END

FILMED

DTIC

1.0

4.5 2.8 2.5
5.0 3.2 2.2
6.0 3.6
1.1   4.0 2.0

1.8

1.25 1.4 1.6
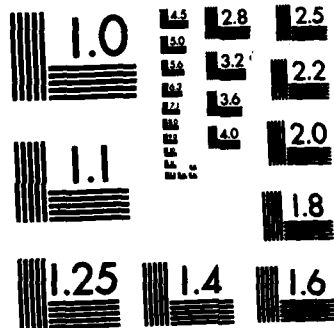
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN EXTENDED MICROCOMPUTER-BASED NETWORK
OPTIMIZATION PACKAGE

by

Michael Edward Finley

October 1982

Advisor: G. G. Brown

Approved for public release, distribution unlimited

83 02 018 056

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
	AD-A124 659	

4. TITLE (and Subtitle)	5. TYPE OF REPORT & PERIOD COVERED
An Extended Microcomputer-Based Network Optimization Package	Master's Thesis October 1982
	6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)	8. CONTRACT OR GRANT NUMBER(s)
Michael Edward Finley	

9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Naval Postgraduate School Monterey, California 93940	

11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE
Naval Postgraduate School Monterey, California 93940	October 1982
	13. NUMBER OF PAGES 88

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report)
	Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Network, generalized network, microcomputer, optimization, network with gains, linear programming, minimum cost network flow, transshipment model, transportation model, mathematical programming.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The capacitated generalized transshipment problem is the most general and universally applicable member of the class of network optimization models. This model subsumes, as specializations, the capacitated and uncapacitated transportation problems as well as the pure network specializations of these models, which include the personnel assignment problem, the maximum flow, and shortest path formulations. The generalized

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73
S/N 0102-014-6601 |

network problem, in turn, can be viewed as a specialization of a linear programming problem having at most two non-zero entries in each column of the constraint matrix. A detailed description is given of the implementation of an efficient algorithm and its supporting data structures, used to solve large-scale, minimum-cost generalized transshipment problems on an Apple II (64K) microcomputer. A suite of advanced techniques for managing minimum-cost network flow models and inherent data elements will also be discussed.

An Extended Microcomputer-Based Network
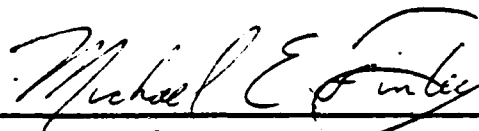Optimization Package

by

Michael Edward Finley
Lieutenant Commander, Supply Corps, United States Navy
B.A., Cornell University, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the
NAVAL POSTGRADUATE SCHOOL
October, 1982

Author: _____

Approved by: _____ GERALD G. BROWN
                                        Thesis Advisor

_____
                                        Second Reader

_____
Chairman, Department of Operations Research

_____
Dean of Information and Policy Sciences

3

ABSTRACT

The capacitated generalized transshipment problem is the most general and universally applicable member of the class of network optimization models. This model subsumes, as specializations, the capacitated and uncapacitated transportation problems as well as the pure network specializations of these models, which include the personnel assignment problem, the maximum flow, and shortest path formulations. The generalized network problem, in turn, can be viewed as a specialization of a linear programming problem having at most two non-zero entries in each column of the constraint matrix. A detailed description is given of the implementation of an efficient algorithm and its supporting data structures, used to solve large-scale, minimum-cost generalized transshipment problems on an Apple II (64K) microcomputer. A suite of advanced techniques for managing minimum-cost network flow models and inherent data elements will also be discussed.

4

TABLE OF CONTENTS

LIST OF FIGURES

NOTATION

Except as otherwise indicated in the text, the following notational conventions have been used for all mathematical expressions:

Vectors:	Lower case Latin (e.g., u)
Vector Components:	Lower case Latin with subscript (e.g., u_i)
Matrices:	Upper case Latin (e.g., A)
Matrix Column:	Upper case Latin with superscript column index (e.g., N^k)
Matrix Row:	Upper case Latin with subscript row index (e.g., A_i)
Matrix Element:	Lower case Latin with subscript row and column indices (e.g., a_{ij})
Set:	Upper case script (e.g., $\mathscr{AR}$)
Scalars:	Lower case script (e.g., g) if emphasis is required; otherwise lower case Latin (e.g., i)

I. INTRODUCTION

Since the development of the Simplex method by George Dantzig, and the introduction of the transportation model by Tjalling Koopmans (both circa 1947), network models have enjoyed wide use. Perhaps two reasons for this attention are the frequent occurrence of situations which are readily modelled as networks and the mathematical and computational elegance which may be achieved through network specializations of the Simplex method. Undoubtedly, the visually appealing graphical description provided by the network formulation has contributed much to the managerial acceptance of these models. Network models have been used in a large number of applications. Jensen and Barnes [Ref. 1] provide a number of examples of network modelling techniques and applications, as do Kennington and Helgason [Ref. 2], Dantzig [Ref. 3], and Bradley [Ref. 4]. Some of these applications deal with military logistic and distribution systems, communications, and pipeline systems, personnel and resource assignments, and production planning.

In recent years, advances in solid state technology have enabled design and production of extremely powerful microprocessor-based computers. The usefulness of these computers to applications of mathematical programming has been largely overlooked by all but a few researchers. The computational efficiency, speed, and elegance of network algorithms and the broad range of application of the generalized network formulation make microcomputer-based network optimization extremely attractive.

The first microcomputer-based network software suite has been designed and implemented on an Apple II Plus. This network system exhibits more capability than any other package available (on any host computer) at this writing. It is designed as an integrated suite of programs capable of solving pure capacitated, non-linear, elastic (with fixed charges), and capacitated generalized network problems and includes a user-friendly interface which facilitates data input and manipulation.

The capacitated generalized transshipment problem is the most general and universally applicable of the network optimization models. This model class embraces, as specializations, the capacitated transportation problem and the pure network class of models. As viewed here, the object of these formulations is to determine in what manner a good or commodity should flow through a network such that flow is conserved at each node and the total cost of flow through the network is minimized.

(LP)   minimize $f(x)$ cost

   s.t. $Ax = b$ conservation constraints

     $lb \leq x \leq cp$ bounds on flow

This problem can easily be formulated as a linear program (LP), however, the network specialization provides significant computational savings, often producing solutions in one hundredth the time [Ref. 2] required by the equivalent linear program. Additionally, the network formulation, when viewed pictorially as a collection of arcs and nodes, has an obvious intuitive appeal, making it more readily accepted and understood by non-analysts [Ref. 4].

There is an important difference between the arcs of a pure network and the arcs of a generalized network; associated with each arc of a generalized network is a multiplier which acts on the flow through that arc. The arc multipliers may serve to transform the units of flow or they may change the amount of flow through an arc [e.g., Ref. 5]. For example, if we wish to represent the conversion of steel into automobile chasses at the rate of 1/10 ton of steel per chassis, the arc multiplier converting tons of steel into chasses would be 10. Ten automobile chasses can be manufactured from each ton of steel. Alternatively, if flow on an arc is in terms of investment dollars from one year to the next at an annual rate of return of 12 percent, the multiplier associated with the arc representing that investment would be 1.12.

In the spirit of Bradley, Brown, and Graves [Ref. 6] and Brown and McBride [Ref. 7], the network formulation may be described as a directed graph G defined by a set of nodes ND and a set of arcs AR. Henceforward, n will be referred to as the number of arcs in a network and m will represent the number of nodes. The conceptual constraint matrix A is thus m rows by n columns.

Members of the arc set are indexed by k and are defined as an ordered pair (tail, head) or (source node, destination node). Associated with each arc there is a cost per unit flow c_k, a lower bound on flow lb_k, an upper bound on flow, or capacity, cp_k. The flow on arc k is represented by x_k.

The generalized network model employs an arc multiplier m_k which represents a gain or loss in material flowing across arc k. When this

11

arc multiplier is unity for all arcs in the network, the model is then a pure network specialization of the generalized network model.

Each node can be designated as a supply node (material enters the network), a demand node (material leaves the network), or a transshipment node (material just passes through).

The problem is to minimize the total cost of flow on all the arcs, such -that the flow on each of the arcs remains within the stipulated bounds, demands are met from available supplies, and flow is conserved at each node. Conservation of flow requires that the flow leaving a node minus the flow entering a node equals the external flow or requirement of that node. In generalized networks, the flow entering a node is the sum of the flows on the incoming arcs multiplied by the associated arc gains. These requirements can be written as:

(GNP)   min $\sum\limits_{k \in AR} c_k x_k$

s.t. $\sum\limits_{\substack{k \in AR \\ \text{leaving } i}} x_k - \sum\limits_{\substack{k \in AR \\ \text{entering } i}} m_k x_k = b_i$, $\forall i \in ND$ conservation of flow

$lb \leq x \leq cp$, bounds on flow

where $0 < b_i$ = supply quantity, if i is a supply node

$0 > b_i$ = - (demand quantity), if i is a demand node

$0 = b_i = 0$, if i is a transshipment node.

The convention followed here is that supplies are represented by positive magnitudes and demands are negative. This algebraic template can accommodate a model with inequality flow constraints, insuring that no more than the available supply will be utilized and that no less than

12

the demand will be provided. Slack arcs represent the difference between available supply and that portion actually used, and surplus arcs measure the extent to which shipments exceed demand. When slack and surplus arcs are utilized in this manner, a formulation which uses inequality constraints can be transformed to the equality constraints of (GNP). Slack and surplus arcs are considered to be "logical" arcs as opposed to the "structural" arcs of the original problem.

(GNP) is a specialization of (LP). If each column of the constraint matrix A in (LP) corresponds to an arc, then it has at most two non-zero entries; those entries can be scaled to be +1 (for the tail) and some other number $-m_k$ (at the head). Thus, the constraint matrix of (GNP) contains elements that are either 0, 1 or $-m_k$ (each m_k is admissably distinct). When the Ax matrix multiplication of (LP) is enforced, we obtain the flow conservation constraint found in (GNP). There is thus one conservation of flow constraint in (LP) for every node i: $A_i x$, where A_i is the row in A corresponding to the ith node in (GNP), having row entries of +1 for every arc originating at node i, and $-m_k$ for every arc terminating at node i. There is also a pair of bounds in (LP) and in (GNP) for every arc in the network.

(GNP) is sufficiently broad to enable any linear program, with at most two non-zero coefficients associated with each variable (column), to be treated as a generalized network. Even when the linear program is not entirely composed of network columns, Brown and Wright [Ref. 8] have shown that many real-life linear programs contain a large embedded network structure which, once discovered, can be exploited to improve solution efficiency [Ref. 7].

13

Solution speed and efficient data storage techniques are two of the most attractive features of the network model. Due to the obvious sparseness of the network constraint matrix, the use of a node-arc incidence matrix is not a viable method of basis representation for data manipulation. Space and speed-efficient methods of handling the representation and manipulation of the generalized network problem are introduced in the following sections. The algorithm which will be described is called GENNET [Ref. 7].

II. <u>BASIS REPRESENTATION AND DATA STRUCTURES</u>

The constraint matrix A of (LP) represents a set of m linear equations in n unknowns. If $m < n$, a feasible solution to these constraints may be found by identifying m linearly independent columns of the constraint matrix A. If the variables associated with the remaining $n-m$ columns of A are set to zero, the values of the variables corresponding to the m linearly independent columns may be uniquely found by solving the resulting set of exactly determined simultaneous equations. Any set of m linearly independent columns of A is referred to as a basis. The solution to the set of linear equations obtained by setting to zero the variables corresponding to the $n-m$ columns of A not included in the basis (i.e., non-basic columns), is called a basic solution.

A unique characterizing feature of the pure transshipment problem is the triangular nature of its bases [e.g., Ref. 3]. Triangular bases are particularly convenient because they are easily solved by direct substitution [e.g., Ref. 9]. The solution of the generalized transshipment problem is somewhat more difficult because of a complication introduced by flow multipliers.

What is now known as the capacitated transshipment problem was first posed, with a discussion of solution methods, by Koopmans [Ref. 10]. Dantzig [Ref. 3] provides the first general exposition of solution technique and basis representation of the capacitated generalized transshipment problem, referring to it as the "Weighted Distribution Problem." Dantzig shows that the basis of a generalized transshipment problem has

15

unique qualities similar to the basis of the pure network. These qual-
ities will be exploited in the problem representation and solution
approach developed here.

A graphical representation of a generalized network basis results in
a familiar node and arc display [e.g., Ref. 2]. The graph thus obtained
is not necessarily a tree, as in the case of pure networks, but may be a
forest whose members are either trees or trees with one cycle (or loop).
If slack variables are admissible, then the network must also include
slack arcs, each incident with only one node. Figure 1 displays a node
with a slack arc.

Figure 1. Node with a Slack Arc

Danzig [Ref. 3] shows that each component of the basis is either a
rooted tree or a subgraph with one cycle. Figure 2 displays typical
components of a generalized network basis. A rooted tree may be viewed
as a subgraph with a slack arc (and thus one cycle) at the root. Each
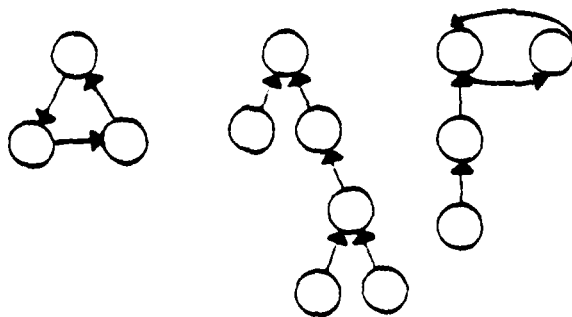
Figure 2. Typical Components of a Generalized Network Basis

16

component of the basis is a connected subgraph with precisely one cycle.
The basis representation of a generalized network is "block diagonal," as
pictured below. The diagonal entries are submatrices corresponding to

$$
\begin{bmatrix}
B^1 & & & & \\
& B^2 & & & \\
& & \cdot & & \\
& & & \cdot & \\
& & & & \cdot \\
& & & & & B^g
\end{bmatrix}
$$

the component trees/subgraphs of the generalized network basis. Those
elements representing rooted trees are upper triangular, as they are in
the pure network case. Dantzig [Ref. 3] shows that basis components
corresponding to subgraphs having one cycle may be put in "nearly tri-
angular" form, with only one column having a non-zero term remaining
below the diagonal. If the variable associated with that "peculiar"
column is treated as a parameter, values for the variables associated
with the other columns can be obtained in terms of that parameter. These
expressions (in terms of the one unknown variable) can be uniquely
solved, thereby obtaining a complete basic solution [Ref. 3]. A solution
method for generalized networks suggests itself; exploit the triangularity
of rooted basis components with efficient data structures and use the
method proposed by Dantzig for nearly triangular basis elements.

To illustrate these concepts suppose we have the basis representation displayed in Figure 3.

rooted tree with slack arc at root

Subgraph with one cycle
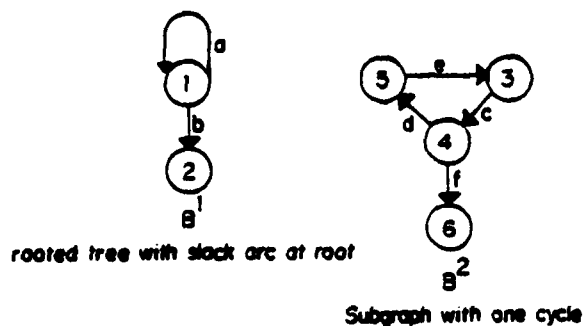
Figure 3. Generalized Network Basis Representation

There are six nodes and thus six arcs in this basis, one of which is a slack arc (arc a) corresponding to the root node of its basis tree. The matrix representation of this basis is shown in Figure 4.

Arc	a	b	c	d	e	f
Node						
1	1	1				
2		$-m_b$				
3			1		$-m_e$	
4			$-m_c$	1		1
5				$-m_d$	1	
6						$-m_f$

Figure 4. Matrix Representation of a Basis

18

The multiplier associated with slack and artificial arcs is -1 and a multiplier of +1 is used for surplus arcs. Thus, $m_a = -1$ since arc "a" is a slack arc. Permuting the rows of this basis matrix produces:

Arc	a	b	c	d	e	f
Node						
1	1	1				
2		$-m_b$				
4			$-m_c$	1		1
5				$-m_d$	1	
3			1		$-m_e$	
6						$-m_f$

$$= \begin{bmatrix} B^1 & 0 \\ 0 & B^2 \end{bmatrix}$$

Figure 5. Triangular/Nearly Triangular Basis

The dotted lines segregate the two basis components B^1 and B^2. Note that B^1 corresponds to a rooted tree and is thus upper triangular. B^2 is "nearly triangular." There is one "singleton" column in the basis which corresponds to the slack arc a.

The sparsity of the constraint matrix and the graphical structure of the model lead us to adopt one-dimensional data structures to represent the problem. These data structures provide economical storage and reduce computational overhead. The two most common types of data structures found in network optimization algorithms are the triple-lable scheme, first proposed by Ellis Johnson [Ref. 11] and the preorder traversal method, as described by Bradley, Brown, and Graves [Ref. 6].

The triple-label scheme employs three functions to represent the basis graph: a predecessor function which provides the parent, father, or immediate ancestor of a node; a successor function which provides the left-most child of a node (sometimes called the eldest son); and a brother function which provides the next left-most node with the same parent (i.e., the next oldest brother or sibling). These functions are exhibited in Figure 6.
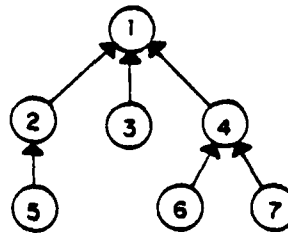
Node	Predecessor	Successor	Brother
1	0	2	0
2	1	5	3
3	1	0	4
4	1	6	0
5	2	0	0
6	4	0	7
7	4	0	0

Figure 6. Triple-Label Scheme

The triple-label scheme has been adopted by several researchers [Ref. 12 for the (pure) transportation network problem and Ref. 13 for generalized networks]. Brown and McBride [Ref. 7] have tested, but not adopted, this data structure. Kennington and Helgason [Ref. 2] and Jensen and Barnes [Ref. 1] repeat textbook explanations of the required basis update actions for the triple-label method of basis representation.

The preorder traversal method uses two functions to represent and update the basis: a predecessor function P and a preorder function IT. The predecessor function provides the same information as in the triple-label scheme, i.e., the "father" node. The predecessor function can be easily constructed from the matrix representation of the basis (e.g., Figure 5). After rearranging the rows and columns and forming triangular/nearly triangular components of the basis, to find the predecessor of node i:

1. Determine the row number of node i, call that row r1.

2. Determine the row of the non-zero off-diagonal element of column r1, call that row r2.

3. The predecessor of node i is the node represented by row r2. If no off-diagonal element is found in step 2, the predecessor of node i is null. (An array P may be used to represent the predecessor function and a distinguished value, e.g., $m + 1$, can indicate nodes with no predecessors.)

Each node and its predecessor define the basic arc (ignoring for the moment the arc's orientation). The predecessor function can be used to construct a "backpath" from any node to the root/cycle of its basis component. The orientation of the arc in the original network can be recorded using the sign of the predecessor. If the arc is oriented from node i to P(i), then $P(i) > 0$; if $P(i) < 0$, then the arc is directed from P(i) to i.

When determining the predecessor of a node i, if a multiplier $(-m_k)$ is discovered on the diagonal of the nearly triangular basis matrix, then $P(i) < 0$. The multiplier value is associated with the "destination end" of a basic arc. The arc k represented by a column having $-m_k$ on the diagonal of the basis matrix is oriented from P(i) to i.

The sign of the predecessor uniquely determines the diagonal entries of the basis matrix. If $P(i) < 0$, the diagonal entry in the row representing node i is $-m_k$; if $P(i) > 0$, the diagonal entry is +1. A predecessor function applied to Figure 3 yields:

Arc	a	b	c	d	e	f	Node	Predecessor
Node								
1	1	1					1	$7 = m + 1$
2		$-m_b$					2	-1
r2 4			$-m_c$	1		1	3	-5
5				$-m_d$	1		4	-3
3			1		$-m_e$		5	-4
r1 6						$-m_f$	6	-4

Figure 7. Predecessor Function

A node may have "successors" or nodes which are further from the root/cycle than the node in question. The set of nodes which are first encountered on all paths from a node, except the path to the root/cycle, are called the "immediate successors" of the node. A basis may alternatively be completely represented by this successor relationship. However, because any node can have a variable number of successors but only one predecessor in the basis, the predecessor function provides a more tractable data structure.

In the example presented in Figures 5 and 7, node 4 is the predecessor of both nodes 5 and 6. Clearly, the predecessor function does not completely represent the triangulated basis. The piece of missing

information is whether node 5 or node 6 is encountered first (top to bottom) in the triangulated basis.

The technique employed here to represent this ordering is an extension to m-trees of the preorder binary tree traversal described by Knuth [Ref. 14]. Graphically, preorder is a dynastic ordering reminiscent of the inheritance of thrones, in which the root or top-most node is listed first and its subtrees are listed in preorder, until every node is listed. Figure 8a displays such a preorder.

Preorder A,B,C,D,E,F,G,H

Figure 8a. Preorder of a Tree

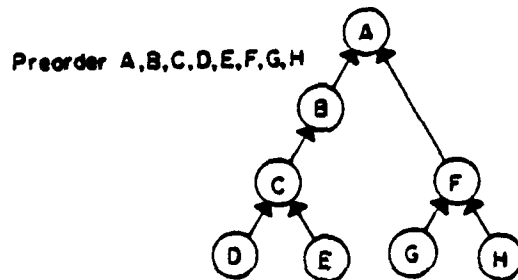Figure 8b illustrates an extension of preorder to the bases of generalized networks. The extension of preorder to "trees" with cyclic roots requires that one node on the cycle be distinguished as the first preorder node. The choice of that distinguished node is arbitrary amongst the cycle nodes.

PREORDER
C, D, E, F, B, G, A, H, I, J
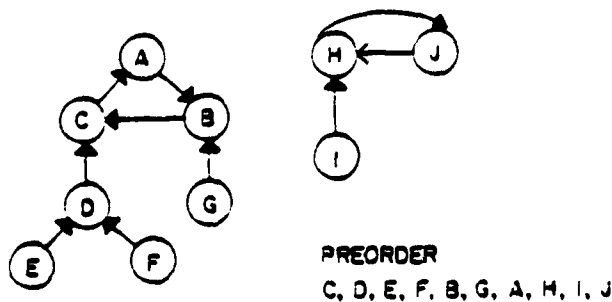
Figure 8b. Extension of Preorder to Generalized Networks

23

The matrix structure of the triangulated basis also induces a preorder. Row i always precedes its successors in the triangulated/nearly triangulated basis and all of its successors precede any row which does not precede row i [Ref. 15].

To summarize, the basis representation we have established is contained in the functions P and IT. P indicates the predecessor of every node in the generalized network. IT provides the preorder-successor of each node. By "iterating" IT (e.g., IT(i), IT(IT(i)), IT(IT(IT(i))) ...), all the preorder-successors of node i may eventually be found. Iteration of the predecessor and preorder functions provides a means of tree traversal. The preorder traversal scheme is considered to be the more elegant and efficient means of basis representation and as such will be used in this description of the GENNET algorithm.

The predecessor and preorder functions are implemented as arrays P and IT. P(i) indicates the predecessor of node i. Similarly, IT(j) indicates the preorder-successor of node j. Associated with each node j is an arc which connects node j with its predecessor P(j). This arc corresponds to a basic variable. The index of, or pointer to, the arc connecting node j and node P(j) is maintained in IVAR(j), an element of a node-length (i.e., $m \times 1$) array IVAR. The sign bit of P(j) records the orientation of the arc connecting nodes j and P(j). If $P(j) > 0$, then the arc connecting node j and its predecessor is oriented from node j to P(j); an arc oriented from the predecessor of node j to node j is indicated by $P(j) < 0$.

Several "housekeeping" arrays are used to support the basis data structures. An array D is maintained to store the depth of each node.

24

Jensen and Barnes [Ref. 1] seem to dismiss the preorder traversal scheme of basis representation for generalized networks due to the apparent difficulty of assigning depths to the nodes of a cycle. This difficulty is overcome by defining the depth of all cycle nodes to be 0. The depth of a node not on a cycle is one more than the depth of its predecessor.

Additional node-length arrays include U which contains the values of the dual variables (or simplex multipliers) and X which contains the value of the right-hand side for each node's conservation of flow equation.

A node-length array FAC is used to store cycle factors and array PC stores information about the incoming non-basic column and is used to effect a simplex pivot. These arrays will be discussed later.

To associate arc numbers with the node pairs connected by an arc, we maintain two arrays: T and H. T is an arc-length array which stores the tail, or source node, of each arc. If the arcs in T are sorted so that all arcs with the same head node are listed contiguously, a space savings can be effected by only storing, for each node i, the location of the first arc whose head, or destination, is i. This information is stored in a node-length array H; $H(i)$ contains the index of the first arc in T whose head is node i. Thus, the tails of all the arcs whose head is node i are: $T(H(i))$, ... $T(H(i + 1) - 1)$. If $H(i) = H(i + 1)$, then no arcs terminate at node i.

Associated with each arc are arrays which describe the arc's characteristics. These arrays are naturally indexed by arc number. The array CP stores the capacity or upper bound on flow, C contains the cost per unit flow, and MUL contains the arc multiplier or gain.

It is only necessary to consider upper bounds in the solution procedure as long as we "transform out" any lower bounds required by the model. For an arc k, from node i to node j with lower bound lb_k, this transformation is easily performed as follows:

$$CP(k) <== CP(k) - lb_k$$

$$Obj <== Obj + lb_k * C(k) \qquad \text{where Obj is the value of the objective function;}$$

if $i \neq j$

$$X(i) <== X(i) - lb_k$$

$$X(j) <== X(j) + lb_k * MUL(k);$$

if $i = j$

$$X(i) <== X(i) - lb_k * MUL(k).$$

An arc k, out of the basis at its upper bound, is "reflected" by logically replacing its flow variable x_k by $(cp_k - x_k)$ and that reflection is recorded using the sign bit of CP, reminiscent of bounded variable simplex methods [e.g., Ref. 3]. Figure 9 summarizes the suggested arrays.

Name	Length	Use
P	Node	Predecessor
IT	Node	Preorder-Successor
D	Node	Depth
X	Node	Right-Hand Side
U	Node	Dual Value
FAC	Node	Cycle Factor
IVAR	Node	Basic Variable
PC	Node	Pivot Column Information
H	Node	Head Node Pointer
T	Arc	Tail Node
CP	Arc	Upper Bound
C	Arc	Cost
MUL	Arc	Arc Multiplier

Figure 9. GENNET Arrays

III. ALGORITHM DESCRIPTION

As with any simplex-based algorithm, the solution of the generalized network problem, (GNP), involves three fundamental operations: priceout, ratio test, and pivot. Non-basic variables (arcs) are individually examined in the priceout to determine if inclusion in the basis can yield a better value of the objective function. Given a favorable incoming variable, a ratio test is performed to determine whether changing flow on the incoming variable will result in the incoming variable achieving its opposite bound or a basic variable being driven to one of its bounds. The variable first reaching a bound is deemed "outgoing." Finally, the incoming variable replaces the outgoing variable in the basis via the pivot operation. We examine each of these fundamental steps and their specialization to generalized networks using the representations and data structures presented in the previous section.

First, however, it is appropriate to review some of the principles of the revised simplex method [e.g., Refs. 16, 17]. Consider the following LP:

(LP') $\quad$ min $\;cx$

$\qquad$ s.t. $Ax = b \qquad$ (A is $m \times n$)

$\qquad\qquad 0 \leq x \leq cp.$

The matrix of technological coefficients A may be partitioned into a basic square sub-matrix B (i.e., B is a basis) and a non-basic sub-matrix N. If B consists of the first m columns of A, then $A = [B, N]$.

28

Similarly, we may partition the variable and cost vectors x and c as:
$x = (x_B, x_N)$ and $c = (c_B, c_N)$. (LP') becomes:

$$\min \quad cx = c_B x_B + c_N x_N$$
$$\text{s.t. } Ax = [B, N]\begin{bmatrix} x_B \\ x_N \end{bmatrix} = Bx_B + Nx_N = b$$
$$x \geq 0.$$

The upper bounds in the original formulation may be taken into account by "reflecting" any variable at its upper bound. That is, the value of a reflected variable is measured as the "distance" from its upper bound cp_k (as opposed to the more natural way of measuring values as distance from the lower bound 0). Any variable at its upper bound cp_k is replaced by $(cp_k - x_k)$; to effect this change, coefficient column k is replaced by its negative $-A^k$ and the right-hand side is transformed to $b - A^k cp_k$.

The solution to the LP is obtained as

$$Bx_B + Nx_N = b$$
$$x_B = B^{-1}b - B^{-1}Nx_N. \tag{1}$$

Substituting this into the cost equation:

$$c_B x_B + c_N x_N$$
$$= c_B B^{-1}b - c_B B^{-1}Nx_N + c_N x_N$$
$$= c_B B^{-1}b + (c_N - c_B B^{-1}N)x_N \quad \text{(the cost of the current solution in terms of } x_N\text{)}.$$

Every basic solution to the LP has the characteristic that $x_N = 0$ and thus the cost of that basic solution is $c_B B^{-1}b$. By changing

29

the current basic solution and thereby making an element of x_N non-zero, the cost of the solution would change by the amount $(c_N - c_B B^{-1} N) x_N$. If $c_N - c_B B^{-1} N < 0$, then the overall cost would be reduced. Thus, elements of x_N for which $c_N - c_B B^{-1} N < 0$ are desirable candidates to enter the basis. $c_N - c_B B^{-1} N$ are called the reduced costs. $c_B B^{-1}$ is called the dual solution or simplex multiplier set.

Reviewing the algebraic development of the ratio test, we begin with the general expression for the basic variables (1).

$$x_B = B^{-1} b - B^{-1} N x_N$$

If $x_k \varepsilon x_N$ is the incoming non-basic variable, we obtain

$$x_B = B^{-1} b - Z^k x_k; \qquad \text{where } Z^k = B^{-1} N^k \text{ is the current incoming column.}$$

To preserve feasibility, each of the x's must remain within its respective bounds $(0, cp_k)$. The three constraining conditions searched for by the ratio test are that the incoming variable:

(a) reaches its opposite bound:

$$x_k = cp_k$$

(b) drives a basic variable to its opposite bound:

$$x_{B_i} - z_{ik} x_k = cp_{B_i} \qquad \text{for } z_{ik} < 0$$

$$x_k = (x_{B_i} - cp_{B_i})/z_{ik}$$

(c) drives a basic variable to its current lower bound:

$$x_{B_i} - z_{ik} x_k = 0 \qquad \text{for } z_{ik} > 0$$

$$x_k = x_{B_i}/z_{ik} \qquad \text{where } B_i \text{ indicates the variable which is basic in row i of the constraint matrix and } z_{ik} \text{ is the corresponding } i^{th} \text{ element of } Z^k.$$

The pivot operation updates the current solution to exhibit the exchange of basis elements. This is accomplished by updating the representation of B^{-1} and the right-hand side $B^{-1}b$ via a pivot, or elementary transformation operation (as B^{-1} is seldom explicitly extant), using the incoming column Z^k. If the incoming variable reaches its opposite bound, then no basis exchange is required; however, that variable must be reflected.

Tableau arithmetic is not the most efficient method for manipulating the problem representation of a linear program, nor is it as insightful as the matrix arithmetic of the revised simplex method. As shown, the current inverse of the basis, B^{-1}, carries with it enough information to generate any current solution. The fundamental principle of the revised simplex method is that using only the current B^{-1} and the original problem coefficients is much more efficient than manipulating a complete tableau.

A further enhancement to this method is made by not storing B^{-1} explicitly but by generating it dynamically as the product of elementary vectors (which may be efficiently stored due to their sparse nature). These same principles will be adhered to in the following specialization of the (revised) simplex method to generalized networks. The mechanism to update our solution will be based on the dynamic generation of the equivalent to a revised simplex B^{-1}.

A. PRICEOUT

As expected, the pricing of non-basic variables is simply the specialization and simplification of simplex pricing. The reduced costs

31

of the non-basic variables are $c_N - c_B B^{-1} N$. Let $u = c_B B^{-1}$ (the dual solution). Then the reduced cost (rc) for a non-basic arc k is:

$$rc_k = c_k - uN^k.$$

N^k is a column of the network constraint matrix having at most two non-zero entries, i.e., +1 (in the row corresponding to the tail of arc k) and $-m_k$ (corresponding to the head of arc k).

The reduced cost of arc k oriented from i to j simplifies to

$$rc_k = c_k - u_i + m_k u_j,$$

or in the data array notation of Section II

$$rc_k = C(k) - U(i) + MUL(K) * U(j).$$

If arc k is a reflected arc (denoted by $CP(k) < 0$), then the sign of rc_k is reversed. (In reflecting a variable x_k, column N^k is multiplied by -1, the proper cost associated with that variable is, therefore, $-C(k)$ and the non-zero coefficients in column N^k change to -1 and $+m_k$).

The pricing simplification is one of the key computational advantages of network models. At most one multiplication, one addition, and one subtraction is required to price a non-basic variable. In the case of pure networks $m_k = 1$, the multiplication may be avoided.

As discussed in Section II, the components of the generalized network basis are triangular or nearly triangular. As a consequence, the value of the dual variables may be found by forward substitution in the system of equations $uB = c_B$. This is a simple matter for triangular elements

32

of the basis and once one variable is determined an equally simple matter for nearly triangular basis elements. A detailed discussion of the computation of dual variables will be presented with the pivot operations.

It is common practice, in didactic presentations of the simplex method, to select the variable with the most negative price as the incoming basis element. This approach requires pricing all the non-basic variables and can be quite time-consuming, especially in network models where the number of arcs far exceed the number of nodes. In fact, any variable which prices favorably will suffice as the incoming candidate and any pricing mechanism which guarantees discovery of all favorably pricing non-basic variables will operate correctly.

Several pricing strategies have been suggested. It has been shown that selection of the best (most negative price) from a limited number of favorably priced candidates is superior to simply choosing the first favorably priced arc [e.g., Ref. 18 for LP and Ref. 5 for GNP].

Some popular pricing strategies are: (a) price out the first g candidates where g is less than the number of non-basic variables; (b) maintain a candidate list as adopted by Glover, et al. [Ref. 5] and by Mulvey [Ref. 19]; and (c) maintain a candidate queue, a strategy which is employed by Bradley, Brown, and Graves [Ref. 6].

The candidate list strategy maintains a list containing at most $g1$ candidates. Each candidate is the most negative arc originating from a node. Every $g2$ pivots (or when no candidate prices favorably) the list is refreshed. If less than $g1$ favorable candidates are found, then $g1$ is set to the number of favorable candidates and $g2$ is halved (unless $g2 = 1$).

Mulvey presents computational results to aid in the selection of $g1$ and $g2$.

The candidate queue is a dynamic list of "interesting nodes" and "good arcs." The queue, as described by Bradley et al. [Ref. 6], employs two arrays, NSA which indicates the head node and ISA which indicates the tail node of candidate arcs. ISA(k) = 0 and NSA(k) = j indicates that node j is an "interesting" node. Arc entries are derived from a scan of all arcs incident to a node, from which the most negatively priced arc is entered. The candidate queue is initialized by placing demand or sink nodes (right-hand side < 0) on the queue as interesting nodes. (If no such nodes are found, the queue is initialized with any node.) A general scan of a head node will select the most favorably priced incident arc and place that arc on the candidate queue. Initially, favorable prices are most commonly caused by inherent infeasibility (i.e., paths do not yet exist between supply and demand nodes). For the first *nne* pivots, known as the opening gambit, the head and tail of each incoming basic arc may be entered on the queue as "interesting" nodes.

For each pivot, the incoming candidate is determined by pricing out *nne* candidates (including all of the arcs incoming to an interesting node). If no favorable arc is found within the *nne* candidates examined, another *nne* are examined, and so forth. If the queue is emptied during the first *nne* pivots, the queue is refreshed by a general scan of *ipg* (a page) of head nodes.

After the end of the opening gambit, the queue is refreshed after each cycle (pass) through the queue by scanning another page of the head nodes. As each arc is priced, the favorably priced candidates are

34

retained in the queue, interesting nodes are replaced by their most favorable incoming arc and unfavorable candidates are dropped from the queue. This pricing strategy finally concludes by pricing every non-basic arc (as every pricing strategy must) to ensure terminal conditions are met for optimality. Suggested values for *ipg*, *nme*, and *nms* are given by Bradley, Brown, and Graves [Ref. 6].

The results of any pricing strategy will be the identification of the head, tail, and arc index of a favorably priced incoming arc (or the determination that there are no favorably priced non-basic arcs).

B. RATIO TEST

At this point, an incoming arc has been identified and the algorithm must now select an appropriate outgoing member of the basis. Conceptually the ratio test identifies the first arc whose flow would reach one of its bounds as flow is incremented on the incoming arc. The ratio test thus identifies an arc whose flow would either increase to its upper bound or decrease to zero if the fullest advantage were taken of the favorable price of the incoming arc. The incoming arc may well be the constraining arc identified by the ratio test and therefore, the basis would not change; however, network flows would be updated with the flow on the incoming arc being "sent" to its opposite bound by reflection.

For an incoming arc k, the ratio test searches for the minimum of:

(a) cp_k

(b) $(x_{B_i} - cp_{B_i})/z_{ik}$ for $z_{ik} < 0$ $\forall\ x_{B_i} \epsilon x_B$

(c) x_{B_i}/z_{ik} for $z_{ik} > 0$ $\forall\ x_{B_i} \epsilon x_B$.

35

We know cp_j $j = 1, \ldots, n$ and the current basic solution x_B. Therefore, the ratio test can easily be performed if z_{ik} $i = 1, \ldots, m$ is known. To determine Z^k (the current incoming column), we must solve the system of equations $BZ^k = N^k$ (remembering to use $-N^k$ if arc k is reflected).

The obvious algebraic solution of this system is $Z^k = B^{-1}N^k$. We can generate N^k on demand since it is the coefficient column associated with arc k (oriented from node i to node j). We determine i and j from the H and T arrays and thereby determine the two non-zero entries of N^k as +1 (corresponding to i) and $-m_k$ or MUL(k) (corresponding to j).

We do not have an explicit representation of B^{-1} and thus must resort to indirect methods of solution. It is in this endeavor that the predecessor relationship proves to be extremely convenient. Using the predecessor array, it is possible to solve for Z^k directly by substitution in the triangular/nearly triangular system of equations $BZ^k = N^k$.

N^k is a non-basic column of the constraint matrix and therefore has at most two non-zero entires: +1 and $-m_k$. N^k can be expressed as:

$$N^k = e_i + (-m_k e_j)$$

where e_i and e_j are the ith and j^{th} unit vectors corresponding to the non-zero entries of N^k. Solving:

$$BQ^i = e_j$$
$$BQ^j = -m_k e_j,$$

will lead us to

$$BZ^k = N^k$$
$$= e_i + (-m_k e_j)$$
$$= BQ^i + BQ^j$$
$$= B(Q^i + Q^j)$$
$$\text{thus,} \quad Z^k = Q^i + Q^j.$$

The entering arc connects nodes i and j. Define the "i-backpath" as the path between node i and its root/cycle. Similarly, define the "j-backpath" as the path between node j and its root/cycle. It will be shown that Q^i and Q^j correspond conceptually to the i-backpath and j-backpath, respectively.

Remember, the basis is made up of disjoint components each of which is triangular or nearly triangular. Working with triangular components first, we find for:

$$BQ^i = e_i \quad \text{(or similarly, } -m_k e_j\text{)}$$

that elements of Q^i beyond the ith component must be zero. This can be seen by solving the triangular system. If e_i is of dimension h x 1 and i < h, then $e_i(h) = 0$, thus $Q^i(h) = 0$. If i < h - 1, then $e_i(h - 1) = 0$, and since $Q^i(h) = 0$, then $Q^i(h - 1) = 0$. The same reasoning applies to $Q^i(i + 1) = Q^i(i + 2) = \ldots = Q^i(h) = 0$. The ith element of Q^i must be +1 or $-m_k$. If there is an entry in the triangular matrix for the predecessor of i, which is in the ith column, it will produce a non-zero multiplication and thus must be offset by an entry in the P(i) row of Q^i, since e_i has only one non-zero entry. That "offsetting" element may also have a predecessor and thus another non-zero entry in Q^i will be appropriate. Q^i will therefore have non-zero entries in rows i, P(i), P(P(i)), etc., (i.e., the i-backpath). The preceding argument also applies to Q^j, which will have non-zero entries in only rows j, P(j), etc.

37

To see this and to determine the non-zero entries of Q, consider the triangular system in Figure 10.

Figure 10. Triangular Basis Component

The values of a and b are dependent on the original arc orientation (recorded by the sign bit of the predecessor array P; if $P(j) > 0$, then $a = +1$ and b is the negative of the multiplier value $(-m_k)$; if $P(j) < 0$, then $a = -m_k$ and $b = +1$, $\forall\ j = i,\ P(i),\ \ldots\ ,$ root. Solving:

$$
\begin{bmatrix}
a_0 & b_1 & & & & & \\
 & a_1 & & & & & \\
 & & \cdot & & & & \\
 & & & \cdot & & & \\
 & & & & b_{i-1} & & \\
 & & & & a_{i-1} & b_i & \\
 & & & & & a_i &
\end{bmatrix}
\begin{bmatrix}
q_0 \\ q_1 \\ \cdot \\ \cdot \\ \cdot \\ q_{i-1} \\ q_i
\end{bmatrix}
=
\begin{bmatrix}
0 \\ 0 \\ \cdot \\ \cdot \\ 0 \\ s
\end{bmatrix}
$$

where $s = 1$ or $-m_k$,

38

we obtain:

$$q_i = s/a_i$$

$$q_{i-1} = - (b_i q_i)/a_{i-1};$$

and in general

$$q_h = - (b_{h+1} q_{h+1})/a_h, \qquad 0 \leq h < i.$$

If we are dealing with a nearly triangular basis element, we again find that the only non-zero entries in Q^i occur in entries corresponding to the backpath between node i and the "root" cycle. Suppose the entering arc is incident to node i which is on the cycle shown in Figure 11.

Figure 11. Cycle

The backpath is thus i, P(i), P(P(i)) ... r where P(r) = i. The nearly triangular basis component is as follows.

$$
\begin{bmatrix}
a_r & & & & & & \\
 & \cdot & & & & & \\
 & & \cdot & & & & \\
 & & & \cdot & b_{P(P(P(i)))} & & \\
 & & & & a_{P(P(i))} & b_{P(P(i))} & \\
 & & & & & a_{P(i)} & b_{P(i)} \\
b_{P(r)} & & & & & & a_i
\end{bmatrix}
$$

39

Solving:

$$
\begin{bmatrix}
a_0 & b_1 & & & & \\
 & a_1 & & & & \\
 & & \ddots & & & \\
 & & & \ddots & & \\
 & & & & b_{i-1} & \\
 & & & a_{i-1} & b_i \\
b_0 & & & & & a_i
\end{bmatrix}
\begin{bmatrix}
q_0 \\ q_1 \\ \cdot \\ \cdot \\ \cdot \\ q_{i-1} \\ q_i
\end{bmatrix}
=
\begin{bmatrix}
0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ s
\end{bmatrix}
\qquad (2)
$$

where $s = 1$ or $-m_k$ and the a's and
b's are as before;

we obtain

$$q_i = (s/a_i)(1/f)$$

$$\text{and} \quad q_h = -\,(b_{h+1}q_{h+1})/a_h, \qquad 0 \le h < i,$$

where $f = 1 - \prod_{h=0}^{i} (b_h/a_h)$.

The solution procedure is described by Dantzig [Ref. 3]. The nearly
triangular system can be easily solved with one of the variables appearing
as a solution parameter thereby creating a triangular system. That
parameter can then be determined as the solution to a pair of two-variable
simultaneous equations. Kennington and Helgason [Ref. 2] give a detailed
derivation of the solution. A similar solution technique is illustrated
herein with the discussion of the dual update.

40

These equations are the same as in the triangular case except for the use, once, of the cycle factor f. The cycle factor (or loop factor, Dantzig) is the same for every node on the cycle and may be computed when the cycle is created. The array FAC is used to store the cycle factor so that it is available for immediate use. The update of FAC will be discussed in the pivot section. Again, the above solution procedure for Q^i applies equally well to determining Q^j.

We have approached this problem as if there are two distinct backpaths. However, the incoming arc may be such that the two backpaths converge. The node at which the backpaths meet is known as the "join." The two separate systems of equations must be solved up to the join using the iterative method discussed above. The separate solutions for q_{join} obtained from the converging backpaths should be added: $q_{join} = q_{join, i} + q_{join, j}$ and only one consolidated backpath need be computed after the join using the same interative formula:

$$q_h = \frac{-b_{h+1} q_{h+1}}{a_h} \ .$$

The q's obtained are the elements of Z^k we need to compute ratios. For use in the pivot, we store these values in the array PC. The array PC contains $B^{-1} N^k$, which, in revised simplex terms, is the incoming non-basic column updated by the current B^{-1}.

Obviously, detection of a join is extremely important in reducing the number of computations required to compute ratios. The depth array D is used to help detect a join in the following way. Let arc k, joining node i and node j, be the entering arc; let i refer to the node which is

41

currently being "visited" on the i-backpath and let j indicate the node currently "visited" on the j-backpath.

We begin with i and j as the origin and terminus of arc k. The depths of i and j are compared and the backpath of the one with the greater depth is iterated using the predecessor relationship. Ratios and the non-zero elements of Z^k are found one at a time as each node on the backpath is visited. If the j-backpath is deeper, we iterate it back until the depth is the same as the i-node. The current i- and j-nodes are checked to see if they are the same node. If they are, the current element of Z^k is computed as the join and the common backpath is iterated, continuing to search for the minimum ratio. If the two nodes are not the same, then the i-backpath and the j-backpath are both iterated once and a join is checked for again.

In summary, the mechanics of the ratio test are: if arc $k(i, j)$ is the entering arc, determine the minimum among:

1. cp_k

2. $(x_{B_i} - cp_{B_i})/z_{ik}$ for $z_{ik} < 0$ $\forall\ x_{B_i} \epsilon x_B$

3. x_{B_i}/z_{ik} for $z_{ik} > 0$ $\forall\ x_{B_i} \epsilon x_B$.

To do this:

(a) Determine cp_k as $CP(k)$.

(b) Determine whether the i-backpath or j-backpath is deeper by checking the depths of nodes i and j, the origin and terminus of the entering arc. Let i refer to the node currently being visited on the i-backpath and j refer to the current node on the j-backpath. Using the predecessor relationship, iterate up each backpath, performing the steps described below on each node encountered. When the current i and j

42

nodes are of the same depth, check for a join. If a join is found, add the values of "q" for that node obtained from each separate backpath to obtain q_{join}. If a join is not found, iterate up one level on both the i- and j-backpath and check for a join again.

(c) Determine the right-hand side of the triangular/nearly triangular system of equations (2). This is accomplished by setting $s = 1$ for the i-backpath (starting at the tail of the entering arc) and setting $s = -m_k$ for the j-backpath (starting at the head of the entering arc) in equation (2).

(d) For each node on the i- and j-backpath, determine the sign of the current node's predecessor $P(i)$. Use this to determine a and b in equation (2) as $P(i) > 0 \Rightarrow a = 1, b = -m_k; P(i) < 0 \Rightarrow a = -m_k, b = 1$ where m_k is the multiplier of the arc joining i and $P(i)$.

(e) Compute

$$q_i = s/a_i \text{ if i is the first node on the backpath;}$$

otherwise,

$$q_i = - (b_{i+1}q_{i+1})/a_i, \text{ where } P(i+1) = i.$$

(f) Determine if a join exists; if so, add the i- and j-backpath values of q to obtain q_{join}.

(g) Determine if the current node is the first node encountered on the root cycle $(D(i) = 0)$. If so, multiply q_i by $1/FAC(i)$.

(h) The q_i thus determined is the z_{ik} required to perform the ratio test

$$x_{B_i}/z_{ik} \qquad \text{if } z_{ik} > 0$$
$$(x_{B_i} - cp_{B_i})/z_{ik} \qquad \text{if } z_{ik} < 0$$

cp_{B_i} is found in $CP(IVAR(i))$.

43

(i) The ratio test may be terminated when a zero ratio is found (a de facto winner) or we completely iterate the i- and j-backpath performing ratio tests. The end of a backpath is signified when the first node with depth zero is encountered for the second time. (For a rooted tree, the root is its own predecessor and will be "encountered" twice in immediate succession by following the predecessor relationship. On the other hand, the nodes of a cycle all have depth zero and once the cycle is completely iterated, you will find a previously encountered node of depth zero.)

C. PIVOT

We have now determined the entering arc (priceout), the leaving arc (ratio test), and have generated the entering column and stored it in array PC. The basis representation and arc flows must now be updated.

If CP(k) is the minimum ratio, then the incoming arc has been determined to be more constraining than any element of the basis. Therefore, the incoming arc k is also the outgoing arc. The update is accomplished by reflecting arc k. The original basis remains unchanged and only the right-hand side need be updated. The general expression for the right-hand side was derived in equation (1)

$$x_B = B^{-1}b - B^{-1}Nx_N,$$

where: $B^{-1}b$ is the old right-hand side
x_N are the values of the non-basic variables.

The non-basic variables are all zero except, conceptually, the incoming variable which is set at its upper bound CP(k). Therefore, only the kth column of $B^{-1}N^k$ is required, which is precisely the information

44

now stored in PC as an artifact of the ratio test. The right-hand side (rhs) update is then:

$$rhs_i = old\ rhs_i - PC(i) * CP(k)$$
$$\longrightarrow X(i) <== X(i) - PC(i) * CP(k). \qquad (\forall i)$$

If the ratio test determines that an exchange of basis elements is required, a more involved update procedure takes place. The basis representation found in P and IT must be updated, as well as the dual variables U and the flow X. The FAC array must be maintained for nodes on cycles and the IVAR array contains the index of the basic arc associated with each node. As such, IVAR must also be updated during the pivot.

To conceptually understand the basis update procedure, return to the graphical representation of the basis. The root/cycle of each basis component is drawn at the top and the immediate successors of each node are depicted below that node (as with a genealogical, vice a biological, tree).

It is important to carefully distinguish between "types" of successors. The "successors" of a node will mean those nodes immediately encountered on all paths from a node except the backpath. "Preorder-successors" will indicate nodes determined by iteration of the preorder function IT. Arcs are drawn to indicate predecessor relationships (i.e., the arcs always point "up"), with the sign of the predecessor being used to record the "correct" orientation of the arc.
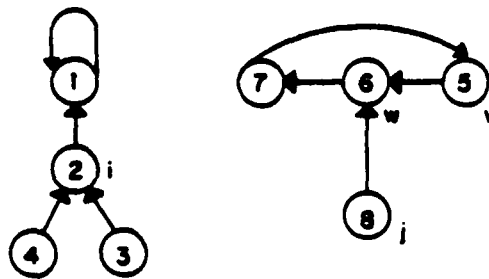
Figure 12. Pre-Pivot Generalized Network Basis

The simple basis pictured above will be used to illustrate the basis
update mechanism when the incoming arc is not the "winner" of the ratio
test. Let the incoming arc be designated as $k_E(i, j)$ and the outgoing
arc be $k_L(v, w)$. Assume the outgoing arc is on the j-backpath (if it
is not, reorient arc k so that it is). Similarly, let node v precede
node w on the j-backpath. Assume the entering arc is $k_E(2, 8)$ and the
leaving arc is $k_L(5, 6)$. The backpath from the destination node j of
the incoming arc to node v of the outgoing arc is called the "j-stem."
The j-stem in the example is along nodes 8, 6, 7, 5. Dropping arc $k_L(5, 6)$
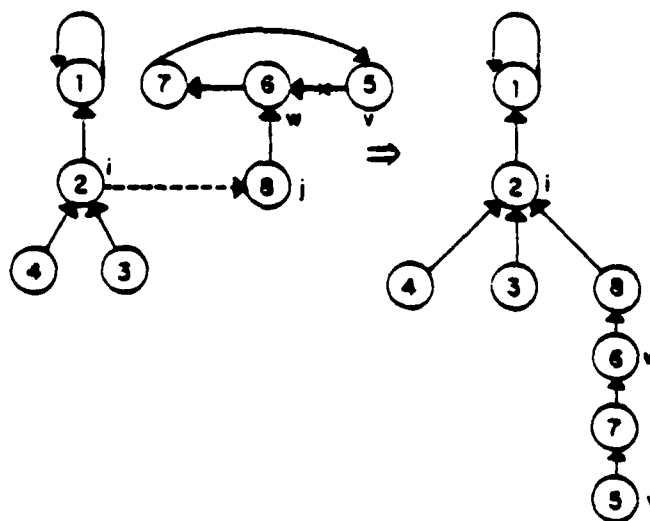and adding arc $k_E(2, 8)$ forms a new basis as shown in Figure 13.

Figure 13. Basis Update Example 1

46

Another example of a basis update is shown in Figure 14.
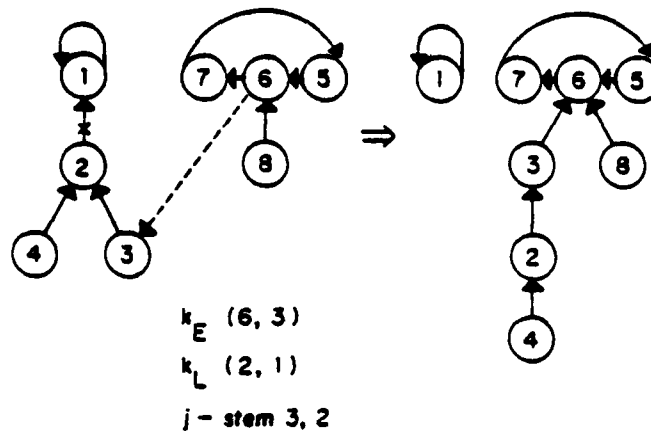

k_E (6, 3)

k_L (2, 1)

j — stem 3, 2

Figure 14. Basis Update Example 2

The update of any basis can be similarly accomplished by application of the following general rules:

1. Reverse the arc orientation of all arcs on the j-stem (which lies on the backpath containing the leaving arc).

2. Orient the entering arc such that it precedes the j-stem and has the same orientation as the redirected j-stem.

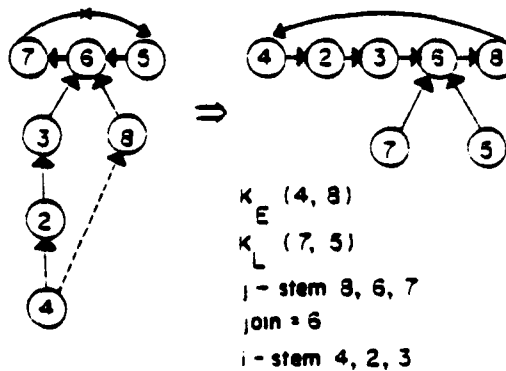The basis update shown in Figure 15 displays how a new cycle is created.

K_E (4, 8)

K_L (7, 5)

j — stem 8, 6, 7

join = 6

i — stem 4, 2, 3

Figure 15. Cycle Creation

47

As noted in the ratio test section, if the i- and j-backpaths merge, the node at which they merge is called the join. If the leaving arc lies beyond the join, then a new cycle is formed, as is the case in the above example. The new cycle becomes the root of its basis component and that component tree is "rehung" from the new cycle. The backpath from the source node (i) of the entering arc to the node whose predecessor is the join is known as the i-stem. In this example, $P(3) = 6 = $ join; the i-stem is therefore composed of nodes 4, 2, and 3. If node i lies on the j-backpath, the i-stem is null.

The graphical display of a basis update must be translated into an algebraic update of the data structures discussed in Section II. Each node affected by the basis update must be "visited" by the algorithm and the associated array values modified. Clearly, it is advantageous to visit a node only once. A one-pass update can be achieved as described below.

The pivot algorithm iterates up the j-stem node by node, updating the arrays: X, IT, P, U, IVAR, and D. If a join is encountered (the existence of a join is predetermined by the ratio test), the algorithm switches to the i-stem and iterates up the i-stem. Upon completion of the i-stem, the algorithm returns and completes iterating the j-stem. The stems are iterated by using the predecessor function P.

The depth D and the dual U must be updated for all of the stem nodes visited by the algorithm, as well as for their preorder-successors.

It is also convenient to modify IT as these nodes are visited. The preorder-successors of a node can be divided into two classes: the left preorder-successors and the right preorder-successors. The left preorder-

48

successors are found by iterating the preorder list IT from the current stem node to the stem node whose predecessor is the current stem node. The right preorder-successors of a node are any unvisited nodes found by further iteration of IT up to the end of the tree whose root is the current stem node. The end of this current tree is found by checking the depth of each preorder-successor of the stem node. If that depth is less than or equal to that of the current stem node, then the start of a new tree is identified.

The update of IT is relatively easy because IT changes only for nodes on the stem and their last left and right preorder-successors.

0. IT (last right preorder-successor of i) = j

1. j-stem update of IT

    a. IT (last left preorder-successor)   <== first right preorder-successor

    b. IT (last right preorder-successor) <== P (stem node)
       If there is no last right preorder-successor, then
       IT (stem node) <== P (stem node)

2. i-stem update of IT

    a. IT (last left preorder-successor)   <== first right preorder-successor

    b. IT (last right preorder-successor) <== node whose predecessor is current stem node.

Step 1b differs from 2b because the predecessor relationship for the j-stem must be reversed.

Some of the nodes encountered in the preorder may belong to the i-stem and its preorder-successors; these nodes must be avoided during the j-stem update, using the "preorder link." We process the part of the j-stem below the join and then process the i-stem (if there is a join).

49

The preorder-successor of the last i-stem node is recorded and this node is termed the "preorder link." Having finished processing the i-stem, we return to the j-stem (or common backpath above the join) and continue processing stem nodes and their preorder-successors. If an i-stem node is encountered, the remainder of the i-stem and their preorder-successors are avoided by immediately skipping to the preorder link. This subtle processing twist significantly reduces the time spent searching for the preorder-successors of the stem nodes.

The i-stem must be visited only if there is a join. If there is no join, then the j-stem is appended to the i-backpath by the entering arc and only the new preorder-successors of i (i.e., the j-stem and their preorder successors) need be updated.

The method we have established for the update of the preorder-successor relationship ensures that all the preorder-successors of a (cycle) node are encountered before the next (cycle) node. While traversing a stem, a stem node may be encountered with a right preorder-successor that has depth zero and is therefore on the root cycle (e.g., Figure 16). We therefore observe that given that the leaving arc is on the backpath of the j-stem, a cycle which is being broken will always be encountered as the right preorder-successor of a stem node.
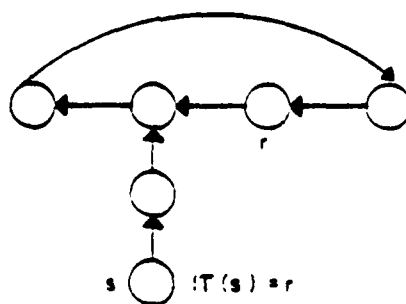

Figure 16. Preorder-Successor with Depth Zero

50

From the development of the computation of Z^k, it can be seen that the arc flows will only change on the backpaths. The array X represents the basic arc flows and must be changed for each arc on the j-stem. If the entering arc forms a cycle, flow changes occur on the i-stem as well as the j-stem. The update is obtained from equation (1):

$$x_B = B^{-1}b - B^{-1}Nx_N$$
$$\longrightarrow X(i) \Longleftarrow X(i) - PC(i) * RATIO. \qquad (\forall i)$$

Only one element of x_N (i.e., x_k) affects the update of the right-hand side. Clearly, if no cycle is created and the minimum ratio is zero, the right-hand side is not changed.

The dual variables, stored in the array U, are determined as

$$u = c_B B^{-1}$$
$$c_B = uB.$$

Therefore, $c_k = u_i - m_k u_j \qquad \forall k(i, j) \in$ basic arcs

$$u_i = c_k + m_k u_j$$
$$u_j = (c_k - u_i)/(-m_k).$$

Enforcing the above relationship will determine the dual variables as the stems are traversed in turn. If a cycle is not created, only the duals for the j-stem nodes and their preorder-successors need be computed. For a node i and associated basic arc k, the update depends on the orientation of arc k.

$$U(i) = C(k) + MUL(k) * U(P(i)) \qquad \text{for } k(i, P(i)); P(i) > 0$$

and

$$U(i) = (C(k) - U(-P(i)))/(-MUL(k)) \qquad \text{for } k(P(i), i); P(i) < 0.$$

51

The preorder traversal scheme (coupled with the reversal of the j-stem predecessor relationships) ensures that the dual of a predecessor node is known prior to its use in updating the dual of an immediate preorder-successor.

When a new cycle is created, the i- and j-backpaths terminate at a new root (the newly created cycle). The duals for this entire new basis component must be recomputed. To do this, the dual variable must be determined for one of the cycle nodes. Algebraically, the situation is analogous to the determination of Z^k during the ratio test. Once the dual of one cycle node is established, the dual variables for the remaining cycle nodes and their preorder-successors may be determined.

The ratio test gives ample warning that a new cycle will be formed. If the leaving arc $k_L(v, w)$ lies above the join, a new cycle will be created as:
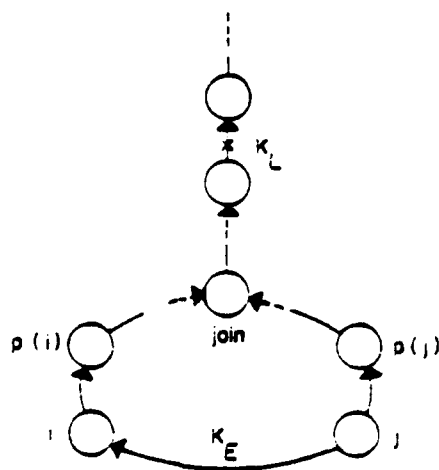
Figure 17. Leaving Arc Above the Join

The new cycle is formed (displaying the new predecessor function) as shown in Figure 18. The nearly triangular system used to compute the

52
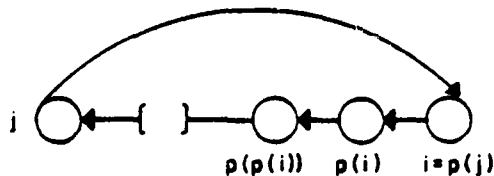
Figure 18. Predecessor Update for New Cycle

dual variables corresponding to this newly formed basis component is

$$uB = c =$$

$$
(u_0, \ldots, u_g)
\begin{bmatrix}
a_0 & b_1 & & & & & \\
 & a_1 & & & & & \\
 & & \cdot & & & & \\
 & & & \cdot & & & \\
 & & & & \cdot & & \\
 & & & & & b_{g-1} & \\
 & & & & & a_{g-1} & b_g \\
 & & & & & & a_g
\end{bmatrix}
= (c_0, \ldots, c_g)
$$

Dantzig [Ref. 3, p. 423] describes how to solve systems such as this,

> " . . . by treating one variable of the loop as the parameter and evaluating all others in terms of it as we proceed around the loop. Upon completion of this circuit a second expression for the parameter will result, and by equating the two expressions we may evaluate it numerically."

Assume the following cycle has been formed:

53
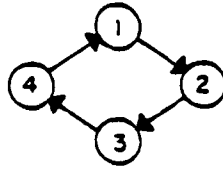
Figure 19. Four Node Cycle

producing this nearly triangular system:

$$(u_4, u_3, u_2, u_1) \begin{bmatrix} a_4 & b_3 & & \\ & a_3 & b_2 & \\ & & a_2 & b_1 \\ b_4 & & & a_1 \end{bmatrix} = (c_4, c_3, c_2, c_1) .$$

Performing the matrix multiplication and solving for u:

$$a_4 u_4 + b_4 u_1 = c_4 \quad \bigg| \quad u_4 = (c_4 - b_4 u_1)/a_4$$

$$b_3 u_4 + a_3 u_3 = c_3 \quad \bigg| \quad u_3 = (c_3 - b_3 u_4)/a_3$$

$$b_2 u_3 + a_2 u_2 = c_2 \quad \bigg| \quad u_2 = (c_2 - b_2 u_3)/a_2$$

$$b_1 u_2 + a_1 u_1 = c_1 \quad \bigg| \quad u_1 = (c_1 - b_1 u_2)/a_1$$

Choosing u_1 as the parameter, we obtain through forward substitution:

$$u_1 = \frac{1}{a_1} (c_1 - b_1 u_2)$$

$$= \frac{1}{a_1} (c_1 - b_1 (\frac{1}{a_2} (c_2 - b_2 u_3)))$$

$$= \frac{1}{a_1} (c_1 - \frac{b_1}{a_2} (c_2 - b_2 (\frac{1}{a_3} (c_3 - b_3 u_4))))$$

$$= \frac{1}{a_1} (c_1 - \frac{b_1}{a_2} (c_2 - \frac{b_2}{a_3} (c_3 - b_3 (\frac{1}{a_4} (c_4 - b_4 u_1)))))$$

So,

$$u_1 = \frac{1}{a_1} \left[c_1 - \frac{b_1}{a_2} (c_2 - \frac{b_2}{a_3} (c_3 - \frac{b_3}{a_4} (c_4 - b_4 u_1))) \right]$$

54

$$= \frac{1}{a_1} \left[c_1 - \frac{b_1}{a_2} \left(c_2 - \frac{b_2}{a_3} \left(c_3 - \frac{b_3 c_4}{a_4} - \frac{b_3 b_4 u_1}{a_4} \right) \right) \right]$$

$$= \frac{1}{a_1} \left[c_1 - \frac{b_1}{a_2} \left(c_2 - \frac{b_2 c_3}{a_3} + \frac{b_2 b_3 c_4}{a_3 a_4} - \frac{b_2 b_3 b_4}{a_3 a_4} u_1 \right) \right]$$

$$= \frac{1}{a_1} \left[c_1 - \frac{b_1 c_2}{a_2} + \frac{b_1 b_2 c_3}{a_2 a_3} - \frac{b_1 b_2 b_3 c_4}{a_2 a_3 a_4} + \frac{b_1 b_2 b_3 b_4}{a_2 a_3 a_4} u_1 \right]$$

$$u_1 = \frac{c_1}{a_1} - \frac{b_1 c_2}{a_1 a_2} + \frac{b_1 b_2 c_3}{a_1 a_2 a_3} - \frac{b_1 b_2 b_3 c_4}{a_1 a_2 a_3 a_4} + \frac{b_1 b_2 b_3 b_4}{a_1 a_2 a_3 a_4} u_1$$

$$\left(1 - \frac{b_1 b_2 b_3 b_4}{a_1 a_2 a_3 a_4} \right) u_1 = \frac{c_1}{a_1} - \frac{b_1 c_2}{a_1 a_2} + \frac{b_1 b_2 c_3}{a_1 a_2 a_3} - \frac{b_1 b_2 b_3 c_4}{a_1 a_2 a_3 a_4}$$

$$u_1 = \frac{\frac{1}{a_1} \left(c_1 - \frac{b_1}{a_2} \left(c_2 - \frac{b_2}{a_3} \left(c_3 - \frac{b_3}{a_4} c_4 \right) \right) \right)}{1 - \frac{b_1}{a_1} \cdot \frac{b_2}{a_2} \cdot \frac{b_3}{a_3} \cdot \frac{b_4}{a_4}} .$$

The denominator is precisely the cycle factor f previously defined as:

$$1 - \prod_{d=1}^{4} - \frac{b_d}{a_d}$$

The sign alternation implied by the $- \frac{b}{a}$ terms can be observed in the derivation of u_1. Thus

$$u_1 = \frac{c_1 - b_1 \left\{ \dfrac{c_2 - b_2 \left(\dfrac{c_3 - b_3 \left(\dfrac{c_4}{a_4} \right)}{a_3} \right)}{a_2} \right\}}{a_1} \times \frac{1}{f} .$$

Letting $u_4' = \frac{c_4}{a_4}$, $\qquad\qquad u_3' = \frac{c_3 - b_3 u_4'}{a_3}$

55

$$u_2' = \frac{c_2 - b_2 u_3'}{a_2} , \qquad u_1' = \frac{c_1 - b_1 u_2'}{a_1}$$

we obtain

$$u_1 = \frac{c_1 - b_1 \left\{ \dfrac{c_2 - b_2 \left(\dfrac{c_3 - b_3 \ u_4'}{a_3} \right)}{a_2} \right\}}{a_1} \times \frac{1}{f}$$

$$= \frac{c_1 - b_1 \left(\dfrac{c_2 - b_2 u_3'}{a_2} \right)}{a_1} \times \frac{1}{f}$$

$$= \frac{c_1 - b_1 (u_2')}{a_1} \times \frac{1}{f}$$

$$u_1 = u_1' \times \frac{1}{f} .$$

Looking at a similar triangular system

$$(u_4'', u_3'', u_2'', u_1'') \begin{vmatrix} a_4 & b_3 & & \\ & a_3 & b_2 & \\ & & a_2 & b_1 \\ & & & a_1 \end{vmatrix} = (c_4, c_3, c_2, c_1) ,$$

we find that it is easily solved as

$$u_4'' = \frac{c_4}{a_4} , \qquad u_3'' = \frac{c_3 - b_3 u_4''}{a_3}$$

$$u_2'' = \frac{c_2 - b_2 u_3''}{a_2} , \qquad u_1'' = \frac{c_1 - b_1 u_2''}{a_1} .$$

The solutions for the u" are the same expressions assumed for u'.

The values for u' may therefore be obtained as the solution of a similar

56

triangular system; the dual variables associated with a cycle (and hence, a nearly triangular system) may be obtained from u'.

To obtain the key dual value u_1, we must therefore determine u_4', u_3', u_2', u_1', and finally u_1. In general, if we have a cycle composed of nodes 1, ... , g, the determination of the dual of one of these nodes, u_1, is found by modified forward substitution as illustrated above. With:

$$u_1 = u_1' \times (1/f)$$

$$\text{where} \quad f = 1 - \prod_{r=1}^{g} - (b_r/a_r)$$

$$\text{and} \quad u_g' = c_g/a_g$$

$$u_r' = (c_r - b_r u_{r+1}')/a_r \quad \forall r = g-1, \ldots , 1;$$

c_g is the cost of the arc associated with node g.

Theoretically, we may start at any node on the cycle, iterate around the cycle computing u' and accumulating the terms of $\prod - \frac{b}{a}$; in practice we choose to start at the terminus of the entering arc (i.e., the j-node). After visiting all the cycle nodes, the cycle factor f is computed and stored in FAC for each node on the cycle and the key dual variable is determined.

The dual of i is the keystone. From the dual of i, we can compute the dual of j and those of the entire j-stem, as well as those of the i-stem.

As stated, we want to start at the j-node and proceed around ending at the i-node. This traversal cannot be conveniently supported by either the pre-pivot or post-pivot predecessor relationships. It is necessary to follow the pre-pivot j-stem predecessor relationship and the reverse of the i-stem predecessor relationship.

57

To perform this intricate maneuver, the cycle factor and key dual value of a newly created cycle is computed immediately after the ratio test and before the basis update. First, the pre-pivot j-stem predecessor relationship is iterated from the j-node to the join, computing a partial product of the cycle factor. The i-stem is then iterated, performing no computations other than storing the reverse predecessor relationship. (A convenient place to do this is provided in the array U, which now contains obsolete dual values and must be recomputed subsequently.) Once at the end of the i-stem, the reverse i-stem path may be followed to complete the computation of the cycle factor and the dual value of the i-node. After this newly formed cycle has been traversed, the pivotal update can be accomplished, following both the i- and j-stem and iterating IT to update preorder-successors.

The update of the depth array is straightforward. If a new cycle is formed, the nodes on the new cycle are assigned depths of zero. The depths of the left and right preorder-successors of the i- and j-nodes are updated as: $D(s) = D(P(s)) + 1$ (after the update of the j-stem pred-ecessor relationship). If no new cycle is formed, then $D(j) = D(i) + 1$ and the preorder-successors of the j-stem nodes are updated accordingly. Again, the preorder ensures that $D(P(s))$ is available when it is time to compute $D(s)$.

IVAR contains the index of the basic arc associated with each node. It represents the arc which connects a node with its predecessor. If a node has no predecessor (i.e., a single root node), then IVAR is set to $n + 1$ (the number of arcs + 1). During the pivot, IVAR(j) is set to the index of the incoming arc. As the predecessor relationship of the j-stem

58

is reversed, IVAR of each j-stem node must be updated accordingly. The reversal of the j-stem arcs corresponds to the reordering of the rows and columns of the current basis to obtain a triangular/nearly triangular basis. IVAR must also exhibit this reordering of the basis elements.

To summarize the pivot steps for $k_E(i, j)$ and $k_L(v, w)$:

(0) Determine if a new cycle is to be formed (predicted by the ratio test: the leaving arc lies above the join). If a new cycle is formed, compute the cycle factor and the dual of i, storing the results in FAC and U(i) respectively.

(1) Bring $k_E(i, j)$ into the basis by setting P(j) <== i (with appropriate sign indicating arc orientation); IVAR(j) <== entering arc, and set IT (last of the right preorder-successors of i) <== j.

(2) Iterate the j-stem, which includes the join. As the j-stem is iterated, the predecessor relationship is reversed, with P and IVAR updated accordingly. If the j-stem is on a cycle, the depths, D, of the stem nodes are set to zero. Otherwise, the depth of j is D(i) + 1 and the preorder-successors of j are assigned D(s) = D(P(s)) + 1. The dual of j is computed as:

$$U(j) = C(k) + MUL(k) * U(P(i)) \qquad \text{if } P(j) > 0 \text{ or}$$

$$U(j) = (C(k) - U(- P(i)))/(- MUL(k)) \qquad \text{if } P(j) < 0;$$

where k is the entering arc.

The duals of the rest of the j-stem and their preorder-successors are computed similarly.

If the minimum ratio is non-zero, X is also updated at this point, using equation (1). Each j-stem node is visited in turn by using the predecessor relationship (updating this relationship as it is

59

traversed) and the preorder-successors of each j-stem node are visited using the preorder relationship, updating IT appropriately.

(3) The i-stem is traversed if a new cycle is formed. During this traversal, the dual variables are updated (replacing the reverse predecessor path temporarily stored here) as well as updating the values of X. As each i-stem node is visited, the left and right preorder-successors of the i-stem node are visited in preorder, updating D, IT, and U. IVAR entries for the i-stem and its preorder-successors remain unchanged.

D. FURTHER CONSIDERATIONS

Priceout, ratio test, and pivot provide the mechanism to move from one basic solution to a "better" basic solution. Primal Simplex methods must be designed to seek feasibility as well as optimality. The two most common methods of achieving an optimal, basic, feasible solution are the Big-M method and the two-phase simplex method. The Big-M method uses artificial arcs with very high costs to satisfy feasibility initially and the solution proceeds from this costly artificial start. In essence, the Big-M costs dominate the model costs ensuring that an optimal solution will have minimal flow on artificial arcs.

The two-phase method first solves a related problem with the same set of constraints whose objective is to minimize the sum of the flow on artificial arcs. If an optimal solution is found to the phase 1 problem with an objective function value of zero, then all arcs pricing non-zero are eliminated from further consideration in phase 2. The phase 2 objective function, which is the original objective function of the model, is introduced and the optimal solution is sought.

60

An all artificial start proceeds by assigning IT(i) = i, P(i) = i,

D(i) = 0 for i = 1, ... , m nodes, and IVAR (i) = n + 1 for

i = 1, ... , m. Each node is conceptually assigned an artificial arc

which satisfies the conservation of flow requirement at that node. The

initial basis for m nodes is graphically depicted as:

Figure 20. All Artificial Start

Each of these artificial arcs is conceptually assigned a multiplier

of 1. Thus, the dual associated with each node is assigned a value equal

to the cost of the artificial arc. The flow on each of the artificial

arcs is set to satisfy conservation of flow requirements.

Demand nodes exhibit a negative external flow; to preserve non-

negativity of the right-hand side, the following adjustments must be

made:

$$X(i) \Leftarrow - X(i) \quad \text{Right-hand side}$$

$$P(i) \Leftarrow - P(i) \quad \text{Arc orientation}$$

$$U(i) \Leftarrow - U(i) \quad \text{Dual values}$$

The Big-M method assigns a very large (Big-M) cost to each of the

artificial arcs described above. These costs are represented by the ini-

tial dual values assigned to each node. If Big-M is chosen sufficiently

large, an optimal minimum cost solution will not include these very

costly artificial arcs with non-zero flow in the basis. The algorithm

will replace these artificial arcs with the less costly "real" arcs of

61

the problem. The choice of the Big-M cost is important. It must be
large enough to drive the artificial arcs out of the basis, but not so
large as to cause numerical (floating point) truncation errors. An
initial choice of twice the largest arc cost has proven to be effective
in most cases.

An alternative to the Big-M method is the two-phase simplex approach.
A temporary cost of 1 is assigned to each artificial arc and a temporary
cost of 0 is assigned to the remaining arcs. The algorithm solves this
modified problem to attain an initial feasible solution. A minimum-cost
feasible solution will have non-zero flow only on arcs with cost zero.
If an artificial remains in the basis with non-zero flow at the end of
phase 1, the problem is infeasible. Once phase 1 is complete, all arcs
with non-zero reduced costs are eliminated from further consideration,
the correct costs are restored to the arcs and the phase 2 problem is
solved, which is the network flow problem of interest restricted to admit
only feasible basic solutions.

It is often found that many basic arcs have no successors. This is
especially true when dealing with problems that have numerous sinks. A
basis aggregation enhancement to primal network algorithms has been
proposed by Bradley, Brown, and Graves [Ref. 6], and has been adopted in
this implementation of the GENNET algorithm. The pivotal update of the
various arrays represents much of the work performed by this algorithm.
The dual variables, U, and the depth, D, of leaf nodes (i.e., nodes with
no successors) are uniquely determined by the knowledge of the leaf
node's predecessor and the arc which connects the leaf node to its
predecessor. In consequence, these values may be easily generated

62

when required and need not be updated by every pivot. Additionally, the preorder-successor, IT, of this node need no longer be maintained. While it is certainly true that any value may be generated rather than updated, the key is to choose values which may be easily restored.

Brown and McBride [Ref. 7] employ an array XM to facilitate the "aggregation" of nodes. An "aggregated node" is a node with no successors whose depth and dual variable are not explicitly maintained and must be generated when required. The entries in XM indicate the number of successors of each node which are not currently explicitly maintained. If $XM(P(r)) = 0$, then node r is not an aggregated node.

If an aggregated node is encountered during priceout, its dual is generated based on the dual of its predecessor and the direction, multiplier, and cost of the connecting basic arc. Similarly, the depth of an aggregated node r is generated as $D(r) = D(P(r)) + 1$. If an aggregated node, r, is the origin or terminus of the entering arc, it is "disaggregated" by restoring its dual and depth, and decrementing $XM(P(r))$, the number of aggregated successors of $P(r)$. IT, for the disaggregated node r, is updated by: storing the preorder-successor of $P(r)$, TEMP <== IT $(P(r))$; making r the preorder-successor of $P(r)$, IT $(P(r))$ <== r; and making the previous preorder-successor of $P(r)$ the preorder-successor of r, IT (r) <== TEMP.

The outgoing arc may isolate either its head or tail node with no preorder-successors and either node may then be aggregated.

Brown and McBride [Ref. 7] and Bradley et al. [Ref. 6] report signifi-cant computational savings using this aggregated node concept, especially when dealing with real-world problems involving few sources and many sinks.

63

IV. MICROCOMPUTER IMPLEMENTATION

An important emphasis of mathematical programming has long been on
the development of computer codes which will solve large-scale problems
efficiently. The network model, as a specialization of the linear
program, was one of the early breakthroughs in this area. As demonstrated,
the unique character of the network model allows for efficient data
storage techniques and greatly simplifies the computational requirements
of the simplex method. The obvious result is that much larger problems
can be solved using a network formulation, rather than a linear program,
on the same computer; solution times and hence computational cost are
reduced. As computer memory becomes cheaper and mainframe computer
central processing units (CPU's) become faster, the problem size and speed
emphasis of mathematical programming will perhaps diminish. Indeed,
problems of more than one million variables have already been solved
using a model introduced by Geoffrion and Graves [Ref. 20].

Advances in computer technology have permeated almost every aspect of
life in the United States. Exhaustive arithmetic calculations can be
made by anyone with a $10.00 calculator. For less than $100.00, a
programmable calculator/computer may be obtained. These radical develop-
ments have been made possible by advances in solid state electronics and
the advent of the microprocessor. A microprocessor is a collection of
many thousand electronic logical gates created as microscopic circuits on
small pieces of silicon, known as chips [Ref. 21]. Microprocessors are
perhaps best known as the controlling device in home and arcade video

64

games. They also serve more practical purposes as industrial process controllers and as the CPU of the microcomputer.

The most popular microprocessors used in today's small computers are the 8080 manufactured by Intel, the Z80 made by Zilog, the 6800 manufactured by Motorola, and the 6502 developed by MOS Technology [Ref. 22]. Each of these are 8-bit microprocessors, indicating that the basic memory unit is 8 bits wide (a byte). These microprocessors have 16 address lines. Each of these lines may be in one of two states: high or low (on or off; 0 or 1). As such, these microprocessors may address 2^{16}, or 65,536, separate memory locations. A microcomputer with 65,536 addressable memory locations is known, somewhat inaccurately, as a 64K (K stands for kilobyte) microprocessor.

The addressable memory in a microprocessor is not all available to the user. Most microcomputer systems reserve some of that memory for the use of the monitor, various languages, and operating systems. The Apple II Plus microcomputer, in its 64K configuration running the Apple UCSD (University of California at San Diego) Pascal Operating System, has a maximum space of 39,900 bytes for a user program and variables [Ref. 23].

In the last two years, 16- and 32-bit microprocessors have emerged, as well as some microprocessors with 20 or more address lines (making them capable of addressing more than one million memory locations) [Ref. 21]. Certainly, size distinctions between microcomputers and minicomputers have diminished and microcomputers may soon be challenging mainframes in many applications.

E. M. L. Beale, in his 1980 Blackett Memorial Lecture on the relationship between operations research and computers [Ref. 24], recognizes the

emergence of microcomputers and their potential as a powerful tool. There has been a plethora of software developed for microcomputers for use in the statistical and trend analysis areas of operations research. There is, however, a dearth of mathematical programming software available to the microcomputer user and surprisingly little published research in this area.

The potential of small computers has not escaped all mathematical programmers and researchers. In 1979, a feasibility study [Ref. 25] was conducted to explore the possibility of implementing mathematical programming algorithms on minicomputers. The study implemented two shortest path algorithms on two different minicomputers. It concluded that minicomputers, although slower than mainframes, were acceptable vehicles for shortest path algorithms. The study also hypothesized that modern minimum cost flow network algorithms may also be excellent candidates for minicomputer implementation.

Also in 1979, F. P. Wyman [Ref. 26] reported the implementation of an out-of-kilter algorithm for the solution of pure minimum cost network flow models. Additional implementations of "textbook" style PERT, CPM, and SIMPLEX codes have been reported in hobby computer journals [e.g., Ref. 27].

R. H. Duff [Refs. 28, 29] reported the development of a comprehensive microcomputer-based network optimization package. Duff's microcomputer package is capable of solving pure minimum cost network flow problems, elastic network problems (in which flow conservation may be violated for a "price"), and nonlinear network problems. The algorithms chosen were state-of-the-art optimization algorithms designed to minimize storage

66

requirements and execution time. Algorithms with these characteristics
are obvious candidates for implementation on microcomputers where memory
is a limited commodity and CPU processing is in the one to three megahertz
(MHz) range.

To complete Duff's network optimization package and make it the
most versatile network optimization package available on anything but a
mainframe computer (and perhaps not even there), all that is needed is an
efficient generalized network algorithm.

The generalized network algorithms presented by Jensen and Barnes
[Ref. 1] and Kennington and Helgason [Ref 2] are thought to be too
cumbersome and inefficient for the "lean" world of microcomputing. Elam
et al. [Ref. 30] report the development of a fast and efficient general-
ized network code, but descriptions of that code are spread throughout
the literature [Refs. 5, 30, 31, 32] and provide no clear explanation of
the basis update mechanism. As such, Brown and McBride's [Ref. 7] GENNET
Algorithm, as described in detail in the previous sections, has been
chosen as the most suitable addition to the microcomputer network optimiza-
tion package. For the sake of brevity, the microcomputer-based network
optimization package herein described will be referred to as Micronet.

The host computer for Micronet is an Apple II Plus with 64K of
memory. This is certainly not the most powerful microcomputer (with some
competitors featuring addressing capabilities of 16 megabytes and running
at speeds of 8 MHz), but it is one of the most popular with a population
of more than 400,000 units [Ref. 33].

It is important to recognize that the host machine is not at issue
here. This project was begun in 1979 and the Apple II was representative

67

of the technology emerging at that time. The Apple II has continued to be an extremely popular and representative 8-bit microcomputer. The purpose of Micronet is to explore the capabilities and suitability of a microcomputer as a tool of mathematical programming. Software design techniques used in this optimization package are applicable to other microcomputer systems as well. Larger and faster microcomputers will only enhance the capabilities of software systems such as Micronet, making mathematical programming on a microcomputer not only a viable, but an attractive option.

In this spirit, the Apple II Plus continues to be used as the host computer for the further development of microcomputer-based network codes. The programming language used is UCSD PASCAL. This language is fast becoming one of the most popular microcomputer-based high level computer languages. Certainly BASIC ranks as the most popular, however, versions of BASIC, both interpreted and compiled, lack standardization. UCSD PASCAL, while all implementations are not identical, provides a more standardized vehicle for the development of microcomputer programs. Duff [Ref. 28] discusses the choice of PASCAL in great detail; his reasoning remains sound and will not be repeated.

It is appropriate, however, to discuss some limitations of the host microcomputer and the Apple implementation of UCSD PASCAL; these limitations profoundly impact the design of any microcomputer mathematical programming code.

Pure network problems are characterized by columns of the constraint matrix which have non-zero entries of only +1 or -1. This unique characteristic eliminates the need for floating point arithmetic and

68

eliminates the requirements for multiplication and division. As such, mathematical precision is not an issue in pure network codes. Generalized networks possess a similar structure with each column of the constraint matrix having a +1 and a multiplier $-m_k$, where $-m_k$ may be any floating point number. The optimal flows in a generalized network are therefore not necessarily integer, thus floating point, as opposed to integer, arithmetic is required.

The Apple II Plus implementation of UCSD PASCAL provides for a 32-bit representation of floating point numbers [Ref. 34]. A representation using 24 bits for the mantissa of a floating point number and 8 bits for the exponent provides six or seven significant figures of precision with a dynamic range of 10^{-38} to 10^{+38} [Ref. 35]. This precision is roughly equivalent to IBM 360/370 single precision. Although this is a limitation of Apple PASCAL, it is not considered to be severely debilitating as applied to generalized networks. The purpose of the multiplier $(-m_k)$ is to model the transformation of units of flow or change the commodity amount as it flows through an arc [e.g., Ref. 5]. These purposes of arc multipliers can usually be accommodated with two or three significant figures. As such, an arithmetic precision level of 10^{-4} has been chosen for use in this implementation of GENNET.

UCSD PASCAL requires static dimensioning of arrays. Programming techniques which dynamically allocate memory at execution time based on problem size and type cannot be used to provide better memory management. This is not a critical economic issue on a dedicated microcomputer, as it is on a mainframe computer, but it does unnecessarily limit problem size and flexibility.

69

Source statements are compiled into a standard UCSD pseudo-code (p-code) which is then interpreted at execution time. This system allows for a standard p-code across all machines and only requires a machine-dependent run time interpreter. Execution speed is therefore not as fast as would be expected from a truly compiled language, but not as slow as an interpreted BASIC. The most disturbing feature of Apple II PASCAL, when used as a vehicle for mathematical programming, has proven to be compilation speed. The nominal compilation rate for Apple II UCSD PASCAL is two hundred source statements per minute. This means that a large program might take 20 to 30 minutes to compile and that requirement can be quite annoying when a program is in the developmental stage.

Other less significant limitations, which are included for the sake of completeness, are the editable file size and maximum procedure size. The operating system editor accommodates files of sizes up to 40 blocks [Ref. 23]. A block represents 500 bytes of information stored on an Apple II floppy diskette. To compose and edit large programs, several text files must be created and connected together. The maximum size of a compiled procedure or function is 1,200 bytes (plus any local variables). This requires programs to be broken into smaller "pieces" than may be logically convenient. These latter limitations are obviously not major faults, but are interesting "quirks," which must be contended with.

Once the Apple PASCAL operating system is resident in memory, less than 40K bytes are available for user programs and variables [Ref. 23]. As such, any mathematical programmer must be extremely conscious of the classical space-speed tradeoffs. Certainly the efficient data storage

techniques and basis update mechanisms exhibited by the GENNET [Ref. 7]
algorithm are mandated.

The UCSD PASCAL system allows "segmentation" or "overlaying" of
program components for large programs. When a portion of code is no
longer required, it may be "swapped" out of memory and replaced by another
segment of code. To maximize the amount of memory available for problem
representation, it is desirable to have as little memory occupied by
program code as possible. However, as more program overlaying is done,
more disk accesses are required by code swapping. Disk accesses are slow
and the programmer is again presented with the ubiquitous space-speed
tradeoff.

Micronet consists of three major components: a master program or
driver, an editor, and a solution module. These components all have
access to a system library containing often-used functions. An online
use manual, or "help" feature, is eventually envisioned for Micronet but
has not been completely implemented as yet. The solution module has
several submodules for solving the various types of network flow problems:
pure minimum cost (GNET), nonlinear (NLPNET), elastic with fixed charges
(ENET), and generalized networks (GENNET). A conceptual view of Micronet
is presented in Figure 21.

Duff [Ref. 28] gives a detailed description of organization, use,
and the characteristics of each component of Micronet (with the exception
of GENNET). The driver, editor, and solution module have been appropri-
ately modified/amended to allow Micronet to solve generalized networks.
A brief description of Micronet will be given here for continuity.

71

MICRONET ORGANIZATION
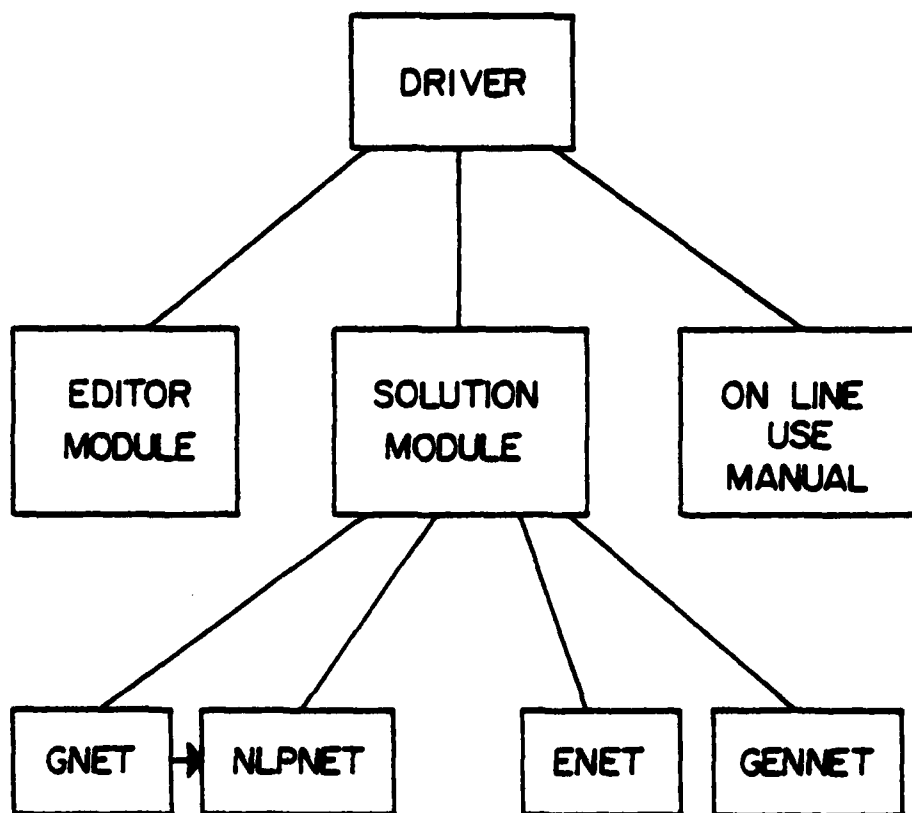
Figure 21. Micronet Organization

The master program coordinates the operation of the optimization package by passing control to the editor for data manipulation, to the solution module to solve a network model, or to the PASCAL operating system to exit Micronet. Upon completion of work by the solution or editor module, control is passed back to the driver program. The driver is chained to the solution module and editor so that when control is passed between one of the three main package components, only that component is in memory.

72

The purpose of the editor is to create, alter, transfer, and browse data files. Data files are essentially network problem description files consisting of a header record and records for nodes and arcs. Nodes may be specifically assigned attributes (e.g., flow bounds, external flow requirements), or the problem may be described using artificial arcs representing the external flow characteristics of sources and sinks. Problem files may be created from the keyboard or read from a text file in SHARE format [e.g., Ref. 36]. Examples of the three types of records follow:

Header Record

Problem Name:	GENTRANS
Problem Type:	GENERALIZED
Number of Nodes:	15
Number of Arcs:	30
Date Created:	1 Oct 81
Date Last Updated:	15 Nov 81

Arc Record (A Generalized Arc)

Arc Name:	A
Source Node:	1
Destination Node:	5
Lowerbound:	2.0
Upperbound:	100.0
Initial Flow:	0
Unit Cost:	10.50
Multiplier:	1.18

Node Record

Node Name:	Phoenix
Node Number:	5
Node Kind:	Demand
Net Flow:	3.0
Lower Range:	0
Upper Range:	0
Lower Penalty:	0
Upper Penalty:	0

The ranges and penalties indicated on the node record are applicable to elastic programming and are discussed by Duff [Ref. 28].

As the information is entered into the problem record, it is screened for consistency. Once created, a file can be altered, browsed, removed, or transferred by choosing the appropriate editor option.

If the solution module is chosen from the command level, the user is prompted to insert a disk, containing the problem file, into one of the Apple II's disk drives. Micronet then displays a catalog of the disk and the user selects the problem to be solved. At this point, the header record is read and based on the problem type, the appropriate solution submodule is automatically invoked. Figure 22 displays the solution path selection logic.

If the problem type is a generalized network, the GENNET solution module is chosen. This module implements the previously described GENNET Algorithm in a segmented PASCAL Program. The program is split into four segments as shown below:

Main Program

Input ⟶ Initialization ⟶ Solution ⟶ Report

74

The segmentation of the program represents this space-speed trade-off decision that must be made. The main program calls each of the segments in turn, "swapping" out of memory the previously used segment. The main program declares all global variables and thereby provides for communication between segments. These variables remain in memory throughout the use of the solution module. Variables local to the individual segments occupy memory only when that segment is active.

The input segment first presents the user with a menu requesting a destination for the results (disk file, printer, serial interface, or terminal), whether a pivot-by-pivot trace is required, and whether a listing of the problem arcs and nodes is desired. The input segment establishes the maximum problem size at 100 nodes and 500 arcs. The input file is read, and problem variables are initialized. Lower bounds are translated out and the resulting cost of the lower bound modification is recorded.

During this phase of the solution procedure, the user is required to interact with the computer by answering menu-driven questions, inserting a problem disk, and choosing a problem. Great care has been exercised to ensure that an incorrect user response will illicit a helpful (or admonishing) message from the program rather than a premature program termination. Having read the problem, the header record, arc records, and node records have been converted to the streamlined data structures of GENNET.

During the initialization segment, a Big-M start is set up, with the initial basis appearing as a forest of one-trees. Big-M is determined by doubling the largest arc cost of the problem. The arcs are sorted by head node and stored in that order in the array T. T(k) contains the

75
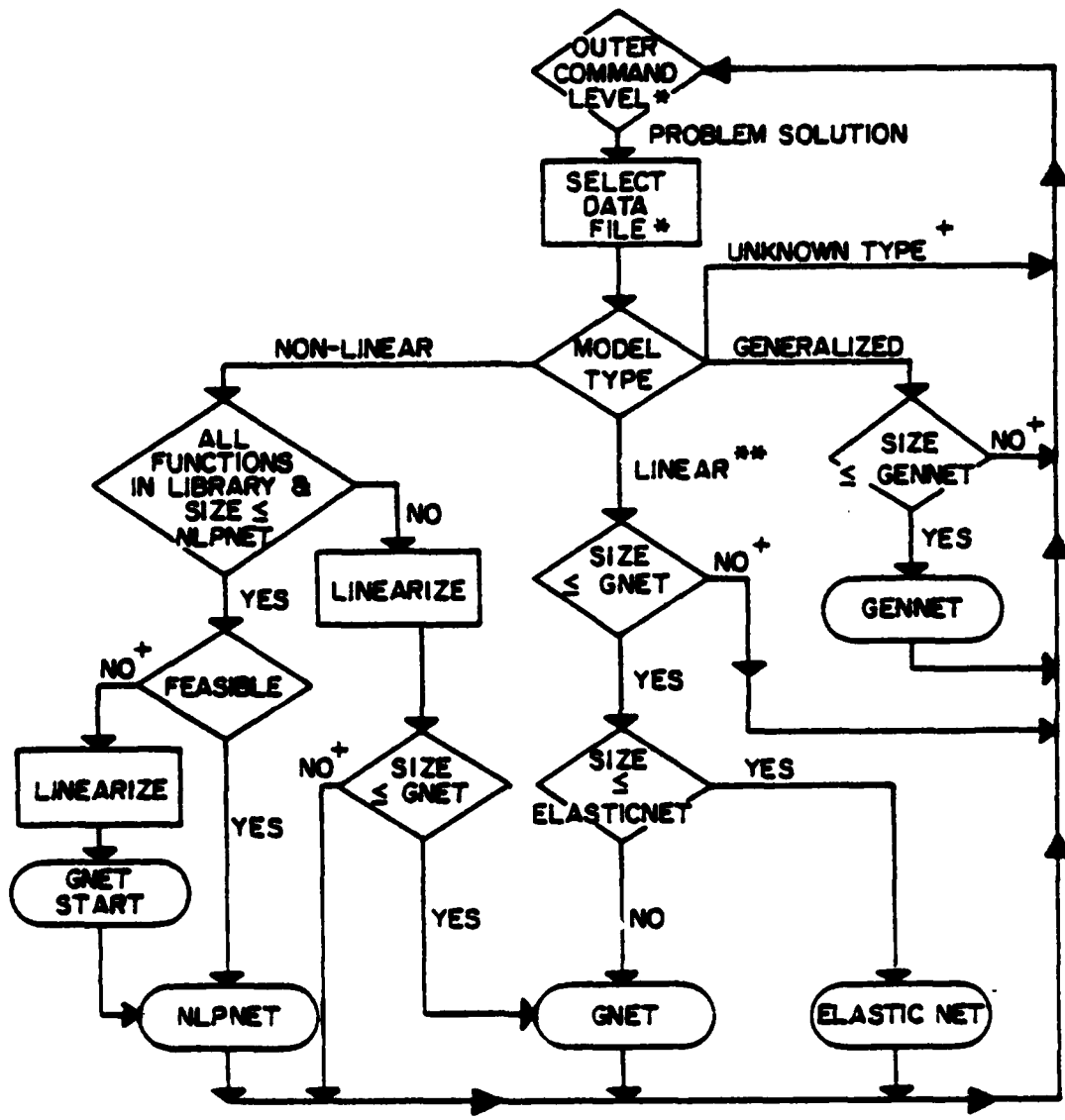
Figure 22. Solution Module Selection Logic

tail of arc k. The head node pointers in H are also set up at this time

and the arrays describing arc characteristics are sorted corresponding to

the order of T. Non-negativity of the right-hand side is enforced at this

point, with the dual variables and predecessors being appropriately updated.

Having initialized the problem, the program now executes the solution

segment. Two new arrays must be introduced at this point for operation

of the candidate queue, making this the most space-critical segment. As

such, the coding in this segment is terse to maximize efficiency and

acceptable problem size. Larger problems could be accommodated if the

solution module were segmented into smaller components such as priceout,

ratio test, and pivot. This would, however, cause multiple memory-to-disk

"swaps" for each pivot, and disk operations are extremely slow compared

with core-memory operations. The decision has been made to sacrifice

problem size in favor of solution speed. To perform memory "swaps" for

each pivot is considered to be prohibitively slow.

The operation of the solution module is split into the three logical

steps of the simplex method: priceout, ratio test, and pivot. The

candidate queue of "interesting nodes and good arcs" is initialized with

the sink nodes--if none are found, the queue may be initialized with any

node. Completion of the solution module is determined by exceeding a

preset maximum number of pivots or by pricing out every non-basic arc and

determining that none price favorably.

If the pivot count has not been exceeded and the solution module ends

with a possible optimal solution, the final report segment is called.

This final segment fi.st determines if there are any artificial arcs left

in the basis with non-zero flow. If there are, the solution is not

feasible. The sum of the flow on the artificials is recorded and the problem is resolved using a larger Big-M cost. After this solution iteration, if a non-feasible solution is again obtained, the total flow on the artificial arcs is compared to the previous total. If the artificial flow has decreased, Big-M is again increased and the problem is again resolved. If the artificial flow has not decreased from the previous iteration, the solution process is terminated and the problem is declared infeasible.

Figure 23 displays the input arrays of a small generalized network problem and Figure 24 exhibits the report of an optimal solution. The final flows indicated on the arcs are flows in addition to arc lower bounds.

[500 ARC 100 NODE VERSION]
DATE: 19 APR 82

FILE: PROB:GT2.NET   CREATED: 18 SEP 81   UPDATED: 14 APR 82
NUMBER OF NODES = 15   NUMBER OF ARCS = 30

ARC LIST ...

ARC NAME	FROM NODE	TO NODE	UNIT COST	GAIN	UPPER BOUND	LOWER BOUND
1	4	3	33.84	0.99	1000.00	0.00
2	2	3	15.47	1.00	1000.00	0.00
3	1	5	53.54	0.74	1000.00	0.00
4	2	5	26.76	0.74	1000.00	0.00
5	3	5	73.49	1.00	1000.00	0.00
6	5	5	52.52	1.00	1000.00	0.00
7	3	6	35.12	0.91	1000.00	0.00
8	5	6	11.12	1.00	1000.00	0.00
9	4	7	59.56	1.17	1000.00	0.00
10	2	7	88.38	1.06	1000.00	2.00
11	4	8	84.12	1.00	1000.00	0.00
12	2	8	21.86	0.92	1000.00	0.00
13	4	9	3.46	1.00	1000.00	0.00
14	3	9	29.72	1.00	1000.00	0.00
15	4	10	6.12	1.00	1000.00	0.00
16	2	10	31.08	0.96	1000.00	0.00
17	3	10	1.07	1.07	1000.00	0.00
18	5	10	44.44	1.00	1000.00	0.00
19	1	11	67.15	0.91	1000.00	0.00
20	2	11	59.83	0.79	1000.00	0.00
21	3	11	50.46	1.17	1000.00	0.00
22	5	11	71.42	1.00	1000.00	0.00
23	2	12	8.88	1.18	1000.00	0.00
24	1	13	28.22	0.83	1000.00	0.00
25	4	13	77.34	1.00	1000.00	0.00
26	3	13	45.60	1.00	1000.00	0.00
27	5	13	20.67	0.88	1000.00	3.00
28	4	14	37.76	1.13	1000.00	0.00
29	2	14	18.16	0.98	1000.00	0.00
30	3	15	67.62	1.00	1000.00	0.00

NODE LIST ... [FOR THOSE NODES EXPLICITLY IN THE DATA FILE]

NODE NAME	NODE NUMBER	NET FLOW	NODE STATUS
1	1	22.86	SUPPLY
2	2	177.14	SUPPLY
6	6	-19.39	DEMAND
7	7	-3.64	DEMAND
8	8	-24.92	DEMAND
9	9	-9.38	DEMAND
10	10	-14.07	DEMAND
11	11	-56.91	DEMAND
12	12	-2.45	DEMAND
13	13	-30.93	DEMAND
14	14	-21.76	DEMAND
15	15	-16.55	DEMAND

Figure 23. Generalized Network Problem

OPTIMALITY OBTAINED IN 23 PIVOTS

ARC	FROM	TO	COST	UP-BOUND	DUAL	FLOW ABOVE LB
24	1	13	28.2200	1000.00	-49.7936	22.8600
2	2	3	15.4700	1000.00	-32.9223	115.089
7	3	6	35.1200	1000.00	-48.3923	15.4126
ART	4	4	0.00000	INF	-220.000	0.00000
4	2	5	26.7600	1000.00	-80.6517	7.24931
8	5	6	11.1200	1000.00	-91.7718	5.36449
10	2	7	88.3800	1000.00	-114.436	1.43396
12	2	8	21.8600	1000.00	-59.5450	27.0870
14	3	9	29.7200	1000.00	-78.1123	9.38000
17	3	10	1.07000	1000.00	-46.2264	13.1495
21	3	11	50.4600	1000.00	-84.4891	48.6410
23	2	12	8.88000	1000.00	-35.4257	2.07627
26	3	13	45.6000	1000.00	-93.9923	11.9562
29	2	14	18.1600	1000.00	-52.1248	22.2041
30	3	15	67.6200	1000.00	-116.012	16.5530

VALUE OF THE OBJECTIVE FUNCTION= 8.94934E3

::
FINAL DUMP * * * PIVOTS:23 DEGENERATE PIVOTS:3 CYCLES CREATED:4
::

Figure 24. Generalized Network Solution

V. CONCLUSIONS AND RECOMMENDATIONS

This project was initiated for two reasons. The first was to better understand generalized networks and to gain an appreciation for and familiarity with the data structures and design of large-scale mathematical programming algorithms. Secondly, an objective was to explore the suitability of a microcomputer as a tool of operations research.

The design and implementation of a large-scale mathematical programming project has been presented here in great detail (only the memory size of the Apple II microcomputer limits the code to medium-sized problems). In mathematical programming in general and microcomputer programming in particular, the requirement to use sparse data structures and efficient computational mechanisms cannot be overemphasized. The programmer must vigorously search for ways to condense coding segments and use mathematical simplification/insight to reduce object code and computational overhead. At the same time, a computer program must have an easily accommodated user interface if it is to be of real value.

While the programmer must be terse and exceedingly efficient when designing a solution module, that same programmer must be lavish when designing the man-machine interface. The primary communications interface for the microcomputer is the keyboard. Mis-stroked keys must be screened, and incorrect user inputs must be tolerated by the program. Single stroke responses to menus have been found to be the best method of communication. When numerical data is required, the user must be provided the luxury of easily amending the input.

81

The network optimization package, as described in this thesis, accommodates both the lean solution module and forgiving user interface requirements. The editor component of the package provides the primary man-machine interface and is designed to prompt and edit all inputs. The solution modules have been coded very succinctly and utilize the extremely efficient data structures of which the GENNET algorithm is exemplary.

This project has demonstrated that serious mathematical programming can be accommodated by microcomputers. The primary drawback of micro-computers in this endeavor is considered to be the slow compilation rate. However, the disadvantages associated with slow compilation represent an initial development cost and as such are considered acceptable, given the utility of the resulting product.

The integrated structure of Micronet eases the burden of data input requirements. Laborious keyboard sessions may be avoided if problems, existing as textfiles on mainframes, are transferred to a microcomputer textfile. The Micronet editor has the ability to create a problem file of arc and node records from a textfile in SHARE format [e.g., Ref. 36]. Alternatively, a computer-based problem generation technique could be employed. In so doing, a microcomputer (or any computer) is capable of solving problems which are much larger than can be reasonably created from a keyboard. Real-life problems to be solved by microcomputers may well be based on microcomputer data files. In this case, data input requirements could again be automated.

Microcomputers continue to evolve at an extremely rapid rate. Prices are falling while memory addressing capability and CPU speed are being enhanced. Some microcomputers can address up to 16 megabytes of

memory, perform double precision arithmetic using a 64-bit representation of floating point numbers, and have internal clock speeds eight times as fast as the Apple II. As such, it is considered pointless to discuss solution speeds, precision, and problem size. The current code accommodates problems of sizes up to 100 nodes and 500 arcs, with execution times averaging .5 seconds per simplex iteration. Certainly any published figures could easily be eclipsed by new entries in the microcomputer market, costing little more than the original list price of a fully equipped Apple II. This project has demonstrated that efficient algorithms can be constructed and implemented on microcomputers for the broad class of problems which may be modelled by a generalized network. The sparse data structures and computational efficiency afforded by modern mathematical programming codes are well-suited for implementation on microcomputers.

Managers, scientists, small businesses, and government agencies purchasing microcomputers for other than mathematical programming purposes can feasibly add a microcomputer optimization package. Small colleges and businesses often cannot afford access to a mainframe computer optimization package. The development of microcomputer-based mathematical programs make these management and decision-making tools available to a much wider audience.

Mathematical programming software for microcomputers is currently in scarce supply, while the population and availability of microcomputers is growing at a breathtaking rate. As managers and operations researchers, we must take full advantage of the power and flexibility of the microcomputer and begin to export mathematical programming methods to the large audience of microcomputer users. It is doubtful that mainframe

computers will be replaced as the primary tool of the operations researcher
and mathematical programmer. However, the advantages and techniques of
scientific management through mathematical programming should be made
available to the "common man" (or at least the common manager) through
the vehicle of microcomputers.

LIST OF REFERENCES

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

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

3. Dantzig, G., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.

4. Bradley, G., "Survey of Deterministic Networks," AIIE Transactions, Vol. 7, No. 1 (1974), pp. 60-87.

5. Glover, F., Hultz, J., Klingman, D., and Stutz. J., "Generalized Networks: A Fundamental Computer-Based Planning Tool," Management Science, Vol. 24, No. 12, (1978) pp. 1209-1220.

6. Bradley, G., Brown, G., and Graves, G., "Design and Implementation of Large Scale Primal Transshipment Algorithms," Management Science, Vol. 24 (1977), No. 1, pp. 1-34.

7. Brown, G. G., and McBride, R. D., "Solving Generalized Networks," manuscript (54 pgs.) presented at ORSA-TIMS Meeting, Detroit, Michigan, 21 April 1982.

8. Brown, G., and Wright, W., "Automatic Identification of Embedded Structure in Large-Scale Optimization Models," in Large-Scale Linear Programming, (G. Dantzig, M. Dempster, and M. Kallio, Eds.), International Institute for Applied Systems Analysis, Laxenberg, Austria, 1981, pp. 781-808.

9. Anton, H., Elementary Linear Algebra, John Wiley & Sons, New York, 1977.

10. Koopmans, T. C., "Optimum Utilization of the Transportation System," Proceedings of the International Statistical Conferences, Washington, D.C. (1947), published in Volume 5 (1949), (Also in Scientific Papers of Tjalling C. Koopmans, Springer-Verlag, New York (1970), p. 184.)

11. Johnson, E., "Networks and Basic Solutions," Operations Research, Vol. 14, No. 4 (1966), pp. 619-623.

12. Glover, F., Karney, D., and Klingman, D., "The Augmented Predecessor Index Method for Locating Stepping Stone Paths and Assigning Dual Prices in Distribution Problems," Transportation Science, Vol. 6, No. 2 (1972), pp. 171-179.

13. Glover, F., Klingman, D., and Stutz, J., "Extension of the Augmented Predecessor Index Method to Generalized Network Problems," Transportation Science, Vol. 7, No. 4 (1973), pp. 377-384.

14. Knuth, D. E., The Art of Computer Programming, Vol. 1 (Fundamental Algorithms), Addison-Wesley, Reading, Massachusetts, 1968.

15. Graves, G. W., "Theory of Permutation Triangulation with Application to Network Flow Problems," Working Paper No. 267, Western Management Science Institute, UCLA, (May 1977).

16. Luenberger, D. G., Introduction to Linear and Nonlinear Programming, Addison-Wesley, 1973.

17. Orchard-Hays, W., Advanced Linear-Programming Computing Techniques, McGraw-Hill, New York, 1968.

18. Beale, E. M. L., Mathematical Programming in Practice, John Wiley & Sons, New York, 1968.

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

20. Geoffrion, A. M. and Graves, G. W., "Multicommodity Distribution System Design by Benders Composition," Management Science, Vol. 20, No. 5 (January 1974), p. 822.

21. Osborne, A., An Introduction to Microcomputers, Vol. 1, McGraw-Hill, 1980.

22. Scanlon, L. J., 6502 Software Design, Howard W. Sams & Co., 1981.

23. Luehrmann, A. and Peckham, H., Apple Pascal, A Hands On Approach, McGraw-Hill, 1981.

24. Beale, E. M. L., "The Blackett Memorial Lecture 1980. Operational Research and Computers: A Personal View," Journal of the Operational Research Society, Vol. 31, pp. 761-767, September 1980.

25. Elam, J., Klingman, D., and Mulvey, J., "An Evaluation of Mathematical Programming and Minicomputers," European Journal of Operational Research, Vol. 3, pp. 30-39, January 1979.

26. Wyman, F. P., "3000 Arcs for 3000 Bucks: How to Justify a Personal Computer for Your OR/MS Department," Interfaces, Vol. 9, pp. 75-80, August 1979.

27. Shore, M. L., "Shortest Paths," Creative Computing, November 1980, pp. 108-113.

28. Duff, R. H., _A Microcomputer-Based Network Optimization Package_, M.S. Thesis, Naval Postgraduate School, Monterey, CA, September 1981.

29. Brown, G. G. and Duff, R. H., "A Microcomputer-Based Network Optimization Package." (Presentation and live demonstration). CORS/ORSA/TIMS Meeting, Toronto, Canada, 4 May 1981.

30. Elam, J., Glover, F., and Klingman, D., "A Strongly Convergent Primal Simplex Algorithm for Generalized Netowrks," _Mathematics of Operations Research_, Vol. 4, No. 1 (1979), pp. 39-59.

31. Glover, F. and Klingman, D., "A Note on Computational Simplifications in Solving Generalized Transportation Problems," _Transportation Science_, Vol. 7, No. 4 (1973), pp. 351-361.

32. Glover, F., Hutz, J., Klingman, D., and Stutz, J., "A New Computer-Based Planning Tool," Research Project CCS 289, Center for Cybernetic Studies, University of Texas at Austin, 1977.

33. Libes, S., "Bytelines--News and Speculation about Personal Computers," _BYTE--The Small Systems Journal_, Vol. 7, No. 8 (August 1982), p. 446.

34. Espinsoa, C., _Apple II Reference Manual_, Apple Computer Inc., 1977.

35. Miller, A. R., _Pascal Programs for Scientists and Engineers_, SYBEX, 1981.

36. Clasen, R. J., "The Numerical Solution of Network Problems Using the Out-of-Kilter Algorithm," RAND Corporation Memorandum, RM-5456-PR Santa Monica, CA, March 1968.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center   2
   Cameron Station
   Alexandria, VA 22314

2. Library, Code 0142   2
   Naval Postgraduate School
   Monterey, CA 93940

3. Chairman
   Department of Operations Research (55)   1
   Naval Postgraduate School
   Monterey, CA 93940

4. Professor Gerald G. Brown (55Bw)   100
   Department of Operations Research
   Naval Postgraduate School
   Monterey, CA 93940

5. Professor Alan R. Washburn (55Ws)   1
   Department of Operations Research
   Naval Postgraduate School
   Monterey, CA 93940

6. Major Richard H. Duff, USMC   1
   H & HS-18 MACG-18 (S-4)
   1st MAW FMFPAC
   FPO San Francisco, CA 96603

7. Professor R. D. McBride   1
   School of Business Administration
   University of Southern California
   Los Angeles, CA 90007

8. Lieutenant Commander Michael E. Finley, SC, USN   2
   1552 Beachview Drive, Lake Christopher
   Virginia Beach, VA 23464

9. Professor George B. Dantzig   1
   Operations Research Department
   Stanford University
   Stanford, CA 94305

END

FILMED

3-83

DTIC