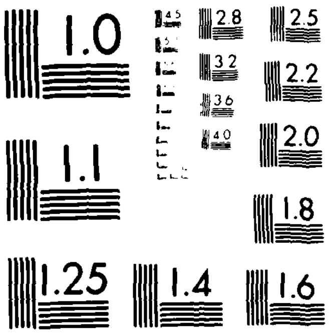
AD-A153 986 A PRIMAL SIMPLEX APPROACH TO PURE PROCESSING NETWORKS (U) TEXAS UNIV AT AUSTIN CENTER FOR CYBERNETIC STUDIES C H CHEN ET AL. FEB 85 CCS-496 N00014-81-C-0236 1/1

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1963 A

2

Research Report CCS 496

A PRIMAL SIMPLEX APPROACH TO PURE PROCESSING NETWORKS

by

C.-H. Chen
M. Engquist

CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712

85

Research Report CCS 496

A PRIMAL SIMPLEX APPROACH TO PURE PROCESSING NETWORKS

by

C.-H. Chen
M. Engquist

February 1985

This paper was partly supported by ONR Contract N00014-81-C-0236, ONR Contract N00014-82-K-0295, USARI Contract MDA 903-83-K-0312 and a grant from the University Research Institute at The University of Texas. Reproduction in whole or in part is permitted for any purpose of the United States Government.

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
College of Business Administration 5.202
The University of Texas at Austin
Austin, Texas 78712-1177
(512) 471-1821

A PRIMAL SIMPLEX APPROACH TO PURE PROCESSING NETWORKS

BY

C.-H. Chen, M. Engquist

Abstract

Pure processing network problems are minimum cost flow problems in which the flow entering or leaving a node may be constrained to do so in given proportions. In this paper, new theoretical results concerning pure processing networks are developed, and based on these results, two new primal simplex variants are presented. One of these variants has been implemented and tested against a general purpose linear programming code. A large class of problems is identified for which the specialized code is an order of magnitude faster than the general purpose code.

Key Words:

Networks
Processing Networks
Linear Programming

✓

PER LETTER

A-1

1. INTRODUCTION

Network problems which allow proportional flow restrictions on the arcs entering or leaving some nodes are called processing network problems. Processing network structure arises in a number of application areas including energy modelling [20], assembly models [28], and management of working capital [7]. A processing network model used in manpower planning is described in [4,5]. Processing network terminology was introduced by Koene [19], and a survey of applications is contained in [19].

In this paper we consider pure processing network problems. For such problems, conservation of flow holds both at nodes and along arcs. A given linear programming (LP) problem can be transformed to pure processing network form in three steps which are roughly described as: create a new row in the LP tableau which is the negative of the sum of the original rows, scale the columns of the new tableau so that the positive components of each column sum to one, and split each non-network column into two new columns containing its positive and negative parts, respectively. These two new columns are forced to correspond to equal variables by the addition of a new constraint. Further details of the transformation are found in [19]. Since the transformation allows any LP problem to be formulated as a pure processing network, it seems unlikely that efficient solution techniques based entirely on the graph traversal methods used to solve pure network problems are possible. Rather, the appropriate strategy is to handle the pure network structure using graphical methods while the non-network part of the problem is handled separately. Basis partitioning in the primal simplex algorithm is the technique we use for this purpose. The proportional flow restrictions can be formulated as non-network variables (side variables) or as non-network constraints (side constraints). We find the side variable formulation preferable since it leads to a working basis of lower dimension. For one of the simplex variants we discuss, the dimension of the working basis equals the number of basic side variables which, for some problems, can be quite small. On the other hand, when the number of basic side variables increases to some point as yet unknown, our partitioning approach will break even with general purpose LP solution methods.

Pure network problems have been solved 150-200 times faster than the general purpose LP code, APEX III, using a specialized primal simplex code [13]. This has motivated the study of pure network problems with side constraints or side variables. The papers [11,12] show that problems with a single side constraint can be solved 25-50 times faster than APEX III. For problems having multiple side constraints or side variables, computational results have been quite limited, and those which are available are less encouraging. Primal partitioning methods for pure network problems with multiple side constraints have been investigated by Klingman and Russell [18] and Chen and Saigal [6]. A recent computational study by Ali et al. [1] showed that on multicommodity network flow problems, a primal partitioning code ran about three times faster than MINOS [25] on problems with up to 31 binding linking constraints.

A primal partitioning algorithm for pure network problems with both side constraints and side variables was developed by Glover and Klingman [14]. In [14], an implementation of the side constraint case is discussed along with preliminary computational results. McBride [22] has extended the methods of [14] to the solution of generalized network problems with both side variables and constraints. An implementation of this algorithm was tested against MINOS with the result that the specialized code was about five times faster overall. Four of the problems solved in McBride's study were generalized processing network problems. Although such problems are quite similar to those studied in the present paper, they allow gains or losses of flow along arcs and are somewhat more difficult to solve. On these generalized processing network problems, the specialized code ran about two and one-half times faster than MINOS.

Koene [19] proposed special primal algorithms for both pure and generalized processing networks based on a side variable formulation. These algorithms have not been implemented. Engquist and Chen [8,9] developed a primal partitioning approach which used some of the ideas of [19] applied to a side constraint formulation, and they presented preliminary computational comparison of a specialized code with MINOS. It should be noted that the algorithm of [8,9] differs considerably from the primal simplex variants introduced in the present paper. McBride [21] developed a hybrid primal partitioning technique for generalized processing networks which starts the problem solution using the side constraint formulation and switches to the side variable formulation one column at a time as the corresponding side constraints attain feasibility. Two test problems were solved for which the hybrid approach proved beneficial compared to either the side variable or side constraint methods.

2. PROBLEM FORMULATION

In this paper, we utilize the processing network structure shown in Figure 1. Node v is known as a splitting node, and associated with each arc (v,w) of Figure 1 is a value α_{vw}, $0 < \alpha_{vw} < 1$. The flow on arc (v,w) is required to equal α_{vw} times the flow entering node v. For pure processing networks, conservation of flow holds at node v, and thus

$$\sum_{z=1}^{t} \alpha_{vw(z)} = 1 \tag{2.1}$$

must hold. It is convenient to associate $\alpha_{vv}=1$ with splitting node v.

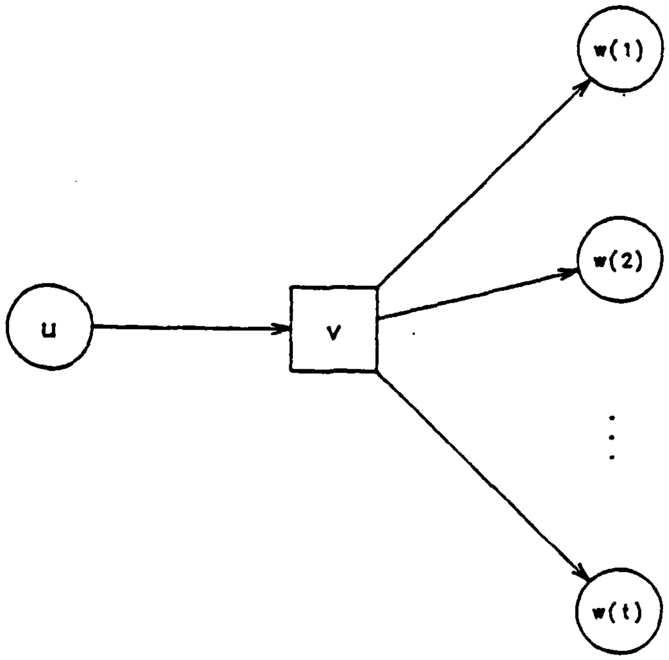


Figure 1. A Splitting Node

We note that splitting nodes are represented graphically as squares while other nodes are shown as circles. The arcs of Figure 1 leaving node v are called processing arcs, while the arc entering node v is termed an allocation arc. The nodes w(1), w(2), ..., w(t) along with the splitting node are called processing nodes.

In the LP formulation of the pure processing network problem (2.3) - (2.6), the processing arcs are represented by a single column, called a processing column, in the constraint matrix. This column has the form

$$\begin{array}{ll} \alpha_{vv} & \text{in row } v \\ -\alpha_{vw(z)} & \text{in row } w(z),\ z = 1, 2, \ldots, t \\ 0 & \text{elsewhere.} \end{array} \qquad (2.2)$$

A network arc is represented by a network column which contains only two nonzero values, a 1 and a -1. The row containing the 1 corresponds to the tail node of the arc, while the row containing the -1 corresponds to the head node. The column corresponding to an allocation arc is called an allocation column.

In [19], the definition of pure processing networks includes the structure formed when the direction of the arcs in Figure 1 is reversed. By complementing flows with respect to their capacities and adjusting supply values appropriately, this structure can be reduced to the one shown in Figure 1. Thus, there is no loss of generality in restricting attention to the structure of Figure 1.

The pure processing network problem (problem PPN) is :

$$\text{minimize} \quad c_N x_N + c_P x_P \tag{2.3}$$

$$\text{subject to:} \quad A_N x_N + A_P x_P = b \tag{2.4}$$

$$0 \le x_N \le h_N \tag{2.5}$$

$$0 \le x_P \le h_P \tag{2.6}$$

The mxn matrix A_N is the node arc incidence matrix for a pure network N, while the mxp matrix A_P contains the processing columns. Vector b contains the supply values, while c_N and c_P contain unit costs for the vectors of decision variables x_N and x_P. The capacities are the components of the simple upper bound vectors h_N and h_P. In Figure 1, if arc (u,v) corresponds to column r of A_N and the corresponding processing column (2.2) is column s of A_P, then it is assumed that the capacities $[h_N]_r$ and $[h_P]_s$ are equal.

We assume, without loss of generality, that a slack arc and artificial arcs with Big-M costs have been introduced into the network N so that it is connected and the matrix A_N has rank m. We also assume that each row of $[A_N, A_P]$ contains at most one non-zero component from the columns of A_P. The latter assumption is for notational convenience only and does not restrict the application of our methods.

An example PPN problem is shown in Figure 2. This problem is uncapacitated, and the arc labels in rectangles are costs. Costs on the processing arcs are zero. Arc labels in semicircles are used to indicate the α_{vw} values of (2.2).

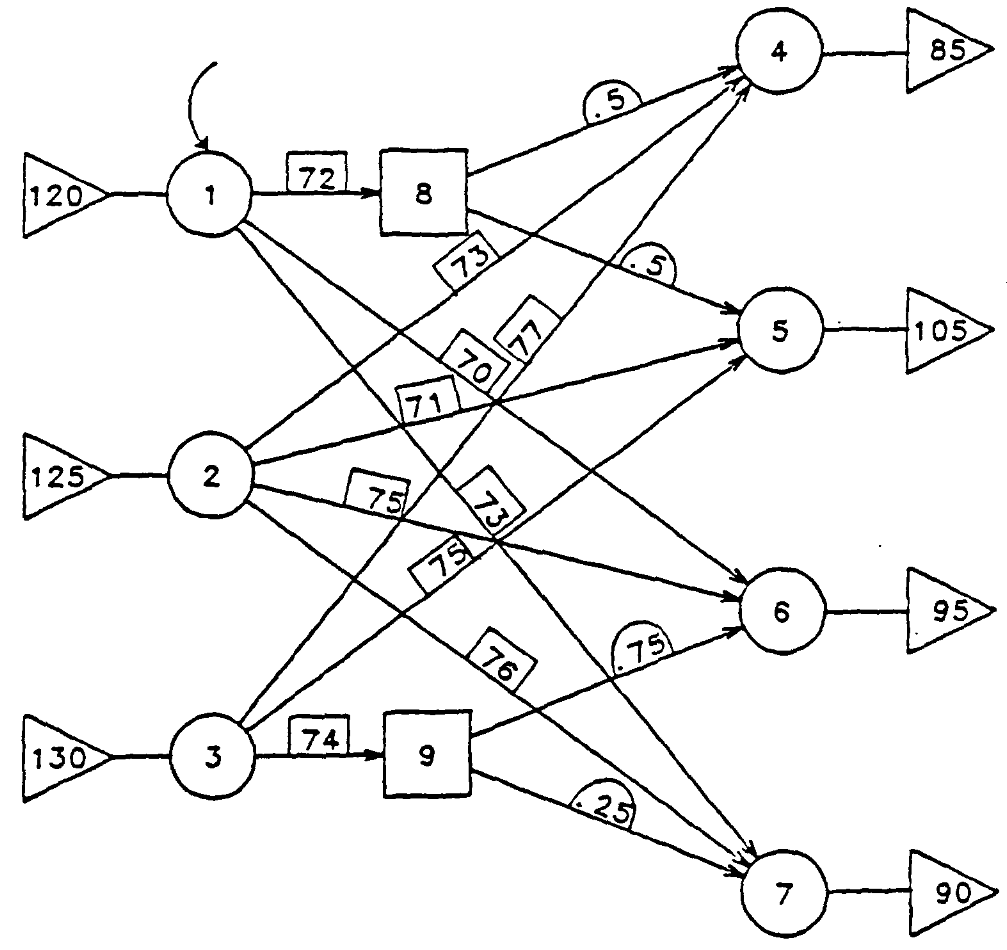


Figure 2. A PPN Example

3. BASIS STRUCTURE

Until further notice, we let B denote a basis matrix for PPN. The column corresponding to the slack arc, which contains a single nonzero value, must be contained in B. Otherwise, the rows of B sum to zero. Let B' be the matrix obtained from B by omitting the slack arc column. Let r be the number of columns of B from A_P. The next theorem and its proof are taken from [19].

Theorem 1. The partial network of N corresponding to columns of B' from A_N consists of r+1 trees if and only if B contains r columns from A_P.

Proof. The number of arcs in a tree is one less than the number of nodes. This fact implies that m-(r+1) network columns from B' correspond to r+1 trees and vice versa, and the result follows.

We let

$$B = [B_1, B_2] \tag{3.1}$$

where B_1 contains the columns of B from A_N and B_2 contains r columns of B from A_P. The r+1 trees whose existence is guaranteed by Theorem 1 are termed basis trees. The slack arc is incident to one of these basis trees and the basis tree with the slack arc adjoined is called the basis quasi-tree. For the remaining r basis trees, root nodes are chosen arbitrarily. The resulting r rooted trees are called the rooted basis trees. In Figure 3, the basis quasi-tree and the rooted basis trees of an optimal basis for the problem of Figure 2 are shown. Optimal flows are shown on the arcs. We note that one of the rooted basis trees is a single node in this example.

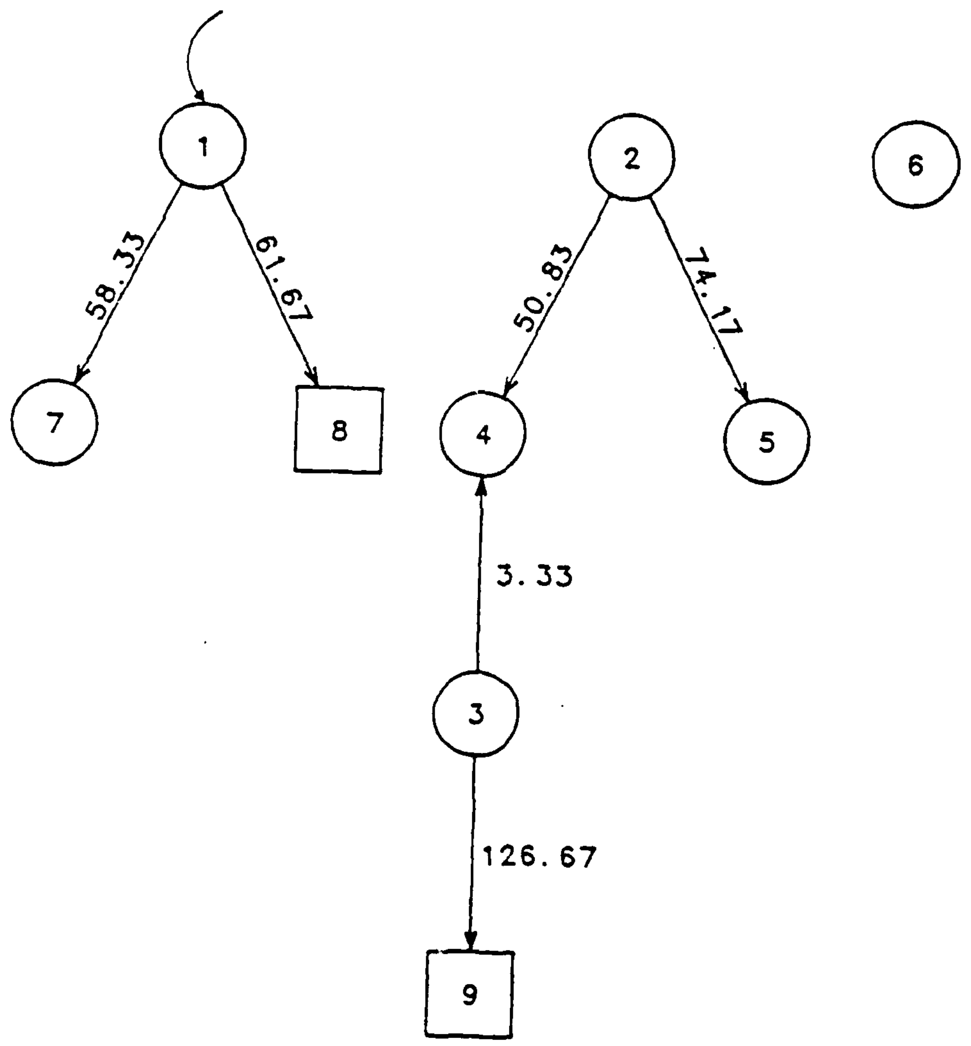


Figure 3. Optimal Basis Trees

We assume that the rows of $[A_N, A_P]$ are arranged so that the last r rows correspond to the roots of the rooted basis trees. The first m-r rows of B_1 form a matrix T while the remaining r rows form matrix D. Likewise, the first m-r rows of B_2 form a matrix C while the last r rows form matrix F. The resulting partition is

$$B = \begin{bmatrix} T & C \\ D & F \end{bmatrix} \tag{3.2}$$

We note that T corresponds graphically to a collection of quasi-trees.

In order to state the next theorem, we temporarily drop the assumption that B is a basis for PPN. Instead, we assume that B contains m columns of $[A_N, A_P]$ including the slack arc column and that B is partitioned as in (3.1) and (3.2) so that T corresponds to a collection of quasi-trees.

We define the matrix Q by

$$Q = F - DT^{-1}C. \tag{3.3}$$

Theorem 2. B is a basis matrix for PPN if and only if Q is nonsingular.

Proof. It follows that

$$B = \begin{bmatrix} T & 0 \\ D & Q \end{bmatrix} \begin{bmatrix} I & T^{-1}C \\ 0 & I \end{bmatrix} \tag{3.4}$$

where I denotes identity matrices of appropriately chosen dimension. From (3.4) we obtain

$$\det B = \det T \det Q. \tag{3.5}$$

Since T corresponds to a collection of quasi-trees, $\det T \neq 0$. The result follows directly.

Theorems 1 and 2 suggest the following approach to obtaining a basis matrix B for PPN: m-r columns of A_N which form r trees and one quasi-tree are chosen along with r columns of A_P such that the resulting matrix Q is nonsingular.

For a basis B, the matrix Q of (3.3) is called the working basis. Before proceeding with an investigation of working basis structure, we make a preliminary definition. Suppose that the rooted basis trees are numbered from 1 to r. The ith tree indicator function is defined to be 1 on the nodes of rooted basis tree i and 0 on the nodes of all other basis trees. We further suppose that the numbering of the rooted basis trees is done so that tree i corresponds via its root node to the ith row D_i of matrix D.

Lemma 1. The ith tree indicator function restricted to non-root nodes equals $-\dot{D}_i T^{-1}$.

Proof. The matrix T represents a number of quasi-trees. Suppose that one of these quasi-trees, say τ, is selected. In the first case, the slack arc of τ and the corresponding component of D_i is -1. Forward substitution yields the desired result. In the second case, the slack arc is oppositely directed, the corresponding component of D_i is 1, and forward substitution again yields the result. If the nodes of τ are contained in rooted basis tree j, $j \neq i$, or in the basis quasi-tree, D_i assigns a zero value to the slack arc of τ and the result follows.

We now consider basis matrix B partitioned as in (3.1), and we suppose that column j of B_2 is the processing column with splitting node v. P_{ij} is defined to be the set of processing nodes in rooted basis tree i which correspond to nonzero values in column j of B_2.

Theorem 3. For a basis B, the elements q_{ij} of the working basis Q satisfy

$$q_{ij} = \sum_{w \varepsilon P_{ij}} \alpha_{vw} \tag{3.6}$$

where the sum in (3.6) is defined to be zero in case P_{ij} is empty.

Proof. If the root node of rooted basis tree i is a processing node corresponding to column j, then an α value is contributed to the sum in (3.6) by the first term on the right hand side of (3.3). That the contribution of other processing nodes in rooted basis tree i is given by (3.6) follows directly from (3.3) and Lemma 1.

Remark. It follows from (3.6) that the matrix Q is independent of the choice of root nodes for the rooted basis trees.

When column a of $[A_N, A_P]$ enters basis B in the primal simplex algorithm, it is necessary to compute the updated column y where y is the solution to

$$By = a. \tag{3.7}$$

We next indicate how the structure of B can be used in calculating y. The vector y is partitioned as

$$y = [y_1, y_2] \tag{3.8}$$

so as to be compatible with (3.1). Then (3.7) can be rewritten as

$$B_1 y_1 + B_2 y_2 = a. \tag{3.9}$$

We suppose that a is partitioned as $\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$ where a_1 contains the first m-r rows of a in (3.9) and a_2 contains the remaining r rows. Using (3.2), it follows from (3.9) that

$$y_1 = T^{-1} a_1 - T^{-1} C y_2 \tag{3.10}$$

and

$$F y_2 = a_2 - D y_1 . \tag{3.11}$$

Upon combining (3.10) and (3.11), we have

$$Q y_2 = a_2 - D T^{-1} a_1 . \tag{3.12}$$

If we interpret the vector a as a supply vector, it is possible to give a flow interpretation to (3.12). Using Theorem 3 and Lemma 1, the ith row of (3.12) may be interpreted as equating the flow into the ith rooted basis tree due to y_2 with the supply to that rooted basis tree. Similarly, the first term on the right hand side of (3.10) can be interpreted as a flow due to supply vector a_1, while the second term is a flow generated by induced supplies at processing nodes which result from y_2 values.

We partition the vector π of dual variables relative to B as $\pi=[\pi_1, \pi_2]$ where π_1 corresponds to the first m-r rows of B and π_2 corresponds to the last r rows of B. The vector c_B of basic costs is partitioned as $[c_1, c_2]$ in order to be compatible with (3.1). The dual variables satisfy the equation

$$\pi B = c_B. \tag{3.13}$$

When the partition of B from (3.2) is used in (3.13), the following equations result.

$$\pi_1 = c_1T^{-1} - \pi_2DT^{-1} \tag{3.14}$$

$$\pi_2F = c_2 - \pi_1C. \tag{3.15}$$

By combining (3.14) and (3.15) we obtain

$$\pi_2Q = c_2 - c_1T^{-1}C. \tag{3.16}$$

From Lemma 1, it follows that the second term on the right hand side of (3.14) can be interpreted as assigning the ith component of π_2 to all nodes, except the root, in rooted basis tree i. For nodes in the basis quasi-tree, a value of zero is assigned. For any node in rooted basis tree i, other than the root node, the first term on the right hand side of (3.14) represents the cost of sending a unit flow from that node to the root node, while the second term is the cost of adding a unit of supply to this root. The jth components in (3.16) can be interpreted as equating two ways of computing the cost of increasing supply at the root nodes of rooted basis trees due to a unit increase in the variable of the jth basic processing column. On the left hand side of (3.16) this cost is computed by adding the costs incurred at root nodes which are caused by increases in supply to the corresponding trees. On the right hand side of the equation the cost of the jth basic processing column is added to costs incurred by sending proportional flows from the processing nodes of the jth basic processing column to the roots of rooted basis trees.

Theorem 4. If the entering column a is from A_N and the arc e corresponding to a has both end nodes in basis tree τ, then

(i) $y_2 = 0$.

(ii) The leaving column corresponds to an arc on the cycle formed in τ by e.

(iii) The working basis Q is unchanged by this pivot.

Proof: (i) We verify that $a_2 - DT^{-1}a_1$ is zero in (3.12). If τ is the basis quasi-tree, then $a_2 = 0$ and DT^{-1} is zero on the nodes of τ by Lemma 1. If τ is rooted basis tree i, the proof breaks down into two cases both of which use Lemma 1. First, suppose that one end node of e is the root node. Then the ith components of a_2 and $-DT^{-1}a_1$ have absolute value one and differ in sign. Second, suppose that both end nodes of e are non-root nodes. Then $a_2 = 0$ and the ith component of $-DT^{-1}a_1$ is the sum of terms +1 and -1.

(ii) By (i), $y_2 = 0$ and (3.10) reduces to $y_1 = T^{-1}a_1$ which is essentially the formula for the updated entering column in a pure network.

(iii) This follows directly from Theorem 3 since the sets P_{ij} in (3.6) are unchanged.

Remark. If A_N in (2.4) is changed to represent a generalized network rather than a pure network, the analogue of Theorem 4 fails. This indicates that more is involved in going from pure to generalized processing networks than one might suspect at first glance.

4. PRIMAL SIMPLEX VARIANTS

In this section we present two primal simplex variants -- Algorithms 1 and 2 -- for PPN. Algorithm 1 appears to be more general in that no special assumptions concerning PPN basis structure are made. For Algorithm 2, we assume that all processing columns remain in the basis during all iterations. We show, however, that this assumption is not restrictive, and we indicate how this algorithm allows standard linear programming methods to be used in updating the working basis.

Before stating these algorithms, we outline the situations which arise in updating the basis trees and the working basis Q during the basis exchange step of the primal simplex algorithm. Before the basis exchange is executed, we assume that τ_0 is the basis quasi-tree and τ_i, $i = 1, 2, \ldots, r$

are the rooted basis trees. Those basis trees which have been changed during the exchange step will be designated by means of an asterisk. If a change to τ_i, $i \neq 0$, results in a change to one of the sets P_{ij} in (3.6), then row i of Q must be updated. The main cases to be considered are as follows:

(i) The entering column is a processing column and the leaving column is a network column (arc). If the leaving arc is in basis tree i, then row i of Q will be updated (unless $i = 0$) and an additional row and column will be adjoined to Q.

(ii) The entering column is a network column (arc) while the leaving column is processing column k of B_2. If the entering arc is incident to τ_i and τ_j, then these two trees are joined to form τ_i^*. Row i of Q will be updated (unless $i = 0$) and row j and column k of Q will be deleted.

(iii) Both the entering column and the leaving column are processing columns. The column of Q corresponding to the entering column is replaced by one corresponding to the leaving column.

(iv) Both the entering and leaving columns are network columns (arcs). As in case (ii), τ_i and τ_j are joined via the entering arc to form τ_i^*. If the leaving arc is contained in τ_k, then τ_k splits into two trees upon its removal. One of these trees becomes τ_j^* and the other becomes τ_k^*. If i, j, and k are nonzero and distinct, then three rows of Q will be updated. Otherwise, special cases occur in which at most two rows of Q are updated.

One of the special cases mentioned in (iv) occurs when both the entering and leaving arcs are in the same basis tree as covered by Theorem 4. For this case, no updating of rows of Q is necessary.

We proceed now with the first primal simplex variant.

Algorithm 1

0. Obtain an initial basis. Set up the initial basis tree and working basis. Compute initial dual variables and basic solution.
1. Price nonbasic columns until an entering column is found. If no entering column exists, stop--the current basic solution is optimal.
2. Compute y_1 and y_2 using (3.10) and (3.12).
3. Perform the ratio test. Update basic solution values.
4. Update basis trees and working basis (basis exchange step--see preceding discussion).
5. Update π_1 and π_2 using (3.14) and (3.16). Go to Step 1.

Although algorithms which allow for working basis updates as general as those required in Step 4 of Algorithm 1 have been implemented [14, 22], such procedures remain relatively untested compared to standard LP basis updating procedures. For this reason, we have specialized Algorithm 1 so that the only case which occurs in Step 4 is the one in which both the entering and leaving columns are network columns (case iv). This means that at most three rows of Q will be updated during each basis exchange step. We will show in Theorem 5 that these rows can be replaced one at a time using the usual LP column replacement technique applied to Q^T, the transpose of Q.

The fundamental idea underlying Algorithm 2 is that any basis matrix for PPN must contain at least one member of any given pair consisting of an allocation column and its corresponding processing column (see Figure 1). Thus, it is possible to assume that the initial basis for PPN contains all the columns of A_p. Also, the flow on an allocation arc and the value of the corresponding processing variable are always the same. This allows us to modify the ratio test so that whenever a processing column would be the leaving column, we choose the corresponding allocation column to leave instead. Note that the allocation column must be basic in this situation, since otherwise the pivot would lead to the impossible situation in which both the allocation column and the processing column are nonbasic. In Algorithm 2 then, the only columns to enter or leave the basis are network columns, and these are the only columns which need to be priced or considered in the ratio test.

In order to describe the basis exchange step of Algorithm 2, it will be useful to visualize the basis trees as hanging downward from their roots. The node incident to the slack arc in the basis quasi-tree is taken as the root there. Thus, if two basis trees τ_i and τ_j are joined by an entering arc, the resulting tree τ_i^* will retain the root of τ_i while τ_j will hang below τ_i in τ_i^*. Also, when a leaving arc is deleted from a basis tree τ_k, an upper tree τ_{k1} which contains the root of τ_k and a lower tree τ_{k2} are formed.

We introduce the vector λ to represent certain quantities which may be thought of as pseudo node potentials.

$$\lambda = c_1 T^{-1} \tag{4.1}$$

It will be useful to extend λ by defining $\lambda_j = 0$ for root nodes j of rooted basis trees. By an abuse of notation, this extension will also be denoted as λ.

Algorithm 2

0. Obtain an initial basis which includes all processing columns. Set up the initial basis trees and working basis. Compute initial dual variables and basic solution.

1. Price nonbasic arcs until an entering arc e is found. If no entering arc exists, stop--the current basic solution is optimal.

2. If both end nodes of e are not in a common basis tree τ, go to Step 3. Otherwise, restrict the ratio test and flow update to the arcs on the cycle formed in τ by e. Update λ on the tree hanging below e after the leaving arc is removed. Go to Step 6.

3. Compute y_1 and y_2 using (3.10) and (3.12).

4. Perform the ratio test. Update basic solution values.

5. Update basis trees and working basis. (The details follow for this step when e is incident to τ_i and τ_j, the leaving arc is in τ_k, and i, j, k are nonzero and distinct. The remaining cases involve updating at most two rows of Q and the details are omitted.) First, τ_j hangs below τ_i via arc e to form τ_i^* and λ is updated on τ_j. Row i of Q is updated to form Q*. Next, the leaving arc is removed from τ_k to form an upper tree τ_{k1} and a lower tree τ_{k2}. The lower tree becomes τ_j^* and λ is updated on τ_j^*. Row j of Q* is updated to form Q**. Finally, τ_{k1} becomes τ_k^* and row k of Q** is updated to form Q***.

6. Update π_2 using (3.16). Compute π_1 using

$$\pi_1 = \lambda - \pi_2 DT^{-1} \tag{4.2}$$

where λ has been previously updated. Go to Step 1.

Implementation of Algorithm 2 is discussed in Section 5. There, a method for determining the initial basis and a pricing strategy are presented along with other techniques.

In step 2 of Algorithm 2, pivots in which both end nodes of the entering arc are in a common basis tree are treated separately. This step is justified by Theorem 4, and its implementation is discussed in the next section. Pivots of this type will be referred to as pure network pivots, while all other pivots will be called processing network pivots.

Updating of λ on a tree which is rehung is done just like the updating of node potentials in the pure network case. This means that a certain constant must be added to λ values on this tree.

The use of LP updating procedures in the basis exchange step of Algorithm 2 is justified by the following theorem. Again, only the case in which i, j, and k are nonzero and distinct is covered, although the remaining cases can be treated similarly.

Theorem 5. In Step 5 of Algorithm 2, matrices Q* and Q** are nonsingular.

Proof. It follows from Theorem 3 and the way that Q* is defined that row i of Q* is the sum of rows i and j of Q. Q is nonsingular by Theorem 2 and this implies the nonsingularity of Q*. Since Q*** is the working basis after the pivot, it is also nonsingular by Theorem 2. It follows from Theorem 3 and the way that Q** and Q*** are defined that row k of Q*** is the difference of row k and row j of Q**. Thus, Q** is also nonsingular.

We note that the values of π_1 in (4.2) can be computed as they are needed in Step 1 of Algorithm 2. If the tail node u of arc e is in τ_i, the head node v of arc e is in τ_j, and the cost of e is denoted as c_e, then the reduced cost of e is

$$c_e - \lambda_u + \lambda_v - [\pi_2]_i + [\pi_2]_j \tag{4.3}$$

Since it is possible for i or j in (4.3) to be 0, we define $[\pi_2]_0$ to be 0.

5. IMPLEMENTATION

An implementation of Algorithm 2, which we call PROCNET, was coded in FORTRAN. Problem data storage in PROCNET is accomplished by means of arrays for arc costs, capacities, and head nodes. Also, arrays containing the nonzero values in processing columns and the positions of these values are used. The costs of processing columns, components of c_p in (2.3), are assumed to be zero, since such costs can be placed on the allocation arcs instead.

PROCNET incorporates the basis trees into a single, larger tree which, following [14], we call the master basis tree. The root of this tree is called the master root, and all basis trees have their roots connected to the master root by artificial arcs known as external arcs. These external arcs are introduced solely for ease in handling the basis trees. The slack arc of the basis quasi-tree is disregarded since it plays no role in the implementation. The master basis tree is maintained by means of the predecessor, depth, thread, and reverse thread functions [2, 3, 15, 16, 17].

In PROCNET, the transposed working basis Q^T is maintained in LU factored form by means of the Harwell LA05 routines [26, 27]. The procedure used in PROCNET to obtain an initial basis for PPN is based on heuristics described in [4, 10]. The arc data for each processing arc is generated in order to apply the procedure. The resulting pure network with proportional flow restrictions relaxed is solved first. Next, the flow values of the relaxed solution on the allocation arcs are used to create a new pure network problem with nonzero lower bounds on the processing arcs. If the flow value on the allocation arc (u, v) of Figure 1 is x, then the lower bound on arc (v, w(z)) is set to $0.7\alpha_{vw(z)}x$. The value 0.7 was chosen during preliminary testing and was not changed throughout the tests described in Section 6. The pure network problem with lower bounds is then solved, and we say that the optimal flows for this problem on the allocation arcs are approximate allocation values. These approximate allocation values become the flows on the allocation arcs in the initial PPN basis as described next. For any allocation arc whose approximate allocation value is between the bounds given in PPN, PROCNET creates a parallel allocation arc. If the approximate allocation value for such an arc is $\bar{x}$ and the capacity of the arc is $\bar{h}$, then this arc is assigned a new capacity $\bar{h} - \bar{x}$ while its parallel arc is given a capacity $\bar{x}$. Both of these arcs have costs equal to the cost of the allocation arc in PPN. The allocation arc is nonbasic at 0 while its parallel arc is nonbasic at capacity in the initial PPN basis. Similarly, an allocation arc whose approximate allocation value is at a PPN bound is set nonbasic at this bound. The approximate allocation values induce proportional flows in the processing arcs and these in turn induce supplies at the processing nodes. The pure network problem which has these induced supplies as well as the original supplies of PPN and which has the processing arcs removed is then solved. The solution of the latter pure network problem is accomplished by means of the network simplex algorithm with an all-artificial initial basis, where the artificial arcs have Big-M costs. Since the feasibility of the pure network problem is not guaranteed, its optimal basis tree is likely to contain artificial arcs with positive flow. This optimal basis tree becomes the basis quasi-tree for the initial PPN basis. All processing columns are included in the initial basis and the rooted basis trees consist of the p splitting nodes. It follows that the initial Q is the p x p identity matrix and Theorem 2 guarantees that we have created a PPN basis.

Next, we discuss other special techniques used in implementing the steps of Algorithm 2. Following Sections 3 and 4, rooted basis trees are numbered 1 through p. The basis quasi-tree is numbered 0. Further, any node in a given basis tree is assigned the number of that tree, and the resulting node function is called treenum. During pricing, treenum is used to provide the i and j values in (4.3). Two candidate lists, L1 and L2, are maintained in PROCNET--L1 for pure network pivots and L2 for processing network pivots. Treenum is used to determine the candidate list on which a given pivot eligible arc is placed. PROCNET repeats Step 2 of Algorithm 2 for all eligible arcs from L1 before updating π_2 in Step 6. The length of L1 was set at 100 and the length of L2 was set at 30. After all pivots from L1 have been made, the best pivots from L2 (up to 20 pivots) are made. This logic for L2 follows [23]. The parameter values used for L1 and L2 remained fixed during the computational tests described in Section 6.

The ratio test for pure network pivots is implemented by using the depth and predecessor functions to identify the cycle determined by the entering arc. The only arcs for which ratios are computed are on this cycle. For processing network pivots, y_2 in (3.12) is computed using the LA05 routines. The y_1 values in (3.10) are then computed with the aid of the reverse thread function. Since processing columns are always basic, only the y_1 values are used in the ratio test.

In step 5, when arc e is incident to τ_i and τ_j, this arc is adjoined to the master basis tree and the external arc between the master root and τ_j is removed. If the leaving arc is f and the subtree below f is τ, then f is removed from the master basis tree and an external arc from the master root to τ is adjoined. The node functions on τ_j and τ are updated as in pure networks [2, 3, 15, 16, 17]. The α values (2.2) of each basis tree are linked to facilitate updating rows of Q.

6. COMPUTATION

All test problems in this study were solved by PROCNET and MINOS. These are both in-core FORTRAN codes, and testing was done using the FTN 4 compiler with optimization level 2 on the CDC 170/750 at The University of Texas. The execution times reported are in central processor seconds and are exclusive of input and output.

Although MINOS is designed for linearly constrained problems with nonlinear objectives, none of the nonlinear subroutines were used here. When restricted to linear problems, MINOS uses the revised simplex algorithm with Phase I-II and maintains the basis in LU factored form. The PARTIAL PRICE parameter for MINOS was set to 20 and the basis was reinverted every 60 iterations. Other MINOS parameters were set to default values [24].

Parameter settings used for PROCNET in addition to these provided in Section 5 are given next. The Big-M value used in the starting procedure was 99999. A reduced cost tolerance of 10^{-5} was used, and pivots with minimum ratio less than 10^{-10} were treated as degenerate. The matrix Q^T was reinverted each time 60 column updates had been performed. In the LA05 routines, pivot elements less than 0.1 times the largest element in the pivot row were excluded. LA05 default values were used for other parameters.

6.1 Test problems and discussion of results.

The class of allocation/processing (AP) network problems previously described in [8, 9] was used for computational tests. These problems have a dual block angular form where the subproblems corresponding to diagonal blocks are transportation problems and the coupling columns are processing columns. The AP problems considered here have transportation subproblems which may be sparse, whereas those of [8, 9] were dense.

The problem data for AP problems is randomly generated. As these problems are generated, a feasible flow is created. The capacity of each arc having finite capacity is set to a parameter μ times the feasible flow generated for that arc. Although other problem data was randomly generated as previously stated, the total supply was fixed at 10000 for all test problems. Two cost ranges were used for the test problems and they are described as follows. Cost range A has costs on the allocation arcs in the range 100 to 150 and other arc costs in the range 1 to 100. Cost range B has costs on the allocation arcs in the range 1 to 100 with other arc costs in the range -100 to -1.

The main test problem data is given in Table 1. Each row of this table represents three problem groups, and every problem in these three groups has the same network topology. Each problem group contains two problems--one with cost range A and one with cost range B. For the problems described in Table 1, only the allocation arcs may have a finite capacity.

Table 1. AP PROBLEM DATA

Problem Groups	Rows (m)	Columns (n + p)	Processing Columns (p)	Nonzeros per Proc. Column
1 - 3	781	2410	10	7
4 - 6	1121	3510	10	8
7 - 9	876	2550	50	6
10 - 12	1201	3650	50	7
13 - 15	1001	3300	100	5
16 - 18	1276	4300	100	4
19 - 21	1051	3750	150	4
22 - 24	1351	4650	150	4
25 - 27	1276	4400	200	4
28 - 30	1501	5000	200	4

Computational results for Problem Groups 1 - 30 are presented in Table 2. The times and iteration counts reported are average values for the two problems in each group. The results are presented in this way since no clear pattern emerged regarding the two cost ranges. PROCNET start time is the time required to create the initial PPN basis. PROCNET iterations begin with the initial PPN basis and include both pure and processing network pivots. The number of basic allocation arcs at optimality is determined by PROCNET. For an implementation of Algorithm 1 this would be the dimension of the working basis at optimality. Over the 30 problem groups, the ratio of total MINOS time to total PROCNET time is 10.05.

Table 2. Computational Results for AP Problems

Problem Group	μ	PROCNET Start Time	PROCNET Total Time	PROCNET Iterations	Basic Alloc. Arcs at Opt.	MINOS Total Time	MINOS Iterations
1	1.1	1.9	4.9	154	10	91	1600
2	2.0	1.7	13.9	767	10	99	1736
3	∞	2.7	14.2	693	10	103	1846
4	1.1	3.1	7.8	211	10	188	2381
5	2.0	2.9	24.2	919	10	203	2540
6	∞	2.2	23.7	942	10	207	2623
7	1.1	2.8	15.6	540	14	251	3920
8	2.0	2.4	45.1	1737	44	353	5075
9	∞	1.9	42.2	1633	49	360	4969
10	1.1	4.6	33.0	1006	16	537	6112
11	2.0	3.6	68.3	2178	42	756	7900
12	∞	2.9	85.7	2829	49	731	7455
13	1.1	4.6	31.6	943	26	449	6114
14	2.0	3.8	51.4	1646	69	580	7116
15	∞	2.9	71.4	2365	90	553	6702
16	1.1	7.6	25.5	541	29	562	6687
17	2.0	5.9	49.2	1467	76	634	7520
18	∞	5.1	70.1	2322	93	599	6940
19	1.1	3.5	14.8	377	87	240	3216
20	2.0	3.4	34.4	1200	91	256	3584
21	∞	2.8	41.3	1483	124	224	3271
22	1.1	3.9	19.7	465	96	356	3828
23	2.0	4.0	48.8	1414	130	361	3964
24	∞	3.5	62.1	2002	140	349	3856
25	1.1	8.6	49.8	981	48	794	9049
26	2.0	6.2	100.3	2338	129	911	9899
27	∞	4.7	132.0	3563	165	831	8769
28	1.1	8.6	60.6	1174	48	1114	9818
29	2.0	6.3	98.9	2209	139	1236	10958
30	∞	5.0	145.3	3900	191	1009	8965
Totals			1485.8			14937	

Degeneracy has been a cause for concern in the solution of pure network problems. For example, it was reported in [3] that on some test problems, more than 90% of the pivots were degenerate. For the AP processing network problems in Table 1, degenerate pivots were far less prevalent. For these problems, 7.2% of the pivots were degenerate when solved by PROCNET. The percentage of degenerate pivots does increase with problem size, however, and it is possible that degeneracy may play more of a role as larger processing network problems are solved.

An alternative strategy for handling candidate list L1 was coded into a modified version of PROCNET. This alternative strategy takes only the best pivots (up to 50) from L1 before taking pivots from L2. With this modified version of PROCNET, the problems in Problem Groups 28, 29, and 30 were solved again. The sum of the solution times for these three problem groups was decreased by about 3%.

6.2 Effects of changes in capacity.

Total solution times for problem groups from Table 1 with given μ values are reported in Table 3. As Table 3 shows, the total time ratio exhibits a wide variation with μ. Apparently, the initial PPN basis generated by PROCNET gives a better start for smaller values of μ. These results indicate that PROCNET will be highly efficient on problems where capacity for processing activities, such as assembly or refining, is quite limited, while capacity for network activities, such as shipment of raw materials or finished products, is essentially unlimited.

Table 3. Solution Times vs. μ Values

Problem Groups	μ	PROCNET Total Time	MINOS Total Time	MINOS/PROCNET Total Time Ratio
1, 4,...,28	1.1	263.3	4582	17.40
2, 5,...,29	2.0	534.5	5389	10.08
3, 6,...,30	∞	688.0	4966	7.22

The results of Table 3 led us to investigate problems in which all arcs have finite capacities. Only two new problem groups were considered because of budgetary restrictions on computer time. They are Problem Groups 31 and 32 and they have the same problem data as Problem Groups 8 and 11, respectively, except that all arcs have finite capacities generated with $\mu = 2.0$. As the data from Table 4 shows, the transition to the new problem groups causes the MINOS/PROCNET total time ratio to increase sharply over corresponding values for Problem Groups 8 and 11. We note that for Problem Group 32 this ratio is 17.55. The increases in this ratio are explained by an accompanying sharp increase in the time required by MINOS to achieve feasibility.

Table 4. Further Computational Results for AP Problems

Problem Group	μ	PROCNET Start Time	PROCNET Total Time	PROCNET Iter-ations	Basic Alloc.Arcs at Opt.	MINOS Total Time	MINOS Iter-ations
31	2.0	2.9	83.5	3403	50	771	10470
32	2.0	4.4	93.6	3640	50	1643	16256

6.3 Effects of changes in number of processing columns.

In Table 5, the average pivot time is introduced as a measure of the efficiency of a given code in carrying out a simplex iteration. Obviously, the total solution time depends on this measure and on the number of pivots required to reach optimality. Because the majority of work involved in a pivot for PROCNET is expended on working basis operations, it was anticipated that a downward trend in the average pivot time ratio might occur as p increases, and this is confirmed in Table 5. On the other hand, the effectiveness of the initial basis of PROCNET and its pricing strategy have resulted in fewer pivots than required by MINOS. Surprisingly, the total time ratio remains above 10 for all values of p except p = 150.

The percentage of pure network pivots and the average pivot time for PROCNET in Table 5 generally show an inverse relationship as might be expected.

Table 5. Performance Values vs Number of Processing Columns.

Problem Groups	Process Columns (p)	PROCNET % Pure Network Pvts.	PROCNET Avg.Pivot Time	MINOS Avg.Pivot Time	MINOS/PROCNET Avg.Pivot Time Ratio	MINOS/PROCNET Ttl.Time Ratio
1-6	10	36	0.0201	0.0700	3.48	10.05
7-12	50	22	0.0274	0.0843	3.08	10.31
31-32	50	28	0.0241	0.0903	3.75	13.63
13-18	100	21	0.0290	0.0822	2.84	11.29
19-24	150	25	0.0288	0.0822	2.86	8.08
25-30	200	17	0.0387	0.1026	2.65	10.04

6.4 Future implementation and testing.

Computational comparison of PROCNET and MINOS in the present study has been limited by the core storage requirements of MINOS, which are considerably greater than those of PROCNET. It is, however, highly desirable to conduct tests using larger problems to determine more clearly the class of problems for which PROCNET is effective. We plan to test a new version of PROCNET against the general purpose LP code MPSX/370. The new version of PROCNET will itself be coded in PL/I and included in an extended control language program of the MPSX/370 system. High level MPSX modules will replace the LA05 routines for handling the working basis. We also plan to implement Algorithm 1 using a modification of the LA05 routines to maintain the working basis. This implementation should execute faster than the present PROCNET because it will have a working basis whose average dimension is smaller.

7. CONCLUSION.

Basis structure and related simplex calculations were studied for partitioned pure processing network bases. An explicit representation of the working basis was presented. This representation allows a working basis row to be generated by graph tracing techniques applied to a corresponding basis tree. Two new primal simplex variants were defined, and for certain pivot types it was shown how tree operations in the basis exchange step result in the replacement of at most three working basis rows. For one of the primal simplex variants, the working basis update is accomplished by means of standard LP column replacement techniques applied to the transposed working basis. This variant was implemented and tested against MINOS on 64 randomly generated problems containing up to 200 processing columns. The specialized code is more than ten times faster than MINOS, and it is particularly effective on tightly capacitated problems.

ACKNOWLEDGMENTS.

We would like to thank Michael Chang for his help with the implementation described in Section 5 and Lawrence Seiford for his editorial assistance.

REFERENCES

1. A. Ali, D. Barnett, K. Farhangian, J. Kennington, B. McCarl, B. Patty and P. Wong, "Multicommodity Network Problems: Applications and Computations," IIE Transactions, Vol. 16, pp. 127-134, 1984.

2. R. Barr, F. Glover and D. Klingman, "Enhancements of Spanning Tree Labeling Procedures for Network Optimization," INFOR, Vol. 17, pp. 16-34, 1979.

3. G. Bradley, G. Brown and G. Graves, "Design and Implementation of Large Scale Primal Transshipment Algorithms," Management Science, Vol. 24, pp. 1-34, 1977.

4. A. Charnes, W.W. Cooper, D. Divine, W. Hinkel, J. Koning and V. Lovegren, "A Sea-shore Rotation Goal Programming Model for Navy Use," Research Report CCS 429, Center for Cybernetic Studies, The University of Texas, Austin, 1982.

5. A. Charnes, W.W. Cooper, B. Golany, V. Lovegren, W. Mayfield and M. Wolfe, "The GPSSR System to Support Management of Policy and Execution of The U.S. Navy's Sea-shore Rotation Program," Research Report CCS 495, Center for Cybernetic Studies, The University of Texas, Austin, 1984.

6. S. Chen and R. Saigal, "A Primal Algorithm for Solving a Capacitated Network Flow Problem with Additional Linear Constraints," Networks, Vol. 7, pp. 59-79, 1977.

7. R. Crum, D. Klingman and L. Tavis, "An Operational Approach to Integrated Working Capital Planning," Journal of Economics and Business, Vol. 35, pp. 343-378, 1983.

8. M. Engquist and C.-H. Chen, "Efficient Tree Handling Procedures for Allocation/Processing Networks," Research Report CCS 437, Center for Cybernetic Studies, The University of Texas, Austin, 1982.

9. M. Engquist and C.-H. Chen, "Computational Comparison of Two Solution Procedures for Allocation/Processing Networks," to appear in a forthcoming Mathematical Programming Study.

10. F. Glover, R. Glover and F. Martinson, "A Netform System for Resource Planning in the U.S. Bureau of Land Management," to appear in Journal of The Operational Research Society.

11. F. Glover, J. Hultz and D. Klingman, "Improved Computer-based Planning Techniques," Interfaces, Vol. 8, pp. 16-25, 1978.

12. F. Glover, D. Karney, D. Klingman and R. Russell, "Solving Singly Constrained Transshipment Problems," Transportation Science, Vol. 12, pp. 277-297, 1978.

13. F. Glover and D. Klingman, "Capsule View of Future Developments on Large-scale Network and Network-related Problems," Research Report CCS 238, Center for Cybernetic Studies, The University of Texas, Austin, 1975.

14. F Glover and D. Klingman, "The Simplex SON Algorithm for LP/Embedded Network Problems," Mathematical Programming Study, Vol. 15, pp. 148-176, 1981.

15. F. Glover, D. Klingman and J. Stutz, "Augmented Threaded Index Method for Network Optimization," INFOR, Vol. 12, pp. 293-298, 1974.

16. P. Jensen and J. Barnes, Network Flow Programming, John Wiley and Sons, New York, 1980.

17. J. Kennington and R. Helgason, Algorithms for Network Programming, John Wiley and Sons, New York, 1980.

18. D. Klingman and R. Russell, "On Solving Constrained Transportation Problems," Operations Research, Vol. 23, pp. 91-107, 1975.

19. J. Koene, "Minimal Cost Flow in Processing Networks, a Primal Approach," Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 1982.

20. A. Manne, R. Richels and J. Weynant, "Energy Policy Modelling: A Survey," Operations Research, Vol. 27, pp. 1-36, 1979.

21. R. McBride, "Solving Generalized Processing Network Problems," Working Paper, School of Business, University of Southern California, Los Angeles, 1982.

22. R. McBride, "Solving Embedded Generalized Network Problems," to appear in European Journal of Operations Research.

23. J. Mulvey, "Pivot Strategies for Primal-Simplex Network Codes," Journal of The Association for Computing Machinery, Vol. 25, pp. 266-270, 1978.

24. B. Murtagh and M. Saunders, "MINOS User's Guide," Technical Report SOL 77-9, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, 1977.

25. B. Murtagh and M. Saunders, "Large Scale Linearly Constrained Optimization," Mathematical Programming, Vol. 14, pp. 41-72, 1978.

26. J. Reid, "FORTRAN Subroutines for Handling Sparse Linear Programming Bases," Report AERE-R8269, Computer Science and Systems Division, AERE Harwell, Oxfordshire, England, 1976.

27. J. Reid, "A Sparsity-exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases," Mathematical Programming, Vol. 24, pp. 55-69, 1982.

28. E. Steinberg and H. Napier, "Optimal Multi-level Lot Sizing for Requirements Planning Systems," Management Science, Vol. 26, pp. 1258-1271, 1980.