariants may be obtained by the group averaging principle, and adapts this principle to the one-parameter continuous group of dilatations.

3. Group averaging principles and dilatation groups.

Let C denote the complex number system, let $\alpha(\cdot): S \to C$ denote an arbitrary functional on S, and let $\mathscr{G}$ be a *finite* group of transformations $T: S \to S$, with parametric representation $l \to T_l; l \in \{1, 2, \cdots, N\}$.

LEMMA 3.1. *If Q is any symmetric function from $C^N \to C$, then the rule*

$$\alpha^*(s) = Q(\alpha(T_1(s)), \alpha(T_2(s)), \cdots, \alpha(T_N(s))), \qquad s \in S,$$

defines an invariant functional under $\mathscr{G}$.

[3] Not always: e.g., algebraic topology investigates the equivalence of topological manifolds under the group of homeomorphic (one-to-one, onto, bicontinuous) transformations; here, the appropriate elementary invariants are "group-valued," i.e., they assign to each manifold certain homeomorphically invariant groups (homotopy and homology groups). In this case, Z is a set of *algebraic systems*, viz., groups (more precisely, isomorphic-equivalence classes of groups; cf. [5]).

Proof. Since $\mathscr{G}$ is a group, the correspondence $T_l \to T_l T_k$ is a bijection (one-to-one onto mapping) for each fixed $k \in \{1, \cdots, N\}$. Thus $\{T_1 T_k(s), \cdots, T_N T_k(s)\}$ is a permutation of $\{T_1(s), \cdots, T_N(s)\}$ for all $s \in S$, and since Q is symmetric,

$$\alpha^*(T_k(s)) = Q(\alpha(T_1 T_k(s)), \cdots, \alpha(T_N T_k(s))) = Q(\alpha(T_1(s)), \cdots, \alpha(T_N(s))) \tag{3.1}$$

for all $s \in S$, all $k \in \{1, \cdots, N\}$.

COROLLARY 3.1.1 (Group averaging principle). *The rule*

$$\alpha^*(s) = \frac{1}{N}\sum_{l=1}^{N} \alpha(T_l(s)), \qquad s \in S, \tag{3.2}$$

defines an invariant functional under $\mathscr{G}$.[4]

Corollary 3.1.1 can be extended in various ways to certain infinite groups. All such extensions obviously presuppose some meaningful notion of summation over infinite sets, e.g., a notion of integration. In this setting, analogues of (3.2) are provided by expressions of the following kind:

$$\frac{1}{\mu(\mathscr{G})}\int_{\mathscr{G}} \alpha(T(\alpha))\, d\mu(T), \qquad \mu(\mathscr{G}) < \infty, \tag{3.3a}$$

$$\int_{\mathscr{G}} \alpha(T(\alpha))\, d\mu(T), \qquad \mu(\mathscr{G}) \leqq \infty, \tag{3.3b}$$

$$\lim_{\mathscr{G}_n \uparrow \mathscr{G}} \frac{1}{\mu(\mathscr{G}_n)}\int_{\mathscr{G}_n} \alpha(T(\alpha))\, d\mu(T), \qquad \mu(\mathscr{G}) = \infty, \quad \mathscr{G}_{n+1} \supset \mathscr{G}_n, \quad \bigcup_{n=1}^{\infty} \mathscr{G}_n = \mathscr{G}, \tag{3.3c}$$

where the integrals are taken with respect to a measure function μ given on some suitable collection of "measurable" subsets of the group manifold $\mathscr{G}$. However, invariance of the functionals obtained from these expressions does not now follow automatically, but depends in a crucial way upon the measure μ. For example, if R is an arbitrary transformation in $\mathscr{G}$, then the correspondence, $T \to TR$, called a right "translation" of $\mathscr{G}$, is once again bijective. Therefore, with reference to (3.3a) and (3.3b),

$$\alpha^*(R(s)) \triangleq \int_{\mathscr{G}} \alpha(TR(s))\, d\mu(T) = \int_{\mathscr{G}} \alpha(T'(s))\, d\mu(T'R^{-1}) \tag{3.4}$$

However, the final step required for invariance, viz.,

$$\int_{\mathscr{G}} \alpha(T'(s))\, d\mu(T'R^{-1}) = \int_{\mathscr{G}} \alpha(T'(s))\, d\mu(T') \triangleq \alpha^*(s), \tag{3.5}$$

goes through if and only if the measure μ is preserved under right translations of $\mathscr{G}$.[5] Similarly, the invariance of functionals arising from (3.3c) also depends upon

[4] Corollary 3.1.1 is really a "summation principle," since the factor $1/N$ is irrelevant to the invariance of α^*. However, certain extensions of this result (including one considered in this paper) do depend in an essential way on averaging processes as opposed to summation processes.

the properties of μ, although right translation invariance is unnecessarily restrictive in this case and can be replaced by the weaker sufficient condition,

$$\mu(R(\mathscr{G}')) = J(R)\mu(\mathscr{G}'), \tag{3.6}$$

where $\mathscr{G}'$ is an arbitrary measurable subset of $\mathscr{G}$, $R(\mathscr{G}')$ is the right translation of $\mathscr{G}'$ under R (i.e., $R(\mathscr{G}') = \{V \in \mathscr{G} | V = TR, T \in \mathscr{G}'\}$, and $J(R)$ is a scalar depending on R.

For a broad class of transformation groups $\mathscr{G}$, (3.6) is satisfied by ordinary Lebesgue measure on a suitable parameter space of $\mathscr{G}$. In particular, this holds for any dilatation group and leads to a basic averaging principle for the construction of scale invariant functionals.

DEFINITION 3.1. $\mathscr{G}$ is called a *dilatation group* if and only if $\mathscr{G}$ is isomorphic to the group $(R_1^+, \cdot)$ of positive real numbers under multiplication "$\cdot$", i.e., if and only if there exists a bijection $T:(0, \infty) \to \mathscr{G}$ such that $T_{\sigma_1}T_{\sigma_2} = T_{\sigma_1 \cdot \sigma_2}$ for all $\sigma_1, \sigma_2 \in (0, \infty)$.

LEMMA 3.2. *Let $\mathscr{G} = \{T_\sigma : S \to S | \sigma \in (0, \infty)\}$ be a dilatation group and let μ denote ordinary Lebesgue measure on $(0, \infty)$. Furthermore, let $\alpha(T_\sigma(s))$ be locally integrable in σ for each $s \in S$, and let*

$$\bar{\alpha}^*(s) = \overline{\lim_{L\to\infty}} \left[\frac{1}{L} \operatorname{Re} \int_0^L \alpha(T_\sigma(s))\, d\mu(\sigma)\right] + i\, \overline{\lim_{L\to\infty}} \left[\frac{1}{L} \operatorname{Im} \int_0^L \alpha(T_\sigma(s))\, d\mu(\sigma)\right], \tag{3.7}$$

$$\underline{\alpha}^*(s) = \underline{\lim}_{L\to\infty} \left[\frac{1}{L} \operatorname{Re} \int_0^L \alpha(T_\sigma(s))\, d\mu(\sigma)\right] + i\, \underline{\lim}_{L\to\infty} \left[\frac{1}{L} \operatorname{Im} \int_0^L \alpha(T_\sigma(s))\, d\mu(\sigma)\right]. \tag{3.8}$$

Then for all $\lambda \in (0, \infty)$, and all $s \in S$,

$$\bar{\alpha}^*(T_\lambda(s)) = \bar{\alpha}^*(s), \tag{3.9}$$

$$\underline{\alpha}^*(T_\lambda(s)) = \underline{\alpha}^*(s), \tag{3.10}$$

i.e., $\bar{\alpha}^*(s)$ and $\underline{\alpha}^*(s)$ are invariant under $\mathscr{G}$.

Proof. Let $\sigma' = \sigma \cdot \lambda$. Then

$$\int_0^L \alpha(T_\sigma T_\lambda(s))\, d\mu(\sigma) = \int_0^L \alpha(T_{\sigma\cdot\lambda}(s))\, d\mu(\sigma) = \frac{1}{\lambda}\int_0^{\lambda L} \alpha(T_{\sigma'}(s))\, d\mu(\sigma').$$

Therefore,

$$\bar{\alpha}^*(T_\lambda(s)) = \overline{\lim_{L\to\infty}} \left[\frac{1}{\lambda L} \operatorname{Re} \int_0^{\lambda L} \alpha(T_{\sigma'}(s))\, d\mu(\sigma')\right] + i\, \overline{\lim_{L\to\infty}} \left[\frac{1}{\lambda L} \operatorname{Im} \int_0^{\lambda L} \alpha(T_{\sigma'}(s))\, d\mu(\sigma')\right]$$
$$= \bar{\alpha}^*(s).$$

Equation (3.10) follows in the same way.

4. Plane figures and scale invariant Fourier transform averages. The balance of this paper consists of a specific application of the foregoing considerations within the context of a "pattern classification" problem. From now on, S will be a set of

[5] Invariant measure is the analogue of the equal weight measure inherent in the finite sum (3.2); cf. [1].

In the case of compact finite-dimensional Lie groups, Weyl [1] describes a simple and systematic method of obtaining invariant measures from ordinary Lebesgue measure.

plane figures, $\alpha(\omega;\cdot):S \to C$ will give the value of the Fourier transform of $s \in S$ as ω ranges over $\Omega = R_2$, and $\mathscr{G}$ will be a particular dilatation group of transformations, $T:S \to S$, described shortly. To begin with, the following definitions are required.

DEFINITION 4.1. A *plane figure* is any nonempty, compact (i.e., closed and bounded), Lebesgue measurable subset of R_2.

DEFINITION 4.2. Let $\bar{x}(s) = (\bar{x}_1(s), \bar{x}_2(s))$ denote the centroid of a plane figure s. Then the rule

$$\phi(x;s) = \begin{cases} 1, & x + \bar{x}(s) \in s, \\ 0, & x + \bar{x}(s) \notin s, \end{cases} \tag{4.1}$$

defines the *centroidal characteristic function* of s as x ranges over R_2 and the rule

$$\alpha(\omega;s) = \int_{R_2} \phi(x;s)\, e^{i\langle\omega,x\rangle}\, d\mu_2(x), \qquad \omega \in R_2, \tag{4.2}$$

defines the (centroidal) *Fourier transform* of s, where $\langle\omega, x\rangle$ = the standard inner product on $R_2 = (\omega_1 x_1 + \omega_2 x_2)$ and $\mu_2(x)$ = Lebesgue product measure on R_2.

Notice that the support of the characteristic function $\phi(\cdot\,;s)$ is simply the image of s under a translation that takes the centroid of s to the origin. It follows that $\phi(\cdot\,;s)$ and the associated Fourier transform $\alpha(\cdot\,;s)$ are automatically invariant under the similarity subgroup of translations.[6] For present purposes, however, translation invariance of the Fourier transform is merely an incidental benefit of Definition 4.2; the principal virtue of this definition is that it leads to the fewest complications later on when averages are taken over the dilatation group described below.

For all $\sigma \in (0, \infty)$, the correspondence

$$x \to \bar{x}(s) + \sigma\cdot[x - \bar{x}(s)] \tag{4.3}$$

produces a linear, and therefore continuous, map of R_2 onto itself. Since compactness and measurability are preserved under continuous maps, it follows that the set

$$T_\sigma(s) = \{y = \bar{x}(s) + \sigma[x - \bar{x}(s)] \mid x \in s\} \tag{4.4}$$

belongs to S whenever $s \in S$; i.e., (4.4) describes a transformation $T_\sigma: S \to S$. Evidently T_σ changes the "size" of s, without altering its "shape", "orientation", or the location of its centroid. For example,

$$\begin{aligned} \bar{x}(T_\sigma(s)) &= \frac{\int_{T_\sigma(s)} y\, d\mu_2(y)}{\int_{T_\sigma(s)} d\mu_2(y)} = \frac{\int_s (\bar{x}(s) + \sigma[x - \bar{x}(s)])\sigma^2\, d\mu_2(x)}{\int_s d\mu_2(x)} \\ &= \bar{x}(s), \qquad \text{for all } \sigma \in (0,\infty), \text{ all } s \in S. \end{aligned} \tag{4.5}$$

Furthermore, we have the following lemma.

LEMMA 4.1. *As σ ranges over $(0, \infty)$, the transformations $T_\sigma: S \to S$ defined by* (4.4) *form a dilatation group $\mathscr{G}_D$.*

Proof. Definition 3.1 requires that the correspondence, $\sigma \to T_\sigma$, be an isomorphism on $(R_1^+, \cdot)$.

[6] Not to be confused with the "right translations of a group" considered in § 3.

Let $s = \{x, -x\}$, where x is an arbitrary nonzero vector in R_2. Then for $\sigma, \lambda \in (0, \infty)$,

$$T_\sigma(s) = \{\sigma \cdot x, -\sigma \cdot x\}, \quad T_\lambda(s) = \{\lambda \cdot x, -\lambda \cdot x\}; \tag{4.6}$$

consequently, $T_\sigma(s) = T_\lambda(s) \Rightarrow \sigma = \lambda$; it follows that $T_\sigma = T_\lambda \Rightarrow \sigma = \lambda$, i.e., the correspondence $\sigma \to T_\sigma$ is bijective.

Furthermore, for general $s \in S$, it follows easily from (4.4) that

$$T_\sigma T_\lambda(s) = T_{\sigma \cdot \lambda}(s). \tag{4.7}$$

Thus, $\sigma \to T_\sigma$ is an isomorphism on $(R_1^+, \cdot)$.

The following self-evident lemma leads to a useful representation for the Fourier transform of $T_\sigma(s)$.

LEMMA 4.2. *For all* $\sigma \in (0, \infty)$, *and all* $s \in S$,

$$\phi(y; T_\sigma(s)) = \phi\left(\frac{y}{\sigma}; s\right), \qquad y \in R_2. \tag{4.8}$$

LEMMA 4.3. *For all* $\sigma \in (0, \infty)$, $s \in S$, *the Fourier transform of* $T_\sigma(s)$ *is given by*

$$\alpha(\omega; T_\sigma(s)) = \sigma^2 \int_{R_2} \phi(x; s)\, e^{i\sigma\langle \omega, x\rangle}\, d\mu_2(x). \tag{4.9}$$

Proof. From Definition 4.2 and Lemma 4.2,

$$\begin{aligned} \alpha(\omega; T_\sigma(s)) &= \int_{R_2} \phi(y; T_\sigma(s))\, e^{i\langle \omega, y\rangle}\, d\mu_2(y) \\ &= \int_{R_2} \phi\left(\frac{y}{\sigma}; s\right) e^{i\langle \omega, y\rangle}\, d\mu_2(y). \end{aligned} \tag{4.10}$$

Let $x = y/\sigma$. Then $\langle \omega, y\rangle = \sigma\langle \omega, x\rangle, d\mu_2(y) = \sigma^2\, d\mu_2(x)$ and (4.9) follows from (4.10).

COROLLARY 4.3.1. *For each fixed* $\omega \in R_2$ *and* $s \in S$, $\alpha(\omega; T_\sigma(s))$ *is continuous in* σ, *and therefore locally integrable on* $(0, \infty)$.

The final lemma of § 3 now gives the following theorem.

THEOREM 4.1. *Let* $\bar{\alpha}^*(\omega; s)$ *and* $\underline{\alpha}^*(\omega; s)$ *denote, respectively, the* lim sup *and* lim inf *of*

$$\frac{1}{L}\int_0^L \alpha(\omega; T_\sigma(s))\, d\mu_1(\sigma) = \frac{1}{L}\int_0^L \sigma^2 \int_{R_2} \phi(x; s)\, e^{i\sigma\langle \omega, x\rangle}\, d\mu_2(x)\, d\mu_1(\sigma) \tag{4.11}$$

as $L \to \infty$.[7] *For each* $\omega \in R^2$, *the corresponding functionals* $\bar{\alpha}^*(\omega; \cdot)$ *and* $\underline{\alpha}^*(\omega; \cdot)$ *from S into the extended complex plane are invariant under the group* $\mathscr{G}_D$.

Proof. Lemmas 3.2, 4.1, and Corollary 4.3.1 give the result.

For elementary polygonal figures, such as rectangles, rhombi, etc., it is feasible to compute $\bar{\alpha}^*$ and $\underline{\alpha}^*$ from (4.11) by writing the Fourier transform as an iterated integral over the coordinates of $x = (x_1, x_2)$.[8] These simple examples show

[7] In the sense of (3.7) and (3.8).

[8] This procedure is straightforward but not well-suited for general figures. A different approach is employed later on.

that $\bar{\alpha}^*$ and $\underline{\alpha}^*$ are nontrivial invariants (i.e., not constant on all of S). They also lead to an interesting conjecture about the behavior of $\bar{\alpha}^*$ and $\underline{\alpha}^*$ for general figures, namely, that these invariants have (i) a common, nonzero, real value only if ω is perpendicular to a ray passing from the centroid through a corner in the boundary of s, (ii) have distinct complex values or a common infinite "value" only if ω is perpendicular to a flat edge in the boundary of s, (iii) have the common value zero for all remaining $\omega \neq 0$. It turns out that this conjecture is correct for the general *polygonal* figure, and derives from a still more fundamental sensitivity of $\bar{\alpha}^*$ and $\underline{\alpha}^*$ to chordal width irregularities. The proof is developed in the next series of lemmas.

The following result shows that $\bar{\alpha}^*$ and $\underline{\alpha}^*$ are determined for all $\omega \neq 0$ by their values on the unit circle.

LEMMA 4.4. *For every* $k \in (0, \infty)$,

$$\bar{\alpha}^*(k\omega; s) = (1/k^2)\bar{\alpha}^*(\omega; s), \tag{4.12}$$
$$\underline{\alpha}^*(k\omega; s) = (1/k^2)\underline{\alpha}^*(\omega; s).$$

Proof. From Lemma 4.3 and the definition of $\bar{\alpha}^*$,

$$\bar{\alpha}^*(k\omega; s) = \overline{\lim_{l \to \infty}} \frac{1}{L} \int_0^L \sigma^2 \int_{R_2} \phi(x; s)\, e^{i\sigma\langle k\omega, x\rangle}\, d\mu_2(x)\, d\mu_1(\sigma).$$

Put $\sigma' = k\sigma$. Then

$$\bar{\alpha}^*(k\omega; s) = \frac{1}{k^2} \overline{\lim_{L \to \infty}} \frac{1}{kL} \int_0^{kL} (\sigma')^2 \int_{R_2} \phi(x; s)\, e^{i\sigma'\langle \omega, x\rangle}\, d\mu_2(x)\, d\mu_1(\sigma')$$
$$= \frac{1}{k^2} \bar{\alpha}^*(\omega; s).$$

The conclusion for $\underline{\alpha}^*(\omega; s)$ follows in the same way.

In view of Lemma 4.4, nothing is lost by restricting $\bar{\alpha}^*(\,\cdot\,; s)$ and $\underline{\alpha}^*(\,\cdot\,; s)$ to the unit circle, i.e., by limiting ω-values to unit vectors of the form

$$e(\theta) = (\cos\theta, \sin\theta), \qquad \theta \in [0, 2\pi). \tag{4.13}$$

More precisely, the original family of quantitative invariants, $\bar{\alpha}^*(\omega; s), \underline{\alpha}^*(\omega; s)$, for $\omega \in R_2$, and its subfamily, $\bar{\alpha}^*(e(\theta); s), \underline{\alpha}^*(e(\theta); s)$, for $\theta \in [0, 2\pi)$, have exactly the same power to discriminate among the equivalence classes induced by the dilatation group, $\mathscr{G}_D$. This is intuitively satisfying, if one considers that the boundary points of "ordinary" plane figures form one-dimensional manifolds, and that these manifolds, in effect, determine the figures to which they belong. It is therefore reasonable to suspect that a manifold of quantitative figural invariants is "unnecessarily large" if its dimension exceeds unity.

The problem now is to find an effective technique for computing $\bar{\alpha}^*(\theta; s)$ and $\underline{\alpha}^*(\theta; s)$[9] for general s, or at least for general polygonal figures. As noted earlier, the standard approach of treating the Fourier transform as an iterated integral over x_1 and x_2 is workable only for simple polygonal figures, and even here, it obscures the underlying reason for the observed behavior of $\bar{\alpha}^*$ and $\underline{\alpha}^*$. In order to

[9] From now on, it will be convenient to replace $e(\theta)$ by θ in the arguments of α, $\bar{\alpha}$, and $\underline{\alpha}^*$.

understand this behavior, it is necessary to compute the Fourier transform, $\alpha(\theta; T_\sigma(s))$, as an iterated integral with respect to an orthogonal coordinate frame attached to the unit vector $e(\theta)$. When this is done, it becomes clear that the behavior of $\bar{\alpha}^*$ and $\underline{\alpha}^*$ is governed by singularities in the figural width variation along a line parallel to $e(\theta)$.

For fixed $\theta \in [0, 2\pi)$ and for fixed $\xi \in R_1$, the set of points

$$L(\xi, \theta) = \{x \in R_2 | x = \xi e(\theta) + \eta e(\theta + \pi/2);\ \eta \in R_1\} \tag{4.14}$$

is a line perpendicular to $e(\theta)$ and passing through the point $\xi e(\theta)$. The intersection of this line with the support of ϕ for a given figure, s, is either empty or forms a replica of a chord of s. Evidently, the "length" of this chord is simply the Lebesgue measure of the set (cf. Def. 4.2 and Fig. 2)

$$\{\eta \in R_1 | \phi(\xi e(\theta) + \eta e(\theta + \pi/2); s) = 1\}.$$

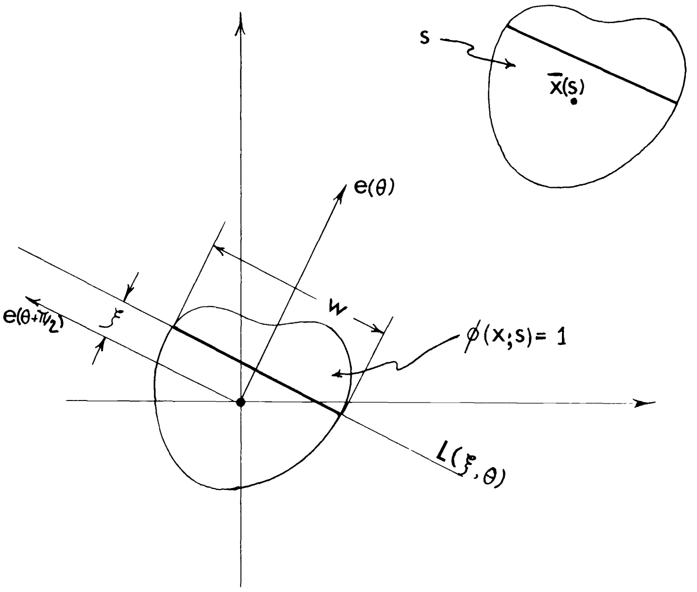


FIG. 2

DEFINITION 4.3. Let μ_1 = ordinary Lebesgue measure on R_1. Then the rule

$$W(\xi, \theta; s) = \mu_1\{\eta | \phi(\xi e(\theta) + \eta e(\theta + \pi/2); s) = 1\} \tag{4.15}$$

defines the *chordal width function* for a plane figure s.

Definition 4.3 is motivated by the following lemma and its consequences.

LEMMA 4.5.

(a) *For all* θ, s, $W(\cdot, \theta; s): R_1 \to R_1$ *is nonnegative, bounded, measurable, and has compact support.*

(b) *For all* $\sigma \in (0, \infty)$, *the Fourier transform of* $T_\sigma(s)$ *is given by*

$$\alpha(\theta; T_\sigma(s)) = \sigma^2 \int_{R_1} W(\xi, \theta; s)\, e^{i\sigma\xi}\, d\mu_1(\xi), \tag{4.16}$$

i.e., the (two-dimensional) Fourier transform of $T_\sigma(s)$ is gotten from the product of σ^2 with the (one-dimensional) Fourier transform of $W(\cdot, \theta; s)$.

(c) *The invariants $\bar{\alpha}^*(\theta; s)$ and $\underline{\alpha}^*(\theta; s)$ are, respectively, the $\overline{\lim}$ and $\underline{\lim}$ of*

$$(4.17) \qquad \frac{1}{L}\int_0^L \sigma^2 \int_{R_1} W(\xi, \theta; s)\, e^{i\sigma\xi}\, d\mu_1(\xi)\, d\mu_1(\sigma)$$

as $L \to \infty$.

Proof. (a) Definitions 4.1 and 4.3 give the result.

(b) Let

$$(4.18) \qquad x = \xi e(\theta) + \eta e(\theta + \pi/2),$$

i.e., $x_1 = \xi \cos\theta - \eta \sin\theta$ and $x_2 = \xi \sin\theta + \eta \cos\theta$. Then $\langle e(\theta), x\rangle = \xi$, and Lemma 4.3 gives

$$(4.19) \qquad \alpha(\theta; T_\sigma(s)) = \sigma^2 \int_{R_2} \phi(x(\xi, \eta, \theta); s)\, e^{i\sigma\xi}\, d\mu_2(x(\xi, \eta, \theta)).$$

Furthermore, for each θ, the transformation, $(\xi, \eta) \leftrightarrow (x_1, x_2)$, defined by (4.18), is orthogonal. Consequently, it preserves the Lebesgue product measure μ_2, and thus

$$\begin{aligned} &\int_{R_2} \phi(x(\xi, \eta, \theta); s)\, e^{i\sigma\xi}\, d\mu_2(x(\xi, \eta, \theta)) \\ (4.20) \qquad &= \int_{R_2} \phi(x(\xi, \eta, \theta); s)\, e^{i\sigma\xi}\, d\mu_2(\xi, \eta) \\ &= \int_{R_1} e^{i\sigma\xi}\left(\int_{R_1} \phi(x(\xi, \eta, \theta); s)\, d\mu_1(\eta)\right) d\mu_1(\xi). \end{aligned}$$

Equation (4.16) now follows immediately from (4.19), (4.20), and Definition 4.3.

(c) Equations (4.11) and (4.16) give the desired result.

Lemma 4.5 establishes the desired relationship between the chordal width function W, and the invariants $\bar{\alpha}^*$ and $\underline{\alpha}^*$. The following result shows that this relationship is either trivial or is dominated by singularities in the derivative, $W' = (d/d\xi)W(\cdot, \theta; s)$.

LEMMA 4.6. *If $W'(\cdot, \theta; s)$ is absolutely continuous on R_1, then $\bar{\alpha}^*(\theta; s) = \underline{\alpha}^*(\theta; s) = 0$.*

Proof. W has compact support; therefore, for some $M > 0$, $|\xi| \geqq M \Rightarrow W(\xi, \theta; s) = W'(\xi, \theta; s) = W''(\xi, \theta; s) = 0$. Furthermore, since W' is absolutely continuous, W'' exists almost everywhere and belongs to $\mathscr{L}^1(-\infty, \infty)$. Consequently, two successive integrations by parts give

$$\begin{aligned} \int_{R_1} W(\xi, \theta; s)\, e^{i\xi\sigma}\, d\mu_1(\xi) &= \int_{-M}^{M} W(\xi, \theta; s)\, e^{i\xi\sigma}\, d\mu_1(\xi) \\ &= -\frac{1}{\sigma^2}\int_{-M}^{M} W''(\xi, \theta; s)\, e^{i\xi\sigma}\, d\mu_1(\xi) \\ &\triangleq -\tilde{W}(\sigma)/\sigma^2, \end{aligned}$$

where $\tilde{W}$ is continuous on R_1 and $\lim_{\sigma\to\infty} \tilde{W}(\sigma) = 0$ (cf. [6]). It follows that

$$\lim_{L\to\infty} \frac{1}{L} \int_0^L \sigma^2 \int_{R_1} W(\xi, \theta; s)\, e^{i\xi\sigma}\, d\mu_1(\xi)\, d\mu_1(\sigma) = \lim_{L\to\infty} \frac{1}{L} \int_0^L -\tilde{W}(\sigma)\, d\mu_1(\sigma) = 0.$$

The next section explores the causes and consequences of singularities in the ξ-derivative of W for "ordinary" plane figures, and in particular, for polygonal figures.

5. Ordinary plane figures and polygonal figures. Definition 4.1 embraces many pathological sets which do not conform to the intuitive concept of what a plane figure ought to be. Thus, it is possible to find bounded measurable sets with chordal width functions W, which for certain values of θ are everywhere discontinuous on their support or which oscillate infinitely often or which are continuous with a derivative nowhere, etc., all within the broad characterization of W given by Lemma 4.5. This kind of set is essentially uninteresting for present purposes; it is excluded from the following subclass of "ordinary" plane figures.

DEFINITION 5.1. A figure s is an *ordinary* plane figure if and only if

(a) s consists of a simple (i.e., non-self-intersecting), closed, piecewise twice continuously differentiable Jordan curve and its interior, or

(b) s consists of a finite union or difference of sets satisfying condition (a).

Since the subclass of ordinary plane figures is mapped onto itself by any similarity transformation, and in particular by any dilatation, it is possible to exclude more general plane figures from further consideration.

If s is an ordinary plane figure, then the singularities of its chordal width function fall into three main categories:

(I) jump discontinuities in $W(\cdot, \theta; s)$;

(II) "corners" in $W(\cdot, \theta; s)$, i.e., points of continuity ξ, where the left and right sided ξ-derivatives of W are finite but unequal;

(III) points of continuity ξ, where $W(\cdot, \theta; s)$ has an infinite right or left sided ξ-derivative.

Jump discontinuities in W can result from flat spots, or "sides", in the boundary of s (or more precisely, the boundary of ϕ's support), as illustrated by Fig. 3; corners

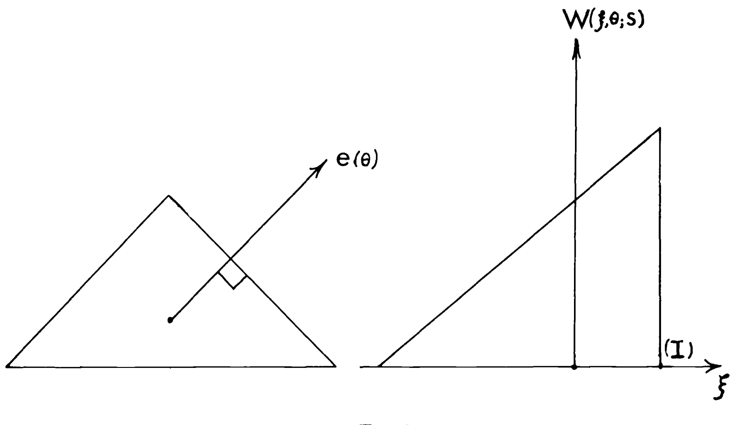


FIG. 3

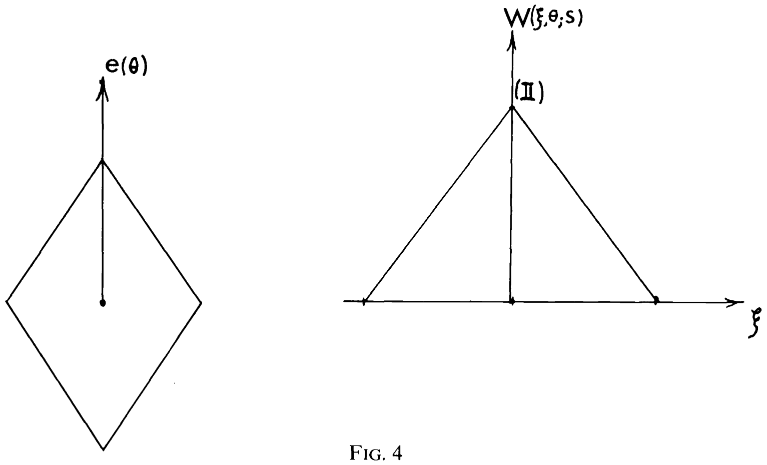


FIG. 4

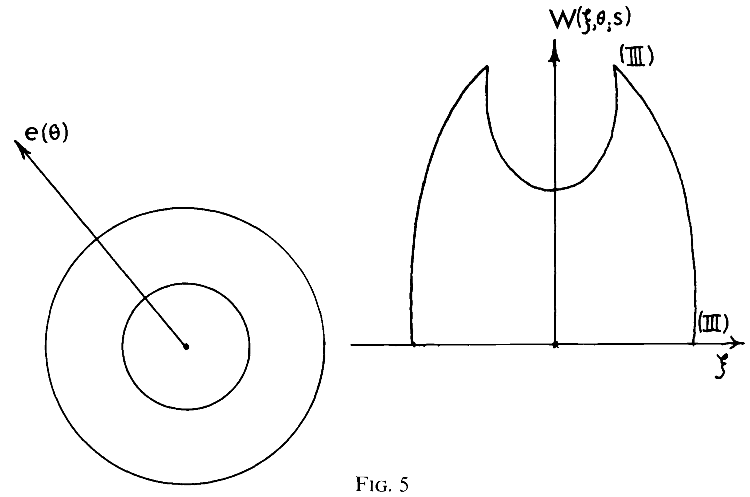


FIG. 5

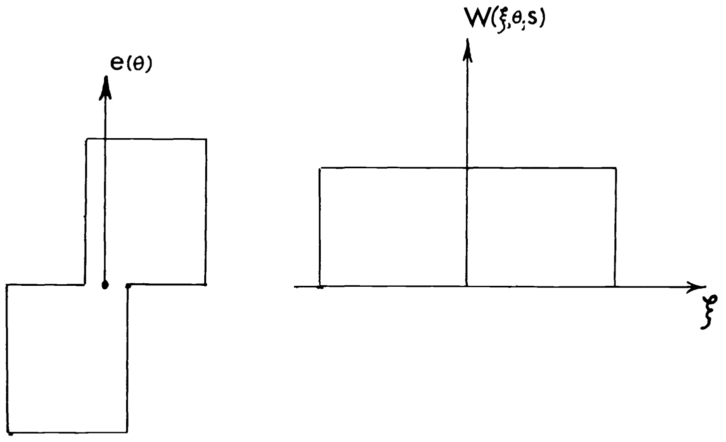


FIG. 6

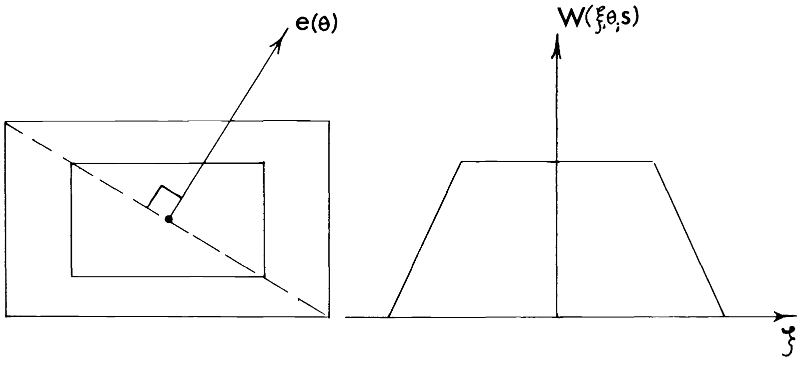


FIG. 7

in W typically reflect corners in the boundary of s (Fig. 4); and finally, infinite right or left sided derivatives of W can occur when the line $L(\xi, \theta)$, as in (4.14), is tangent to the boundary of s (Fig. 5). It should be noted, however, that flat spots or corners may exist in the boundary of s without consequential irregularities in W (Figs. 6 and 7). This point will receive further attention in the next section; for the moment, the main objective is to assess the influence of W's singularities on the behavior of $\bar{\alpha}^*$ and $\underline{\alpha}^*$. The following results go part of the way toward making this determination.

THEOREM 5.1. *Suppose that $W(\cdot, \theta; s)$ has a (distributional) derivative of the following form*:

$$W'(\xi, \theta; s) = w'(\xi, \theta; s) + \sum_{k=1}^{N} [W'(\xi_k(\theta, s))]h(\xi - \xi_k) + \sum_{k=1}^{N} [W(\xi_k(\theta, s))]\delta(\xi - \xi_k), \tag{5.1}$$

where

$w'(\cdot, \theta; s) =$ *absolutely continuous function*,

$\{\xi_1(\theta, s) < \cdots < \xi_N(\theta, s)\} =$ *finite subset of W's support*,

$h(\cdot) =$ *Heaviside unit step function*,

$\delta(\cdot) =$ *Dirac delta distribution*,

$[W(\xi_k(\theta, s))] =$ *jump in W across $\xi_k(\theta, s)$*,

$[W'(\xi_k(\theta, s))] =$ *jump in W' across $\xi_k(\theta, s)$*.

Then

$$\frac{1}{L}\int_0^L \alpha(\theta; T_\sigma(s))\, d\mu(\sigma) = \frac{iL}{2}[W(0)] - [W'(0)] + \sum_{\xi_k \neq 0} \frac{[W(\xi_k(\theta, s))]}{\xi_k(\theta, s)} e^{i\xi_k(\theta,s)L} + \frac{1}{L}\left\{ i \sum_{\xi_k \neq 0} \left(\frac{[W(\xi_k(\theta, s))] + \xi_k(\theta, s)[W'(\xi_k(\theta, s))]}{(\xi_k(\theta, s))^2} \right) \cdot (e^{i\xi_k(\theta,s)L} - 1) \right. \tag{5.2}$$

(5.2 cont.)
$$-\int_0^L \int_{R_1} \omega''(\xi, \theta; s)\, e^{i\xi\sigma}\, d\mu(\xi)\, d\mu(\sigma)\Bigg\},$$

where the first two terms are absent if $0 \notin \{\xi_1(\theta, s), \cdots, \xi_N(\theta, s)\}$.

Proof. By a straightforward application of the integration by parts procedure used in the proof of Lemma 4.6, and the distributional identity, $(d/d\xi)h(\xi) = \delta(\xi)$, the result is obtained. Details are omitted in the interest of brevity.

COROLLARY 5.1.1.

(a)
$$\bar{\alpha}^*(\theta, s) = -[W'(0)] + \overline{\lim_{L\to\infty}} \left(\frac{iL}{2}[W(0)] + \sum_{\xi_k \neq 0} \frac{[W(\xi_k(\theta, s))]}{\xi_k(\theta, s)} e^{i\xi_k(\theta,s)L}\right),$$

$$\underline{\alpha}^*(\theta, s) = -[W'(0)] + \underline{\lim}_{L\to\infty} \left(\frac{iL}{2}[W(0)] + \sum_{\xi_k \neq 0} \frac{[W(\xi_k(\theta, s))]}{\xi_k(\theta, s)} e^{i\xi_k(\theta,s)L}\right);$$

(b) $W(\cdot, \theta; s)$ *is continuous* $\Rightarrow \bar{\alpha}^*(\theta; s) = \underline{\alpha}^*(\theta; s) = -[W'(0)]$.

These conclusions follow at once from the fact that $e^{i\xi_k(\theta,s)L}$ is bounded and the fact that
$$\lim_{L\to\infty} \frac{1}{L} \int_0^L \int_{R_1} w''(\xi, \theta; s)\, e^{i\xi\sigma}\, d\mu(\xi)\, d\mu(\sigma) = 0,$$
since $w'' \in \mathscr{L}^1(-\infty, \infty)$.

Corollary 5.1.1 completely characterizes $\bar{\alpha}^*$ and $\underline{\alpha}^*$ for the class of ordinary plane figures whose width functions exhibit *only* the first two types of singularities listed at the beginning of this section. For all practical purposes, this class consists entirely of polygonal figures, as we shall see next.

DEFINITION 5.2. A figure s is a *polygonal figure* if and only if

(a) s consists of a simple closed polygon (cf. [7]) and its interior, or

(b) s is a finite sum or difference of sets satisfying condition (a).

THEOREM 5.2. *If* s *is a polygonal figure, then* $W(\cdot, \theta; s)$ *satisfies condition* (5.1) *of Theorem* 5.1 *for all values of* θ.

Proof. The conclusion is obvious for triangular figures. Since every simple closed polygon can be "triangulated" (cf. [7]), it follows that every polygonal figure is a finite union of triangular figures, t_i, whose pairwise intersections have zero measure, i.e.,
$$\mu(t_i \cap t_j) = 0, \qquad i \neq j.$$

From Definition 4.3 and the elementary properties of measure, it therefore follows that
$$W(\xi, \theta; s) = \sum_i W(\xi - \bar{x}_i + \bar{x}, \theta; t_i),$$

where $\bar{x}_i$ = centroid of t_i and $\bar{x}$ = centroid of $\bigcup_i t_i$. Each term in this finite sum has the property (5.1), consequently the sum also has the property (5.1).

Taken together, Theorems 5.1 and 5.2 provide the means for computing the invariants $\underline{\alpha}^*$ and $\bar{\alpha}^*$ for any polygonal figure via elementary trigonometrical processes. The case of a rectangle offers a convenient illustration; e.g., when $\theta = 0$, the width function of the rectangle in Fig. 8a is constant on its support with jump discontinuities at the endpoints due to the flat edges in the boundary of the

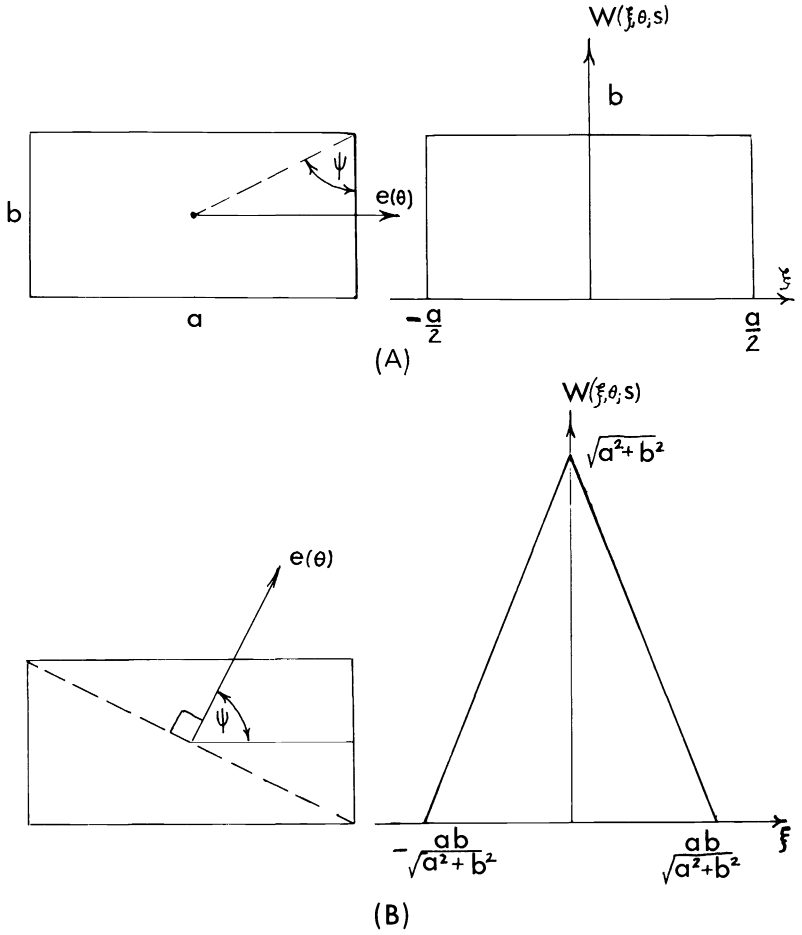


FIG. 8

rectangle. Part (a) of Corollary 5.1.1 applies in this case and gives

$$
\begin{aligned}
\bar{\alpha}^*(0, \text{“}\square\text{”}) &= \overline{\lim_{L\to\infty}}\left[\frac{b}{(-a/2)}e^{-iaL/2} + \frac{(-b)}{(a/2)}e^{iaL/2}\right] \\
&= \overline{\lim_{L\to\infty}}\left[-\frac{4b}{a}\cos(aL/2)\right] = +\frac{4b}{a}.
\end{aligned}
\tag{5.3}
$$

In the same way,

$$
\underline{\alpha}^*(0, \text{“}\square\text{”}) = \underline{\lim_{L\to\infty}}\left[-\frac{4b}{a}\cos(aL/2)\right] = -\frac{4b}{a}.
\tag{5.4}
$$

When θ falls in the range $(0, \pi/2)$, $W(\cdot, \theta; \text{“}\square\text{”})$ is continuous; furthermore, $[W'(0)] = 0$ except for $\theta = \tan^{-1}(a/b)$, where

$$
[W'(0)] = 2(a^2 + b^2)/ab
\tag{5.5}
$$

due to the corner in the boundary of the rectangle (cf. Fig. 8b). Consequently, part (b) of Corollary 5.1.1 applies here and gives

$$(5.6)\quad \bar{\alpha}^*(\theta; \text{“}\square\text{”}) = \underline{\alpha}^*(\theta; \text{“}\square\text{”}) = \begin{cases} \dfrac{2(a^2+b^2)}{ab}, & \theta \in (0, \pi/2);\ \theta = \tan^{-1}(a/b), \\ 0, & \theta \in (0, \pi/2);\ \theta \neq \tan^{-1}(a/b). \end{cases}$$

Continuing in this way, one arrives at the following complete description of $\bar{\alpha}^*(\theta; \text{“}\square\text{”})$ and $\underline{\alpha}^*(\theta; \text{“}\square\text{”})$:

$$(5.7)\qquad \bar{\alpha}^*(\theta; \text{“}\square\text{”}) = \underline{\alpha}^*(\theta; \text{“}\square\text{”}) = 0,$$

with exceptions as listed in Table 1 ($\psi \in (0, \pi/2)$, $\psi = \tan^{-1}(a/b)$).

TABLE 1
Exceptions to (5.7)

θ	$\bar{\alpha}^*$	$\underline{\alpha}^*$
0	$4b/a$	$-4b/a$
ψ	$2\dfrac{(a^2+b^2)}{ab}$	$2\dfrac{(a^2+b^2)}{ab}$
$\pi/2$	$4a/b$	$-4a/b$
$\pi - \psi$	$2\dfrac{(a^2+b^2)}{ab}$	$2\dfrac{(a^2+b^2)}{ab}$
π	$4b/a$	$-4b/a$
$\pi + \psi$	$2\dfrac{(a^2+b^2)}{ab}$	$2\dfrac{(a^2+b^2)}{ab}$
$3\pi/2$	$4b/a$	$-4b/a$
$2\pi - \psi$	$2\dfrac{(a^2+b^2)}{ab}$	$2\dfrac{(a^2+b^2)}{ab}$

There are several things worth noting about the foregoing example. First, if the rectangle in Fig. 8 is rotated about its centroid through an angle β, then the corresponding description of $\bar{\alpha}^*$ and $\underline{\alpha}^*$ is obtained by replacing θ with $\theta + \beta$ in (5.7). In other words, if R_β denotes the rotational (similarity) transformation in question, then

$$(5.8)\qquad \begin{aligned} \bar{\alpha}^*(\theta, R_\beta(\text{“}\square\text{”})) &= \bar{\alpha}^*(\theta + \beta, \text{“}\square\text{”}), \\ \underline{\alpha}^*(\theta, R_\beta(\text{“}\square\text{”})) &= \underline{\alpha}^*(\theta + \beta, \text{“}\square\text{”}). \end{aligned}$$

This is, in fact, a general property of the scale invariants $\bar{\alpha}^*$ and $\underline{\alpha}^*$, as is easily seen from Definition 4.3 and (4.17).

Second, the entries in Table 1 are all real numbers. This is *not* a general property of $\bar{\alpha}^*$ and $\underline{\alpha}^*$; it does occur for figures with biaxial symmetry (e.g., the rectangle, all equilateral $2N$-gonal figures, etc.) because of a fortuitous combination of positive and negative exponents appearing in the summations in part (a) of Corollary 5.1.1.

Finally, if one constructs a polar coordinate representation of the ordered pairs $(\theta, \bar{\alpha}^*)$ for those values of θ where $(\bar{\alpha}^* + \underline{\alpha}^*)/2 \neq 0$ in Table 1, the result is something which "looks like" a scaled rotated image of the original rectangle (Fig. 9). This also is not a general characteristic of $\bar{\alpha}^*$ and $\underline{\alpha}^*$; however, it does occur once again for biaxially symmetric polygonal figures since the corners of such figures are arranged in diametrically opposing pairs.

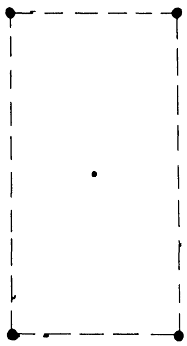


FIG. 9

6. Open questions. If the boundary of an ordinary plane figure contains a subarc of nonzero curvature, the associated width function $W(\cdot, \theta; s)$ will generally exhibit type (III) singularities (infinite one-sided derivatives) at certain points within its support (e.g., at boundary points) for some or all values of θ. As it stands, Theorem 5.1 says nothing about $\bar{\alpha}^*(\theta; s)$ and $\underline{\alpha}^*(\theta; s)$ for these values of θ, and the method employed in its proof does not appear to extend to width functions with type (III) singularities. The basic question is this: does Corollary 5.1.1 remain valid when the component function w' in (5.1) has (finitely many) type (III) singularities? This question is not resolved here. However, an affirmative answer seems plausible in the light of two considerations, viz., (a) a heuristic argument based on

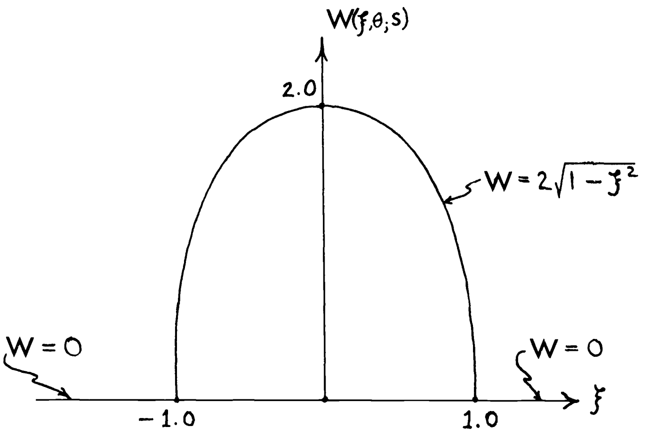


FIG. 10

polygonal approximations of s (this leads to an unresolved limit interchange question), and (b) explicit calculations for the special case of the circle.

With regard to (b) above, consider the unit circle with width function

$$(6.1) \qquad W(\xi, \theta; \text{“}\odot\text{”}) = \begin{cases} 2\sqrt{1-\xi^2}, & |\xi| \leqq 1, \text{all } \theta, \\ 0, & |\xi| > 1, \text{all } \theta \end{cases}$$

(cf. Fig. 10). The one-dimensional Fourier transform of (6.1) is

$$(6.2) \qquad 2\int_{-1}^{1} \sqrt{1-\xi^2} \cdot e^{i\xi\sigma}\, d\mu(\sigma) = \frac{K}{\sigma} J_1(\sigma),$$

where $K = 4\sqrt{\pi}\Gamma(3/2)$, Γ = gamma function, and J_1 = first order Bessel function (cf. [8, p. 321]). Consequently, it follows from Lemma 4.5, part (c), that $\bar{\alpha}^*$ and $\underline{\alpha}^*$ are, respectively, the $\overline{\lim}$ and $\underline{\lim}$ as $L \to \infty$ of

$$\frac{K}{L}\int_0^L \sigma J_1(\sigma)\, d\sigma .$$

But in view of the elementary Bessel identity, $(d/d\sigma)J_0(\sigma) = -J_1(\sigma)$ [9], an integration by parts gives

$$\frac{1}{L}\int_0^L \sigma J_1(\sigma)\, d\sigma = \frac{1}{L}\int_0^L J_0(\sigma)\, d\sigma - J_0(L) .$$

Therefore, since $\lim_{L\to\infty} \int_0^L J_0(\sigma)\, d\sigma = 1$ and $\lim_{L\to\infty} J_0(L) = 0$ ([8, p. 665] and [9]), it follows that $\bar{\alpha}^* = \underline{\alpha}^* = 0$ for the circle. This result shows that the type (III) singularities in the width function of the circle make no contribution to the values of $\bar{\alpha}^*$ and $\underline{\alpha}^*$, and suggests that the same conclusion may hold for more general figures as well.

Another open question concerns the completeness of the invariants $\bar{\alpha}^*$ and $\underline{\alpha}^*$ (cf. § 2). If the foregoing conjecture about Corollary 5.1.1 is correct, then $\bar{\alpha}^*$ and $\underline{\alpha}^*$ are certainly not complete with respect to the class of all ordinary plane figures, since both invariants would then vanish identically for any figure with a smooth curved boundary. On the other hand, it is not difficult to prove that $\bar{\alpha}^*$ and $\underline{\alpha}^*$ are complete with respect to the class of equilateral polygonal figures. Between these two extremes lies the class of all polygonal figures, and here the completeness question is not so easily resolved.

Section 5 offered two examples of polygonal figures which demonstrate that certain corners or edges in the boundary of a figure will not produce a singularity in the associated width function (Figs. 6 and 7). Such corners or edges cannot be detected in $\bar{\alpha}^*$ and $\underline{\alpha}^*$, at least not explicitly; nevertheless, it may be possible to infer their presence indirectly, as the only possible explanation of the values assumed by $\bar{\alpha}^*$ and $\underline{\alpha}^*$ on the “visible” edges and vertices of the polygonal figure in question. In other words, the inherent edge-vertex redundancy in $\bar{\alpha}^*$ and $\underline{\alpha}^*$ may prove sufficient to distinguish this kind of polygonal figure from any other non-scale-equivalent polygonal figure. If not, then the subclass of *convex* polygonal figures appears to be the next most significant and likely candidate for completeness, since the edges and vertices of convex figures are always visible in $\bar{\alpha}^*$ and $\underline{\alpha}^*$.[10]

Apart from the foregoing special considerations, there are a number of general questions growing out of the present investigation. For example, what factors determine whether a given integral transform, when averaged over a given group of figural transformations, will produce a nontrivial (perhaps even complete) invariant with respect to a given class of figures? The scale-averaged Fourier transform gives a nontrivial invariant for certain polygonal figures; is it unique in this respect among integral transforms? Will the Fourier transform produce significant results when averaged over other transformation groups, or do these groups call for their own specific transforms? Finally, are any of these questions resolvable in terms of simple structural relationships abstracted from the transforms, groups, and figural classes under consideration?

REFERENCES

[1] H. Weyl, *The Classical Groups: Their Invariants and Representations*, 2nd ed., Princeton Univ. Press, Princeton, N.J., 1953.

[2] W. Pitts and W. McCulloch, *How we know universals: the perception of auditory and visual forms*, Bull. Math. Biophys., 9 (1947), pp. 127–147.

[3] M. Minsky and S. Pappert, *Perceptions*, MIT Press, Cambridge, Mass., 1969.

[4] G. Birkhoff and S. MacLane, *A Survey of Modern Algebra*, 2nd ed., Macmillan, New York, 1953.

[5] A. H. Wallace, *An Introduction to Algebraic Topology*, Pergamon Press, New York, 1957.

[6] E. C. Titchmarsh, *Introduction to the Theory of the Fourier Integral*, 2nd ed., Oxford Univ. Press, New York, 1948.

[7] E. Hille, *Analytic Function Theory*, vol. 1, Blaisdell, New York, 1959.

[8] I. S. Gradshteyn and I. M. Ryshik, *Table of Integrals, Series and Products*, 4th ed., Academic Press, New York, 1965.

[9] H. Hochstadt, *Special Functions of Mathematical Physics*, Holt, Rinehart, and Winston, New York, 1966.

[10] It is perhaps worth noting that the problem of undetected edges and vertices can be alleviated to some extent if one is willing to tamper with the definition of W, e.g., if (4.15) is replaced by $W(\xi, \theta; s) = \mu_1\{\eta \geqq 0 | \phi(\xi e(\theta) + \eta e(\theta + \pi/2); s) = 1\}$, then (4.17) gives a new and different pair of invariants. Under certain conditions, this "half-plane masking" device will eliminate cancellations of width function singularities due to diametrically opposing pairs of congruent edges and/or vertices. For example, every edge in Fig. 6 becomes visible in the new invariants (on the other hand, the vertices of the square ring in Fig. 7 remain invisible).

SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

ON OPTIMAL PROCESSOR SCHEDULING FOR MULTIPROGRAMMING*

L. J. BASS†

Abstract. This paper investigates the problem of scheduling a processor to optimize throughput in a multiprogramming environment. A deterministic model is used to study the scheduling of a batch of k programs residing in main memory of a system consisting of a single processor and k input–output devices in such a way as to minimize the time to complete all k jobs.

It is shown that for any set of independent programs a preemptive strategy is not necessary to obtain the minimum running time for the entire batch. There is always an interrupt driven schedule which is as good as the best preemptive schedule.

It is also shown that processor bound programs are easy to schedule. A lower bound on the completion time for any set of programs is observed, and it is shown that with processor bound programs the lower bound can always be obtained. An algorithm for obtaining this bound is given.

These results provide some insight into the workings of the dynamic scheduling algorithms in use in many modern computer systems.

Key words. multiprogramming, scheduling, processor bound programs

A considerable amount of work has been done to place bounds on the performance of multiple processor systems which process sequences of CPU-bound programs. A survey of this work is presented in [1]. This paper considers a sequence of programs which use both the processor and an input-output device. The objective is to complete the total processing of a batch of k programs which reside in the primary memory in the shortest possible time. We show that an interrupt driven schedule will produce the minimum completion time. We also demonstrate an optimal strategy for scheduling a set of programs where two of the programs are processor bound. This last result should help clarify the workings of the dynamic scheduling algorithms in use in many current systems ([2]–[5]).

1. The model.

We define a program P_i to be a finite sequence of integers $T_{i,1}, t_{i,1}, \cdots, t_{i,n_i-1}, T_{i,n_i}$, where $t_{i,j} > 0$ for $j \leqq n_{i-1}$; $T_{i,j} > 0$ for $1 < j < n_i$; and $T_{i,j} \geqq 0$ for $j = 1$ or $j = n_i$. The $T_{i,j}$ are the compute times of the programs and the $t_{i,j}$ are the wait (input-output) times. A program, then, is given by a fixed sequence of compute, wait, compute, wait, etc.

The hardware portion of our model consists solely of one processor. We will not model any contention for channels or devices, and so we will assume programs just wait instead of doing input-output.

The computer system we consider consists of k independent programs together with the one processor and channels. A program, P_i, moves through the system by being assigned the processor one unit at a time. When it has been assigned $T_{i,1}$ units of processor time, it then waits $t_{i,1}$ units regardless of any other activity in the system. This is equivalent to assuming the existence of k input-output devices. After $t_{i,1}$ units of waiting, it then enters into competition for the processor again. When it has been assigned $T_{i,2}$ units, it again begins waiting. A program continues

* Received by the editors September 29, 1972, and in revised form May 11, 1973.

† Department of Computer Science, University of Rhode Island, Kingston, Rhode Island 02881.

this process until its sequence is exhausted. While program i is waiting, another program may be assigned the processor, but only one program may be assigned the processor at any one time (see Example 1). The mechanism by which a program is assigned the processor is called the scheduler, and the scheduler, in assigning the processor to a program, uses no time. We are interested in the algorithm by which the scheduler assigns the processor.

Example 1. *Action of the scheduler*. Let P_1 be given by $T_{1,1} = 4$, $t_{1,1} = 3$, $T_{1,2} = 1$, $t_{1,2} = 3$, $T_{1,3} = 2$ and P_2 be given by $T_{2,1} = 1$, $t_{2,1} = 3$, $T_{2,2} = 3$. Let $-$ denote the assignment of the processor for one unit and $\cdot$ denote waiting for one unit.
Then some possible schedules are:

```
(1) P1     - - - - · · ·   - · · · - -      Total Time = 14
    P2   - · · ·   - - -

(2) P1   -   - - - · · · - · · · - -        Total Time = 14
    P2     - · · · - - -

(3) P1   - - - - · · · - · · · - -          Total Time = 13.
    P2           - · · · - - -              (optimal)
```

Let $P_1, \cdots, P_k$ be some fixed set of independent programs. We are interested in the schedule which produces the shortest time to completion of all the programs. We will denote this time by $R(P_1, \cdots, P_k)$. Our results, then, pertain to scheduling for throughput or utilization. It can be argued that turnaround (minimizing average time to completion) is the best strategy for scheduling at computer installations, but we are only concerned with scheduling after the programs have reached the multiprogrammed state, i.e., have entered the system. It is entirely feasible to decide which programs to introduce into the system on a basis other than throughput and still to schedule the processor on a throughput basis. Indeed, this is what is done at many computer installations.

Notation. As mentioned, $R(P_1, \cdots, P_k)$ is the minimum possible time to completion of all k programs. We will need the notion of assigning the processor to a particular program for one unit of time. Fix i. We will define $R_i(P_1, \cdots, P_k)$. In defining $R_i(P_1, \cdots, P_k)$, the processor is assigned to program i for one unit of time, at the first compute interval (if it is positive). More precisely, if $T_{i,1} = 0$, then $R_i(P_1, \cdots, P_k) = R(P_1, \cdots, P_k)$. If $T_{i,1} > 0$, then let program P_i' be given by the sequence $T_{i,1} - 1$, $t_{i,1}$, $T_{i,2}, \cdots, T_{i,n_i}$, and then $R_i(P_1, \cdots, P_i, \cdots, P_k) = 1 + R(P_1, \cdots, P_i', \cdots, P_k)$. Intuitively, $R_i(P_1, \cdots, P_k)$ is the time achieved by the schedule of assigning the processor to P_i for one unit and then choosing the optimum schedule for the remaining time. Clearly, $R(P_1, \cdots, P_k) = \min_{1 \leqq i \leqq k} R_i(P_1, \cdots, P_k)$.

2. Interrupt driven schedules. We have, by our definitions, created a model with no-cost, preemptive scheduling. Our first theorem shows that a general preemptive schedule is not necessary. The minimum time to completion for a set of programs can be achieved by using an interrupt driven strategy, i.e., reassign the processor only upon the completion of a compute period by some program or completion of a wait period by some program. Before proving this theorem, we need to prove several lemmas. These lemmas, except for Lemma 3, are technical and serve only to show that the model is well-behaved.

LEMMA 1. *Let* $1 \leqq i \leqq k$. *Let* P_i' *be given by* $T_{i,1} + 1, t_{i,1}, T_{i,2}, \cdots, T_{i,n_i}$. *Then* $1 + R_i(P_1, \cdots, P_i, \cdots, P_k) \geqq R_i(P_1, \cdots, P_i', \cdots, P_k)$, *and* $1 + R(P_1, \cdots, P_i, \cdots, P_k) \geqq R(P_1, \cdots, P_i', \cdots, P_k)$.

Proof. $1 + R_i(P_1, \cdots, P_i, \cdots, P_k) \geqq 1 + R(P_1, \cdots, P_i, \cdots, P_k) = R_i(P_1, \cdots, P_i', \cdots, P_k) \geqq R(P_1, \cdots, P_i', \cdots, P_k)$.

LEMMA 2. *Let* $1 \leqq i, j \leqq k$. *Let* P_j' *be given by* $T_{j,1} + 1, t_{j,1}, T_{j,2}, \cdots, T_{j,n_j}$. *Then* $1 + R_i(P_1, \cdots, P_j, \cdots, P_k) \geqq R_i(P_1, \cdots, P_j', \cdots, P_k)$.

Proof. If $i = j$ or $T_{i,1} = 0$, then this follows from Lemma 1, so we can assume $i \neq j$ and $T_{i,1} \geqq 1$. Let P_i' be given by $T_{i,1} - 1, t_{i,1}, \cdots, T_{i,n_i}$. Then we have

$$\begin{aligned} 1 + R_i(P_i, \cdots, P_i, \cdots, P_j, \cdots, P_k) &= 2 + R(P_1, \cdots, P_i', \cdots, P_j, \cdots, P_k) \\ &\geqq 1 + R(P_1, \cdots, P_i', \cdots, P_j', \cdots, P_k) \\ &= R_i(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k). \end{aligned}$$

It has occasionally been asked what is the effect on total system performance of making one program more efficient. The next lemma shows that it can never hurt to shorten the first compute interval of the program. An easy consequence of Lemma 3 is that it never hurts to shorten the computing portion of the program.

LEMMA 3. *Let* $1 \leqq i \leqq k$, *and let* P_i' *be given by* $T_{i,1} + 1, t_{i,1}, T_{i,2}, \cdots, T_{i,n_i}$. *Then* $R(P_1, \cdots, P_i', \cdots, P_k) \geqq R(P_1, \cdots, P_i, \cdots, P_k)$.

Proof. The proof follows by induction on $\sum T_{i,j} + \sum t_{i,j}$. If this sum is 1, then the result is clear. Assume the theorem is false, and let $\sum T_{i,j} + \sum t_{i,j}$ be the least for which it fails. Then for some j, we must have

$$R(P_1, \cdots, P_i, \cdots, P_k) > R(P_1, \cdots, P_i', \cdots, P_k) = R_j(P_1, \cdots, P_i', \cdots, P_k).$$

If $i = j$, we have

$$R(P_1, \cdots, P_i, \cdots, P_k) > R_i(P_1, \cdots, P_i', \cdots, P_k) = 1 + R(P_1, \cdots, P_i, \cdots, P_k),$$

a contradiction, and so we have $i \neq j$. Furthermore, if for all $j \neq i$ we have $T_{j,1} = 0$, then we can take $j = i$ and derive the same contradiction. Thus we can take $T_{j,1} > 0$.

Now let P_j' be given by $T_{j,1} - 1, t_{j,1}, \cdots, T_{j,n_j}$. Then by the induction hypothesis and the above, we have

$$\begin{aligned} R(P_1, \cdots, P_i, \cdots, P_k) &> R_j(P_1, \cdots, P_i', \cdots, P_k) \\ &= 1 + R(P_1, \cdots, P_j', \cdots, P_i', \cdots, P_k) \\ &\geqq 1 + R(P_1, \cdots, P_j', \cdots, P_i, \cdots, P_k) \\ &\geqq R(P_1, \cdots, P_j, \cdots, P_i, \cdots, P_k). \end{aligned}$$

This contradiction proves the lemma.

Now we are prepared to prove that with the appropriate assignment of the processor we can allow a program to compute until either it voluntarily yields the processor or another program finishes waiting. We first prove a lemma which says that if assigning the processor to program j is best at one point, then it is also best at other points while the processor is computing on the first compute interval of program j. For notational convenience in both the statement of Lemma 4 and its proof, we ignore the fact that if $T_{i,1} = 0$, then $t_{i,1}$ decrements as we assign the processor, and hence we are dealing with a different program, P_i, at each step. This is a technical point which has no critical effect upon either the lemma or the proof.

LEMMA 4. *Let $i, j \leqq k$ and let $S \leqq T_{j,1}$. If $R_i(P_1, \cdots, P_k) = R(P_1, \cdots, P_k)$ and if for all $p \leqq S$, P_i' is given by $T_{i,1} - 1, \cdots, T_{i,n_i}$ and P_j' is given by $T_{j,1} - p, t_{j,1}, \cdots, T_{j,n_j}$ and $R(P_1, \cdots, P_i', \cdots, P_j', \cdots, P_k) = R_j(P_1, \cdots, P_i', \cdots, P_j', \cdots, P_k)$, then for any $p \leqq S$, we have*

$$R(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k) = R_i(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k)$$
$$= R_j(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k).$$

Proof. The proof is by induction on p. For the purposes of this proof, let P_i' be given by $T_{i,1} - 1, \cdots, T_{i,n_i}$. If $p = 0$, then if the lemma is not true, we have

$$\begin{aligned} R_j(P_1, \cdots, P_k) > R_i(P_i, \cdots, P_k) &= 1 + R(P_1, \cdots, P_i', \cdots, P_j, \cdots, P_k) \\ \text{(by hypothesis on } j) \quad &= 1 + R_j(P_1, \cdots, P_i', \cdots, P_k) \\ \text{(by Lemma (3)} \quad &\geqq R_j(P_1, \cdots, P_i, \cdots, P_k) \end{aligned}$$

This is a contradiction. Now let the lemma be true for $p = n - 1 < S$. Let P_j' be given by $T_{j,1} - p, t_{j,1}, \cdots, T_{j,n_j}$ and P_j'' be given by $T_{j,1} - p + 1, t_{j,1}, \cdots, T_{j,n_j}$. Now, for the sake of contradiction, assume

$$R_i(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k) > R(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k)$$

Then we have (by hypothesis on j)

$$\begin{aligned} R(P_1, \cdots, P_i', \cdots, P_j'', \cdots, P_k) &= R_j(P_1, \cdots, P_i', \cdots, P_j'', \cdots, P_k) \\ \text{(by definition)} \quad &= R_i(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k) \\ \text{(by hypothesis to be contradicted)} \quad &\geqq 1 + R(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k) \\ \text{(by Lemma 1)} \quad &\geqq R(P_1, \cdots, P_i, \cdots, P_j'', \cdots, P_k) \\ \text{(by induction hypothesis)} \quad &= 1 + R(P_1, \cdots, P_i', \cdots, P_j'', \cdots, P_k). \end{aligned}$$

This is a contradiction, and hence

$$R_i(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k) = R(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k).$$

Now, to prove the lemma, we need to show

$$R_j(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k) = R_i(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k).$$

Assume for the sake of contradiction this is false. Then by the result just proved, we must have

$$\begin{aligned} R_j(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k) &> R_i(P_1, \cdots, P_i, \cdots, P_j', \cdots, P_k) \\ \text{(by definition)} \quad &= 1 + R(P_1, \cdots, P_i', \cdots, P_j', \cdots, P_k) \end{aligned}$$

$$\begin{aligned}
&\text{(by hypothesis on } j) && = 1 + R_j(P'_1, \cdots, P'_i, \cdots, P'_j, \cdots, P_k) \\
&\text{(by Lemma 3)} && \geqq R_j(P_1, \cdots, P_i, \cdots, P'_j, \cdots, P_k).
\end{aligned}$$

This contradiction proves Lemma 4.

THEOREM 1. *Let* $Q = \{j; T_{j,1} = 0\}$. *Then*

(a) *if* $Q = \{1, \cdots, k\}$, *let* $l = \min_{j\in Q}(t_{j,1})$ *and for all* $i \leqq k$, *let* P'_i *be given by*

$$P'_i = \begin{cases} 0, t_{i,1} - l, T_{i,2}, \cdots, T_{i,n_i} & \text{if } t_{i,1} > l, \\ T_{i,2}, \cdots, T_{i,n_i} & \text{if } t_{i,1} = l. \end{cases}$$

Then $R(P_1, \cdots, P_k) = l + R(P'_1, \cdots, P'_k)$.

(If none of the programs wants the processor, then delay until one of them finishes waiting.)

(b) *If* $Q \neq \{1, \cdots, k\}$, *let*

$$l = \begin{cases} \min_{j\in Q}(t_{j,1}) & \text{if } Q \neq \varnothing, \\ \sum_{j\leqq k} T_{j,1} & \text{if } Q = \varnothing. \end{cases}$$

Then there exists a j such that if P'_i *is given by*

$$P'_i = \begin{cases} P_i & \text{if } T_{i,1} \neq 0 \quad \text{and} \quad j \neq i, \\ T_{j,1} - \min(l, T_{j,1}), t_{j,1}, T_{j,2}, \cdots, T_{j,n_j} & \text{if } i = j, \\ 0, t_{i,1} - \min(l, T_{j,1}), T_{i,2}, \cdots, T_{i,n_i} & \text{if } i \neq j, T_{i,1} = 0, \quad \text{and} \quad t_{i,1} \neq l, \\ T_{i,2}, t_{i,2}, \cdots, T_{i,n_i} & \text{otherwise}, \end{cases}$$

then $R(P_1, \cdots, P_k) = \min(l, T_{j,1}) + R(P'_1, \cdots, P'_k)$.

(Here l is the time until one of the programs finishes waiting—an arbitrary large number of no programs are waiting— and $T_{j,1}$ is the time until program j begins waiting. Thus, the P'_i represent what happens to programs P_i if the processor is assigned to program P_j until either it yields the processor or another program begins waiting).

Proof. The proof of part (a) is obvious and will be omitted. The proof of part (b) proceeds by induction on $\sum T_{i,j} + \sum t_{i,j}$. If this sum is 1, the theorem is trivially true, so assume part (b) holds for $\sum T_{i,j} = n$ and consider $\sum T_{i,j} + \sum t_{i,j} = n + 1$.

Suppose the theorem is false. For some i such that $T_{i,1} \neq 0$, we have $R_i(P_1, \cdots, P_k) = R(P_1, \cdots, P_k)$. If $T_{i,1} = 1$, we are done; thus, $T_{i,1} > 1$. Let P''_i be given by $T_{i,1} - 1, t_{i,1}, \cdots, T_{i,n_i}$ and then by the induction hypothesis, there is a j such that $R(P_1, \cdots, P''_i, \cdots, P_k) = \min(T_{j,1}, l - 1) + R(P'_1, \cdots, P''_i, \cdots, P'_k)$, where the P'_i are given by the hypothesis of part (b). If $i = j$, we are done, and so it must be the case that $i \neq j$.

Since we are assuming the theorem is false, it must also be the case that for some $p \leqq \min(T_{j,1}, l)$, we have P'_j given by $T_{j,1} - p, t_{j,1}, \cdots, T_{j,n_j}$ and $R_j(P'_1, \cdots, P'_j, \cdots, P'_k) > R(P'_1, \cdots, P'_j, \cdots, P'_k)$. This contradicts Lemma 4, and we are done.

3. Scheduling processor bound programs.

We now examine the problem of deciding on the optimal schedule where some of the programs are processor bound.

We first examine the case where only two programs are involved, and then we utilize the results obtained with two programs to examine the problem of more than two programs.

It is easily seen that any optimal scheduling strategy for arbitrary programs must examine the whole of the programs involved (see Example 2). What we shall now prove, for the case of two programs, is that if we know certain information about the programs involved, then the optimal schedule can be determined by examining just the first compute interval of each program and that, furthermore, the optimum schedule is given by assigning the process to the program whose first compute interval is smallest.

Example 2. *Examination of entire programs.* Let P_1 be given by $T_{1,1} = 10$, $t_{1,1} = 3$, $T_{1,2} = 10$, $t_{1,2} = 3$, $T_{1,3} = 10$, $t_{1,4} = 3$, $T_{1,5} = 10$, and P_2 be given by $T_{2,1} = 1$, $t_{2,1} = 3$, $T_{2,2} = 1$, $t_{2,2} = 3$, $T_{2,3} = 1$. Then $R(P_1, P_2) = T_{1,1} + t_{1,1} + T_{1,2} + t_{1,2} + T_{1,3} + t_{1,4} + T_{1,5}$, and any scheduling strategy based on only initial portions of the programs of less than 3 compute segments will fail to recognize this. This example is easily extended to arbitrarily long initial segments.

First we note that for programs $P_1, \cdots, P_k$, a lower bound on the time to completion is the maximum of the sum of the processor times over all the programs and the maximum of the times the individual programs would take when running alone. Formally,

$$R(P_1, \cdots, P_k) \geqq \max\left(\sum_{i \leqq k} \sum_{j \leqq n_i} T_{i,j}, \max_{i \leqq k}\left(\sum_{j \leqq n_i} T_{i,j} + \sum_{j < n_i} t_{i,j}\right)\right).$$

What we shall prove, initially, is that for the case of two programs which are processor bound, the lower bound is achievable and if the lower bound is the sum of the processor times, then it can be achieved by a variation on choosing the shortest compute interval first.

We say two programs P_1 and P_2 are *processor bound* with respect to each other if

$$T_{1,i} \geqq t_{2,j} \quad \text{for } 1 < i \leqq n_1 \text{ and } 1 \leqq j < n_2$$

and

$$T_{2,i} \geqq t_{1,j} \quad \text{for } 1 < i \leqq n_2 \text{ and } 1 \leqq j < n_1$$

and either

$$T_{1,1} \geqq t_{2,1} \quad \text{or} \quad T_{2,1} \geqq t_{1,1}.$$

Intuitively, this definition says that, with the possible exception of one of the initial compute times, all of the compute times of P_1 are greater than any of the wait times of P_2 and vice versa. With two such programs, the lower bound can always be achieved, and if the lower bound is the sum of the processor times, then the appropriate strategy is to assign the processor to the program with the smallest compute segment relative to the other program's first wait interval. We formalize and prove this in the next theorem.

THEOREM 2. *Let P_1 and P_2 be processor bound with respect to each other and let*

$$R(P_1, P_2) > \max_{i=1,2}\left(\sum_{j \leqq n_i} T_{i,j} + \sum_{j < n_i} t_{i,j}\right).$$

Then

$$R(P_1, P_2) = \sum_{i=1,2} \sum_{j \leqq n_i} T_{i,j}.$$

Furthermore, if $T_{1,1} - t_{2,1} \geqq T_{2,1} - t_{1,1}$, *then* $R_2(P_1, P_2) = R(P_1, P_2)$.

Proof. The proof proceeds by induction on $\sum_{i=1,2} \sum_{j \leqq n_i} T_{i,j}$. If this sum is 1, the theorem is clear. Assume the theorem is true for all programs such that $\sum_{i=1,2} \sum_{j \leqq n_i} T_{i,j} < n$, and we will show it to be true for programs such that $\sum_{i=1,2} \sum_{j \leqq n_i} T_{i,j} = n$. Let $T_{1,1} - t_{2,1} \geqq T_{2,1} - t_{1,1}$. If $T_{2,1} = 0$, then since P_1 and P_2 are processor bound with respect to each other, it must be that $T_{1,1} \geqq t_{2,1}$. In this case, we have $R_2(P_1, P_2) = R(P_1, P_2) = t_{2,1} + R(P'_1, P'_2)$, where P'_1 is given by $T_{1,1} - t_{2,1}, t_{1,1}, \cdots, T_{1,n_1}$ and P'_2 is given by $T_{2,2}, t_{2,2}, \cdots, T_{2,n_2}$. P'_1 and P'_2 are processor bound with respect to each other and $R(P'_1, P'_2) > \max(\sum_{i \leqq j \leqq n_1} T_{1,j} + \sum_{j < n_1} t_{1,j} - t_{2,1}, \sum_{2 < j < n_2} T_{2,j} + \sum_{2 \leqq j < n_2} t_{2,j})$. Thus, by induction,

$$R(P'_1, P'_2) = \sum_{i=1,2} \sum_{1 \leqq j \leqq n_i} T_{i,j} + T_{1,1} - t_{2,1}.$$

Combining these two equalities, we obtain

$$R_2(P_1, P_2) = \sum_{i=1,2} \sum_{1 < j \leqq n_1} T_{i,j} + T_{1,1},$$

as desired. If $T_{2,1} > 0$ then, letting P'_2 be given by $T_{2,1} - 1, t_2, 1, \cdots, T_{2,n_2}$ and observing that P_1 and P'_2 are processor bound with respect to each other, we have by the induction hypothesis,

$$R_2(P_1, P_2) = 1 + R(P_1, P'_2) = 1 + \sum_{i=1,2} \sum_{2 \leqq j \leqq n_i} T_{k,j} + T_{1,1} + T_{2,1} - 1,$$

and the theorem is proved.

Example 3. *Shortest first is not always optimum.* Let P_1 be given by $T_{1,1} = 3$, $t_{1,1} = 3, T_{1,2} = 1$ and P_2 by $T_{2,1} = 6, t_{2,1} = 3, T_{2,2} = 5$. Then $R(P_1, P_2) > T_{1,1} + t_{1,1} + T_{1,2}$ and $R(P_1, P_2) > T_{2,1} + t_{2,1} + T_{2,1}$.

Scheduling by shortest first yields:

```
P1  — — — · · · —
                                                   (18 units).
P2        — — —  — — —  · · ·  — — — — —
```

Even recognizing that P_1 should not be chosen when it has no wait time left yields:

```
P1  — — — · · ·           —
                                                   (17 units).
P2        — — — — — — · · · — — — — —
```

The optimum schedule is given by choosing P_2 first and yields:

```
P1                — — — · · ·      —
                                                   (15 units).
P2  — — — — — — · · · — — — — —
```

It is easily seen (Example 3) that this strategy does not work for more general programs. Now we consider the case where we have k programs, two of which are processor bound with respect to each other and the rest of which have only the restriction that the wait times are not excessive. In this case, the lower bound can be achieved by assigning a low priority to the two compute bound programs.

THEOREM 3. *Let $P_1, \cdots, P_k$ be programs such that $\sum_{i=1,2} \sum_{j \leqq n_1} T_{i,j} > \max_{i \leqq k} \sum_{1 \leqq j \leqq n_i} t_{i,j}$ and such that P_1 and P_2 are processor bound with respect to each other. Then $R(P_1, \cdots, P_k) = \sum_{i \leqq k} \sum_{j \leqq n_i} T_{i,j}$.*

Proof. When competition occurs for the processor, assign it to all of the $P_3, \cdots, P_k$ that are in competition before assigning it to either P_1 or P_2. When only P_1 and P_2 are in competition, assign the processor based on the rule in Theorem 2. The first clause in the hypothesis guarantees that no program does enough waiting to absorb the processor times of P_1 and P_2, and the proof that the processor remains busy is an obvious extension of the proof of Theorem 2. Note that a single processor bound program would also suffice. In fact, it is possible to consider any set of programs which are processor bound with respect to each other instead of the two which we used.

4. Conclusions. These results are not surprising to practical programmers. The algorithms used in [2], [3] and [5] were developed, apparently, strictly on the basis of intuitition. The results presented here, however, do provide a formal analysis and some insight into the observed behavior without reference to arbitrary probabilistic assumptions.

These practical algorithms all work by using the past behavior of the program to attempt to predict the future behavior of the program. Theorem 3 indicates that if the future behavior of the program is predicted to be processor bound, then the program should be given low priority in competing for the processor and this is what was initially done [5].

Acknowledgment. The author would like to thank Peter J. Denning and the referees for their constructive comments.

REFERENCES

[1] R. L. GRAHAM, *Bounds on multiprocessing anomalies and related packing algorithms*, Proc. of 1972 Spring Joint Computer Conference, vol. 40, AFIPS Press, Montvale, N.J., pp. 205–217.

[2] B. S. MARSHALL, *Dynamic calculation of dispatching priorities under OS/360 MVT*, Datamation, Aug. 1969, pp. 93–97.

[3] K. D. RYDER, *A heuristic approach to task dispatching*, IBM Systems J., 8 (1970), pp. 189–198.

[4] S. SHERMAN, F. BASKETT AND J. C. BROWNE, *Trace driven modelling and analysis of CPU scheduling in a multiprogramming system*, Comm. ACM, 15 (1972), pp. 1063–1069.

[5] D. F. STEVENS, *On overcoming high-priority paralysis in multiprogramming systems: A case history*, Comm. ACM 11 (1968, pp. 539–541.

SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

AN ALGORITHM FOR THE EXTREME RAYS OF A POINTED CONVEX POLYHEDRAL CONE*

WALTER B. McRAE† AND ERNEST R. DAVIDSON‡

Abstract. An algorithm for exhibiting the extreme rays of a pointed convex polyhedral cone is described. The cone is assumed to be initially defined by a system of homogeneous linear inequalities. The method differs from prior procedures in two respects. First, faces of lower dimension than facets for the polar convex cone are used to serially determine the extreme rays; second, the method allows for the possibility of symmetry considerations involving the extreme rays to reduce storage requirements and computation effort.

Key words. extreme ray, polyhedral cone, linear programming

1. Introduction. In this paper an algorithmic procedure is described for exhibiting a complete set of extreme rays for a pointed convex polyhedral cone [1] which is defined by a system of homogeneous linear inequalities. That is, if $\{\bar{e}_i\}_{i=1}^{P}$ is a set of d-dimensional column vectors whose components define normals to a set of d-dimensional hyperplanes, then the inequalities $\bar{e}_i^T\bar{y} \geqq 0$ define the coordinates y of points in the polyhedral cone bounded by these hyperplanes. If the rank of the matrix $\bar{E} = (\bar{e}_1, \bar{e}_2 \cdots)$ is d, the cone is pointed. The problem, then, is to find the extreme rays $\bar{y}$ of the cone (i.e., those $\bar{y}$ which are contained in the intersection of at least $d - 1$ linearly independent hyperplanes and which lie in the surface of the cone).

An example of this type of problem arises in connection with the Slater hull problem, which is a limited version of the N-representability problem ([2]–[5]). In one version of the Slater hull problem, the matrix $\bar{E}$ has $\binom{r}{N}$ columns of dimension $\binom{r}{2}$. Each column i is characterized by a different set of N distinct integers in the range 1 to r. The element $e_{J,i}$ [$J = k + j(j-1)/2$ for $r \geqq j > k \geqq 1$] is 1 if k and j are both in the set of N integers and 0 otherwise. The extreme rays associated with this particular matrix $\bar{E}$ are far from simplicial. The cone does have a high degree of symmetry, however, since all columns e_i are equivalent under the permutation group on r objects. Even for small integers such as $r = 9$ and $N = 4$, the number of extreme rays for this cone becomes astronomical ($>10^8$).

For very small values of r and N, the double description algorithm as outlined by Koler [6] is an efficient way to generate the extreme rays $\bar{y}$ of $\bar{E}$. For slightly larger dimensions, however, this method fails because it requires simultaneous generation of all extreme rays. On the CDC 6400 computer, this prevents its use when the number of $\bar{y}$ greatly exceeds 10^4.

As will be discussed in more detail later, many of the extreme rays are equivalent under the symmetry operations which send $\bar{E}$ into itself. The double

* Received by the editors April 1, 1972. This research was supported in part by the National Science Foundation under Grant no. GP-2734.

† Department of Chemistry, University of Georgia, Athens, Georgia 30601.

‡ Department of Chemistry, University of Washington, Seattle, Washington 98195.

description algorithm cannot easily be modified to take account of this simplification because this symmetry is not present during the intermediate stages.

The algorithm outlined in this paper, then, is designed to allow easy use of symmetry and to allow sequential rather than simultaneous generation of extreme rays. It is generally less efficient than the double description method for problems where both will work, but it allows results to be generated for problems too large to handle by the double description procedure.

2. An algorithm for diagonal N-representability.

2.1. Mathematical preliminaries. In the discussion to follow, the symbol $\bar{x}$, unless specified otherwise, will be used to denote either an element x of an abstract Euclidean space or the column matrix representation of x relative to some basis. Moreover, the symbol $\bar{x}^T\bar{y}$ will be used to indicate either the scalar product of two elements x and y of an Euclidean space or their conventional matrix inner product.

The solution set for a finite system of m linear homogeneous inequalities in n unknowns constitutes a convex polyhedral cone [1], called the polar cone of E, which may be symbolized as

$$C(E) = \{\bar{y}|\bar{E}^T\bar{y} \geqq 0\}. \tag{1}$$

Here $\bar{E}^T$ is an $m \times n$ matrix whose rows are the normals at the origin to the hyperplanes bounding the closed half-spaces.

Equation (1), however, is not the only way a convex polyhedral cone may be characterized. A set C in R^n is a convex polyhedral cone if C can be written as the sum of a finite number of half-lines L_i [7, p. 65], i.e., $C = \sum_{i=1} L_i$.

According to this latter definition, there exists a finite set of elements $G = \{\bar{g}_i\}_{i=1}^m$, where $\bar{g}_i$ generates the half-line L_i such that

$$C(G) = \left\{ \bar{\rho} \in R^n | \bar{\rho} = \sum_{i=1}^{m} \omega_i \bar{g}_i, \omega_i \geqq 0 \right\}.$$

If the elements of G are represented as the columns of a matrix $\bar{G}$, then C is given as

$$C(G) = \{\bar{\rho}|\bar{\rho} = \bar{G}\bar{\omega}, \bar{\omega} \geqq 0\}. \tag{2}$$

Now by a theorem stated by Weyl [8], if E is a finite set of vectors in R^n, then there exists a finite set G in R^n such that

$$\bar{C}(E) = C(G) \quad \text{and} \quad \bar{C}(G) = C(E). \tag{3}$$

The explicit characterization of the polar cone $\bar{C}(E)$ is thus given by $C(G)$. The algorithm developed here for computing G for polyhedral cones $\bar{C}(E)$ is based on solving the equation $\bar{C}(G) = C(E)$. In order to exhibit the set G, it is convenient to restrict the discussion to the case where $\bar{C}(E)$ is pointed[1] because a pointed convex polyhedral cone is the convex hull of its extreme rays.

[1] No assumptions are made here concerning the pointedness of the convex cone $C(E)$. In the case, however, that a positive basis for $\bar{C}(E)$ is to be used to characterize the interior of the convex hull of the set E, e.g., in the diagonal N-representability problem, then $C(E)$ must also be pointed.

In order to describe the algorithm for exhibiting these extreme vectors, the facial structure of the convex polyhedral cone must be considered.

The algorithm is based, in part, on two well-known results [9, Chap. 3, Chap. 11].

(I) Each $(n - 2)$-face F of a convex polyhedral cone C of dimension n is contained in precisely two facets F_1 and F_2 of C, and $F = F_1 \cap F_2$.

(II) If F_1 is a face of the convex polyhedral cone C and if F_2 is a face of the convex polyhedral cone F_1, then F_2 is a face of C.

In the spirit of (I), two facets are set to be adjacent if they contain in common an $(n - 2)$-face.

2.2. The basic algorithm. The basic iterative procedure of the algorithm may be given a geometric interpretation based on (I). If a facet of $C(E)$ is known, then any subfacet within this facet is contained in precisely one adjacent facet. Hence, if the supporting hyperplane containing the initial facet can be "rotated" about this subfacet, it may be brought into coincidence with the adjacent facet.

In order to describe this pivotal procedure more precisely, let E be a finite set of m elements in $R^n (m \geqq n)$ and let $\bar{y}$ be the normal to a facet F of $C(E)$; then $\bar{y}$ satisfies

$$\bar{E}^T \bar{y} = \bar{\eta}, \qquad \bar{\eta} \geqq 0,$$

or

$$\bar{C}^T \bar{y} = 0, \tag{4a}$$

$$\bar{D}^T \bar{y} > 0, \tag{4b}$$

where $\bar{C}^T$ is the matrix of rank $n - 1$ whose rows are the rows of $\bar{E}^T$ for which equality with zero holds, and $\bar{D}^T$ is the matrix constructed from the remaining rows of $\bar{E}^T$. If $\bar{z}$ is the element of F normal to the subfacet H of F, then $\bar{z}$ satisfies

$$\left|\begin{matrix} \bar{y}^T \\ \bar{C}^T \end{matrix}\right| \bar{z} = \left|\begin{matrix} 0 \\ \bar{\xi}_c \end{matrix}\right|, \qquad \bar{\xi}_c \geqq 0, \tag{5}$$

where the rank of rows of $\bar{C}^T$ on which equality with zero holds is $n - 2$. Here $\bar{\xi}_c$ denotes the elements $\bar{E}^T \bar{z} = \bar{\xi}$ formed from the submatrix $\bar{C}$ of $\bar{E}$. To show that such $\bar{z}$ exist, observe that F is the convex polyhedral cone generated as the positive hull of the elements of $\bar{E}$ contained in $\bar{C}$. Moreover, within the $(n - 1)$-dimensional linear space determined by the supporting hyperplane normal to $\bar{y}$, F is pointed because the rank of $\bar{C}$ is $n - 1$.

The normal to the adjacent facet intersecting F in H may be written as $\bar{v} = \alpha\bar{y} + \beta\bar{z}$. Now since $\bar{v}$ is normal to a facet of $C(E)$, for all columns $\bar{e}_k$ of $\bar{E}$, necessarily

$$\begin{aligned} \bar{e}_k^T \bar{v} &= \alpha \bar{e}_k^T \bar{y} + \beta \bar{e}_k^T \bar{z} \geqq 0, \\ &= \alpha \eta_k + \beta \xi_k \quad \geqq 0. \end{aligned} \tag{6}$$

Since an adjacent facet exists, there is at least one $\bar{e}_m$ in $\bar{C}$ for which $\xi_m > 0$.

Since for every $\bar{e}_n$ in $\bar{C}$,

$$\bar{e}_n^T \bar{v} = \beta \xi_n > 0, \tag{7}$$

it follows that $\beta > 0$. Similarly, for at least one $\bar{e}_n$ in $\bar{D}$,

$$\bar{e}_n^T \bar{v} = \alpha \eta_n + \beta \xi_n = 0. \tag{8}$$

As $\eta_j > 0$ for all $\bar{e}_j$ in $\bar{D}$, it follows from (8) that $\xi_n/\eta_n = -\alpha/\beta$ for at least one element of $\bar{D}$. For every $\bar{e}_j$ in $\bar{D}$, necessarily

$$\bar{e}_j^T \bar{v} = \alpha \eta_j + \beta \xi_j \geqq 0 \quad \text{or} \quad -\alpha/\beta \leqq \xi_j/\eta_j. \tag{9}$$

Hence, if $\alpha = -\xi_n$ and $\beta = \eta_n$ are chosen so that

$$\xi_n/\eta_n = \min_{\bar{e} \in \bar{D}} \{\xi_j/\eta_j\}, \tag{10}$$

then (9) will be satisfied for all $\bar{e}_j$ in $\bar{D}$. Furthermore, because $\eta_j = 0$ and $\xi_j \geqq 0$ for every $\bar{e}_j$ in $\bar{C}$, it follows that $\bar{v} = -\xi_n \bar{y} + \eta_n \bar{z}$ satisfies (6) for every $\bar{e}_k$ in $\bar{E}$, and thus the hyperplane normal to $\bar{v}$ supports $C(E)$. Finally, notice that the intersection of this hyperplane with $C(E)$ is indeed a facet. It obviously contains H and hence at least $n - 2$ linearly independent elements of E. However, the additional element represented by $\bar{e}_n$ that has been included is necessarily linearly independent of these because if it were not, then $\bar{e}_n^T \bar{y}$ would be zero, contrary to the fact that $\bar{e}_n$ was chosen as a row of $\bar{D}$. The intersection thus contains $n - 1$ linearly independent elements of E and is hence a facet. It will frequently happen that n in (10) is not uniquely defined because several elements or D give the same minimum ratio. This is not a problem, however, since every choice of n among this degenerate set leads to the same facet of $C(E)$.

In order to generate every facet of a convex polyhedral cone, it will be necessary to systematically determine each of the subfacets contained in the specified facet. If these subfacets can be found, the rotation procedure described may be used to determine each adjacent facet. In addition, if these adjacent facets are accumulated in a list with the initial facet being the first list item, all facets may be constructed by moving through the list sequentially and repeating the procedure. Of course, only those facets generated which are different from the existing list items will be entered in the list at each iteration. The process of generating all adjacent facets and comparing them with a list to determine those which will be retained may be referred to as *scanning*. The algorithm ends when the last list item is scanned without producing new list entries.

The fact that every facet of $C(E)$ is encountered using this procedure follows because $\bar{C}(E)$ is taken to be pointed. It is possible to identify a convex polytope $\bar{P}(E)$ with $\bar{C}(E)$ by specifying a normalization for the extreme vectors of $\bar{C}(E)$. If the vertices and edges of $\bar{P}(E)$ are viewed as constituting the vertices and edges of a graph to be naturally identified with $\bar{P}(E)$ and hence $\bar{C}(E)$, the resulting graph is $(n - 1)$-connected [9, Chap. 3, Chap. 11].

In the context of this polytopal graph, the process of scanning corresponds to determining all those graph vertices not previously known that are adjacent to (i.e., possess an edge in common with) that graph vertex identified with the facet of $C(E)$ being scanned. Furthermore, rotating about one subfacet in each of a sequence of adjacent facets specifies a sequence of adjacent vertices in the graph known as a *path*. Because the polytopal graph is connected, such a path exists connecting any two vertices. Hence missing a facet of $C(E)$ is impossible because this would require that the associated graph vertex either be isolated or be an element of a component of the graph which is disconnected from that component containing the vertices which have been encountered in scanning.

2.3. Generation of subfaces. The essential feature of the procedure used for exhibiting each of the subfacets in a particular facet may be illustrated by considering the more general problem of determining a complete set of extreme rays for a pointed convex polyhedral cone (in an $(n-p)$-dimensional space) defined as the intersection of p $(p < n)$ orthogonal hyperplanes,

$$H_m = \{\bar{x} \in R^n | \bar{a}_m^T \bar{x} = 0\}, \qquad m = 1, \cdots, p,$$

and q closed half-spaces,

$$\bar{H}_n = \{\bar{x} \in R^n | \bar{b}_n^T \bar{x} \geqq 0\}, \qquad n = 1, \cdots, q,$$

where it is assumed that the vectors a_m, $m = 1, \cdots, p$, are linearly independent of the vectors b_n, $n = 1, \cdots, q$.

In other words, this convex cone is the solution set to the system of equations

$$(11) \qquad \begin{vmatrix} \bar{A} \\ \bar{B} \end{vmatrix} \bar{z} = \begin{vmatrix} 0 \\ \bar{\eta} \end{vmatrix}, \qquad \bar{\eta} \geqq 0,$$

where $\bar{B}\bar{z} = \bar{\eta}$, with $\bar{B}$ the $q \times n$ matrix whose rows are the components of the vectors $\bar{b}_n$ and $\bar{A}$ the $p \times n$ matrix whose rows are the components of the vectors $\bar{a}_m$. Let us denote this convex cone by $\bar{C}_A(B)$ and observe that $\bar{C}_A(B)$ is polar to the convex cone $C_A(B)$ generated as the positive hull of the elements in R^n represented by the rows of $\bar{B}$. Moreover, both of these convex cones reside in the $(n-p)$-dimensional linear space determined as the intersection of the specified p orthogonal hyperplanes.

For the purpose of exhibiting a complete set of extreme vectors for $\bar{C}_A(B)$, consider first the case that $p + q = n$. In this case, the matrix $\begin{vmatrix} \bar{A} \\ \bar{B} \end{vmatrix}$ is necessarily nonsingular if $\bar{C}_A(B)$ is pointed. Hence, letting $\bar{U}$ be the inverse matrix and $\bar{z}$ any element of $\bar{C}_A(B)$, observe that

$$U \begin{vmatrix} \bar{A} \\ \bar{B} \end{vmatrix} \bar{z} = \bar{z} = \bar{U} \begin{vmatrix} 0 \\ \bar{\eta} \end{vmatrix} = \sum_{i=1}^{q} \eta_i \bar{u}^{i+p},$$

where $\bar{u}^i$ represents column i of $\bar{U}$. Thus in this simple case, the last q columns of $\bar{U}$ constitute a positive basis for $\bar{C}_A(B)$ and represent a complete set of extreme vectors. In this case, $\bar{C}_A(B)$ and $C_A(B)$ are called *simplicial*.

The other case to consider in this context occurs when $p + q = m$ with $m > n$. The rank of the matrix $\begin{vmatrix} \bar{A} \\ \bar{B} \end{vmatrix}$ is still necessarily n if $\bar{C}_A(B)$ is pointed. Thus let $\bar{U}$ be a generalized inverse, i.e., let $\bar{U}$ be an $m \times m$ nonsingular matrix satisfying

$$\bar{U}\begin{vmatrix}\bar{A}\\ \bar{B}\end{vmatrix} = \begin{vmatrix}I_n\\ 0\end{vmatrix}, \qquad \bar{U}^{-1}\begin{vmatrix}I_n\\ 0\end{vmatrix} = \begin{vmatrix}\bar{A}\\ \bar{B}\end{vmatrix},$$

where I_n is the $n \times n$ identity matrix. If $\bar{z}$ is any element of $\bar{C}_A(B)$, then

$$\bar{U}\begin{vmatrix}\bar{A}\\ \bar{B}\end{vmatrix}\bar{z} = \begin{vmatrix}\bar{z}\\ 0\end{vmatrix} = \bar{U}\begin{vmatrix}0\\ \bar{\eta}\end{vmatrix}, \qquad \bar{\eta} \geqq 0.$$

If $\bar{U}$ is now partitioned as

$$\bar{U} = \begin{vmatrix}\bar{R} & \bar{T}\\ \bar{Q} & \bar{S}\end{vmatrix},$$

where R, T, Q and S are, respectively, $n \times p$, $n \times q$, $(m - n) \times p$ and $(m - n) \times q$ matrices, then clearly

$$\begin{vmatrix}\bar{z}\\ 0\end{vmatrix} = \begin{vmatrix}\bar{T} & \bar{\eta}\\ \bar{S} & \bar{\eta}\end{vmatrix}.$$

Hence for any $\bar{z}$ in $\bar{C}_A(B)$, the associated vector $\bar{\eta}$ must satisfy the homogeneous system of equations

$$(12) \qquad \bar{S}\bar{\eta} = 0$$

in addition to the necessary nonnegative condition

$$(13) \qquad \bar{\eta} \geqq 0.$$

Conversely, for any $\bar{\eta}$ satisfying (12) and (13) it is easily demonstrated that if

$$(14) \qquad \bar{z} = \bar{T}\bar{\eta},$$

then $\bar{z}$ satisfies (11). Observe, however, that the solution set to (12) and (13) is the convex polyhedral cone resulting from the intersection of the positive orthant in a q-dimensional Euclidean space and the $m - n$ hyperplanes defined by (12). Denoting this convex cone by $\bar{C}(\eta)$, it is easily verified that the linear transformation in (14) defines a one-to-one correspondence preserving dimension between faces of $\bar{C}(\eta)$ and $\bar{C}_A(B)$. Thus $\bar{C}(\eta)$ is a pointed convex cone of dimension $n - p$. Moreover, an element of $\bar{C}_A(B)$ is an extreme vector if and only if the image of this element is an extreme vector for $\bar{C}(\eta)$. Thus a complete set of extreme vectors for $\bar{C}_A(B)$ may be obtained as the images of a complete set of extreme vectors for $\bar{C}(\eta)$. The task of exhibiting such a set for this latter convex cone is, however, a tractable problem using established procedures if the number of extreme rays is not too large (less than 10,000 for a CDC 6400 computer). Specifically, the algorithm developed by Kohler [6], which is a variant of the double description procedure of Motzkin [10], may be used for this purpose.

2.4. Imbedding sequence for subfaces. Practically, then, a complete set of extreme vectors can be exhibited for a convex polyhedral cone defined according to (11) if the set is not too large. This result may be used to develop the procedure for determining each of the subfacets contained in a specified facet of the convex cone $C(E)$ in R^n.

In general terms, the procedure utilizes the fact that any face of $C(E)$ is a convex polyhedral cone to construct a sequence of faces of $C(E)$ satisfying the imbedding condition,

$$F^k \subset F^{k+1} \subset \cdots F^{n-2} \subset F,$$

where F is any facet and F^k is some k-face. A unique normal at the origin to any face in this sequence may be determined to within a multiplicative constant by requiring that this vector be orthogonal to the normal to each face of greater dimension. It is easily seen that the k-dimensional linear space resulting from the intersection of the orthogonal hyperplanes determined by each of these normals contains F^k. This result is readily cast in the format of (11); and because the polar F^k is pointed in this linear space, either a matrix inversion or the double description method may be used to determine a normal to each facet of F^k where the facets of F^k are just the $(k - 1)$-faces of $C(E)$ contained in F^k. The normals constructed in this manner are then used in conjunction with the rotation procedure described to generate a list of normals to the k-faces adjacent to F^k which still are imbedded in $F^{k+1} \subset \cdots \subset F$. The process then continues by scanning this list to eventually generate the normals to every k-face of F^{k+1}. The normals to the facets of F^{k+1} are then used to generate normals to each $(k + 1)$-face adjacent to F^{k+1}. If this list is then scanned, normals to each $(k + 1)$-face in F^{k+2} may be obtained, and so on, until eventually normals to each subfacet in F are obtained.

At each stage, when the search for faces adjacent to F^d is begun, all faces of lower dimension left from the scan of the previous F^d are invalid and must be discarded. Ideally, at each stage, a program would select the best one of the three possible ways of finding the imbedded F^{d-1} faces. If F^d is simplicial, the F^{d-1} can best be found by inverting a matrix. If F^d has few enough facets, the double description procedure can be used to find them all simultaneously. Finally, if F^d has many facets, a scanning procedure can be initiated to find them sequentially. The algorithm, as actually implemented, applied the double description method to find the faces of F^{n-3}.

A feature of this algorithm which requires emphasis is the fact that the scan of the lists associated with faces of lower dimension need not be completed in order to generate some elements for the lists associated with faces of higher dimension. Admittedly, there will be no assurance that any of the lists are complete if this is done; nonetheless, it is an important capability in view of the symmetry considerations to be discussed in the next section.

2.5. Initial faces. To conclude § 2, a brief description of the method used to find an initial facet is given. This procedure is required not only for the purpose of constructing an initial sequence of imbedded faces, but also in the scanning process whenever it is necessary to find an initial subface of a F^d face.

Several investigators have considered the problem of constructing a solution to a system of inequalities [11], [12], but in general, the treatments given have been within a context where only nonnegative solutions are considered and where the system of inequalities is necessarily presumed to be inhomogeneous. These restrictions present no great difficulty, however, because the homogeneous inequalities encountered in the case of a pointed convex polyhedral cone are easily transformed to meet these conditions.

With reference to the requirements of the algorithm discussed, consider a system of linear inequalities of the form

$$(15) \qquad \bar{A}\bar{z} = \begin{vmatrix} \bar{y}_1 \\ \cdot \\ \cdot \\ \bar{y}_k \\ \bar{C} \end{vmatrix} = \begin{vmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ \bar{\eta} \end{vmatrix}, \qquad \eta \geqq 0,$$

where $\bar{A}$ is an $m \times n\,(m \geqq n)$ matrix of rank n and where the vectors $\bar{y}_i, i = 1, \cdots, k$, are linearly independent. In order to convert this system to an inhomogeneous one, observe that if $\bar{z}$ is any nonzero solution to (15), and 1_n is a row matrix of 1's, then necessarily

$$\bar{1}_n\bar{A}\bar{z} = \left(\sum_{i=1}^{m} \bar{a}_i\right)\bar{z} = \bar{v}\bar{z} = \sum_{i=1}^{m-k} \eta_i = \alpha > 0,$$

where $\bar{a}_i$ represents row i of $\bar{A}$. This strict inequality for any solution other than zero may be written because the solution set under the given conditions on $\bar{A}$ constitutes a pointed convex polyhedral cone, (i.e., no vector exists which is orthogonal to all the rows of $\bar{A}$, because the rank of $\bar{A}$ is n). It also follows from this fact that for any $\bar{z} \neq 0$, a fixed arbitrary positive value for α may be chosen; hence let $\alpha = 1$ for all $\bar{z} \neq 0$. Finally, notice that for any $\bar{z}$ satisfying (15), obviously

$$\left(\sum_{i=1}^{m} \bar{a}_i\right)\bar{z} = \left(\sum_{i=1}^{m-k} \bar{c}_i\right)\bar{z}.$$

Thus if $\bar{s} = \sum \bar{c}_i$, the following equivalent inhomogeneous inequalities may be considered:

$$(16) \qquad \bar{A}'\bar{z} = \begin{vmatrix} \bar{y}_1 \\ \cdot \\ \cdot \\ \bar{y}_k \\ \bar{C} \\ \bar{s} \end{vmatrix} \quad \bar{z} = \begin{vmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ \eta \\ \beta \end{vmatrix}, \qquad \eta \geqq 0, \quad \beta \geqq 1.$$

These inequalities may now be converted to an equivalent system of inhomogeneous inequalities for which a nonnegative solution is sought by utilizing the fact that the rank of $\bar{A}$ is maximal. Thus the vectors $\bar{y}_i$, $i = 1, \cdots, k$, may be augmented with $n - k$ rows of $\bar{C}$ to form a set of n linearly independent vectors. Letting $\bar{V}$ be the nonsingular matrix generated from these vectors, (22) may be written as

$$\begin{vmatrix} \bar{V} \\ \bar{s} \\ \bar{B} \end{vmatrix} \bar{z} = \begin{vmatrix} 0 \\ \vdots \\ \bar{\delta} \\ \beta \\ \bar{\xi} \end{vmatrix}, \qquad \bar{\delta} \geqq 0, \quad \bar{\xi} \geqq 0, \quad \beta \geqq 1,$$

where $\bar{B}$ represents the $m - n + k$ rows of $\bar{C}$ not in $\bar{V}$, and $\bar{\delta}$ corresponds to the rows of $\bar{C}$ in $\bar{V}$. If $\bar{U}$ is the matrix inverse to $\bar{V}$, then

$$\begin{vmatrix} \bar{V} \\ \bar{s} \\ \bar{B} \end{vmatrix} \bar{z} = \begin{vmatrix} \bar{V} \\ \bar{s} \\ \bar{B} \end{vmatrix} \bar{U}\bar{U}^{-1}\bar{z} = \begin{vmatrix} I_n \\ \bar{s}\bar{U} \\ \bar{B}\bar{U} \end{vmatrix} \bar{U}^{-1}\bar{z} = \begin{vmatrix} I_n \\ \bar{D} \end{vmatrix} \bar{x} = \begin{vmatrix} 0 \\ \vdots \\ 0 \\ \bar{\delta} \\ \beta \\ \bar{\xi} \end{vmatrix},$$

where $\bar{x} = \bar{U}^{-1}\bar{z}$ and $\bar{D} = \begin{vmatrix} \bar{s}\bar{U} \\ \bar{B}\bar{U} \end{vmatrix}$. The last matrix equality requires that $\bar{x}$ be nonnegative with the first k components identically zero. Hence the reduced system of inequalities becomes

$$\bar{D}'\bar{x}' = \begin{vmatrix} \beta \\ \bar{\xi} \end{vmatrix}, \qquad \bar{\xi} \geqq 0, \quad \beta \geqq 1, \quad \bar{x}' \geqq 0,$$

where $\bar{D}'$ is the matrix obtained from $\bar{D}$ be deleting the first k columns and $\bar{x}'$ represents the last $n - k$ components of $\bar{x}$.

The inequalities in (17), (or, more accurately, the transpose of this system) are exactly the type of linear inequalities for which the ingenious lexicographic procedure of Gale [12] is designed to construct a basic solution. This is the procedure which has been utilized in the algorithm with only minor modifications in the pivotal steps to eliminate the necessity of divisions.

2.6. Symmetry considerations. The role of permutational symmetry in the diagonal N-representability problem is considerable. In fact it is the presence of this symmetry which makes the algorithm described here (when modified to account for this property) particularly applicable to this problem [13]. For the purpose of this paper, however, the only assumption concerning symmetry which will be made is that there exists a finite group G defined over R^n which is homomorphic to the permutation group of the finite set E which positively spans $C(E)$. As a consequence of this assumption [14, pp. 181–186], the set E may be decomposed exhaustively into equivalence classes with respect to G. Here by an *equivalence class* is meant a subset of E (possibly E itself) consisting of those elements in E which are equivalent under G, where $\bar{e}$ and $\bar{e}'$ in E are said to be *equivalent* (under

G) if there exists some P in G such that $P\bar{e} = \bar{e}'$. Observe that knowledge of one element in an equivalence class is sufficient to obtain all other elements.

The faces of $C(E)$ may also be categorized into equivalence classes, and hence only one representative face of each equivalence class need be retained in order to completely characterize the facial structure of $C(E)$. More important for the purposes of the algorithm, however, is the fact that when scanning a specified k-face, it is only necessary to rotate about one representative for each equivalence class (defined relative to a subgroup $G(F^k)$ of G) of $(k-1)$-faces in this k-face in order to generate one representative for each equivalence class of the adjacent k-faces. This has the obvious consequence of reducing the number of items in the lists associated with the algorithm if the scanning process is modified so that only normals which are not equivalent are retained. This modification, of course, requires a procedure for determining when two normals in a prescribed normalization are not equivalent, and this may be a very difficult problem if the order of the group G is large. For example, in the diagonal N-representability problem, introducing this capability required development of a constructive procedure for determining when two square matrices $\bar{A}$ and $\bar{B}$ were related according to $\bar{A} = \bar{P}^T\bar{B}\bar{P}$, where $\bar{P}$ is a permutation matrix. Because this problem is essentially equivalent to determining when two finite graphs are isomorphic, a detailed account of the procedure will be given in a subsequent paper.

In order to demonstrate that the existence of G does in fact induce equivalence class structure on the faces of $C(E)$, consider first the facets. If $\bar{P}$ is the matrix identified with a member P of G in an orthogonal representation and $\bar{Q}$ is the permutation matrix identified with a corresponding element of the permutation group on E under the aforementioned homomorphism, then $\bar{E}\bar{Q} = \bar{P}\bar{E}$. If $\bar{y}$ is the normal to any facet of $C(E)$, it follows that

$$\begin{aligned} \bar{E}^T\bar{y} &= \bar{E}^T\bar{P}^{-1}\bar{P}\bar{y} = \bar{E}^T\bar{P}^T\bar{P}\bar{y} \\ &= \bar{Q}^T\bar{E}^T\bar{z} = \bar{\eta}, \qquad \bar{\eta} \geqq 0, \quad \bar{z} = \bar{P}\bar{y}, \end{aligned}$$

or

$$\bar{E}^T\bar{z} = \bar{Q}\bar{\eta}, \qquad \bar{Q}\bar{\eta} \geqq 0.$$

If $\bar{C}(\bar{y})$ is the submatrix of $\bar{E}$ for which $\bar{C}^T\bar{y} = 0$, then $\bar{P}\bar{C}$ is the corresponding matrix for $\bar{z}$. The fact that $\bar{z}$ is a basic solution is then easily shown if its is recalled that the rank of matrix is invariant to multiplication by a nonsingular matrix. Hence $\bar{z}$ is normal to a facet of $C(E)$ and the set of solutions $\{\bar{z}|\bar{z} = \bar{P}\bar{y}, P \in G\}$ defines an equivalence class of facets.

A useful definition of the equivalence class structure of the faces of dimension lower than facets within the context of the imbedding sequence may be illustrated by considering the $(n-3)$-faces contained in a given $(n-2)$-face which in turn is contained in a facet of $C(E)$. If $\bar{y}_1$ is a normal to the facet and $\bar{y}_2$ a normal in this facet to the subfacet then, according to the construction in the algorithm, a normal $\bar{y}_3$ to an $(n-3)$-face in the given subfacet satisfies

$$\begin{vmatrix} \bar{y}_1^T \\ \bar{y}_2^T \\ \bar{C}^T \end{vmatrix} \bar{y}_3 = \begin{vmatrix} 0 \\ 0 \\ \bar{\eta} \end{vmatrix}, \qquad \bar{\eta} \geqq 0.$$

Here it is to be recalled that $\bar{C}^T$ is the matrix whose rows are the representations of the elements of the subset C of E whose positive hull is the specified subfacet, i.e., $\bar{C}^T$ is the submatrix of rows of $\bar{E}^T$ satisfying $\bar{C}^T\bar{y}_1 = \bar{C}^T\bar{y}_2 = 0$.

Analogous to the manner in which an equivalence class was exhibited for the facets of $C(E)$, an equivalence class of $(n - 3)$-faces in this subfacet, characterized by $\bar{y}_3$ as a representative element, may be constructed if a subgroup of G is determined which is homomorphic to the permutation group of the set C. Such a group is the subset G^{n-2} of G consisting of those elements of G which leave $\bar{y}_1$ and $\bar{y}_2$ invariant, i.e., for any P in G^{n-2},

$$\bar{P}\bar{y}_1 = \bar{y}_1 \quad \text{and} \quad \bar{P}\bar{y}_2 = \bar{y}_2.$$

Equivalently, G^{n-2} may be regarded as the subgroup of G^{n-1} which sends F^{n-2} into itself. The fact that the image of any element in C under a member of G^{n-2} is also in C is easily established. Letting $\bar{e}^T$ be any row of $\bar{C}^T$ and $\bar{P}$ the matrix associated with a member of G^{n-2} in an orthogonal representation, then we have

$$0 = \bar{e}^T\bar{y}_1 = \bar{e}^T\bar{P}^T\bar{P}\bar{y}_1 = (\bar{P}\bar{e})^T\bar{y}_1,$$

and similarly,

$$(\bar{P}\bar{e})^T\bar{y}_2 = 0,$$

Hence $(\bar{P}\bar{e})^T$ is a row of $\bar{E}^T$ which is contained in $\bar{C}$. Analogous arguments can be used to define equivalence classes for the $(k - 1)$-faces or their normals in any k-face. Thus for the imbedding sequence, $F^k \subset F^{k+1} \cdots \subset F^{n-1}$, the subgroups are also imbedded as $G^{k+1} \subset \cdots \subset G$. The equivalence class of F^k relative to G^{k+1} is defined as $\{F^k | \bar{C}(F^k) = \bar{P}\bar{C}(F_1^k), P \in G^{k+1}\}$.

Finally, in order to illustrate why one one representative from each equivalence class for the k-faces contained in a specified $(k + 1)$-face need be retained in order to generate at least one representative for each of the equivalence classes characterizing adjacent $(k + 1)$-faces, the preceding example may be used to show that any two normals to $(n - 3)$-faces in the same equivalence class give rise necessarily to equivalent $(n - 2)$-faces.

Assume that $\bar{z}_3$ and $\bar{y}_3$ satisfy (5), where $\bar{z}_3 \neq \bar{y}_3$ and where $\bar{P}\bar{y}_3 = \bar{y}_2$, $\bar{P}\bar{z}_3 = \bar{y}_3$, $P \in G^{n-2}$; then for some α and β determined by the rotation procedure previously described, the vector $\bar{u} = \alpha\bar{y}_2 + \beta\bar{y}_3$ is normal to an $(n - 2)$-face adjacent to that determined by $\bar{y}_2$. Observe, however, that

$$\begin{aligned}\bar{u} &= \alpha\bar{y}_2 + \beta\bar{P}\bar{z}_3\\ &= \alpha\bar{P}\bar{y}_2 + \beta\bar{P}\bar{z}_3\\ &= \bar{P}(\alpha\bar{y}_2 + \beta\bar{z}_3)\\ &= \bar{P}\bar{v}.\end{aligned}$$

By construction, for any P in G^{n-2}, necessarily $\bar{P}\bar{y}_1 = \bar{y}_1$, and hence $\bar{y}_1^T\bar{v} = 0$. Moreover, since $P \in G^{n-2} \subset G^{n-1}$, the elements of E in the facet determined by $\bar{y}_1$ are merely permuted by P, and hence $\bar{v}$ is a normal in this facet to an adjacent subfacet. Furthermore, because only two facets intersect in each subfacet, $\bar{v}$ is the only normal which may be obtained from $\bar{y}_2$ and $\bar{z}_3$. Extending this result to faces of general dimension, the conclusion follows that normals to inequivalent $(k + 1)$-faces under G^{k+2} adjacent to a specified $(k + 1)$-face can arise only from rotations about inequivalent k-faces under G^{k+1} within the given $(k + 1)$-face. The converse of this statement does not hold, however, as equivalent $(k + 1)$-faces (under G^{k+2}) may arise from inequivalent k-faces (under G^{k+1}).

When rotation has been carried out about one representative element for each equivalence class of k-faces contained in a $(k + 1)$-face, there can exist no adjacent $(k + 1)$-face which is not equivalent to those adjacent $(k + 1)$-faces generated in scanning the specified k-face. To show this, assume such an inequivalent adjacent $(k + 1)$-face exists. Necessarily, this adjacent $(k + 1)$-face contains a k-face in common with the specified $(k + 1)$-face. However this k-face must be an element of one of the characteristic equivalence classes for the k-faces in the original $(k + 1)$-face. Thus by the preceding result, the $(k + 1)$-face obtained by rotation was carried out about the k-face representing the appropriate equivalence class. Hence by contradiction, the assumed inequivalent adjacent $(k + 1)$-face cannot exist.

For the general group G, comparison of the list of known $\bar{y}$ with a new $\bar{y}$ can be facilitated if a standard element of each equivalence class can be defined and kept in the list. One definition of this standard element for small groups might be the lexographically largest element of the equivalence class. That is, for $\{\bar{z}|\bar{z} = \bar{P}\bar{y}, \bar{P} \in G\}$, define $\bar{z}_0 = \max \bar{z}$, where $\bar{z}_1 > \bar{z}_2$ if $z_{i,1} = z_{i,2}$ for all i less than j, and $z_{j,1} > z_{j,2}$ ($\bar{z}_1$ exceeds $\bar{z}_2$ in the first element in which they differ). In general, $\bar{z}_0$ can only be found by generating the whole equivalence class of $\bar{y}$. Clearly two elements are in the same equivalence class if and only if their standard elements are the same.

3. Discussion. A computerized version of this algorithm has been prepared and applied to several elementary fermion and boson diagonal N-representability problems. The results of these calculations have been reported elsewhere [13]. In the most complex case considered ($r = 9$, $N = 5$) the matrix E had dimension 36×126. About 30 hours of CDC 6400 computer time was spent to find the adjacent facets to 195 of the facets. In this way, 1089 equivalence classes were found which contained 2.3×10^8 facets. For the simple cases $(r, N) = (3, 6)$, $(3, 7)$, $(3, 8)$ the dimensions of E are $(15, 20)$, $(21, 35)$, $(28, 56)$. The rotation algorithm required 4.3, 58.5, and 80.9 seconds per equivalence class, respectively, or 0.25, 0.46, and 0.030 seconds per facet. By comparison, the double description algorithm required only 0.04 and 0.17 seconds per facet for the first two cases, but this method was not feasible for the 52,000 facets of the third case.

In any particular application the choice of the minimum dimension for scanning (or, indeed, whether to use the double description method directly for the facets) must be based on expediency. As mentioned previously, the double description method is superior in the absence of symmetry when the expected number of extreme rays is small enough that they can all be kept in a computer at once. With some modifications for using disk storage, the amount of core-storage required can be kept small at the cost of extensive input-output. For the problems for which this new algorithm was designed, the extreme vectors had dimensions in the range of 20–40 and were expected to exceed 10^6 in number. Hence, even disk storage was not large enough to hold all of them at once.

The inefficiency in the scanning procedure described here arises because every facet is generated many times. If there are K facets with an average of L subfacets, then the process of scanning will generate each facet an average of L times. Since L is at least $n - 1$, for high dimensions the process becomes very inefficient.

If high symmetry is present, the situation may be reversed. Suppose each equivalence class contains an average of R facets. Then scanning one element of each equivalence class will produce only KL/R facets. While it is true that only K/R of these are needed, if L/R is much smaller than unity the scanning procedure will become more efficient than the double description algorithm which always generates K facets.

Because of the descent in symmetry and the decrease in the number of adjacent faces as the scanning procedure is carried to lower dimensional faces, beyond some face of minimum dimension the use of the double description method becomes preferred. Ideally this should be dynamically adjusted by the program itself.

Acknowledgment. The authors wish to thank Professor V. Klee for many helpful discussions.

REFERENCES

[1] A. J. Goldman and A. W. Tucker, *Polyhedral convex cones*, Ann. Math. Studies, 38 (1956), p. 19.

[2] R. M. Erdahl and H. Kummer, *The N-representability problem*, Report of the Density Matrix Seminar, Queen's University, Kingston, Ontario, June 17–July 12, 1968.

[3] E. R. Davidson, *Linear inequalities for density matrices. I*, J. Math Phys., 10 (1969), p. 725.

[4] M. L. Yoseloff, *A combinatorial approach to the diagonal N-representability problem*, Ph.D. thesis, Princeton University, Princeton, New Jersey, 1970.

[5] P. O. Löwdin, *Quantum theory of many-particle systems. I. Physical interpretation by means of density matrices, natural spin orbitals, and convergence problems in the method of configuration interaction*, Phys. Rev., 97 (1955), p. 97.

[6] D. A. Kohler, *Projections of convex polyhedral sets*, Ph.D. thesis, University of California, Berkeley, California, 1967.

[7] G. Hadley, *Linear Programming*, Addison-Wesley, Reading, Mass., 1962, p. 65.

[8] H. Weyl, *The elementary theory of convex polyhedra*, Ann. Math. Studies, 24 (1950), p. 3.

[9] B. Grunbaum, *Convex Polytopes*, Interscience Publishers, New York, 1967.

[10] T. S. Motzkin, H. Raiffa, G. L. Thompson and R. M. Thrall, *The double description method*, Ann. Math. Studies, 28 (1953), p. 51.

[11] G. Debreu, *Nonnegative solutions of linear inequalities*, Internat. Econom. Rev., 5 (1964), p. 178.

[12] D. Gale, *How to solve linear inequalities*, Amer. Math. Monthly, 76 (1969), p. 589.

[13] W. M. McRae and E. R. Davidson, *Linear inequalities for density matrices. II*, J. Math. and Phys., 13 (1972), p. 1527.

[14] M. Eisen, *Elementary Combinatorial Analysis*, Gordon and Breach, New York, 1969.

SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

SET MERGING ALGORITHMS*

J. E. HOPCROFT† AND J. D. ULLMAN‡

Abstract. This paper considers the problem of merging sets formed from a total of n items in such a way that at any time, the name of a set containing a given item can be ascertained. Two algorithms using different data structures are discussed. The execution times of both algorithms are bounded by a constant times $nG(n)$, where $G(n)$ is a function whose asymptotic growth rate is less than that of any finite number of logarithms of n.

Key words. algorithm, algorithmic analysis, computational complexity, data structure, equivalence algorithm, merging, property grammar, set, spanning tree

1. Introduction. Let us consider the problem of efficiently merging sets according to an initially unknown sequence of instructions, while at the same time being able to determine the set containing a given element rapidly. This problem appears as the essential part of several less abstract problems. For example, in [1] the problem of "equivalencing" symbolic addresses by an assembler was considered. Initially, each name is in a set by itself, i.e., it is equivalent to no other name. An assembly language statement that sets name A equivalent to name B by implication makes C equivalent to D if A and C were equivalent and B and D were likewise equivalent. Thus, to make A and B equivalent, we must find the sets (equivalence classes) of which A and B are currently members and merge these sets, i.e., replace them by their union.

Another setting for this problem is the construction of spanning trees for an undirected graph [2]. Initially, each vertex is in a set (connected component) by itself. We find edges (n, m) by some strategy and determine the connected components containing n and m. If these differ, we add (n, m) to the tree being constructed and merge the components containing n and m, which now are connected by the tree being formed. If n and m are already in the same component, we throw away (n, m) and find a new edge.

A third application [3] is the implementation of property grammars [4], and many others suggest themselves when it is realized that the task we discuss here can be done in less than $O(n \log n)$ time.

By way of introduction, let us consider some of the more obvious data structures whereby objects could be kept in disjoint sets, these sets could be merged, and the name of the set containing a given object could be determined. One possibility is to represent each set by a tree. Each vertex of the tree would correspond to an object in the set. Each object would have a pointer to the vertex representing it, and each vertex would have a pointer to its father. If the vertex is a root, however, the pointer would be zero to indicate the absence of a father. The name of the set is attached to the root.

* Received by the editors August 10, 1972, and in revised form May 18, 1973. This research was supported by the National Science Foundation under Grant GJ-1052 and the Office of Naval Research under Contract N00014-67-A-0071-0021.

† Department of Computer Sciences, Cornell University, Ithaca, New York 14850.

‡ Department of Electrical Engineering, Princeton University, Princeton, New Jersey 08540.

Given the roots of two trees, one can replace the representation of two sets by a representation for their union by making the pointer at one root point to the other root and, if necessary, updating the name at the root of the combined tree. Thus two structures can be merged in a fixed number of steps, independent of the size of the sets. The name of the set containing a given object can be found, given the vertex corresponding to the object, by following pointers to the root.

However, by starting with n trees, each consisting of a single vertex, and successively merging the trees together until a single tree is obtained, it is possible to obtain a representation for a set of size n, which consists of a chain of n vertices. Thus, in the worst case, it requires time proportional to the size of the set to determine which set an object is in.

For purposes of comparison, assume that initially there are n sets, each containing exactly one object, and that sets are merged in some order until all items are in one set. Interspersed with the mergers are n instructions to find the set containing a given object. Then the tree structure defined above has a total cost of n for the merging operation and a cost bounded by n for determining which set contains a given object (total cost n^2 for n look ups).

Methods based on maintaining balanced trees (see [5], e.g.) have a total cost of $n \log n$ for the merging operations and a cost bounded by $\log n$ for determining which set contains a given object (total cost $n \log n$ for n look ups).

A distinct approach is to use a linear array to indicate which set contains a given object. This strategy makes the task of determining which set contains a given object finite. By renaming the smaller of the two sets in the merging process, the total cost of merging can be bounded by $n \log n$.

A more sophisticated version of the linear array replaces the set names in the array by pointers to header elements. This method, based on the work of Stearns and Rosenkrantz [3], uses $n \underbrace{\log \log \cdots \log}_{k}(n)$ steps for the merging process and a fixed number of steps independent of n for determining which set contains a given element. Here k is a parameter of the method and can be any fixed integer.

In what follows, we shall make use of a very rapidly growing function and a very slowly growing function which we define here. Let $F(n)$ be defined by

$$F(0) = 1,$$

$$F(i) = F(i-1)2^{F(i-1)} \quad \text{for} \quad i \geqq 1.$$

The first five values of F are 1, 2, 8, 2048 and 2^{2059}.

The slowly growing function $G(n)$ is defined for $n \geqq 0$ to be the least number k such that $F(k) \geqq n$. Both set merging algorithms presented here have asymptotic growth rates of at most $O(nG(n))$.

Initially, let $S_i = \{i\}$, $1 \leqq i \leqq n$. The S_i's are *set names*. Given a sequence of two types of instructions,

(i) MERGE(i, j), (ii) FIND(i),

where i and j are distinct integers between 1 and n, we wish to execute the sequence from left to right as follows. Each time an instruction of the form MERGE(i, j) occurs, replace sets S_i and S_j by the set $S_i \cup S_j$ and call the resulting set S_j. Each time an instruction of the form FIND(i) occurs, print the name of the set currently containing the integer i. It is assumed that the length of the sequence of instructions is bounded by a constant times n. Both set merging algorithms presented here have asymptotic growth rates bounded by $nG(n)$. The first algorithm actually requires $nG(n)$ steps for certain sequences. For the second algorithm, it is unknown whether $nG(n)$ is in fact its worst case performance. Recently, Tarjan [6] has shown the algorithm to be worse than $O(n)$.

2. The first set merging algorithm. The first set merging algorithm represents a set by a data structure which is a generalization of that used in [3] to implement property grammars. The basic structure is a tree similar to that shown in Fig. 1, where the elements of the set are stored at the leaves of the tree. Links are assumed to point in both directions.

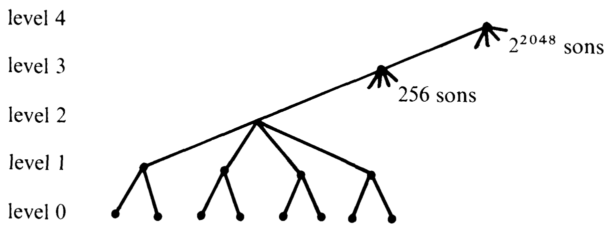


FIG. 1. *Data structure for set representation*

Each vertex at level i, $i \geqq 1$, has between one and $2^{F(i-1)}$ sons and thus at most $F(i)$ descendants which are leaves. We define a *complete* vertex as follows:

(i) Any vertex at level 0 is complete.

(ii) A vertex at level $l \geqq 1$ is complete if and only if it has $2^{F(l-1)}$ sons, all of which are complete.

Otherwise, a vertex is *incomplete*.

The data structure is always maintained, so that no vertex has more than one incomplete son. Furthermore, the incomplete son, if it exists, will always be leftmost, so it may be easily found. Attached to each vertex is the level number, the number of sons, and the number of descendant leaves. The name of the set is attached to the root.

The procedure COMBINE, given below, takes two data structures of the same level l and combines them to produce a single structure of level l. If there are too many leaves for a structure of level l, then COMBINE produces two structures of level l, one with a complete root and one with the remaining leaves. To simplify understanding of the algorithm, the updating of set names, level numbers, number of sons and the number of descendant leaves for each vertex has been omitted.

Procedure COMBINE (s_1, s_2).

Assume without loss of generality that s_2 has no more sons than s_1 (otherwise permute the arguments).

1. If both s_1 and s_2 have incomplete sons, say v_1 and v_2, then call COMBINE (v_1, v_2). On completion of this recursive call of COMBINE, the original data structure is modified as follows. If originally the total number of leaves on the subtrees with roots v_1 and v_2 did not exceed the number of leaves for a complete subtree, then the new subtree with root v_1 contains all leaves of the original two subtrees. The new subtree with root v_2 consists only of the vertex v_2.

If originally the total number of leaves exceeded the number for a complete subtree, then the new subtree with root v_1 is a complete subtree whose leaves are former leaves of the original subtrees with roots v_1 and v_2, and the new subtree with root v_2 is an incomplete subtree with the remaining leaves. If on completion of the recursive call of COMBINE, vertex v_2 has no sons, then delete vertex v_2 from the data structure.

2. Make sons of s_2 be sons of s_1, until either s_2 has no more sons or s_1 has $2^{F(l-1)}$ sons.

3. If s_2 still has sons, then interchange the lone incomplete vertex at level $l - 1$, if any, with the leftmost son of s_2. Otherwise, interchange the incomplete vertex with the leftmost son of s_1.

We now consider the first algorithm for the MERGE–FIND problem.

ALGORITHM 1.

1. Initially n vertices numbered 1 to n are created and treated as structures of level 0. Each vertex has information giving the name of its set, its number of descendant leaves (1), its level (0) and the number of its sons (0). A linear array of size n is created, such that the ith cell contains a pointer to vertex i.

2. The sequence of instructions of the forms MERGE(i, j) and FIND(i) are processed in order of occurrence.

(a) For an instruction of the form FIND(i), go to the ith cell of the array, then to the vertex pointed to by that cell, then proceed from the vertex to the root and print the name at the root.

(b) For an instruction of the form MERGE(i, j), if the roots for structures i and j are not of the same level, then add a chain of vertices above the root of lower level so that the roots of the two trees will have the same level. Let s_1 and s_2 be the two roots. Execute COMBINE(s_1, s_2). If after the execution of COMBINE s_2 has no sons, then discard s_2 and we are finished. Otherwise, create a new vertex whose left son is s_1 and whose right son is s_2.

We now show that Algorithm 1 requires time bounded by a constant times $nG(n)$, provided that the length of the sequence of instructions is itself first bounded by a constant times n. The first step is to observe that Algorithm 1 preserves the property that at most one son of any vertex is incomplete.

LEMMA 1. *At any time during the stimulation of a sequence of* MERGE *and* FIND *instructions by Algorithm* 1, *each vertex has at most one incomplete son. Furthermore, the incomplete son, if it exists, will always be leftmost.*

Proof. The proof proceeds by induction on the number of MERGE instructions simulated. The basis zero is trivial, since each leaf is complete by definition.

For the inductive step, we observe by induction on the number of applications of COMBINE that applying COMBINE to two trees, each of which have the desired property, will result in either producing a single tree with the desired property plus a single vertex or producing a complete tree plus another tree with the desired property. Thus, no call of COMBINE can give a vertex two incomplete sons if no vertex had two such sons previously. Furthermore, if two trees are produced, neither consisting of a single vertex, then one must be complete.

As a consequence of the above observation, if a new vertex with sons s_1 and s_2 is created in step 2(b) of Algorithm 1, the subtree with root s_1 is complete, and the property holds for the new vertex.

LEMMA 2. *If we begin with n objects, no vertex created during Algorithm* 1 *has level greater than* $G(n)$.

Proof. If there were such a vertex v, it could only be created in step 2(b). It would, by Lemma 1, have a complete descendant at level $G(n)$. An easy induction on l shows that a complete vertex at level l has $F(l)$ descendant leaves. Thus, v has at least $F(G(n)) + 1$ descendant leaves, which implies $F(G(n)) + 1 \leqq n$. The latter is false by definition of G.

THEOREM 1. *If Algorithm* 1 *is executed on n objects and a sequence of at most m* MERGE *and* FIND *instructions*, $m \geqq n$, *then the time spent is* $O(mG(n))$.

Proof. By Lemma 2, each FIND may be executed in $O(G(n))$ steps, for a total cost of $O(mG(n))$ for the FIND's. Since there can be at most $n - 1$ MERGE's, the cost of simulating the MERGE operations, exclusive of calls to COMBINE, is $O(n)$. Moreover, by Lemma 2, each MERGE can result in at most $G(n)$ recursive calls to COMBINE, as each call is with a pair of vertices at a lower level than previously. Thus, the cost of all calls to COMBINE is $O(nG(n))$ plus the cost of shifting vertices from one father to another in step 2 of COMBINE.

It remains only to show that the total number of shifts of vertices over all calls of COMBINE is bounded by a constant times $nG(n)$. In executing an instruction MERGE(s_1, s_2), no more than $G(n)$ incomplete vertices are shifted, at most one at each level. Thus we need only count shifts of complete vertices.

Consider step 2 of COMBINE. The new subtree with root s_2 is referred to as the CARRY unless the subtree consists solely of the vertex v_2 in which case we say, "there is no CARRY." The new subtree with root s_1 is referred to as the RESULT. The number of shifts of complete vertices is counted as follows. If a complete vertex is shifted, and there is no CARRY at this execution of COMBINE, charge the cost to the vertex shifted. If there is a CARRY, set the cost of each vertex in the CARRY to zero, and distribute the cost uniformly among the vertices in the RESULT.

Each time a vertex is moved, either its new father has at least twice as many sons as its old father, or there is a CARRY. Thus, a vertex at level i is moved at most $F(i)$ times before a CARRY is produced. Once a CARRY is produced, the root of the RESULT is complete, and its sons are never moved again. Hence, a vertex at level i can accumulate a cost of at most $2F(i)$, that is, $F(i)$ due to charges to itself and $F(i)$ for its share of the costs previously charged to the sons of the root of CARRY.

To compute the total cost of shifting complete vertices, note that a complete vertex at level i has $F(i)$ descendant leaves. Redistribute the cost of each complete vertex uniformly among its descendant leaves. Since no leaf has more than $G(n)$ ancestors, the cost charged to any leaf is bounded by $2G(n)$. Hence, the complete cost of moving vertices is $O(nG(n))$. Since $m \geqq n$ is assumed, we have our result.

COROLLARY. *If we may assume $m \leqq an$ for some fixed constant a, then Algorithm 1 is $O(nG(n))$.*

It should be observed that the set merging algorithm can be modified to handle a problem which is in some sense the inverse of set merging. Initially given a set consisting of integers $\{1, 2, \cdots, n\}$, execute a sequence of two types of instructions:

(i) PARTITION(i), (ii) FIND(i),

where i is an integer between 1 and n. Each time an instruction of the form PARTITION(i) occurs, replace the set S containing i into two sets:

$$S_1 = \{j | j \in S \text{ and } j \leqq i\},$$
$$S_2 = \{j | j \in S \text{ and } j > i\}.$$

Each set is named by the largest integer in the set. Each time an instruction of the form FIND(i) occurs, print the name of the set currently containing the integer i. To handle this partitioning problem, use the same data structure as before, but start with a single tree with n leaves. The leaves in order from left to right correspond to the integers from 1 to n. To execute PARTITION(i), start at the leaf corresponding to the integer i and traverse a path to the root of the tree to partition the tree into two trees.

All vertices to the left of the path are placed on one tree, all vertices to the right of the path are placed on the other. Vertices on the path are replaced by two vertices, one for each subtree, unless the vertex i is the rightmost descendant leaf of the vertex, in which case the vertex is placed on the left subtree. Assume that vertices v and w are on the path and w is a son of v. The sons of v are partitioned as follows. Simultaneously, start counting the sons of v to the left of w, including w, starting with the leftmost son of v, and start counting the sons of v to the right of w. Cease counting as soon as one of the two counts is complete. (The reason for the simultaneous counting is to make the cost of counting proportional to the smaller of the two counts.) The sons of v can now be partitioned at a cost proportional to the smaller of the two sets by moving the smaller number of sons.

The analysis of the running time is similar to that of the set merging algorithm, and thus only a brief sketch is given. Note that a vertex can have at most two incomplete sons, only one of which can be moved in the execution of any PARTITION instruction. Thus at most $G(n)$ incomplete vertices are moved in executing one PARTITION instruction.

To bound the cost of moving a complete vertex, note that each time a vertex is moved, its new father has at most half as many sons as the old father. Thus a vertex at level i can be moved at most $F(i)$ times. Since a complete vertex at level i has $F(i)$ leaves, distributing to its leaves the cost of all moves of a given vertex while it is complete gives at most a cost of one to each of its leaves. Since a leaf has at most $G(n)$ ancestors, the cost of moving all complete vertices is bounded by $nG(n)$.

3. The second set merging algorithm.

We now consider a second algorithm to simulate a sequence of MERGE and FIND instructions.

This algorithm also uses a tree data structure to represent a set. But here, all vertices of the tree, rather than just the leaves, correspond to elements in the set. Moreover, tree links point only from son to father.

ALGORITHM 2.

1. Initially each set $\{i\}$, $1 \leqq i \leqq n$, is represented by a tree consisting of a single vertex with the name of the set at the vertex.

2. To merge two sets, the corresponding trees are merged by making the root of the tree with fewer vertices a son of the root of the other tree. Ties can be broken arbitrarily. Attach the appropriate name to the remaining root.

3. To execute FIND(i), follow the path from vertex i to the root of its tree and print the name of the set. Make each vertex encountered on the path a son of the root. (This is where the algorithm differs substantially from balanced tree schemes!)

The above algorithm, except for the balancing feature of merging small trees into large, was suggested by Knuth [7] and is attributed by him to Tritter. The entire algorithm, including this feature, was implemented by McIlroy [2] and Morris in a spanning tree algorithm. The analysis of the algorithm without the balancing feature was completed by Fischer [8], who showed that $O(n \log n)$ is a lower bound, and Paterson [9], who showed it to be an upper bound. Our analysis shows that the algorithm with the balancing scheme is $O(nG(n))$ at worst. Thus it is substantially better than the one without the balancing.

We now introduce certain concepts needed to analyze Algorithm 2. Let α be a fixed sequence of MERGE and FIND instructions. Let v be a vertex. Define the *rank* of v, with respect to α, denoted $R(v)$, as follows. If in the execution of α by Algorithm 2, v never receives a son, then $R(v) = 0$. If v receives sons $v_1, v_2, \cdots, v_k$ at any time during the execution, then the rank of v is $\max\{R(v_i)\} + 1$. It is not hard to prove that the rank of v is equal to the length of the longest path from v to a descendant leaf in the tree that would occur if the FIND instructions and their attendant movement of vertices were ignored in the execution of α.

LEMMA 3. *If the rank of v with respect to α is r, then at some time during the execution of α, v was the root of a tree with at least 2^r vertices.*

Proof. The proof is by induction on the value of r. The case $r = 0$ is trivial. Assume the lemma to be true for all values up to $r - 1$, $r \geqq 1$. If v is of rank r, at some point v must become the father of some vertex u of rank $r - 1$. This event could not occur during a FIND, for if it did, then u would have previously been a descendant of v with some vertex w between them. But then w is of rank at least r and v of rank at least $r + 1$ by the definition of rank.

Thus, u must be the root of a tree T which is merged with a tree T' having root v. By the inductive hypothesis, T has at least 2^{r-1} vertices, since a root cannot lose descendants and a nonroot cannot gain descendants, and hereafter u will no longer be a root. By step 2 of Algorithm 2, T' has at least as many descendants as T. The resulting tree has at least 2^r vertices and has v as root.

LEMMA 4. *Each time a vertex v gets a new father w during the execution of Algorithm 2, that father has a rank higher than any previous father $u \neq w$ of v.*

Proof. If v, formerly a son of u, becomes a son of w, it must be during a FIND. Then w is an ancestor of u and hence of higher rank by definition.

LEMMA 5. *For each vertex v and rank r, there is at most one vertex u of rank r which is ever an ancestor of v.*

Proof. Suppose there were two such u's, say u_1 and u_2. Assume, without loss of generality, that v first becomes a descendant of u_1. Then u_2 is of higher rank than u_1 by Lemma 4 and the fact that at all times, paths up trees are of monotonically increasing rank. This contradicts the assumption that u_1 and u_2 were of rank r.

LEMMA 6. *There are at most $n/2^r$ vertices of rank r.*

Proof. Each vertex of rank r at some point is the root of a tree with at least 2^r vertices by Lemma 3. No vertex could be the descendant of two vertices of rank r by Lemma 5. This implies that there are at most $n/2^r$ vertices of rank r.

For $j \geqq 1$, define $\mathscr{s}_j$, the *j-th rank group*, as follows:[1]

$$\mathscr{s}_j = \{v \mid \log^{j+1}(n) < R(v) \leqq \log^j(n)\}.$$

Note that the higher the rank group, the lower the ranks of the vertices it contains.

LEMMA 7. $|\mathscr{s}_j| \leqq 2n/\log^j(n)$.

Proof. Since there are at most $n/2^r$ vertices of rank r, we have

$$|\mathscr{s}_j| \leqq \sum_{i=\log^{j+1}n}^{\log^j n} \frac{n}{2^i} \leqq \frac{n}{\log^j n}(1 + \tfrac{1}{2} + \tfrac{1}{4} + \cdots) \leqq \frac{2n}{\log^j n}\,.$$

LEMMA 8. *Each vertex is in some $\mathscr{s}_j$ for $1 \leqq j \leqq G(n) + 1$.*

Proof. By Lemma 6, no vertex has rank greater than $\log n$, so $j \geqq 1$ may be assumed. Thus to prove $j \leqq G(n) + 1$, we need only show that $\log^{G(n)+2}(n) < 0$. From the definition of $G(n)$, $n \leqq F(G(n))$, and so

$$\log^{G(n)+2}(n) \leqq \log^{G(n)+2}F(G(n)).$$

Thus it suffices to show that $\log^{i+2}F(i) < 0$. We shall actually show that $\log^{i+2}2F(i) \leqq 0$. The proof is by induction on i. For $i = 0$, the result is obvious. Assume the induction hypothesis true up to $i - 1$. Then $\log^{i+2}2F(i) \leqq \log^{i+1}(1 + \log F(i)) = \log^{i+1}(1 + \log F(i-1) + F(i-1))$, which is less than or equal to $\log^{i+1}2F(i-1)$, since $F(i-1) \geqq 1$ for $i \geqq 1$ and since $\log x \leqq x - 1$ for integer $x \geqq 1$. Thus by the inductive hypothesis, $\log^{i+2}2F(i) \leqq 0$. From this it follows that $\log^{i+1}F(i) < 0$, and the proof is complete.

THEOREM 2. *Given $n \geqq 2$ objects, the time necessary to execute any sequence of $m \geqq n$* MERGE *and* FIND *instructions on these objects by Algorithm* 2 *is $O(mG(n))$.*

Proof. The cost of the MERGE instructions is clearly $O(n)$. The cost of executing all FIND instructions is proportional to the number of instructions plus the sum, over all FIND's, of the number of vertices moved by that FIND. We now show that this sum is bounded by $O(mG(n))$.

Let v be a vertex in $\mathscr{s}_j$. If before a move of v the father of v is in a lower rank group (smaller value of j), assign the cost to the FIND instruction. Otherwise, assign the cost to v itself. For an instruction FIND(i), consider the path from vertex i to the root of its tree. The ranks of vertices along the path to the root are monotonically increasing, and hence there can be on the path at most $G(n)$ vertices whose fathers are in a lower rank group. Hence no FIND instruction is assigned a cost more than $G(n)$.

By Lemma 7, there are at most $2n/\log^j n$ vertices in $\mathscr{s}_j$, and by Lemma 4, each vertex in $\mathscr{s}_j$ can be moved at most $\log^j n$ times before its new father is in $\mathscr{s}_{j-1}$ or a lower rank group. Thus, the total cost of moving vertices in $\mathscr{s}_j$, not counting moves of a vertex whose father is in a lower rank group, is $2n$. Since there are at most $G(n) + 1$ rank groups, the total cost exclusive of that charged to FIND's is $O(nG(n))$. Hence, the total cost of executing the sequence of MERGE and FIND instructions is $O(mG(n))$.

[1] $\log^0(n)$ is n and $\log^{j+1}(n) = \log(\log^j(n)) = \log^j(\log(n))$. Base 2 logarithms are assumed throughout.

COROLLARY. *If $m \leqq an$ for fixed constant a, then Algorithm 2 requires $O(nG(n))$ time.*

4. An application. One application of the set merging algorithms is to process a sequence of instructions of the forms INSERT(i), $1 \leqq i \leqq n$, and MIN. Start with a set S which is initially the empty set. Each time an instruction INSERT(i) is encountered, adjoin the integer i to the set S. Each time a MIN instruction is executed, delete the minimum element from the set S and print it. We assume that for each i, the instruction INSERT(i) appears at most once in the sequence of instructions, and at no time does the number of MIN instructions executed exceed the number of INSERT instructions executed. Note that as a special case, we could sort k integers from one to n by executing INSERT instructions for each integer, followed by k MIN instructions.

The algorithm which we shall give for this problem is *off-line*, in the sense that the entire sequence of instructions must be present before processing can begin. In contrast, Algorithms 1 and 2 are *on-line*, as they can execute instructions without knowing the subsequent instructions with which they will be presented.

Let $I_1, I_2, \cdots, I_r$ be the sequence of INSERT and MIN instructions to be executed. Note that $r \leqq 2n$. Let l of the instructions be MIN. We shall set up a MERGE-FIND problem whose on-line solution will allow us to simulate the INSERT-MIN instructions. We create l objects M_i, $1 \leqq i \leqq l$, where M_i "represents" the ith MIN instruction. We also create n objects X_i, $1 \leqq i \leqq n$, where X_i "represents" the integer i. Suppose, in addition, that there are two arrays which, given i, enable us to find M_i or X_i in a single step. The following algorithm determines for each i, that MIN instruction, if any, which causes i to be printed. Once we have that information, a list of the integers printed by $I_1, I_2, \cdots, I_r$ is easily obtained in $O(n)$ steps.

ALGORITHM 3.

```
1. Initially use MERGE instructions to create the following sets:
     (i) $S_i$, $2 \leqq i \leqq l$, consists of object $M_{i-1}$ and all those $X_j$ such that
         INSERT($j$) appears between the $i - 1$st and $i$th MIN instructions.
    (ii) $S_1$ consists of all $X_j$ such that INSERT($j$) appears prior to the first
         MIN.
   (iii) $S_\infty$ consists of $M_l$ and all $X_j$'s not placed by (i) or (ii).
2. for $i \leftarrow 1$ until $n$ do
     begin
       FIND($X_i$);[2]
       let $X_i$ be in $S_j$;
       if $j \neq \infty$ then
       begin
         TIME($i$) $\leftarrow j$;
         FIND($M_j$);
         let $M_j$ be in $S_k$;
         MERGE($S_j$, $S_k$)
       end
     end
```

[2] We are taking the liberty of using X_i, M_i and S_i as arguments of FIND and MERGE, rather than integers, as these instructions were originally defined. It is easy to see that objects and set names could be indexed, so no confusion should arise.

The strategy behind Algorithm 3 is to let M_k after the ith iteration of step 2 lie in that set S_j such that the jth MIN instruction is the first one following the kth MIN having the property that none of the integers up to i are printed by the jth MIN. Likewise, X_m will be in S_j if and only if the jth MIN is the first MIN following INSERT(m) which does not print any integer up to i. We can formalize these ideas in the next lemma.

LEMMA 9. (a) *After the i-th iteration of step* 2 *in Algorithm* 3, $S_j, j \neq \infty$, *contains* X_m *(resp.* M_k*) if and only if j is the smallest number such that the j-th* MIN *follows* INSERT(m) *(resp. the k-th* MIN*), and none of* $1, 2, \cdots, i$ *is printed by the j-th* MIN.

(b) TIME(i) $\leftarrow j$ *by Algorithm* 3 *if and only if i is printed by the j-th* MIN.

Proof. We show (a) and (b) simultaneously by induction on i. The basis, $i = 0$, is true by step 1 of Algorithm 3. Part (b) of the induction holds, since if X_i is in S_j when the ith iteration begins, it must be that the jth MIN follows INSERT(i) but does not cause any integer smaller than i to be printed. However, by hypotheses, any MIN's between INSERT(i) and the jth MIN do print out smaller integers. Thus i is the smallest integer available when the jth MIN is executed.

For part (a) of the induction, we need only observe that the set in which an X_m or M_k belongs does not change at iteration i unless it was in S_j. Then, since the jth MIN prints i, Algorithm 3 correctly merges S_j with the set containing M_j, that is the set of the next available MIN (or S_∞ if no further MIN's are available).

THEOREM 3. *A sequence of* INSERT *and* MIN *instructions on integers up to n can be simulated off-line in* $O(nG(n))$ *time.*

Proof. We use Algorithm 3 to generate a sequence of MERGE and FIND instructions. At most $2n$ MERGE's are needed in step 1 and at most $2n$ FIND's and n MERGE's are needed in step 2, a total of at most $5n$ instructions. We can thus, by the corollary to either Theorem 1 or 2, simulate this sequence on-line in $O(nG(n))$ steps. In doing so, we shall obtain the array TIME(i). The order in which integers are printed by the MIN instructions can be obtained in a straightforward manner from TIME in $O(n)$ steps.

Acknowledgment. We are indebted to M. J. Fischer for bringing to our attention an error in an earlier analysis of the second algorithm which attempted to show that the time bound was linear.

REFERENCES

[1] B. A. GALLER AND M. J. FISCHER, *An improved equivalence algorithm*, Comm. ACM, 7 (1964), pp. 301–303.

[2] M. D. MCILROY, Private communication, Sept. 1971.

[3] R. E. STEARNS AND D. J. ROSENKRANTZ, *Table machine simulation*, 10th SWAT, 1969, pp. 118–128.

[4] R. E. STEARNS AND P. M. LEWIS, *Property grammars and table machines*, Information and Control 14 (1969), pp. 524–549.

[5] D. E. KNUTH, *The Art of Computer Programming*, vol. III, Addison-Wesley, Reading, Mass., 1973.

[6] R. E. TARJAN, *On the efficiency of a good but not linear set union algorithm*, Tech. Rep. 72–148, Dept. of Comp. Sci., Cornell University, Ithaca, N.Y., 1972.

[7] D. E. KNUTH, *The Art of Computer Programming*, vol. II, Addison-Wesley, Reading, Mass., 1969. See also L. Guibas and D. Plaisted, *Some combinatorial research problems with a computer science flavor*, Combinatorial Seminar, Stanford Univ., Stanford, Calif., 1972.

[8] M. J. FISCHER, *Efficiency of equivalence algorithms*, Complexity of Computer Computations, Miller et al., eds., Plenum Press, New York, 1972, pp. 153–167.

[9] M. Paterson, unpublished report, University of Warwick, Coventry, Great Britain.

SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

THE HARDEST CONTEXT-FREE LANGUAGE*

SHEILA A. GREIBACH†

Abstract. There is a context-free language L_0 such that every context-free language is an inverse homomorphic image of L_0 or $L_0 - \{e\}$. Hence the time complexity of recognition of L_0 is the least upper bound for time complexity of recognition of context-free languages. A similar result holds for quasirealtime Turing machine languages. Several languages are given such that deterministic and nondeterministic polynomial time acceptance are equivalent if and only if any one of them is deterministic polynomial time acceptable.

Key words. context-free, quasirealtime, complexity, polynomial time recognition

1. Introduction. There have been several studies of the complexity of recognition of context-free languages. It is known that the family of deterministic context-free languages is recognizable in linear time [1] but not in real time [2]. Context-free languages are recognizable off-line in space $\log^2 n$ [3]. They can be recognized by deterministic multitape Turing machines in time complexity n^3 [4]; linear context-free languages can be recognized in time n^2 [5].

It is not known if any of these upper bounds is also a lower bound. For example, it is not known whether there is any context-free language which cannot be accepted in linear time by a deterministic multitape Turing machine.

In this paper we show that there exists a "hardest" context-free language L_0 in the sense that any reasonable deterministic recognition procedure for the class of context-free languages will have as its time complexity the time it takes to recognize L_0 and will have as its tape complexity the space it takes to recognize L_0. The demonstration of this "universal" property of L_0 is constructive in the sense that an algorithm for recognizing L_0 in time $p(n)$ (p a polynomial or other semihomogeneous function[1]) or in space $p(n)$ can be mechanically transformed into an equally efficient recognizer for any other context-free language.

The idea behind this result can be extended to exhibit a specific language $\hat{L}_0$ holding the same relationships to $\mathcal{Q}$ (the family of languages accepted in "realtime" by nondeterministic multitape Turing machines). That is, if $\hat{L}_0$ can be accepted in time $p(n)$ by a deterministic multitape Turing machine for any polynomial p, then all members of $\mathcal{Q}$ can be so recognized. Since Book has shown that the class $\mathcal{P}$ of languages accepted by deterministic Turing machines in polynomial time is equal to $\mathcal{NP}$ (the class of languages accepted in polynomial time by nondeterministic Turing machines) if and only if $\mathcal{Q} \subseteq \mathcal{P}$ [6], it follows that $\mathcal{P} = \mathcal{NP}$ if and only if $\hat{L}_0$ is in $\mathcal{P}$. Furthermore, all reductions involved can actually be performed by polynomially time bounded Turing machines.

* Received by the editors December 1, 1972, and in revised form, April 18, 1973. The research represented in this paper was supported in part by the National Science Foundation under Grant no. GJ-803.

† Department of System Science, University of California at Los Angeles, Los Angeles, California 90024.

[1] A function f is *semihomogeneous* if for every $k_1 > 0$, there is a $k_2 > 0$, such that for all $x > 0$, $f(k_1 x) \leqq k_2 f(x)$.

We show that every context-free language can be expressed[2] as $h^{-1}(L_0)$ or $h^{-1}(L_0 - \{e\})$ for a homomorphism h.[3] The algebraic statement is: the family of context-free languages is a principal AFDL; similarly, $\mathscr{Q}$ is a principal AFDL. By way of contrast, the family of deterministic context-free languages is not a principal AFDL [7].

2. Main results. In this section we exhibit a context-free language L_0 such that every context-free language is an inverse homomorphic image of L_0 or $L_0 - \{e\}$. As one might guess, L_0 is a nondeterministic version of a Dyck set.

DEFINITION 2.1. Let $T = \{a_1, a_2, \bar{a}_1, \bar{a}_2, c, ¢\}$ where $a_1, a_2, \bar{a}_1, \bar{a}_2, c$, and $¢$ are all distinct. Let D be the context-free language generated by $S \to SS$, $S \to a_1 S \bar{a}_1$, $S \to a_2 S \bar{a}_2$, $S \to e$.[4] We call D a *Dyck set on two letters.* Let d be new and let

$$L_0 = \{e\} \cup \{x_1 c y_1 c z_1 d \cdots d x_n c y_n c z_n d \mid n \geqq 1, y_1 \cdots y_n \in ¢D, x_i, z_i \in T^* \text{ for all } i,$$
$$y_i \in \{a_1, a_2, \bar{a}_1, \bar{a}_2\}^* \text{ for } i \geqq 2\}$$

Thus the language L_0 selects one subword from each group set off by d's in such a way that the concatenation of the choices belongs to $¢D$, the Dyck set on two letters, preceded by $¢$. In the construction below, we let L_0 encode derivations in a context-free grammar.

THEOREM 2.1. *If L is a context-free language, there is a homomorphism h such that $L - \{e\} = h^{-1}(L_0 - \{e\})$.*[5]

Proof. We may assume that L does not contain the empty word. Hence there is a grammar $G = (V, \Sigma, P, S)$ such that $L = L(G)$ and $P \subseteq [(V - \Sigma) \times \Sigma(V - \Sigma - \{S\})^*]$. Thus the rules in P are of the form $Z \to ay$ for $\bar{a} \in \Sigma$ and y containing neither terminals nor S; we say that G is in *standard* form [8].

Order $V - \Sigma$ in some way, $V - \Sigma = \{Y_1, \cdots, Y_n\}$, such that $Y_1 = S$. Define functions ξ, $\bar{\xi}$ from productions into $\{a_1, a_2, \bar{a}_1, \bar{a}_2\}^*$ as follows. If p is the production $Y_i \to a$, then $\bar{\xi}(p) = \bar{a}_1 \bar{a}_2^i \bar{a}_1$; if p is the production $Y_i \to a Y_{j_1} \cdots Y_{j_m}$, $m \geqq 1$, then $\bar{\xi}(p) = \bar{a}_1 \bar{a}_2^i \bar{a}_1 a_1 a_2^{j_m} a_1 \cdots a_1 a_2^{j_1} a_1$. If $i \neq 1$, $\xi(p) = \bar{\xi}(p)$; if $i = 1$, $\xi(p) = ¢ a_1 a_2 a_1 \bar{\xi}(p)$.

If $P_a = \{p_1, \cdots, p_m\}$ is the set of all productions whose right-hand sides start with a, then $h(a) = c\xi(p_1)c \cdots c\xi(p_m)cd$; without loss of generality we can assume $P_a \neq \varnothing$ for all a in Σ.

Thus we must show that $h(w) = x_1 c y_1 c z_1 d \cdots d x_k c y_k c z_k d$ with $k = |w|$,[6] $y_i \in \{a_1, a_2, \bar{a}_1, \bar{a}_2\}^*$, $x_i z_i \in T^*$, $1 \leqq i \leqq k$, and $y_1 \cdots y_k \in ¢D$ if and only if

[2] The symbol e is used for the empty string; L^+ is the closure of L under concatenation and $L^* = L^+ \cup \{e\}$.

[3] Notice that for any language L and homomorphism h, if L can be recognized within time $p(n)$ (tape $p(n)$), then $h^{-1}(L)$ can be so recognized, providing p is semihomogeneous [2], [16]; for $p(n) = 2^n$, e.g., this may not be true.

[4] A *context-free grammar* $G = (V, \Sigma, P, S)$ has finite vocabulary V, terminal vocabulary $\Sigma \subseteq V$, initial symbol $S \in V - \Sigma$ and finite production set $P \subset (V - \Sigma) \times V^*$. If $(Z, y) \in P$, written $Z \to y$, and $u, v \in V^*$, then $uZv \Rightarrow uyv$; the relation $\overset{*}{\Rightarrow}$ is the transitive reflexive extension of $\Rightarrow$. The *context-free language* generated by G is $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$.

[5] For a homomorphism h, $h^{-1}(L) = \{w \mid h(w) \in L\}$.

[6] For a word w, $|w|$ is the length of w.

$y_1 \cdots y_k$ encodes the productions used in a left-to-right derivation of w from S.[7]

It is well known (see [9] and [10] for further discussion) that D is the set of all words canceling to e under the cancellation rules $xa_1\bar{a}_1y \simeq e \simeq xa_2\bar{a}_2y$ and that for each w in $\{¢, a_1, a_2, \bar{a}_1, \bar{a}_2\}^*$ there is a unique word of minimal length, which we shall call $\mu(w)$, such that $w \simeq \mu(w)$. Further, w is in $\text{Init}\, D = \{x \mid \exists y, xy \in D\}$ if and only if $\mu(w) \in \{a_1, a_2\}^*$. It suffices to show that for $b_1, \cdots, b_k \in \Sigma$, $Y_{i_1}, \cdots, Y_{i_r} \in (V - \Sigma - \{S\})$, $S \overset{*}{\Rightarrow} b_1 \cdots b_k Y_{i_1} \cdots Y_{i_r}$ if and only if $h(b_1 \cdots b_k) = x_1cy_1czd \cdots dx_kcy_kcz_kd$, where $y_1 \cdots y_k \in \text{Init}\, ¢D$ and $\mu(y_1 \cdots y_k) = ¢a_1a_2^{i_r}a_1 \cdots a_1a_2^{i_1}a_1$. If $p_1, \cdots, p_k$ are the productions used, in order, in a left-to-right derivation of $b_1 \cdots b_k Y_{i_1} \cdots Y_{i_r}$, then $y_i = \xi(p_i)$, $1 \leqq i \leqq k$.

The proof is by induction on k; we shall sketch the induction step and leave other details to the reader. Assume the result for $k \geqq 1$. First, suppose $k \geqq 1$, $S \overset{*}{\Rightarrow} b_1 \cdots b_k Y_{i_1} \cdots Y_{i_r}$, production p is $Y_{i_1} \to b_{k+1} Y_{j_1} \cdots Y_{j_t}$ and $h(b_1 \cdots b_k) = x_1cy_1cz_1d \cdots dx_kcy_kcz_kd$, where $\mu(y_1 \cdots y_k) = ¢a_1a_2^{i_r}a_1 \cdots a_1a_2^{i_1}a_1$.

If we let

$$y_{k+1} = \xi(p) = \bar{a}_1\bar{a}_2^{i_1}\bar{a}_1a_1a_2^{j_t}a_1 \cdots a_1a_2^{j_1}a_1,$$

then $h(b_{k+1}) = x_{k+1}cy_{k+1}cz_{k+1}cd$, and

$$\begin{aligned}\mu(y_1 \cdots y_ky_{k+1}) &= ¢a_1a_2^{i_r}a_1 \cdots a_1a_2^{i_1}a_1y_{k+1} \\ &= a_1a_2^{i_r}a_1 \cdots a_1a_2^{i_2}a_1a_1a_2^{j_t}a_1 \cdots a_1a_2^{j_1}a_1,\end{aligned}$$

while

$$S \overset{*}{\Rightarrow} b_1 \cdots b_k Y_{i_1} \cdots Y_{i_r} \Rightarrow b_1 \cdots b_k b_{k+1} Y_{j_1} \cdots Y_{j_t} Y_{i_2} \cdots Y_{i_r}.$$

On the other hand, suppose $h(b_1 \cdots b_{k+1}) = x_1cy_1cz_1d \cdots dx_{k+1}cy_{k+1}cz_{k+1}d$ and $\mu(y_1 \cdots y_{k+1}) \in ¢\{a_1, a_2\}^*$. Then, examining h, we see that we must have

$$\mu(y_1 \cdots y_k) = ¢a_1a_2^{i_r}a_1 \cdots a_1a_2^{i_1}a_1$$

and

$$\mu(y_{k+1}) = \xi(p) = \bar{a}_1\bar{a}_2^{s}\bar{a}_1a_1a_2^{j_t}a_1 \cdots a_1a_2^{j_1}a_1$$

for some production

$$p: Y_s \to b_{k+1} Y_{j_1} \cdots Y_{j_t}.$$

But $\mu(y_1 \cdots y_ky_{k+1}) \in ¢\{a_1, a_2\}^*$ implies that $s = i_1$. By the induction hypothesis,

$$S \overset{*}{\Rightarrow} b_1 \cdots b_k Y_{i_1} \cdots Y_{i_r},$$

and hence

$$S \overset{*}{\Rightarrow} b_1 \cdots b_{k+1} Y_{j_1} \cdots Y_{j_t} Y_{i_1} \cdots Y_{i_r}.$$

We notice in passing that if we use the proof of Theorem 2.1 to express a context-free language L as $h^{-1}(L_0)$, the homomorphism h applied to a word w in

[7] If in every step of a derivation the leftmost nonterminal is expanded, it is a left-to-right derivation.

L encodes possible derivations for w in a context-free grammar G for L. Thus, for example, it is possible to select a grammar G_0 for L_0 such that the number of derivations of $h(w)$ in G_0 is precisely the number of derivations of w in G; in this sense homomorphism h "preserves multiplicities".[8] Furthermore, G_0 can be so selected that an efficient parser for L_0 (i.e., a recognizer which gives as output for y appropriate representations of derivations of y in G_0) can be converted automatically into an efficient parser for L.

DEFINITION 2.2. An AFDL is a family of languages containing at least one nonempty e-free[9] language and closed under inverse gsm mappings,[10] marked union and star,[11] and removal of endmarkers.[12] The least AFDL containing a language L is denoted by $\mathscr{F}_d(L)$ and is called a *principal* AFDL.

Since a homomorphism is a gsm mapping, the following corollary is immediate.

COROLLARY. *The family of context-free languages forms a principal* AFDL.

By the results of Chandler [11], the family of context-free languages can be accepted deterministically by a family of one-way machines with a *finite* number of tape symbols and instructions which define only context-free languages. So there is a deterministic acceptance scheme for the context-free languages. However, this scheme is of no practical use since it consists of reading an input, performing a finite state translation (a gsm mapping) onto a working tape, and then consulting an oracle at the end.

What is more significant is that we have a rather precise way of saying that the complexity of the family of context-free languages is precisely the complexity of the language L_0. If we have a deterministic machine M accepting a language L in time $p(n)$ and a homomorphism h, then we can construct from M and h an acceptor M' for $h^{-1}(L)$. Basically M' translates its input w symbol by symbol into $h(w)$ and feeds $h(w)$ into M. Thus w will be accepted by M' in time $p(|h(w)|) + |w|$ if $h(w)$ is accepted by M'. Hence if p is semihomogeneous (and polynomials are semihomogeneous) and M is a multitape Turing machine, we can build M' to accept $h^{-1}(L)$ in time $p(n)$. Similarly, if M is an off-line Turing machine accepting L in space $p(n)$, we can construct an off-line Turing machine accepting $h^{-1}(L)$ in space $p(n)$. (Notice that this argument applies not just to Turing machines but also to any "reasonable" deterministic recognition procedure.)

Hence, since we already know that L_0 as a context-free language *is* in $\mathscr{P}$, Theorem 2.1 tells us that the time necessary to recognize deterministically L_0 is

[8] The question of preservation of multiplicities was brought to the author's attention by S. Eilenberg.

[9] A language is *e-free* if it does not contain the empty word.

[10] A *gsm* is a sextuple $M = (K, \Sigma, \Delta, \delta, \lambda, q_0)$ where K, Σ, and Δ are finite sets of *states*, *inputs* and *outputs*, δ, the *transition* function, is a function from $K \times \Sigma$ into K, λ, the *output* function, is a function from $K \times \Sigma$ into Δ^* and $q_0 \in K$. We extend δ and λ to $K \times \Sigma^*$ by $\delta(q, e) = q$, $\lambda(q, e) = e$, $\delta(q, xa) = \delta(\delta(q, x), a)$ and $\lambda(q, xa) = \lambda(q, x)\lambda(\delta(q, x), a)$, for $q \in K$, $x \in \Sigma^*$ and $a \in \Sigma$. If $\delta(q, a) = p$, $\lambda(q, a) = y$, we sometimes write $(q, a) \to (p, y)$ and if $\delta(q, w) = p$, $\lambda(q, w) = y$, we write $(q, w) \overset{*}{\Rightarrow} (p, y)$ for p, $q \in K$, $a \in \Sigma$, $w \in \Sigma^*$ and $y \in \Delta^*$. We define $M(w) = \lambda(q_0, w)$, $M^{-1}(w) = \{y | M(y) = w\}$, $M(L) = \{M(w) | w \in L\}$ and $M^{-1}(L) = \{y | M(y)\} \in L$ for a word w or a language L, and call $M(L)$ a *gsm mapping* and $M^{-1}(L)$ an *inverse gsm mapping* of L.

[11] If $L_1 \cup L_2 \subseteq \Sigma^*$ and $c \notin \Sigma$, then $L_1 \cup cL_2$ is a *marked union* of L_1 and L_2 and $(L_1c)^*$ a *marked star* of L_1.

[12] If $c \notin \Sigma$ and $L \subseteq \Sigma^*c$, then c is an *endmarker* of L, and removing c yields $L/c = \{w | wc \in L\}$.

the (realizable) least upper bound of time complexity for deterministic recognition of the class of context-free languages. We believe L_0 to be the first known instance of such a language.

We can extend our arguments to show that the family $\mathscr{Q}$ is also a principal AFDL. The family $\mathscr{Q}$–called the family of *quasirealtime languages*–is the family of all and only those languages expressible as $h(L_1 \cap L_2 \cap L_3)$ where the L_i are context-free and h is a length preserving homomorphism[13][12]. For our purposes we can take this as a definition of $\mathscr{Q}$. Examining the proof of Theorem 2.1, we see that a similar construction would apply to members of $\mathscr{Q}$.

Instead of one Dyck set we need three. Let D' and D'' be two distinct renamings of D,[14] using vocabularies Σ_2 and Σ_3, where $D \subseteq \Sigma_1^*$.

We define the language $\hat{L}_0$ in words. It contains the empty string and every nonempty word in $\hat{L}_0$ ends with d's. Let $T = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \{c, ¢, d, \$\}$. A word w in T^*d containing $m\,d$'s is in $\hat{L}_0$ if before the first d and then between each consecutive pair of d's in $\hat{L}_0$ one can select, sequentially, a triple of distinct nonempty subwords (x_i, y_i, z_i) such that

(i) $x_1 \cdots x_m \in ¢D$, $y_1 \cdots y_m \in ¢D'$ and $z_1 \cdots z_m \in ¢D''$, and

(ii) for each i, a subword of the form $\$\alpha_i c x_i c \beta_i c y_i c \gamma_i c z_i c \eta_i \$$ lies between the $(i-1)$st and ith d's and $\alpha_i \beta_i \gamma_i \eta_i$ contains *no* occurrence of $\$$ or d.

Thus $\hat{L}_0$ resembles a shuffle of three copies of L_0 with the crucial restriction that each triple selected lie within the *same* pair of $\$$'s. Akin to Theorem 2.1, we have the following result.

THEOREM 2.2. *If $L \subseteq \Sigma^*$ is in $\mathscr{Q}$, there is a homomorphism h such that*

$$L - \{e\} = h^{-1}(\hat{L}_0 - \{e\}).$$

Proof. The proof follows the lines of the previous one. We have $L = (L(G_1) \cap L(G_2) \cap L(G_3))$, where $\lambda : \Delta^* \to \Sigma^*$ is a length-preserving homomorphism and each G_i is a context-free grammar in standard form. For each G_i we define a homomorphism h_i from Δ^* into Σ_i^* similar to the one in the proof of Theorem 2.1; thus $h_i(b) = c\xi_i(p_1)c \cdots c\xi_i(p_m)c$, where ξ_i is the appropriate function mapping productions of G_i into Σ_i and $\{p_1, \cdots, p_m\}$ is the set of all productions of G_i of the form $Z \to by$. Then if $\lambda^{-1}(a) = \{b | \lambda(b) = a\} = \{b_1, \cdots, b_t\}$, we let

$$h(a) = \$h_1(b_1)h_2(b_1)h_3(b_1)\$ \cdots \$h_1(b_t)h_2(b_t)h_3(b_t)\$d.$$

Then $L - \{e\} = h^{-1}(\hat{L}_0 - \{e\})$ as desired.

COROLLARY 1. *Every language in $\mathscr{Q}$ is accepted deterministically in polynomial time if and only if $\hat{L}_0$ is so accepted.*

Now R. Book [6] has shown that $\mathscr{P}$ (the family of languages accepted by deterministic Turing machines in polynomial time) is equal to $\mathscr{NP}$ (the family of languages accepted by nondeterministic Turing machines in polynomial time) if and only if $\mathscr{Q} \subseteq \mathscr{P}$. Hence we have the following corollary.

COROLLARY 2. *$\mathscr{P} = \mathscr{NP}$ if and only if $\hat{L}_0 \in \mathscr{P}$*

Thus the membership problem for $\hat{L}_0$ is polynomially complete in the sense of Karp [13]. The main difficulty in recognizing $\hat{L}_0$ is that a triple x_i, y_i, z_i can be

[13] That is, for any symbol a, $h(a)$ is also a symbol.

[14] A *renaming* is a length-preserving one-to-one homomorphism h: $\Sigma_1^* \to \Sigma_2^*$, where $\Sigma_1 \cap \Sigma_2 = \varnothing$; if $L \subseteq \Sigma_1^*$, then $h(L)$ is a *renaming* of L.

selected only if cx_ic, cy_ic and cz_ic lie between the same pair of $\$$'s: if we could drop this restriction, we would get a member of $\mathscr{P}$. But we cannot–the resulting language is in the intersection closure of the context-free languages which is an AFDL properly contained in $\mathscr{Q}$ [12].

Indeed, results in [14] show that every language in $\mathscr{NP}$ can be obtained by polynomially-bounded erasing from a language $L_1 \cap L_2$ for L_1 and L_2 linear context-free.[15] Let us define

$$L_a = \{w\$_1 w^R | w \in \{a, b\}^*\} \quad \text{and} \quad L_1 = \{w\$_2 w^R | w \in \{0, 1\}^*\}.$$

Instead of the three Dyck sets used to form $\hat{L}_0$, "paste together" pieces of L_a and L_1 to form $\hat{L}_0'$:

$$\hat{L}_0' = \{e\} \cup \{du_1 ¢ v_1 c\alpha_1 cw_1 c\beta_1 cx_1 ¢ z_1 d \cdots du_1 ¢ v_m c\alpha_m cw_m c\beta_m cx_m ¢ z_m d | m \geqq 1,$$
$$\alpha_1 \cdots \alpha_m \in L_a, \beta_1 \cdots \beta_m \in L_1$$
$$\text{for } i \geqq 1, v_i w_i x_i \in \{a, b, 0, 1, \$_1, \$_2, c\}^*, \alpha_i \neq e \neq \beta_i,$$
$$u_i z_i \in \{a, b, 0, 1, \$_1, \$_2, c, ¢\}^*\}.$$

Then combining arguments in [6], [14] and this paper, we have the next corollary.

COROLLARY 3. *$\mathscr{P} = \mathscr{NP}$ if and only if $\hat{L}_0' \in \mathscr{P}$.*

3. Conclusions and open problems.

We have shown that the family of context-free languages is a principal AFDL, although the deterministic context-free languages do not form a principal AFDL [7].

We can show certain other nondeterministic families to be principal AFDL's. Wegbreit's results can be extended to show that the nondeterministic and deterministic context-sensitive languages form principal AFDL's [15]. Similarly, the recursively enumerable languages form a principal AFDL. The various nondeterministic tape and time bounded and deterministic tape bounded Turing machine families described in [16] and [17] are easily seen to form principal AFDL's.[16] However the same question–do they form a principal AFDL–is open in other cases such as the one-way stack [18] and checking automata languages [19], and for various subfamilies of the context-free languages such as the one-counter languages; it is likewise unknown if every linear context-free language can be obtained from one language by inverse gsm mappings and derivatives. We suspect that the answer is no in all the cases mentioned, though for varying reasons.

Of course, the major open questions posed by this work concern the actual time complexity of L_0 and $\hat{L}_0$. These appear as difficult as ever.

Acknowledgments. An earlier version of this paper and other material appears in [7]. The author would like to thank the referees for useful suggestions in preparing this revision.

[15] A homomorphism h is *polynomially bounded* on L if there is a polynomial f such that $|w| \leqq f(|h(w)|)$ for all $w \in L$.

[16] Indeed, in all these cases one can express the family as $\{h^{-1}(L), h^{-1}(L - \{e\}) | h \text{ a homomorphism}\}$ for some generator L.

REFERENCES

[1] S. Ginsburg and S. A. Greibach, *Deterministic context-free languages*, Information and Control, 9 (1966), pp. 602–648.

[2] A. L. Rosenberg, *Real-time definable languages*, J. Assoc. Comput. Mach., 14 (1967), pp. 645–662.

[3] P. M. Lewis, R. E. Stearns and J. Hartmanis, *Memory bounds for recognition of context-free and context-sensitive languages*, IEEE Conference Record on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, 1965, pp. 191–202.

[4] D. H. Younger, *Recognition and parsing of context-free languages in time n^3*, Information and Control, 10 (1967), pp. 189–208.

[5] T. Kasami, *A note on computing time for recognition of languages generated by linear grammars*, Ibid., 10 (1967), pp. 209–214.

[6] R. V. Book, *On languages accepted in polynomial time*, this Journal, 1 (1972), pp. 281–287.

[7] S. Greibach, *Jump pda's, deterministic context-free languages, principal AFDL's and polynomial time recognition*, Proc. of Fifth Annual ACM Symposium on Theory of Computing, Austin, Texas, 1973, pp. 20–28.

[8] ———, *A new normal form theorem for context-free phrase structure grammars*, J. Assoc. Comput. Mach., 12 (1965), pp. 42–52.

[9] N. Chomsky, *Context-free grammars and pushdown storage*, MIT Res. Lab. Electron Quarterly Progress Rep. 65, Cambridge, Mass., 1962.

[10] N. Chomsky and M. P. Schutzenberger, *The algebraic theory of context-free languages*, Computer Programming and Formal Systems, P. Braffort and P. Hirschberg, eds., North-Holland, Amsterdam, 1963, pp. 115–161.

[11] W. J. Chandler, *Abstract families of deterministic languages*, Proc. First ACM Symposium on Theory of Computing, Marina del Rey, Calif., 1969, pp. 21–30.

[12] R. V. Book and S. Greibach, *Quasi-realtime languages*, Math. Systems Theory, 4 (1970), pp. 97–111.

[13] R. M. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, pp. 85–104.

[14] B. Baker and R. V. Book, *Reversal-bounded multipushdown machines*, Proc. 13th Annual IEEE Symposium on Switching and Automata Theory, College Park, Maryland, 1972, pp. 207–211.

[15] B. Wegbreit, *A generator of context-sensitive languages*, J. Comput. System Sci., 3 (1969), pp. 456–461.

[16] R. V. Book, S. Greibach and B. Wegbreit, *Time- and tape-bounded turing acceptors and AFL's*, Ibid., 4 (1970), pp. 606–621.

[17] R. Book, S. Greibach, O. Ibarra and B. Wegbreit, *Tape-bounded turing acceptors and principal AFL's*, Ibid., 4 (1970), pp. 622–625.

[18] S. Ginsburg, S. Greibach and M. A. Harrison, *One-way stack automata*, J. Assoc. Comput. Mach., 14 (1967), pp. 389–418.

[19] S. Greibach, *Checking automata and one-way stack languages*, J. Comput. System Sci., 3 (1969), pp. 196–217.

SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

AN ALGORITHM FOR DETERMINING THE CHROMATIC NUMBER OF A GRAPH*

D. G. CORNEIL AND B. GRAHAM†

Abstract. A heuristic algorithm for the determination of the chromatic number of a finite graph is presented. This algorithm is based on Zykov's theorem for chromatic polynomials, and extensive empirical tests show that it is the best algorithm available. Christofides' algorithm for the determination of chromatic number is described and is used in the comparison tests.

Key words. chromatic number, coloring algorithm, Zykov's theorem

1. Introduction.

This paper presents an algorithm for the determination of the chromatic number of a graph. Empirical evidence indicates that this algorithm is the best available.

The chromatic number of a graph G, denoted $\chi(G)$, is the minimum number of colors required to color the graph such that no two adjacent vertices have the same color. Christofides [1] and Welsh and Powell [10] mention several practical applications of this property of a graph. For example, it can be used to solve an examination scheduling problem; in this application each examination is represented by a node of a graph and an edge joining two vertices indicates that because of a candidate's conflict, the corresponding examinations may not be held at the same time. The chromatic number of this graph is the minimum number of periods required to hold all examinations. In addition, the chromatic number problem belongs to the Cook–Karp class [5] which also includes the subgraph isomorphism, the Hamiltonian circuit and the maximal clique problems. Problems in the Cook–Karp class are polynomially equivalent, i.e., either each of them possesses a polynomially bounded algorithm or none of them does.

The best known algorithms for the determination of chromatic number are Christofides' algorithm [1] and an algorithm based on a corollary to Zykov's theorem for chromatic polynomials [13].[1] As mentioned by Hedetniemi [4], no extensive work has been done to compare these two methods.

Our algorithm is a modification and extension of the Zykov algorithm. We also present the results of extensive empirical tests [3] which were conducted to compare the storage and time requirements of our algorithm with that of Christofides.

2. Our algorithm.

2.1. The Zykov algorithm.

As mentioned in the Introduction, our algorithm is a modification and extension of the Zykov algorithm. Before describing Zykov's method, we introduce the concept of reduction of a graph.

* Received by the editors March 13, 1973. This research was supported by the National Research Council of Canada.

† Department of Computer Science, University of Toronto, Toronto, Canada.

[1] This algorithm will be referred to as Zykov's algorithm.

DEFINITION. Let $G(V, E)$ be a graph with nonadjacent vertices x and y. The *reduced graphs* of G are G'_{xy} and G''_{xy}, where

(i) $G'_{xy} = G'_{xy}(V', E')$; $V' = V$; $E' = E + (x, y)$ (i.e., G'_{xy} is obtained from G by adding the edge (x, y)).

(ii) $G''_{xy} = G''_{xy}(V'', E'')$; $V'' = V - y$;

$$(i,j) \in E'' \quad \text{iff} \quad (i,j) \in E \quad \text{for} \quad i,j \neq x,$$

$$(i,x) \in E'' \quad \text{iff} \quad (i,x) \in E \quad \text{or} \quad (i,y) \in E,$$

(i.e., G''_{xy} is obtained from G by coalescing (identifying) the vertices x and y).

For example, Fig. 1 shows a graph G together with a pair of reduced graphs.

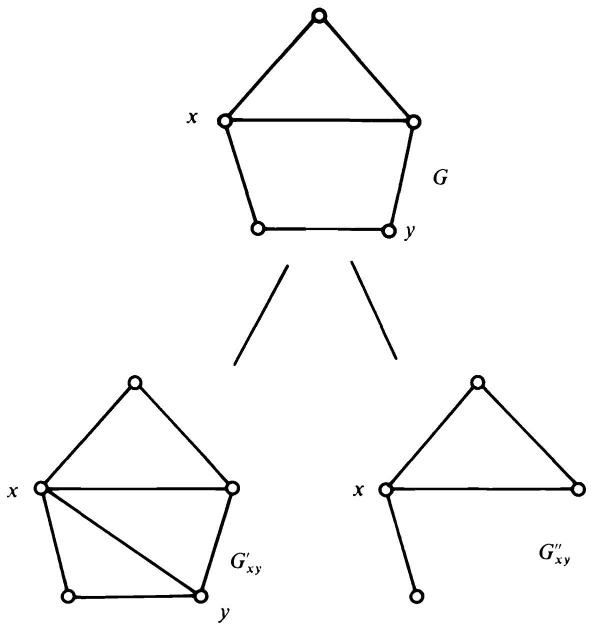


FIG. 1

Obviously any graph which is not complete can be reduced; a complete graph is irreducible, and the chromatic number of the irreducible graph K_α is α. If either G'_{xy} or G''_{xy} is reducible, the process can be continued until all graphs obtained are irreducible; at this stage G is said to be *completely reduced*, and

$$R(G) = \{G_1, G_2, \cdots, G_s\} \tag{1}$$

is the set of irreducible graphs thus obtained. Also let

$$R'(G) = \{G'_1, G'_2, \cdots, G'_{s'}\} \tag{2}$$

denote a set of graphs produced at some intermediate stage of the reduction process. For example,

$$R'(G) = \{G'_{xy}, G''_{xy}\}$$

after just one reduction step.

Theorem 1 below is the corollary to Zykov's theorem [13] on which the Zykov algorithm is based.

THEOREM 1. *If x and y are nonadjacent vertices in G, then* $\chi(G) = \min[\chi(G'_{xy}), \chi(G''_{xy})]$.

Proof. For a proof of this theorem, see [3].

Theorem 1 can be applied recursively, so when G is completely reduced to $R(G)$ (cf. (1)), we have

$$\chi(G) = \min[|G_1|, |G_2|, \cdots, |G_s|]. \tag{3}$$

Similarly when G is partially reduced to $R'(G)$ (cf. (2)), we have

$$\chi(G) = \min[\chi(G'_1), \chi(G'_2), \cdots, \chi(G'_{s'})]. \tag{4}$$

Figure 2 shows an application of the Zykov algorithm. We note from the figure that the algorithm has produced a binary tree (the Zykov tree) where each node of the tree is a graph.

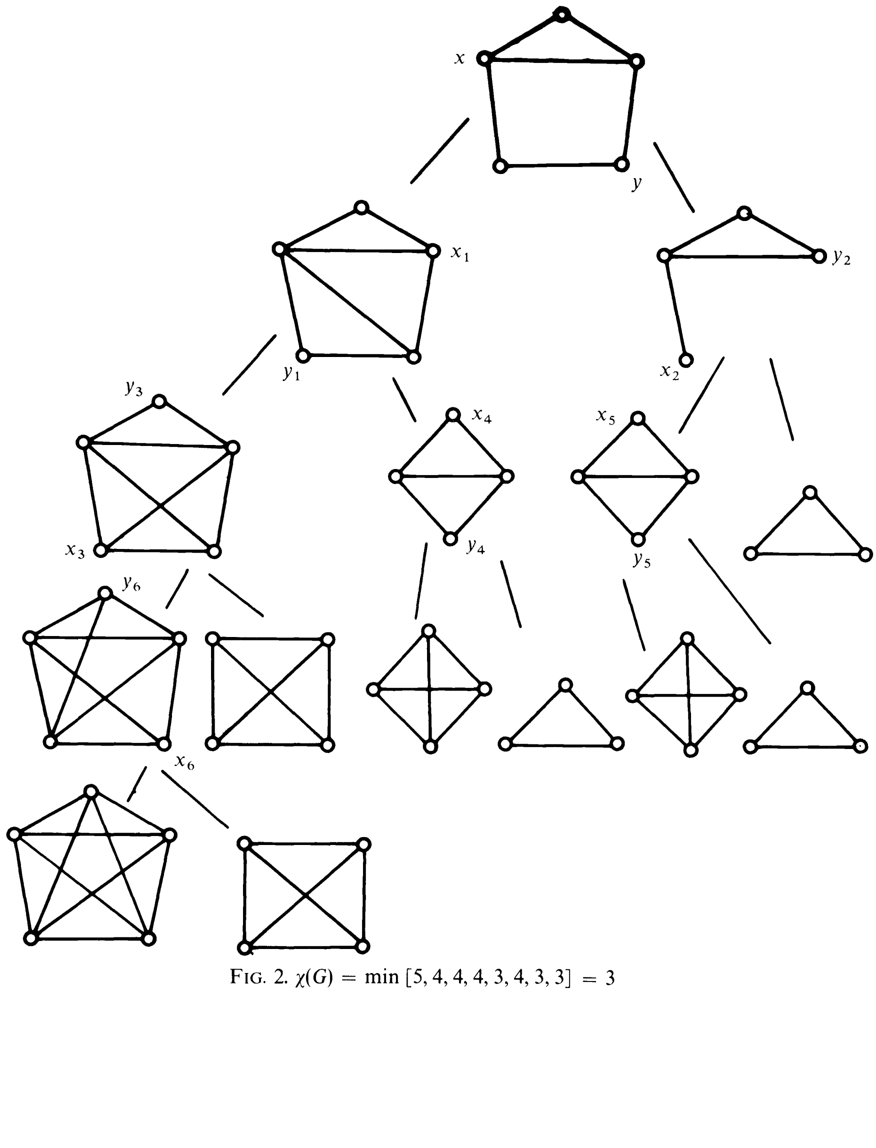


FIG. 2. $\chi(G) = \min[5, 4, 4, 4, 3, 4, 3, 3] = 3$

2.2. The improved algorithm. We will now show how the number of reduction steps required by the Zykov algorithm may be reduced by using a branch and bound approach [7]. In our case the branching is the reduction of G to G'_{xy} and G''_{xy}. We now show how the branching is bounded. Consider Fig. 3, an intermediate stage in the development of Fig. 2. At this stage we know from (4) that

$$\chi(G) = \min [\chi(A), \chi(B), \chi(C)].$$

Graph C is complete (K_3), so $\chi(C) = 3$ and 3 is thus an upper bound for $\chi(G)$; note that each of A and B contains a 3-clique, so that 3 is a lower bound for $\chi(A)$ and $\chi(B)$. Consequently, branching from A and B is unnecessary; we conclude that $\chi(G) = 3$ without finding $\chi(A)$ or $\chi(B)$. In general whenever α has been establisted as an upper bound for $\chi(G)$, then a graph encountered in the reduction of G which is known to contain an α-clique need not be reduced further. This pruning of the Zykov tree is the essence of bounding and is a major step in the development of our algorithm.

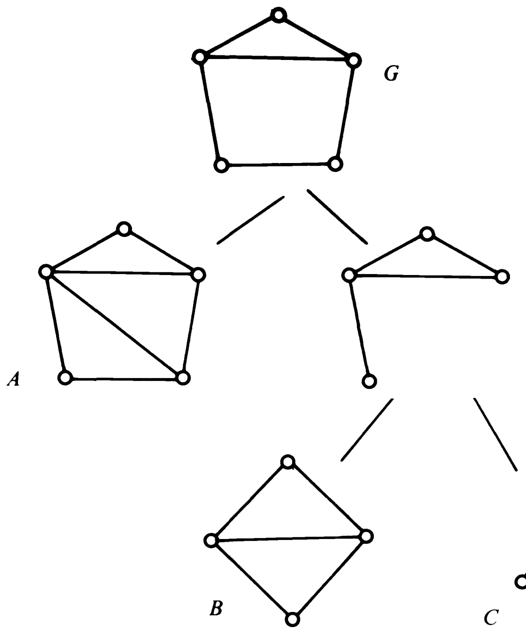


FIG. 3

The basic bounding strategy, once α (an upper bound for $\chi(G)$) has been obtained, is now obvious. It would, however, be poor practice to determine whether a reducible graph H contains an α-clique since the clique finding problem is in the Cook–Karp class mentioned in the Introduction. Instead the strategy is to find an α-cluster[2] in H, where an α-cluster is a set of α vertices which has a high density of edges; let H_α denote the subgraph of H determined by this α-cluster. Next H is reduced to H' and H'' such that H' is formed by adding an edge to H_α.

[2] The present version of the algorithm uses an $O(n^3)$ threshold-matrix method for finding clusters. This method is fully described in [3].

This process is continued with H' until the α-cluster becomes an α-clique, whereupon this branch is terminated. Graphs formed by coalescence (e.g. H'') are treated similarly.

If α is the exact value of $\chi(G)$, then the above procedure will always terminate successfully by building α-cliques. However, if $\alpha > \chi(G)$, then at some stage a graph introduced by coalescence will have only $\alpha - 1$ vertices, implying that $\chi(G) \leqq \alpha - 1$. Whenever this occurs, the value $\alpha - 1$ is substituted for α and the execution of the algorithm continues uninterrupted (i.e., a decrease of α does not require a repetition of any of the previous steps). It terminates when every branch of the Zykov tree introduced by reduction has been forced to contain an α-clique. Upon termination, the value of $\chi(G)$ will be exactly α.

We now present a concise recursive definition of the algorithm. Its most important feature is the procedure Reduce (which might be more aptly labeled "Reduce-if-necessary") which has two parameters: H, a graph, and N, the order of H. Note that the algorithm essentially performs a preorder traversal [6, p. 316] of a pruned Zykov tree such as the one in Fig. 3.

2.3. Recursive definition of our algorithm.

```
Main Program: ($\alpha$ is a global variable)
        Find Initial $\alpha$, an upper bound for $\chi(G)$;[3]
        Reduce $(G, n)$;
        Stop (now $\chi(G) = \alpha$).
  Procedure Reduce $(H, N)$;
          if $\alpha > N$ then $\alpha := N$;
          $\beta := \alpha$; ($\beta$ is a local variable)
          if $H$ is complete then GO TO EXIT;
          Find $H_\alpha$, a cluster on $\alpha$-vertices;
      A:  if $H_\alpha$ is an $\alpha$-clique then GO TO EXIT;
          choose nonadjacent vertices $x$ and $y$ in $H_\alpha$;
          Form $H'_{xy}$ and $H''_{xy}$ by reduction of $H$;
          $H := H'_{xy}$;
          Reduce $(H''_{xy}, N - 1)$;
          If $\beta = \alpha$ then GO TO A
                    else REDUCE $(H, N)$;
          (the else is necessary in case $\alpha$ was changed during the previous step)
  EXIT: END;
```

3. Christofides' algorithm. The Christofides' algorithm [1] depends strongly on the concept of a *Maximal Internally Stable Set* (or MISS) which is defined as follows:

A MISS of a graph is a set of vertices, no two of which are adjacent, and which is maximal with respect to this property. Thus a MISS in G is a clique in $\bar{G}$, the complement of G.

As shown in [8], for some graphs, the number of MISS grows exponentially with n. For example, a graph consisting of t disjoint triangles will have 3^t MISS.

[3] There are several methods for finding an initial upper bound for $\chi(G)$, for example $\alpha := n$. However in [3] and [11] methods for finding a good upper bound for $\chi(G)$ are described.

Having found all the MISS of the graph, Theorem 2 is used to find its chromatic number.

THEOREM 2 (Christofides [1]). *If a graph is r-chromatic, then it can be colored with r colors, coloring first with one color a* MISS *of G, say M_i, next coloring with another color a* MISS *of $G - M_i$ and so on until all the vertices are colored.*

Unfortunately, Theorem 2 does not determine which MISS should be colored at each step, so every MISS must be considered. A modification introduced by Furtado et al [2], while not eliminating the exponential nature of the algorithm, reduces the number of MISS investigated. Whenever a new color is introduced, they concentrate on coloring only one of the uncolored vertices, and consequently they consider only the MISS which contain this vertex. For obvious reasons, the vertex which appears in fewest MISS is selected. They also suggest that it is necessary to use the MISS-finding algorithm only once, since the MISS of G can be used to find the MISS of $G - M_i$.

4. Tests and results. Both our algorithm and Christofides' algorithm were programmed in ALGOL-W [12] and run on an IBM 370 model 165 computer. The most efficient available version of Christofides' algorithm [2] was used for the tests. In order to improve the speed of this algorithm, recursive programming and dynamic storage techniques were not employed.

4.1. Storage comparison. It has been shown [3] that any program based on Christofides' method may require an amount of storage which grows exponentially with n. By use of dynamic storage allocation the maximum storage required by our algorithm is bounded by $O(n^3)$.

4.2. Timing comparison. An extensive empirical comparison of the time requirements of the two algorithms was made [3]. For the test, the following three families of graphs were used.

(i) *Pseudo-random graphs $G(n, \rho)$.*

A pseudo-random graph $G(n, \rho)$ has n vertices and $(n/2)(n - 1) \cdot \rho$ edges. The location of the edges within the graph is determined by some pseudo-random mechanism.

(ii) *Modified Moon–Moser graphs $G(n)$.*

Modified Moon–Moser graphs are a family of graphs with $n = 0 \pmod 3$ vertices. Each graph is formed by constructing $n/3$ disjoint triangles and then adding $n/3 - 1$ additional edges to form a chain of triangles.

(iii) *Starred polygons $G(n, s)$* [9].

A starred polygon $G(n, s)$ is defined as follows:

$$V = \{v_1, v_2, \cdots, v_n\},$$
$$E = \{(v_i, v_k), k = (i + j) \text{ modulo } n, 1 \leqq i \leqq n, 1 \leqq j \leqq s\}.$$

Approximately two hundred test graphs were used. Our algorithm paid the overheads of recursion and dynamic storage allocation; these techniques were not employed in the tested version of Christofides' algorithm.

4.3. Results. Tables 1–3 are representative of the results obtained from the three families of graphs tested. From these tests we conclude:

TABLE 1

Observed tine and storage requirements on pseudo-random graphs

		Times[4]		Core used[5]	
ρ	n	C–G	Christofides	C–G	Christofides
4	10	3.0	3.3	22.0	29.2
	15	11.7	32.7	57.6	94.2
	20	66.3	608.7	268.8	653.0
	25	354.2	6324.2	720.0	4197.5
	30	972.3	—	830.8	—
	35	12785.0	—	1620.0	—
.8	10	1.3	2.7	26.4	23.4
	15	6.0	11.0	38.4	41.8
	20	16.7	27.3	75.6	61.6
	25	177.7	361.3	291.2	308.0
	30	845.3	4909.7	570.4	3323.2
	35	2006.0	—	950.4	—
	40	7825.0	—	1284.7	—

[4] 1 time unit = .01 sec.
[5] 1 unit of core = 1 word = 32 bits.

(i) For all graphs tested, the execution time of our algorithm was either significantly less than that of Christofides or at worst approximately equal to it.

(ii) With one exception, the asymptotic behavior (i.e., the slope of the log (time) vs. n plots) of our algorithm was superior to that of Christofides' algorithm. The exception was for pseudo-random graphs with a high density ($\rho = 0.9$) of edges; in this case the plots had approximately equal slopes.

(iii) For low values of n, the storage requirements of the algorithms are approximately equal, but for larger graphs, Christofides' method requires more storage. This reflects the result that the storage requirement of our algorithm is bounded by $O(n^3)$, while that of Christofides can grow exponentially.

TABLE 2

Observed time and storage requirements on modified Moon–Moser graphs

	Time		Storage	
n	C–G	Christofides	C–G	Christofides
3	0.0	0.0	8	12
6	1.7	1.7	14	25
9	3.3	8.3	20	60
12	5.0	56.7	26	166
15	8.3	705.0	32	533
18	13.3	8781.7	38	2014
21	18.3	—	44	—
—	—	—	—	—
60	278.3	—	122	—
63	320.0	—	128	—

TABLE 3
Observed time and storage requirements on starred polygons with $s = 1$ (i.e., cycles)

n	Time		Core Used	
	C-G	Christofides	C-G	Christofides
3	0.0	0.0	8	12
6	1.7	1.7	14	19
9	5.0	5.0	80	42
12	5.0	20.0	26	97
15	16.7	138.3	224	281
18	11.7	1235.0	38	778
21	48.3	11593.3	440	2862
24	23.3	—	50	—
27	101.7	—	728	—
—	—	—	—	—
54	185.0	—	110	—
60	243.0	—	122	—
63	1320.0	—	3968	—

REFERENCES

[1] N. CHRISTOFIDES, *An algorithm for the chromatic number of a graph*, Comput. J., 14 (1971), pp. 38–39.

[2] A. FURTADO, S. ROSCHKE, C. DOS SANTOS AND J. BAUER, *Finding the optimal colourings of a graph*, unpublished report, Departmento de Informatica Pontificia Universidade Catolica do Rio de Janeiro, Brazil.

[3] B. GRAHAM, *An algorithm to determine the chromatic number of a graph*, M.Sc. thesis, Tech. Rep., no. 47, Dept. of Computer Sci., Univ. of Toronto, Toronto, Canada, 1972.

[4] S. T. HEDETNIEMI, Rev. no. 22063, Comput. Rev., 12 (1971), pp. 446–447.

[5] 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. 85–104.

[6] D. E. KNUTH, *The Art of Computer Programming*, vol. 1, Addison-Wesley, Reading, Mass., 1968.

[7] E. L. LAWLER AND D. E. WOOD, *Branch and bound methods: A survey*, Operation Res., 14 (1966), pp. 699–719.

[8] J. W. MOON AND L. MOSER, *On cliques in graphs*, Israel J. Math., 3 (1965), pp. 23–28.

[9] J. TURNER, *Point symmetric graphs with a prime number of points*, J. Combinatorial Theory, 3 (1967), pp. 136–145.

[10] D. J. A. WELSH AND M. B. POWELL, *An upper bound for the chromatic number of a graph and its application to large scale time-tabling problems*, Comput. J., 10 (1967), pp. 85–86.

[11] M. R. WILLIAMS, *The colouring of very large graphs*, Combinatorial Structures and their Applications, R. Guy et al., eds., Gordon and Breach, New York, 1970, pp. 477–478.

[12] N. WIRTH AND C. A. R. HOARE, *A contribution to the development of ALGOL*, Comm. ACM, 9 (1966), pp. 413–431.

[13] A. A. ZYKOV, *On some properties of linear complexes*, Mat. Sb., 24 (1949), pp. 163–188; English transl., Amer. Math. Soc. Translation no. 79, 1952.