Accelerated Algorithms for Labeling and Relabeling of Trees, with Applications to Distribution Problems

V. SRINIVASAN

The University of Rochester, Rochester, New York

AND

G. L. THOMPSON

Carnegie-Mellon University, Pittsburgh, Pennsylvania

ABSTRACT. Adjacent extreme point problems involving a tree basis (e.g. the transportation problem) require the determination of cycles which are created when edges not belonging to the basis are added to the basis-tree. This paper offers an improvement over the predecessor-index method for finding such cycles and involves the use of a distance function defined on the nodes of the tree, in addition to the predecessor labels. It is shown that the relabeling associated with a basis change can be minimized by defining yet another function called the successor function. The algorithms for labeling and relabeling are then specialized for the specific case of transportation problems.

KEY WORDS AND PHRASES: graph theory, adjacent extreme point methods, transportation problems, network problems

CR CATEGORIES: 5.32, 5.41, 8.3

1. *Introduction*

When solving distribution problems (also known as transportation or Hitchcock problems) using so-called "primal" methods, a sizeable portion of the computational effort is involved in the determination of "cycles" or "stepping stone tours" [3–5, 9] which are created when a nonbasic cell is added to an existing basis. Earlier approaches for finding cycles used a tree search technique [5, 9]. A methodology, which we may call the "crossing out routine" that systematically eliminates rows and columns of the tableau not belonging to the cycle, was formulated independently

This paper was prepared as part of the activities of the Management Sciences Research Group, Carnegie-Mellon University, under contract N00014-67-A-0314-0007 NR 047-048 with the U. S. Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the U.S. Government.

Authors' addresses: V. Srinivasan, University of Rochester, Graduate School of Management, Rochester, NY 14627; G. L. Thompson, Carnegie-Mellon University, Graduate School of Industrial Administration, Schenley Park, Pittsburgh, PA 15213.

by Lourie [15], Gaver and Thompson [10], and Barr, Klingman, and Raike [1]. Glover and Kingman [12] recently gave a "predecessor-index method", which was largely anticipated by Glicksman, Johnson, and Eselson [11], and which appears to be more efficient than the previous approaches.

The present paper offers several extensions and improvements over the Glover–Klingman procedure. First, we extend their method to an arbitrary graph, and then show that by using a distance function in the graph as well as the predecessor function, the retracing portion of their cycle-finding routine can be eliminated. Second, we show that by using still another function defined on the graph, the successor function, it is possible to determine which is the smaller of the two parts into which a basis tree is split when an edge is removed. Third, we show that when the two parts of the tree are connected by another edge it is possible to recalculate the predecessor, distance, and successor functions for just one of the two parts. Clearly, computational effort is minimized by relabeling the smaller of the two parts. However, the most valuable use of the successor function appears to be in the applications of the duality operators for transportation problems that we have derived in a series of papers [17, 18].

In [19] the authors discuss computational experience with the ideas presented here for transportation problems. Specifically, we have made benefit-cost studies on some of the ideas presented in this paper as well as some of the standard methods for solving transportation problems. We have developed a code that uses the best of these methods. Actual computation times for solving a variety of kinds of problems are reported in [19].

There are a variety of other problems for which the methods developed in this paper should be useful. For instance, the shortest (or longest) spanning tree problem [6, 14] involves the checking for cycles. Adjacent extreme point problems—as for instance, certain concave minimization problems—involving a tree basis could also make use of it. Specific examples of the latter in the transportation framework are the time transportation problem [13], the traveling salesman problem [7], and the fixed charge transportation problem [16].

2. *The Predecessor and Distance Functions for Trees*

For a general reference on graph theory see Berge [2].

Let V be a set of n elements called *vertices* or *nodes* and let E be a set of (some of the) pairs (u, v) with $u, v \in V$. A pair (u, v) is called an *edge* between u and v, or also between v and u (no direction is implied). Then $G = (V, E)$ is called a *graph*. A *path* between u and v in G is a list

$$u = w_0, w_1, \cdots, w_R = v \tag{1}$$

where $(w_{j-1}, w_j) \in E$ for $j = 1, \cdots, R$, are distinct edges. The *length* of the path is R. A path is a *cycle* if $u = v$ in (1). A graph is *acyclic* if it has no cycles. A graph is *connected* if there is at least one path connecting each pair of distinct nodes. A *tree* [2] is a connected acyclic graph. Equivalently, a graph is a tree if and only if there is a unique path between each pair of distinct nodes [2]. A *rooted tree* is a tree $T = (V, E)$ together with a distance function $d(v)$ for $v \in V$, having the following properties:

(a) For some node r, $d(r)$ is a minimum ($= L$, say); node r is called the *root*.

(b) If r is a root, v any other node, and (1) is the unique path between r $(= u)$ and v then $d(v) = L + R$.

In a rooted tree there is a unique path from the root r to any other distinct node v; hence there is a unique predecessor $p(v) = u$ of v such that $d(u) = d(v) - 1$ and $(u, v) \in E$. We define $p(r) = \varnothing$ where r is the root and call p the *predecessor function* of the rooted tree.

Clearly any tree $T = (V, E)$ having n vertices can be made into a rooted tree in n different ways by choosing each vertex $r \in V$ in turn to be the root and using the following algorithm for determining the distance and predecessor functions:

ALGORITHM 1. For defining distance and predecessor functions on a tree $T = (V, E)$ given root r.

0. Set $d(r) = L$, where L is an integer, $p(r) = \varnothing$, $M = \{r\}$, $S = \varnothing$.
1. For each $u \in M$:
   (a) Find the set $D(u) = \{v \mid (v, u) \in E, v \in V - S\}$.
   (b) For each $v \in D(u)$ let $d(v) = d(u) + 1$ and $p(v) = u$.
2. Replace S by $S \cup M$; replace M by $\cup_{u \in M} D(u)$.
3. If $M = \varnothing$ stop. Otherwise go to step 1.

In thinking of this algorithm it helps to interpret the set M as being the subset of vertices just "labeled" and the set S as the vertices "labeled and scanned" (see Ford and Fulkerson [8]).

We assert that after completing Algorithm 1, all the nodes would have been labeled. For assume the contrary, that some node, say v, was not labeled. Consider the path in expression (1) from r to v. Clearly, if $v = w_R$ was not labeled then w_{R-1} was not labeled either, since otherwise we would have labeled v in step 1 of the algorithm. By repeating this argument we can show that the root r was not labeled, a contradiction, since we labeled it in step 0 above.

No node v will be labeled more than once, since that would imply G contains a cycle(s) formed from the two (or more) sequences of labelings that led to v.

A rooted tree may be thought of as a *directed graph*, i.e. a graph with directed edges called *arcs* with the direction of each edge being from the vertex of lower distance to the vertex of higher distance. Given any rooted tree with root r, there is a unique *backward path* between any other node v and r of length $R = d(v) - d(r)$, which can be found by means of the following algorithm:

ALGORITHM 2. For finding the backward path between $v \neq r$ and the root r in a rooted tree.

0. Let $R = d(v) - d(r)$, $j = R$, $v_R = v$.
1. If $j - 1 > 0$, replace j by $j - 1$ and go to step 2; otherwise go to step 3.
2. Find $v_j = p(v_{j+1})$. Go to step 1.
3. Let $v_0 = r$ and stop. The backward path is $v = v_R, v_{R-1}, \cdots, v_0 = r$.

To justify this algorithm we merely use property (b) of the distance function and the definition of the predecessor function in Algorithm 1.

If v_k is some node on the backward path between v and the root r, we will call the set $\{v = v_R, v_{R-1}, \cdots, v_k\}$ the backward path between v and v_k. Clearly this is a subset of the backward path between v and r.

As mentioned above, there is a unique path between each pair of distinct vertices

in a tree. Algorithm 2 solves the problem of finding this path when one of the vertices is the root. If u and v are two distinct vertices in $T = (V, E)$, then one way of finding the unique path between u and v is to make one of them, say u, the root, apply Algorithm 1 to find the resulting distance and predecessor functions, and then apply Algorithm 2 to find the backward path from v to u. However, if there already are distance and predecessor functions defined with root r (different from both u and v) on the tree, the following algorithm will efficiently find the path between u and v.

ALGORITHM 3. For finding the unique path between vertices u and v in a tree $T = (V, E)$ with root r.

0. If $d(u) = d(v) = R$, set $u_R = u$, $v_R = v$, $j = R$, $R' = R$, and go to step 2. Otherwise rename the points u and v, if necessary, so that $R' = d(u) > d(v) = R$, and go to step 1.
1. Use Algorithm 2 to find the unique backward path from u to the unique element u_R on the backward path from u to r such that $d(u_R) = d(v) = R$. Set $v_R = v$, $j = R$ and go to step 2.
2. If $v_j = u_j$ go to step 5. Otherwise go to step 3.
3. Find $u_{j-1} = p(u_j)$ and $v_{j-1} = p(v_j)$.
4. Replace j by $j - 1$ and go to step 2.
5. Stop. The unique path from u to v is

$$u = u_{R'}, \cdots, u_R, \cdots, u_{j+1}, u_j = v_j, v_{j+1}, \cdots, v_R = v. \tag{2}$$

The intuitive justification of this procedure is the following: There is a unique backward path from u to r and another from v to r; these two paths will certainly intersect at r, but perhaps also at nodes having positive distance. However, in the backward paths from u to r, u_j can equal v_j only if these two are at the *same* distance from r. Hence step 1 traces back the first part of the backward path from u to that element u_R which is at the same distance from r as v is. Then in step 2 we test to see whether the two elements on the backward paths from u_R and v to the root are equal, stopping as soon as equality is obtained.

The above algorithm typically requires fewer steps than the Glover–Klingman approach which required a complete backtracking of at least one of the paths back to the root r. In [19] we present some computational results which indicate that although each step of Algorithm 3 takes slightly longer than a step of the Glover–Klingman approach, the savings in steps afforded by Algorithm 3 pay off by reducing overall computation time.

3. *The Successor Function and Relabeling*

There are a number of problems that use adjacent extreme point methods (see Section 1), including the distribution problem to be discussed in Section 5, in which a tree $T = (V, E)$ is first constructed on the nodes in a set V. Then an edge $(u, v) \notin E$ is added to the tree. Let $E^* = E \cup \{(u, v)\}$. It is easy to see that this procedure creates a unique cycle in the graph (V, E^*) consisting of the unique path in T from u to v (found by applying Algorithm 3) together with the edge (u, v). Once the cycle has been determined, some edge (u_1, v_1) on the cycle is selected (using some appropriate criterion) to be removed from E^* so that the graph $T' = (V, E^* - \{(u_1, v_1)\})$ becomes a tree again. The problem now is how to relabel the new tree (i.e. how to recompute the functions p and d preserving as much of the old labeling as

possible, so that the new tree becomes a rooted tree (possibly with a different root.) We shall develop an efficient algorithm for doing this.

Consider the original tree $T = (V, E)$ with root r. If we remove the directed edge (u_1, v_1) (where $d(v_1) = d(u_1) + 1$) from E, then the tree splits into two subtrees T_1 and T_2 and the set of nodes V splits into two subsets V_1 and V_2 with u_1 and the root node r in V_1 and v_1 in V_2 (see Figures 1(a) and 1(b)). Since the addition of (u, v) creates a tree, u and v lie on different subtrees. Between these two nodes, we define v to be that node whose backward path (in the original tree) to the root contains both v_1 and u_1. In labeling the new tree we can preserve the labels in V_1 and change those in V_2 or vice versa. In order to do this using minimum effort we should select the set with the fewest vertices.

In order to determine which subset has the fewest vertices, we define the *successor function* on a rooted tree $T = (V, E)$ as follows: for each $v \in V$ let $s(v)$ be the *number* of successors of v in T, where u is a successor of v if $d(u) > d(v)$ and v lies on the unique path from u to the root r. The next algorithm determines the successor function.

ALGORITHM 4. For finding the successor function in a rooted tree $T = (V, E)$ with root r.

0. Let $s(v) = 0$ for all $v \in V$. Let R be the maximum distance that any node $v \in V$ is from the root r.
1. Let $M = \{v \mid d(v) = R, v \in V\}$.
2. If $M = \{r\}$ stop. Otherwise go to step 3.
3. For each $v \in M$ find $u = p(v)$ and replace $s(u)$ by $s(u) + s(v) + 1$.
4. Let $R = R - 1$. Go to step 1.

Notice that this algorithm begins by finding the values of s for the predecessors of nodes most distant from the root r, then labeling the predecessors of the nodes at one less distance, etc. On the last step $s(r)$ is calculated which must be $n - 1$, since all nodes except r are successors of r. Obviously the distance and predecessor func-

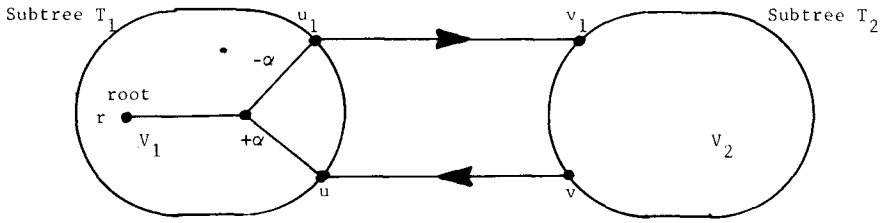

FIG. 1(a). Case I: $| V_1 | = \beta \geq \alpha = | V_2 |$

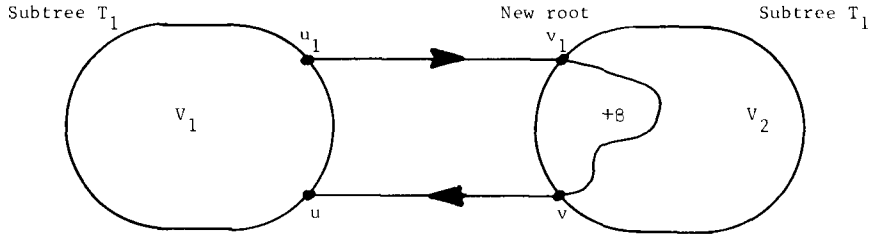

FIG. 1(b). Case II: $| V_1 | = \beta < \alpha = | V_2 |$

tions can be determined in one forward pass through the tree using Algorithm 1, and the successor function can then be determined in a backward pass using Algorithm 4.

Let us return to the two subsets V_1 and V_2 which obtain when the edge (u_1, v_1) is removed from the tree $T = (V, E)$. We denote by $|X|$ the number of elements in set X. Define

$$\alpha = s(v_1) + 1 = |V_2|, \tag{3}$$

$$\beta = n - \alpha = |V_1|. \tag{4}$$

There are two cases:

Case I. $\beta \geq \alpha$. Here $|V_1| \geq |V_2|$ so that we want to retain the root r and relabel V_2. As shown in Figure 1(a), this means we must subtract α from the successor function along the backward path from u_1 to the root r, and add α along the backward path from u to r. As shown in the figure, we need not necessarily go back all the way to r if these two paths join before r (cf. Algorithm 3).

Case II. $\beta < \alpha$. Here $|V_1| < |V_2|$ so that we want to make v_1 the new root of the tree, and relabel V_1. As shown in Figure 1(b) this requires that we add β to the successor function along the backward path from v to v_1.

From the above analysis we can now state the algorithms for relabeling either V_1 or V_2.

ALGORITHM 5(I). For relabeling V_2 holding the labels in V_1 fixed. This is Case I above; see Figure 1(a). The root will remain node r.

0. Find the unique path from u_1 to u using Algorithm 3; subtract α from the successor function values of u_1 and its predecessors that are not predecessors of u; add α to the successor function values of u and its predecessors that are not predecessors of u_1. Let $d(v) = d(u) + 1$, $p(v) = u$, $M = \{v\}$, and $S = \varnothing$.

1. Apply Algorithm 1, steps 1–3, but replacing V by V_2 in the definition of $D(u)$. At the conclusion of the algorithm the functions d and p will be updated on the entire tree.

2. Apply Algorithm 4, but replacing V by V_2 in steps 0 and 1, and replacing $\{r\}$ by $\{v\}$ in step 2. At the conclusion of the algorithm the function s will be updated on the entire tree.

ALGORITHM 5(II). For relabeling V_1 holding the labels in V_2 fixed. This is Case II above; see Figure 1(b). The new root will be v_1.

0. Find the unique backward path from v to v_1 using Algorithm 3 above; add β to the successor function values of all nodes on this path. Let $d(u) = d(v) + 1$, $p(u) = v$, $p(v_1) = \varnothing$, $M = \{u\}$, and $S = \varnothing$.

1. Apply Algorithm 1, steps 1–3, but replacing V by V_1 in the definition of $D(u)$. At the conclusion of the algorithm the functions d and p will be updated on the entire tree.

2. Apply Algorithm 4, but replacing V by V_1 in steps 0 and 1, and replacing $\{r\}$ by $\{u\}$ in step 2. At the conclusion of the algorithm the function s will be updated on the entire tree.

4. *An Example*

Figure 2 corresponds to a tree T with a set of 10 nodes $V = \{1, 2, 3, \cdots, 10\}$ and the set of edges $E = \{(1,4), (2,5), (3,5), (4,5), (4,8), (5,6), (6,7), (6,9), (7,10)\}$. It may be verified that T is a tree, i.e. it is connected and has no cycles.

We make this tree a rooted tree by arbitrarily choosing a root (say, node 6) and assigning an arbitrary value for its distance function (say, $L = 10$). On applying

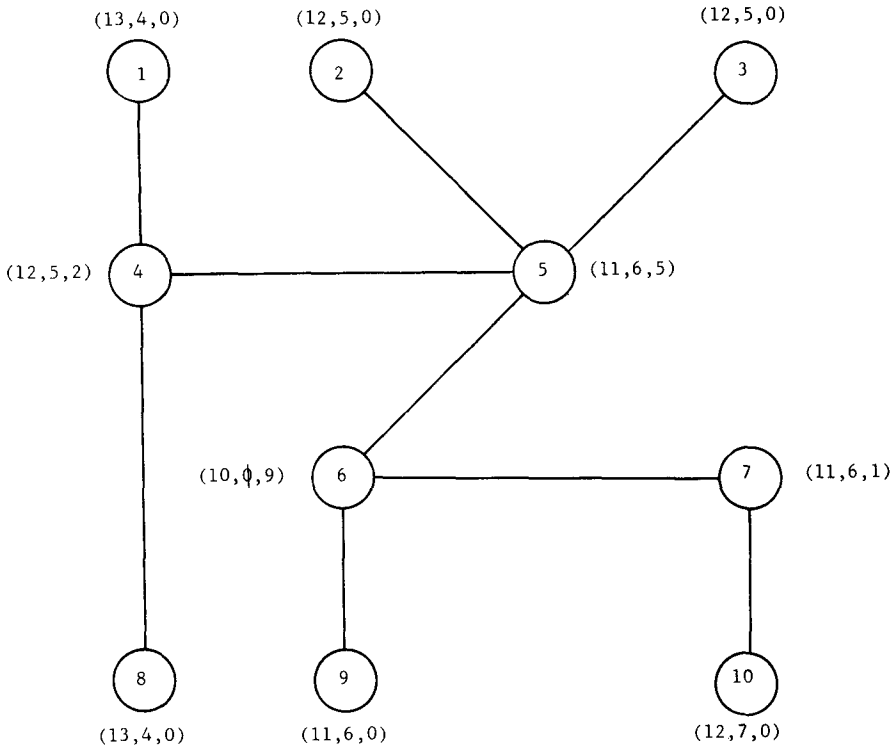

FIG. 2. Initial tree T and its labels. Node label = (distance, predecessor, successor)

Algorithm 1 first and then Algorithm 4, we obtain the distance, predecessor, and successor labels of Figure 2.

To find the backward path from node 1 to the root node 6 we apply Algorithm 2. The reader may verify that path $\{1, 4, 5, 6\}$ is the result.

We now use Algorithm 3 to find the unique path between nodes 1 and 2 to be $\{1, 4, 5, 2\}$, i.e. the set of edges $\{(1, 4), (4, 5), (5, 2)\}$ in that order. This path, together with the edge $(1, 2)$, constitutes the cycle formed when $(1, 2)$ is added to T. If we drop the edge $(4, 5)$ from T, it splits into the two subtrees (see Figure 3) with node $u_1 = 5$ and the root node $r = 6$ in T_1 and the node $v_1 = 4$ in T_2. Adding the edge $(1, 2)$ with $v = 1$ in T_2 and $u = 2$ in T_1 creates the tree T' of Figure 3.

Corresponding to the subtrees T_1 and T_2, $|V_2| = \alpha = s(v_1) + 1 = s(4) + 1 = 3$ and $|V_1| = \beta = n - \alpha = 7$. To obtain the labels for T' with minimum effort, we preserve the labels in T_1 and relabel T_2. This corresponds to Case I (Figure 1(a)) and on applying Algorithm 5(I) we obtain the labels of Figure 3. The reader may verify that direct application of Algorithms 1 and 4 to the tree T' (with root $r = 6$ and $L = 10$) would have yielded exactly the same labels.

Figure 4 is an instance of Case II (Figure 1(b)) where the subtree T_1 gets relabeled. Here the tree T'' is obtained by removing edge $(5, 6)$ from T and adding $(5, 7)$. $|V_2| = \alpha = s(v_1) + 1 = s(5) + 1 = 6$ and $|V_1| = \beta = n - \alpha = 4$. Thus relabeling T_1 is easier and the application of Algorithm 5(II) results in the labels of Figure 4. Node 5 becomes the new root.

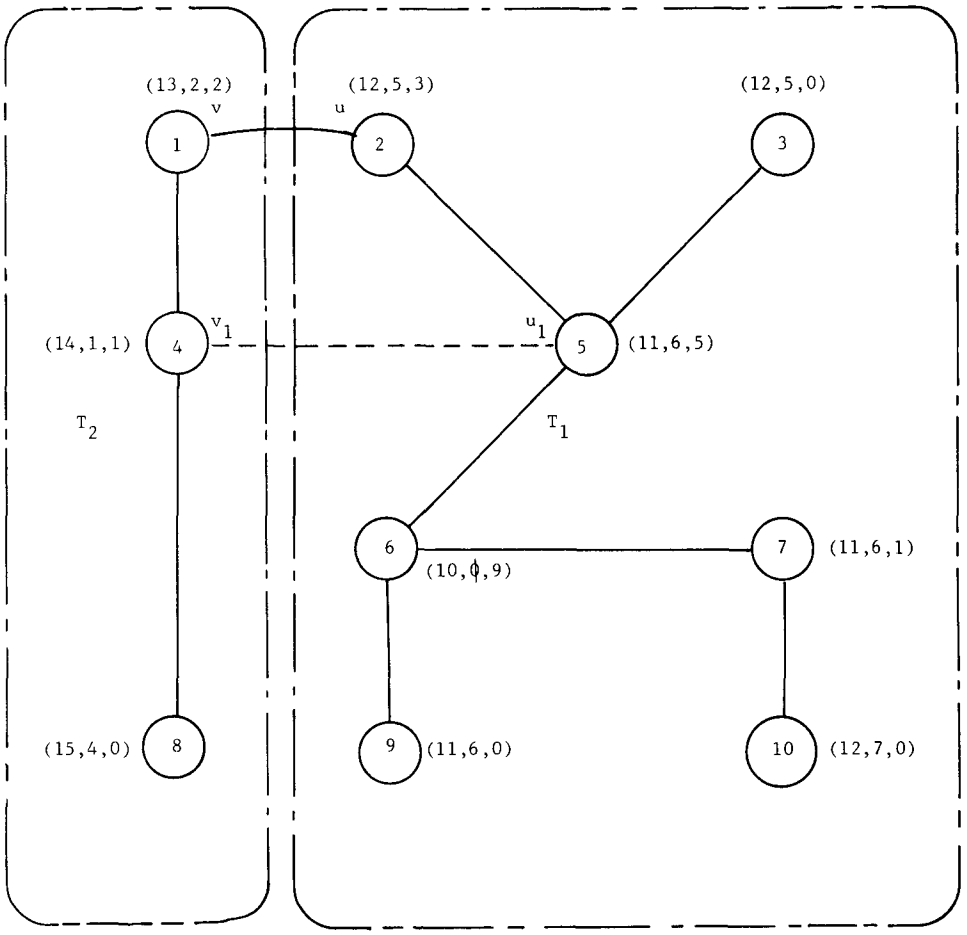

FIG. 3. Relabeled tree T' after adding edge (1, 2) and removing (4, 5) from the tree T of Figure 2

5. *Application to Distribution Problems*

We now apply the preceding theory to the specific case of the transportation problem. This problem may be stated as:

$$\text{Minimize} \sum_{\substack{i \in I \\ j \in J}} c_{ij} w_{ij} \tag{5}$$

subject to

$$\sum_{j \in J} w_{ij} = a_i \qquad \text{for } i \in I = \{1, 2, \cdots, m\}, \text{ the set of rows,} \tag{6}$$

$$\sum_{i \in I} w_{ij} = b_j \qquad \text{for } j \in J = \{1, 2, \cdots, n\}, \text{ the set of columns,} \tag{7}$$

$$w_{ij} \geq 0 \qquad \text{for } i \in I \text{ and } j \in J. \tag{8}$$

A *basic solution* to this problem consists of a *basis set* B of $m + n - 1$ basis *cells* which are pairs (i, j) with $i \in I$ and $j \in J$, and variables w_{ij} satisfying (6), (7), and

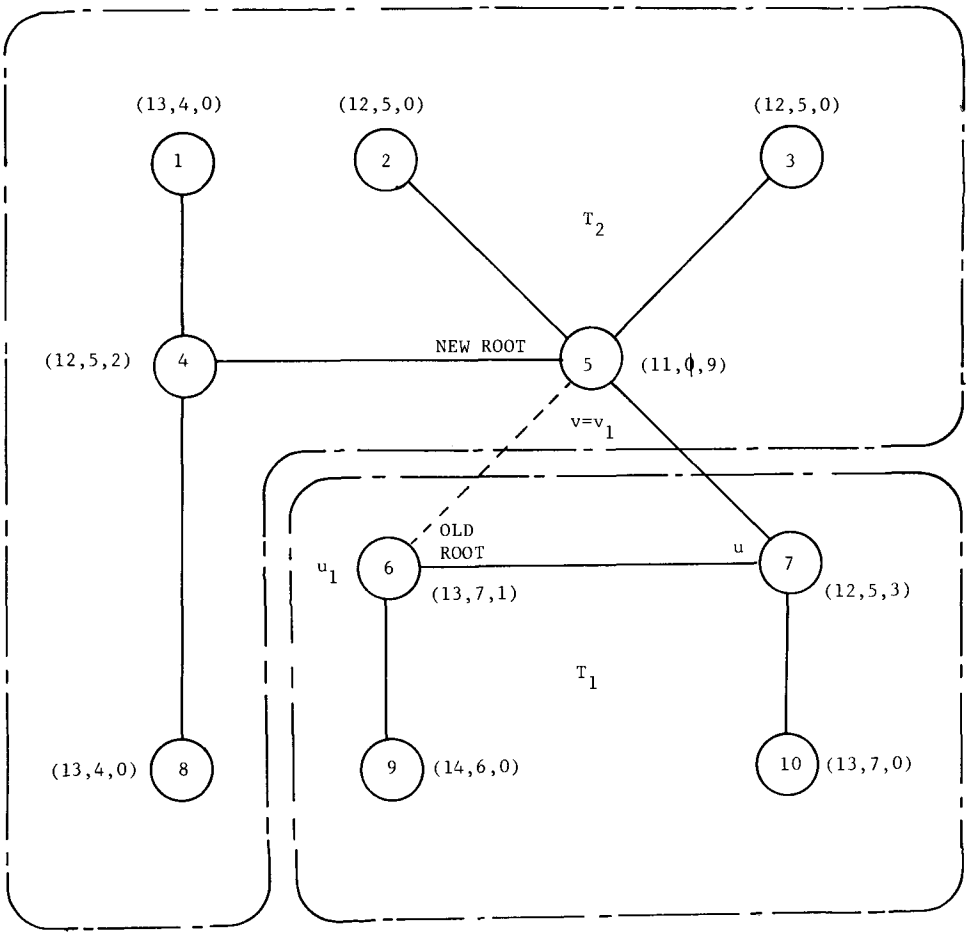

FIG. 4. Relabeled tree T'' after adding edge (5, 7) and removing (5, 6) from the tree T of Figure 3

(8) and also satisfying

$$w_{ij} = 0 \quad \text{if} \quad (i, j) \notin B. \tag{9}$$

If the transportation problem is nondegenerate (see [10]) then it can be shown that

$$w_{ij} > 0 \quad \text{if} \quad (i, j) \in B. \tag{10}$$

A degenerate problem can easily be replaced by an equivalent nondegenerate problem (see e.g. [10]).

The graph of the basic solution consists of the nodes $V = I \cup J$ and the edges $E = B$. It can be shown that the graph $T = (V, E) = (I \cup J, B)$ is a *tree*; see [3, 4, 10]. The graph T also has other properties. It has two classes of nodes I and J and every path in the graph alternately makes use of one, then the other, kind of node. This implies that if an edge is added to the graph T, the resulting cycle will have an even number of nodes. The so-called MODI or row-column sum method of transportation problems [3, 4] involves starting with an initial basic solution; checking to see if it is optimal; if not, adding an edge to the graph (i.e. adding a cell to B);

finding the unique cycle determined in the graph; selecting an edge on the cycle to be removed; etc. The choice of a cell to be added involves the solution to the dual problem. The dual problem to (5)–(8) is given by:

$$\text{Maximize} \sum_{i \in I} a_i x_i + \sum_{j \in J} b_j y_j \tag{11}$$

subject to

$$x_i + y_i \leq c_{ij} \qquad \text{for all } i \in I,\ j \in J, \tag{12}$$

where the dual variables x_i and y_i are unrestricted. Because of condition (10) we also have, using the complementary slackness theorem of linear programming, that

$$x_i + y_j = c_{ij} \quad \text{if} \quad (i, j) \in B. \tag{13}$$

We next explain how to specialize the algorithms of Sections 2 and 3 to the transportation problem. At the same time we will extend them to compute the dual variables x_i and y_j from (13) during the same pass through the graph that we make in computing the distance and predecessor functions. Given a row node i or a column node j we want to compute the distance d_i or e_j, the predecessor p_i or q_j, the dual variable x_i or y_j, and the successor function s_i or t_j. The following algorithm finds the initial values of each of these quantities.

ALGORITHM 6. For computing initial values for d_i, e_j, p_i, q_j, x_i, y_j, s_i, and t_j given basis B. The initial root node in T is arbitrarily chosen to be row 1.

0. Set $d_1 = x_1 = 0$; $p_1 = \varnothing$; $I' = I^* = \{1\}$; $J' = \varnothing$. Go to step 1.
1. If $I^* = \varnothing$ go to step 5. Otherwise set $J^* = \varnothing$ and go to step 2.
2. For each $i \in I^*$ do (a), (b), and (c) below, then go to step 3.
   (a) Find the set $D(i) = \{j \in J - J' \mid (i, j) \in B\}$.
   (b) For each $j \in D(i)$ let $e_j = d_i + 1$, $q_j = i$, $y_j = c_{ij} - x_i$.
   (c) Replace J' by $J' \cup D(i)$ and replace J^* by $J^* \cup D(i)$.
3. If $J^* = \varnothing$ go to step 5. Otherwise set $I^* = \varnothing$ and go to step 4.
4. For each $j \in J^*$ carry out (a), (b), and (c) below and then go to step 1.
   (a) Find the set $E(j) = \{i \in I - I' \mid (i, j) \in B\}$.
   (b) For each $i \in E(j)$ let $d_i = e_j + 1$, $p_i = j$, and $x_i = c_{ij} - y_j$.
   (c) Replace I' by $I' \cup E(j)$ and replace I^* by $I^* \cup E(j)$.
5. Let $s_i = 0$ for all $i \in I$, and $t_j = 0$ for all $j \in J$. Let R be the largest of all the d_i ($i \in I$) and e_j ($j \in J$). If there exists an i such that $d_i = R$ go to step 6. Otherwise go to step 8.
6. Carry out steps (a) and (b) below and then go to step 7.
   (a) Find the set $I_R = \{i \in I \mid d_i = R\}$.
   (b) For each $i \in I_R$ let $j = p_i$ and replace t_j by $t_j + s_i + 1$.
7. Replace R by $R - 1$. If $R = 0$ stop. Otherwise go to step 8.
8. Carry out steps (a) and (b) below and then go to step 9.
   (a) Find the set $J_R = \{j \in J \mid e_j = R\}$.
   (b) For each $j \in J_R$ let $i = q_j$ and replace s_i by $s_i + t_j + 1$.
9. Replace R by $R - 1$. If $R = 0$ stop. Otherwise go to step 6.

This algorithm is a specialization of Algorithms 1 and 4 for graphs. Note that steps 0–4 constitute a "forward pass" through the tree, while steps 5–9 are a "backward pass." The reader should keep in mind that all the nodes (rows or columns) at the same distance from the root node (row 1) will be either all rows or all columns. For

this reason we label just rows in steps 4 and 8 and just columns in steps 2 and 6. Examination of the example in the next section will help in understanding the algorithm.

At the conclusion of this algorithm we will have determined dual variables x_i and y_j satisfying (13). The transportation method proceeds by checking to see if (12) is satisfied. If this inequality is false for at least one cell, then one such cell is selected to enter the basis. The problem now is to determine the cycle which the new cell creates when added to basis B.

Let (u, v) be the cell chosen to enter the basis B which is the set of edges in the tree $T = (I \cup J, B)$. Then we can find the cycle in the graph $T^* = (I \cup J, B \cup \{(u, v)\})$ by Algorithm 7 below, which is a specialization of Algorithm 3.

ALGORITHM 7. For finding the unique path from row u to column v in T and the unique cycle in the graph T^*.

0. Let $S = d_u$; $R = e_v$. If $S > R$, set $v_R = v$ and go to step 1; otherwise set $u_S = u$ and go to step 2.

1. Find the unique backward path from u,

$$u = u_S , u_{S-1} , u_{S-2} , \cdots , u_R ,$$

where $u_{S-1} = p_{uS}$, $u_{S-2} = v_{uS-1}$, etc. (cf. Algorithm 2 with obvious notational changes). Let $j = R$ and go to step 3.

2. Find the unique backward path from v,

$$v = v_R , v_{R-1} , v_{R-2} , \cdots , v_S ,$$

where $v_{R-1} = q_{vR}$, $v_{R-2} = p_{vR-1}$, etc., (cf. Algorithm 2 with obvious notational changes). If $j = S$ go to step 5.

3. If $u_j = v_j$ go to step 7. Otherwise go to step 4.

4. Find $u_{j-1} = q_{uj}$ and $v_{j-1} = q_{vj}$. Replace j by $j - 1$.

5. If $u_j = v_j$ go to step 7. Otherwise go to step 6.

6. Find $u_{j-1} = p_{uj}$ and $v_{j-1} = p_{vj}$. Replace j by $j - 1$ and go to step 3.

7. Stop. The unique path between u and v in T is

$$u_S , u_{S-1} , \cdots , u_j = v_j , v_{j+1} , \cdots , v_R$$

and the unique cycle in T^* determined when (u, v) is added to B is

$$(u_S , u_{S-1}), (u_{S-2} , u_{S-1}), (u_{S-2} , u_{S-3}), \cdots , (v_{R-1} , v_R)$$

together with the cell $(u_S , v_R) = (u, v)$.

Once the cycle has been found, a cell (u_1 , v_1) is selected to leave the set $B \cup \{(u, v)\}$. As in Section 3, the set $B - \{(u_1 , v_1)\}$ consists of two subsets V_1 and V_2 (see Figures 1(a) and 1(b).) In the case of the transportation problem, it can be shown that if u and u_1 are row indices and v and v_1 are column indices, then u and u_1 will either both belong to V_1 or both belong to V_2 , with v and v_1 both belonging to the opposite set. This follows from the facts that: (a) every cycle has an even number of elements; (b) the cell coming in is a "getter" cell while the cell going out is a "giver" cell; (c) the cells on the cycle are alternately marked "givers" and "getters" [10]. If $e_{v_1} > d_{u_1}$, then u and u_1 are in V_1 with v and v_1 in V_2 . In this case formulas (3) and (4) become $\alpha = t_{v_1} + 1$ and $\beta = m + n - \alpha$. However, if $e_{v_1} < d_{u_1}$ then u and u_1 are in V_2 with v and v_1 in V_1 . For this case $\alpha = s_{u_1} + 1$ and $\beta = m + n - \alpha$.

In the algorithms that follow we assume the former case. The extensions to the latter case should be obvious. We can thus relabel whichever subtree is smaller, and we have the following variants of Algorithm 5(I) and 5(II).

ALGORITHM 8(I). For relabeling V_2 holding the labels in V_1 fixed. This corresponds to Figure 1(a). We assume u_1 and u are V_1. The root does not change.

0. Find the unique path from u_1 to u using Algorithm 7; subtract α from the successor function values of u_1 and its predecessors that are not predecessors of u; add α to the successor function values of u and its predecessors that are not predecessors of u_1. Let $e_v = d_u + 1$, $q_v = u$, $y_v = c_{uv} - x_u$, $J' = J^* = \{v\}$, $I' = \varnothing$.
1. Apply Algorithm 6, steps 1–4, starting with step 3, but replacing I and J by $I \cap V_2$ and $J \cap V_2$ in the definition of $D(i)$ and $E(j)$. At the conclusion of the algorithm the predecessor and distance functions will be updated on the entire tree.
2. Apply Algorithm 6, steps 5–9, but replacing I and J by $I \cap V_2$ and $J \cap V_2$ in steps 5, 6, and 8, and stopping in step 7 when $R = e_v$. At the conclusion of the algorithm the successor function will be updated on the entire tree.

ALGORITHM 8(II). For relabeling V_1 holding the labels in V_2 fixed, corresponding to Figure 1(b). We assume u and u_1 are in V_1. The new root is v_1.

0. Find the unique backward path from v to v_1 using Algorithm 7; add β to the successor function values of all nodes on this path. Let $d_u = e_v + 1$, $p_u = v$, $q_{v_1} = \varnothing$, $x_u = c_{uv} - y_v$, $I' = I^* = \{u\}$, $J' = \varnothing$.
1. Apply Algorithm 6, steps 1–4, starting with step 1, but replacing I and J by $I \cap V_1$ and $J \cap V_1$ in the definition of $D(i)$ and $E(j)$. At the conclusion of the algorithm the predecessor and distance functions will be updated on the entire tree.
2. Apply Algorithm 6, steps 5–9, but replacing I and J by $I \cap V_1$ and $J \cap V_1$ in steps 5, 6, and 8, and stopping in step 7 when $R = d_u$. At the conclusion of the algorithm the successor function will be updated on the entire tree.

6. *A Transportation Problem Example*

We consider the example given in [12] with the cost entry c_{24} changed from 6 to 1. Figure 5(a) shows the cost entries, the rim conditions (W stands for warehouses and M for markets), and a basic feasible solution to this problem. A circle in the cell (i, j) denotes that (i, j) is in basis B and the corresponding w_{ij} is shown over the circle. Figure 5(d) shows the tree T corresponding to this basis. We apply Algorithm 6 to obtain the labels shown in Figures 5(b), (c), and (d).

The basic solution of Figure 5(a) is not dual feasible, since it violates the condition (12) for $(i, j) = (2, 4)$. The solution can be improved by bringing $(2, 4)$ into the basis. To do this, we apply Algorithm 7 and find the unique path between $W2$ and $M4$ as ($W2$, $M2$, $W3$, $M4$). The application of Glover–Klingman algorithm [12] would have resulted in the two backward paths $\{(W2, M3, W1)\}$, $\{M4, W3, M2, W2, M3, W1\}$, and after eliminating the common backtracking part $\{(W2, M3, W1)\}$ it obtains the above result. The unique cycle created by the addition of $(2, 4)$ is $\{(2,2),(3,2),(3,4),(2,4)\}$. (See Figures 5(a) and 6(d).) On this cycle the "giver cells" [10] are $(2, 2)$ and $(3, 4)$, so that the minimum giver cell $(2, 2)$ should leave the basis ($w_{22} = 2 < w_{34} = 14$). When the cell $(2, 2)$ is removed from the tree T, we get the two subtrees T_1 and T_2 shown in Figure 6(d). When $(2, 4)$ is added to these subgraphs, we obtain the tree $T' = T - \{(2, 2)\} + \{(2, 4)\}$.

Corresponding to the two subtrees T_1 and T_2, $\alpha = |V_2| = t_{v_1} + 1 = t_2 + 1 = 3$

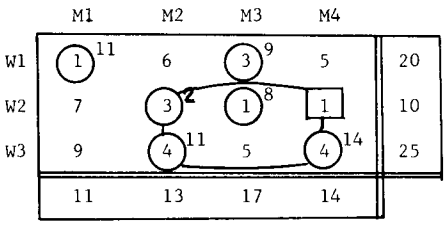

FIG. 5(a). Transportation tableau

	d_i	p_i	s_i	x_i
W1	0	ϕ	6	0
W2	2	3	3	-2
W3	4	2	1	-1

FIG. 5(b). Labels for warehouses

	M1	M2	M3	M4
e_j	1	3	1	5
q_j	1	2	1	3
t_j	0	2	4	0
y_j	1	5	3	5

FIG. 5(c). Labels for markets

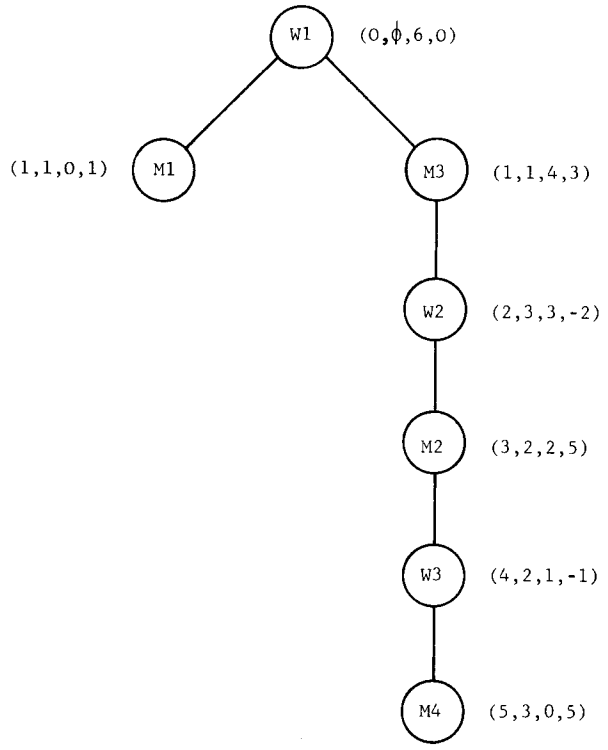

FIG. 5(d). Tree T corresponding to Fig. 5(a). Node label = (distance, predecessor, successor, dual variable)

and $\beta = m + n - \alpha = 3 + 4 - 3 = 4$. Thus relabeling V_2 is easier and we apply algorithm 8(I) to obtain the labels of Figures 6(b), (c), and (d). The new basis and the corresponding w_{ij} are shown in Fig. 6(a). The reader may verify that the labels of Figures 6(b) and 6(c) will be obtained if Algorithm 6 is applied to the tree T' of Figure 6(d). The reader may also verify that (12) is satisfied by the basis of Figure 6(a) so that this solution is, in fact, optimal.

	M1	M2	M3	M4	
W1	(1)[11]	6	(3)[9]	5	20
W2	7	3	(1)[8]	(1)[2]	10
W3	9	(4)[13]	5	(4)[12]	25
	11	13	17	14	

FIG. 6(a) Modified tableau

	d_i	p_i	s_i	x_i
W1	0	ϕ	6	0
W2	2	3	3	-2
W3	4	4	1	-1

FIG. 6(b) Modified warehouse labels

	M1	M2	M3	M4
e_j	1	5	1	3
v_j	1	3	1	2
t_j	0	0	4	2
y_j	1	3	3	3

FIG. 6(c) Modified market labels

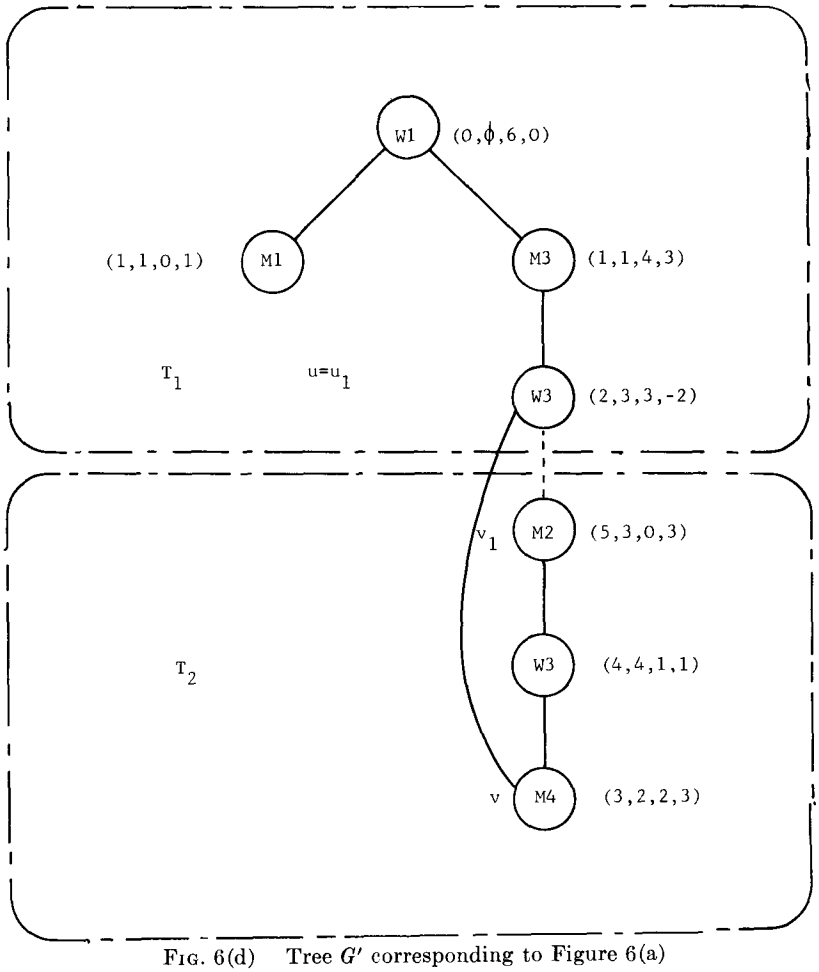

FIG. 6(d) Tree G' corresponding to Figure 6(a)

REFERENCES

1. BARR, R. S., KLINGMAN, D., AND RAIKE, W. M. Computational simplifications through topological structure in network and distribution models. Series in Appl. Math. for Management, Pub. No. AMM-13, Grad. School of Business, U. of Texas, Austin, Texas, 1968.
2. BERGE, C. *The Theory of Graphs and Its Applications.* Wiley, New York, 1962.
3. CHARNES, A., AND COOPER, W. W. *Management Models and Industrial Applications of Linear Programming, Vols. I and II.* Wiley, New York, 1961.
4. DANTZIG, G. B. *Linear Programming and Extensions.* Princeton U. Press, Princeton, N. J., 1963.
5. DENNIS, J. B. A high-speed computer technique for the transportation problem. *J. ACM 5*, 2 (Apr. 1958), 132–153.
6. DIJKSTRA, E. W. A note on two problems in connection with graphs. *Numer. Math. 1* (1959), 269–271.
7. EASTMAN, W. L. Linear programming with pattern constraints. Ph.D. Diss., Harvard U., Cambridge, Mass. 1958.
8. FORD, L. R., AND FULKERSON, D. R. *Flows in Networks.* Princeton U. Press, Princeton, N. J., 1962.
9. FORTRAN Transportation Code, Contributed Program Library, 360D-15.2.010. IBM, New York, 1968.
10. GAVER, D. P., AND THOMPSON, G. L. *Mathematical Models—Programming and Probability.* Brooks/Cole Pub. Co., Belmont, Calif. (to be published, 1973).
11. GLICKSMAN, S., JOHNSON, L., AND ESELSON, L. Coding the transportation problem. *Naval Res. Logist. Quart. 7* (1960), 169–183.
12. GLOVER, F., AND KLINGMAN, D. Locating stepping-stone paths in distribution problems via the predecessor index method. *Transportation Sci. 4* (1970), 220–225.
13. HAMMER, P. L. Time-minimizing transportation problems. *Naval Res. Logist Quart. 16* (1969), 345–357.
14. KRUSKAL, J. On the shortest spanning subtree of a graph and the travelling salesman problem. *Proc. Amer. Math. Soc. 7* (1956), 48–50.
15. LOURIE, J. R. Topology and computation of the generalized transportation problem. *Management Sci. 11* (1964), 177–187.
16. MURTY, K. G. Solving the fixed charge problem by ranking the extreme points. *Operations Res. 16* (1968), 268–279.
17. SRINIVASAN V., AND THOMPSON, G. L. An operator theory of parametric programming for the transportation problem, I. *Naval Res. Logist. Quart. 19* (to appear).
18. SRINIVASAN, V., AND THOMPSON, G. L. An operator theory of parametric programming for the transportation problem, II. *Naval Res. Logist. Quart. 19* (to appear).
19. SRINIVASAN, V., AND THOMPSON, G. L. Benefit-cost analysis of coding techniques for the primal transportation algorithm. Management Sci. Res. Rep. No. 229, Grad. School of Ind. Admin., Carnegie-Mellon U., Pittsburgh, Pa.; to appear in *J. ACM.*

RECEIVED FEBRUARY 1971; REVISED NOVEMBER 1971