Solving Generalized Networks

Gerald G. Brown; Richard D. McBride

Management Science, Vol. 30, No. 12 (Dec., 1984), 1497-1523.

Stable URL:
http://links.jstor.org/sici?sici=0025-1909%28198412%2930%3A12%3C1497%3ASGN%3E2.0.CO%3B2-T

Management Science is currently published by INFORMS.

Your use of the JSTOR archive indicates your acceptance of JSTOR's Terms and Conditions of Use, available at http://www.jstor.org/about/terms.html. JSTOR's Terms and Conditions of Use provides, in part, that unless you have obtained prior permission, you may not download an entire issue of a journal or multiple copies of articles, and you may use content in the JSTOR archive only for your personal, non-commercial use.

Please contact the publisher regarding any further use of this work. Publisher contact information may be obtained at http://www.jstor.org/journals/informs.html.

Each copy of any part of a JSTOR transmission must contain the same copyright notice that appears on the screen or printed page of such transmission.

JSTOR is an independent not-for-profit organization dedicated to creating and preserving a digital archive of scholarly journals. For more information regarding JSTOR, please contact jstor-info@umich.edu.

http://www.jstor.org/
Tue Jan 27 14:04:01 2004

MANAGEMENT SCIENCE
Vol. 30, No. 12, December 1984
Printed in U.S.A.

SOLVING GENERALIZED NETWORKS*

GERALD G. BROWN AND RICHARD D. McBRIDE
Naval Postgraduate School, Monterey, California 93943
School of Business Administration, University of Southern California, Los Angeles, California 90089-4121

A complete, unified description is given of the design, implementation and use of a family of very fast and efficient large-scale minimum-cost (primal simplex) network programs. The class of capacitated generalized transshipment problems solved includes the capacitated and uncapacitated generalized transportation problems and the continuous generalized assignment problem, as well as the pure network flow models which are specializations of these problems. These formulations are used for a large number of diverse applications to determine how (or at what rate) flows through the arcs of a network can minimize total shipment costs. A generalized network problem can also be viewed as a linear program with at most two nonzero entries in each column of the constraint matrix; this property is exploited in the mathematical presentation with special emphasis on data structures for basis representation, basis manipulation, and pricing mechanisms. A literature review accompanies computational testing of promising ideas, and extensive experimentation is reported which has produced GENNET, an extremely efficient family of generalized network systems.
(NETWORKS/GRAPHS—FLOW ALGORITHMS; NETWORKS/GRAPHS—GENERALIZED)

1. Introduction

This paper reports the development of a large-scale primal network code for solving capacitated generalized transshipment problems. The capacitated generalized transshipment problem is the most general of the minimum cost flow models in continuous variables, which include the capacitated and uncapacitated transportation problems and the continuous generalized assignment problem as well as the pure network specializations of these problems. These models are used for a large number of diverse applications that include transportation of goods, design of reservoir, communications, and pipeline systems, assignment of personnel and machinery to jobs, bid evaluation, currency exchange and cash management, production, sales and inventory planning, and many others. For further discussion of these applications see survey articles such as Bradley (1975), or Glover et al. (1977, 1978), and textbooks (as well as their cited references) such as Dantzig (1963), Jensen and Barnes (1980), and Kennington and Helgason (1980).

The capacitated generalized transshipment model and its specializations are minimum-cost network flow problems. The goal is to determine how (or at what rate) flows through the arcs of a network can minimize shipment costs. The network is a directed graph $\underline{G}$, defined by a set of nodes, $\underline{N}$, and a set of arcs $\underline{A}$, with ordered pairs of nodes (tail, head) as elements indexed by $k : (f_k, t_k)$. For each arc there is a shipping cost per unit flow, c_k, a minimum allowable flow (or lower bound), l_k, and a maximum allowable flow (or upper bound, or capacity), u_k. In addition, there are coefficients (or multipliers, or gains, or losses), $\underline{a}_k$ and $\underline{b}_k$ which can change the magnitude of (or amplify, or attenuate) each unit of flow respectively entering and leaving arc k. Each node is either a supply node where units of the good enter the network, a demand node where units leave, or a transshipment node. The problem is to minimize total costs with flows, x_k, that satisfy the associated lower and upper bounds and preserve the conservation of flow at each node:

$$\text{MIN} \sum_{k \in \underline{A}} c_k x_k \tag{GN}$$

$$\text{s.t.} \sum_{\substack{k \in \underline{A} \\ \text{with } f_k = i}} \underline{a}_k x_k + \sum_{\substack{k \in \underline{A} \\ \text{with } t_k = i}} \underline{b}_k x_k = b_i, \qquad i \in \underline{N},$$

$$l_k \leqslant x_k \leqslant u_k, \qquad k \in \underline{A},$$

where:

$$b_i = \begin{cases} \text{supply if } i \text{ is a supply node;} \\ -\text{demand if } i \text{ is a demand node;} \\ 0 \quad \text{otherwise.} \end{cases}$$

*Accepted by Donald Erlenkotter, former Departmental Editor; received April 21, 1982. This paper has been with the authors $5\frac{1}{2}$ months for 2 revisions.

0025-1909/84/3012/1497$01.25
Copyright © 1984, The Institute of Management Sciences

Note that any linear program with at most two nonzero coefficients associated with each variable is a generalized network (GN). Formulation (GN) is a generalized transshipment model (GT) if all $\underline{a}_k$ are $+1$, in which case the corresponding coefficient is called the multiplier $\underline{m}_k = -\underline{b}_k$. For purposes of exposition, we will address (GT) since, assuming the existence of a finite upper bound on each variable, it is possible to transform the (GN) coefficients—by scaling or by reflecting variables with respect to their upper bounds—so that one coefficient is $+1$ for each variable. The other coefficient for each variable then becomes the multiplier value, $\underline{m}_k$, and the generalized transshipment (GT) network flow interpretation results with a node for each constraint and a directed arc for each variable. If a variable has two nonzero coefficients, its arc is directed away from the node corresponding to the constraint with the $+1$ coefficient; a variable with just one nonzero coefficient (± 1) corresponds to an arc forming a self-cycle, leaving and returning to the same node.

Some generalized networks (GN), such as those obtained by relaxing integer restrictions on flow variables, cannot be scaled conveniently to (GT). These (GN) are accommodated by obvious minor modifications in the following (GT) presentation. We have developed codes for solving (GN) and specialized them for (GT) problems.

Generalized networks can be solved as linear programming problems, but contemporary commercial linear programming systems consume much more computer time and data storage region than special purpose network codes. Indeed, the advantage of network codes is so pronounced that it is even worthwhile to develop special linear programming procedures to exploit intrinsic network structure found embedded within more general models (e.g., Kennington and Helgason 1980, McBride 1981, Brown and Graves 1975, and Brown and McBride 1981). Brown and Wright (to appear) and Brown, McBride and Wood (1983) show that many real-life linear programs contain a large embedded generalized network structure.

These models are widely used because they accurately describe a large variety of important applications. Generalized networks not only directly represent gains with $\underline{m}_k > 1$ (e.g. interest return on investment, heat gain, etc.) and losses with $0 < \underline{m}_k < 1$ (e.g., evaporation, voltage drop, attrition, etc.) in the flows, but also admit conversions of units for these flows (e.g., machine time to output pieces, lira to yen, etc.). In addition, $\underline{m}_k < 0$ represents situations without obvious physical flow interpretation (e.g., flows which "enter the head" of arc k in proportion $\underline{m}_k$ of those entering the tail), but which nonetheless provide valuable modelling tools. There has been continuing growth of interest in network models because efficient computer programs have made possible the reliable, economic solution of problems with more variables than virtually any other optimization technique (e.g., the pure network system, GNET (Bradley et al. 1977), has been installed at hundreds of sites worldwide and is now cited as a routine research tool). Perhaps most important, networks are readily accepted by nonanalysts and are consequently extremely popular operations research models.

Although several papers have been written in this general area, and significant computational breakthroughs have been reported, there has not previously been a single, unified description of a complete implementation, nor have "new generation" computer programs been made generally available to the academic community. Here we report the research and computational experiments which have produced GENNET, an extremely efficient family of network optimization systems. GENNET exploits pure network structure embedded in generalized networks, and specializes intrinsically to GNET (Bradley et al. 1977), when both systems are applied to pure networks, (the floating point arithmetic in) GENNET requires about 15 percent more time than GNET. An important objective of this paper is to make these new approaches easily accessible to a wide audience via a clear exposition and concrete examples of efficient FORTRAN programs. Further, the availability of the (GT) and (GN) computer programs will now make it possible for other investigators to reproduce and extend our experimental results.

Bradley, Brown and Graves (1977) trace the historical developments leading to contemporary primal simplex pure network algorithms, their supporting data structures, and efficient implementations such as GNET. For other subclasses of generalized networks, algorithms have been reported by Jewell (1962), Eisemann (1964), Maurras (1972), Glover, Hultz, Klingman and Stutz (1977, 1978, 1974), Balachandran (1976), and Jensen and Bhaumik (1977). An efficient algorithm for large generalized network problems has been developed by Glover, Klingman, Hultz, Stutz, Karney and Elam (1979, 1977, 1978, 1973, 1973). However, their contributions are scattered among the papers referenced; Kennington and Helgason (1980) and Jensen and Barnes (1980) provide textbook descriptions of computations apparently gleaned from these papers, providing an adequate treatment of the graphical algorithm with more computational advice than the seminal presentation by Dantzig (1963) but few details of efficient basis updating.

2. The Approach

Our approach continues with generalized networks in precisely the philosophical vein of the pure network exposition of Bradley, Brown and Graves (1977, p. 3ff.): we seek data structures and algorithms that yield efficient implementations without

abandoning the flexibility of a general large-scale mathematical programming perspective. We introduce few of the details of the general bounded-variable simplex algorithm, and we repeat little of the underlying pure network material; the assiduous reader might well review the prior paper for which this is an intimate companion.

We continue with a brief description of the algebraic specialization of the simplex method for generalized networks. Specific design decisions and experiments carried out with GENNET are described, including computational tests of alternate approaches. Some extensions of GENNET are presented to further exploit special problem structure.

3. Generalized Network Specialization

Efficient primal simplex specialization to the *generalized* network case depends upon the well-known result that any *generalized* network basis can be put in *nearly* (upper) triangular form by simple permutation of rows and columns. This inherent *near* triangularity can be exploited by direct solution of the simplex equations with *modified* forward, or back substitution. Fortuitously, this basis structure also leads to extremely fast solution updates orchestrated in concert with efficient dynamic reorganization of each new basis.

THEOREM (e.g., Dantzig 1963). *Any basis B extracted from a generalized network problem can be put in the form* (1) *by rearranging rows and columns*:

$$\begin{vmatrix} B^1 & & & & \\ & B^2 & & & \\ & & \ddots & & \\ & & & B^l & \\ & & & & \ddots \\ & & & & & B^p \end{vmatrix} \tag{1}$$

where each square submatrix component B^l is either upper triangular or nearly upper triangular with only one element below the diagonal.

Consider the generalized network problem given in Figure 1, a generalized transshipment problem with 2 sources, 10 sinks, 15 nodes, and 30 arcs. We will use this problem to illustrate concepts and efficient solution methods. In this problem the multipliers are all positive. For arc k directed from node i to node j, if flow x_k leaves node i, then $\underline{m}_k x_k$ arrives at node j. When $\underline{m}_k > 1$ the amount arriving at node j is greater than the amount leaving node i. This would be the case in cash flow problems when the arc corresponds to an investment and $\underline{m}_k = (1 + \underline{r}_k)$ with $\underline{r}_k$ the rate of return. When $\underline{m}_k < 1$ then the amount arriving at node j is less than the amount leaving node i. Here the loss could correspond to evaporation, taxation, transmission loss, brokerage fees, seepage or deterioration. Pure network arcs are indicated by $\underline{m}_k = 1$ (and may be even more abundant in real problems than in our example). For clarity, minimum allowable flows are zero, and maximum allowable flows are not specified.

Figure 2 shows a basis for the problem introduced in Figure 1. (In this simple case, there is only one component: $p = 1$.)

A unique subgraph partition of $\underline{G}$ denoted $\underline{G}_B$ corresponds to B. Let $\underline{A}_B = \{a^j \mid a^j$ is an arc associated with b^j, a column of $B\}$, then $\underline{G}_B = [\underline{N}, \underline{A}_B]$ denotes the directed graph associated with the basis B. To each submatrix B^l of B there corresponds a component of $\underline{G}_B$ denoted by $\underline{G}^l = [\underline{N}^l, \underline{A}^l]$. $\underline{N}^l$ is the set of nodes corresponding to the rows of B^l and $\underline{A}^l$ is the set of arcs corresponding to the columns of B^l. It is known

Node	Supply
1	22.86
2	177.14

Node	Demand
6	19.39
7	3.64
8	24.92
9	9.38
10	14.07
11	56.91
12	2.45
13	30.93
14	21.76
15	16.55

Arc k	From f_k	To t_k	Cost c_k	Multiplier $\underline{m}_k$
1	4	3	33.84	0.99
2	2	3	15.47	1.00
3	1	5	53.54	0.74
4	2	5	26.76	0.74
5	3	5	73.49	1.00
6	5	5	52.52	1.00
7	3	6	35.12	0.91
8	5	6	11.12	1.00
9	4	7	59.56	1.17
10	2	7	88.38	1.06
11	4	8	84.12	1.00
12	2	8	21.86	0.92
13	4	9	3.46	1.00
14	3	9	29.72	1.00
15	4	10	6.12	1.00
16	2	10	31.08	0.96
17	3	10	1.07	1.07
18	5	10	44.44	1.00
19	1	11	67.15	0.91
20	2	11	59.83	0.79
21	3	11	50.46	1.17
22	5	11	71.42	1.00
23	2	12	8.88	1.18
24	1	13	28.22	0.83
25	4	13	77.34	1.00
26	3	13	45.60	1.00
27	5	13	20.67	0.88
28	4	14	37.76	1.13
29	2	14	18.16	0.98
30	3	15	67.62	1.00

FIGURE 1. A Single Commodity Generalized Transshipment Problem (GT).

(e.g., Dantzig 1963) that $\underline{G}^l$ is either a rooted tree or a one-tree (a tree with an additional arc forming one cycle). If $\underline{G}^l$ is a rooted tree, then B^l is upper triangular. If $\underline{G}^l$ is a one-tree, then B^l is upper triangular except for one element below the diagonal. Each component can be viewed as having one cycle if we assert that each rooted tree has a self-cycle corresponding to its root node.

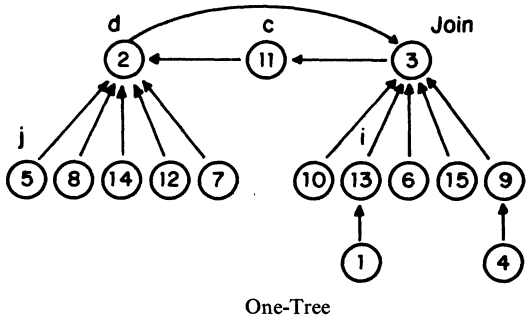
In our example basis, the subgraph $\underline{G}_B$ has only one *component*, (one-tree) shown in Figure 2: $\underline{G}_B$ is a one-tree, with nodes 2, 3 and 11 composing its cycle.

As in the pure network case, this near triangulation and associated subgraph are naturally represented by a *predecessor function* p( ), and a predecessor graph (which does not preserve the orientation of arcs in the original network). The predecessor function can be used iteratively to construct the unique *backpath* from any node to the root (or cycle); the backpath includes all nodes on the cycle. The *immediate successors* of a node, if any, are the first nodes encountered on all paths except the backpath to the root, and all the nodes on these paths are called *successors*.

Note that each basis may have many near triangulations. However, all such near triangulations yield the same predecessor function and graph (where the right to left ordering of successors of any node is immaterial). Thus, the predecessor graph does not completely represent a near triangulation without additional information: an ordering of the rows (nodes). For algebraic reasons, we restrict such partial ordering to preorder (Bradley et al. 1977), in which a node i always precedes its successors, if any, *and* in which all its successors, if any, precede any node which does not precede node i.

Row \ Column	2	4	12	29	23	10	20	21	17	26	24	7	30	14	13
2	1	1	1	1	1	1	1								
5		-.74													
8			-.92												
14				-.98											
12					-1.18										
7						-1.06									
11							-.79	-1.17							
3	-1							1	1	1		1	1	1	
10									-1.07						
13										-1.0	-.83				
1											1				
6												-.91			
15													-1.0		
9														-1.0	-1.0
4															1

A Preorder Near-Triangulation
(The underlined coefficient is below the diagonal.)



One-Tree

FIGURE 2. A Generalized Transshipment Basis (for the problem in Figure 1).

4. Implementation

For didactic reasons, we begin by introducing a complete primal generalized network algorithm using a preorder traversal method. Controversial alternatives are deferred until this paradigm is presented. Hereafter, notation with upper case italic letters followed by parentheses indicates a program data array. For instance, the predecessor *array* is referred to as P( ).

Static arc storage is used for tails T( ), head H( ), costs C( ), multipliers MUL( ), and capacities CP( ). Contiguous storage by tail, or by head node reduces T( ), or H( ) to a hierarchical node-length entry point array. (GENNET uses contiguous storage by head node, as does GNET.) Lower bounds on arc flow are translated out prior to solution, with appropriate adjustment of the initial right-hand side of (GT), and of CP( ). The sign bit of CP( ) is available to indicate arcs *nonbasic* at their upper bounds (*reflected* with flow $-$CP( )).

The predecessor function and its array P( ) are defined so that the basic arcs in a cycle are oriented uniformly in a directed cycle. All basic arcs not on a cycle are oriented so that a backpath is created to a cycle. To obtain this orientation the direction of some arcs must be reversed, and the sign bit of the predecessor array is used to indicate: if P(I) $<$ 0, then the orientation of arc (I, $-$P(I)) is reversed from its original orientation.[1]

A *depth* array D( ) reveals for each node the number of nodes on the backpath before encountering a cycle. Nodes on a cycle have depth zero. *Number of successors*, or *preorder distance* are acceptable substitutes for depth (Bradley et al. 1977), but are not discussed here.

A preorder traversal array IT( ) is maintained so that all preorder successors of a cycle node are encountered before another cycle node. It is convenient to make this a circular list for each near triangular basis component by setting IT( ) of the last preorder node in the component equal to the first preorder node in the component.

The components of $\underline{G}_B$ are not interconnected, or equivalently, the submatrices B^l in (1) do not have common rows or columns. Consequently, the p components of a basis may be represented in a single set of node-length arrays.

The array X( ) contains the values of basic variables, values of dual variables (or simplex multipliers, or node potentials) are stored in U( ), and IVAR( ) gives the

Node	Predecessor	Depth	Traversal	Basic Variable		Dual	Cycle Factor
	P( )	D( )	IT( )	IVAR( )	X( )	U( )	FAC( )
1	13	2	6	24	22.860	16.665	*
2	3	0	5	2	118.169	47.148	− 0.48101
3	11	0	10	21	45.826	31.678	− 0.48101
4	9	2	2	13	0.0	5.418	*
5	− 2	1	8	4	0.0	27.552	*
6	− 3	1	15	7	21.308	− 3.782	*
7	− 2	1	11	10	3.434	− 38.898	*
8	− 2	1	14	12	27.087	27.487	*
9	− 3	1	4	14	9.380	1.958	*
10	− 3	1	13	17	13.150	28.606	*
11	− 2	0	3	20	4.170	− 16.053	− 0.48101
12	− 2	1	7	23	2.076	32.431	*
13	− 3	1	1	26	11.956	− 13.922	*
14	− 2	1	12	29	22.204	29.580	*
15	− 3	1	9	30	16.550	− 35.942	*

FIGURE 3. GENNET Basis Representation Arrays (for Basis in Figure 2).

[1] This is the complement of the discipline used in GNET (Bradley et al. 1977).

location of basic variables in the arc arrays. The array FAC( ) contains the *cycle factors*, defined later. Figure 3 shows these arrays for the basis given in Figure 2.

Generalized networks do not exhibit totally unimodular bases. Consequently, floating point representation is required for X( ), U( ) and FAC( ), and is desirable for arc-length arrays C( ), MUL( ), and CP( ).

Step S1, Priceout

The reduced cost for nonbasic arc k, oriented from f_k to t_k is (given the current dual solution u and column N^k):

$$\begin{aligned} r_k &= c_k - uN^k = c_k - u_{f_k} + \underline{m}_k u_{t_k} \\ &= \mathrm{C}(k) - \mathrm{U}(f_k) + \mathrm{MUL}(k)*\mathrm{U}(t_k). \end{aligned}$$

(If $\mathrm{CP}(k) < 0$, arc k is reflected and the sign of r_k is reversed.) At most, one multiplication, addition and subtraction are required. Note that the multiplication is unnecessary if $|\underline{m}_k| = 1$; further specialization is possible for sets of priced arcs with common attributes. If arc k is a logical arc (slack, artificial, or surplus variable) then $\mathrm{C}(k)$ can be logically generated, rather than explicitly stored, and $r_k = \mathrm{C}(k) \pm \mathrm{U}(f_k)$, depending upon the sign of $\mathrm{P}(f_k)$.

From the example,

$$\begin{aligned} r_{27} &= \mathrm{C}(27) - \mathrm{U}(5) + \mathrm{MUL}(27)*\mathrm{U}(13) \\ &= 20.67 - (27.552) + 0.88(-13.922) \\ &= -19.133. \end{aligned}$$

This variable will be used as the entering nonbasic variable for further illustration.

Step S2, Ratio Test

For the determination of the arc to leave the basis, the system of equations $BZ^k = N^k$ must be solved for the transformed column Z^k. ($-N^k$ is used if arc k is reflected.) Due to the near triangularity of B, this incoming *column transformation* can be combined with the ratio test in a single integrated process.

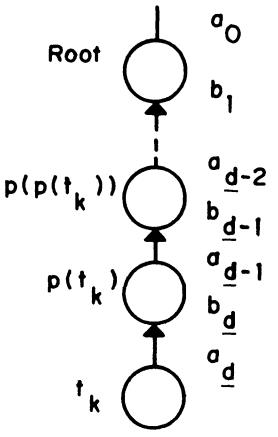
Suppose that N^k has two nonzero coefficients representing an arc oriented from f_k to t_k, and that the arc is not reflected. An apparent complication arises if f_k and t_k are in separate components of B in (1), say B^s and B^t, respectively. In this case two disjoint subsystems must be solved and the results added to determine the nonzero elements in Z^k. The subsystems are:

$$B^s Q^{f_k} = e_{f_k} \quad \text{and} \quad B^t Q^{t_k} = -\underline{m}_k e_{t_k}$$

(e_{t_k} is the t_kth unit vector; Q^{f_k} and Q^{t_k} are disjoint components of Z^k).

This complication is inconsequential. In order to see this, consider solving one of the systems: $B^t Q^{t_k} = -\underline{m}_k e_{t_k}$. The only nonzero elements of Q^{t_k} will be those that correspond to the nodes in the backpath from node t_k. As we shall see, this follows from the manner in which the coefficient $-\underline{m}_k$ in row t_k propagates during substitution solution.

Suppose that $\underline{G}^t$ is a rooted tree. The backpath from node t_k can be denoted by iteration of the predecessor function: $t_k, p(t_k), p(p(t_k)), \ldots,$ root; this sequence is shown in the following figure as $\underline{d}, \underline{d}-1, \ldots, 0$, analogous to the backpath length remaining to the root. The values of a and b for each basic arc depend upon the original orientation of the arc, given by the sign of P( ). For original orientation, $\mathrm{P}( ) > 0$ implies that $a = +1$ and b is the multiplier value, $\mathrm{P}( ) < 0$ reverses these definitions.



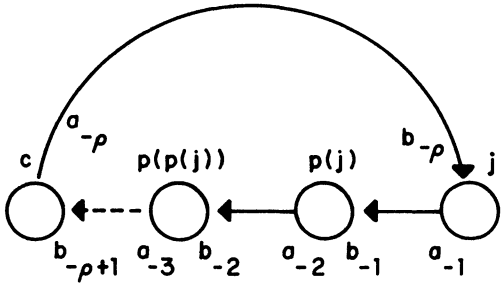
The triangular system corresponding to this backpath is:

$$\begin{vmatrix} a_0 & b_1 & & & & & \\ & a_1 & \ddots & & & & \\ & & \ddots & a_{\underline{d}-2} & b_{\underline{d}-1} & & \\ & & & & a_{\underline{d}-1} & b_{\underline{d}} & \\ & & & & & a_{\underline{d}} & \ddots \end{vmatrix} \quad Q^{t_k} = \begin{vmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ -\underline{m}_k \\ 0 \end{vmatrix}$$

By back substitution, its solution is:

$$q_{\underline{d}} = \frac{-\underline{m}_k}{a_{\underline{d}}}, \qquad q_\delta = \frac{-b_{\delta+1}q_{\delta+1}}{a_\delta} \qquad \text{for} \quad \delta = \underline{d}-1, \ldots, 0.$$

Now suppose that $\underline{G}^t$ is a one-tree. Let node t_k be on the cycle. The backpath is $t_k, p(t_k), p(p(t_k)), \ldots, c$, with $p(c) = t_k$ and length ρ; this sequence is shown in the figure as $-1, -2, \ldots, -\rho$, analogous to the backpath length beyond the cycle start.



The corresponding near triangular system is:

$$\begin{vmatrix} a_{-\rho} & b_{-\rho+1} & & & & & \\ & a_{-\rho+1} & \ddots & & & & \\ & & \ddots & a_{-3} & b_{-2} & & \\ & & & & a_{-2} & b_{-1} & \\ b_{-\rho} & & & & & a_{-1} & \ddots \end{vmatrix} \quad Q^{t_k} = \begin{vmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ -\underline{m}_k \\ 0 \end{vmatrix}$$

By modified back substitution, its solution is:

$$q_{-1} = \frac{-m_k}{a_{-1}} \times \frac{1}{f}, \qquad q_\delta = \frac{-b_{\delta+1} q_{\delta+1}}{a_\delta} \qquad \text{for} \quad \delta = -2, \ldots, -\rho, \quad \text{where}$$

$$f = 1 - \prod_{\delta=-1}^{-\rho} \frac{-b_\delta}{a_\delta}.$$

The cycle factor (or loop factor Dantzig 1963), f, is common to all nodes in the cycle and can be computed when the cycle is created and stored in FAC( ) for all nodes on the cycle so that it is immediately available at this step.

Suppose that the entering arc is (f_k, t_k) with coefficient entries (a_k, b_k) and that the f_k and t_k backpaths converge. Using nomenclature for the backpath sequences introduced above, the corresponding system is:

$$\begin{array}{r|cccccccccccccc|c|c|}
 & a_0 & b_1 & & & & & & & & & & & & & & 0 \\
 & & a_1 & & & & & & & & & & & & & & \vdots \\
 & & & \ddots & & & & & & & & & & & & & \\
 & & & a_{l-1} & b_l & b_w & & & b_r & & & & & & & & \\
 & & & & a_l & & & & & & & & & & & & \\
 & & & & & \ddots & & & & & & & & & & & \\
 & & & & & a_w & & & & & & & & & & & \\
 & & & & & & \ddots & & & & & & & & & & \\
\text{join} & & & & & & a_{d-1} & b_d & & & & & & & & Q^{t_k} = & 0 \\
 & & & & & & & a_d & & & & & & & & & a_k \\
 & & & & & & & & \ddots & & & & & & & & 0 \\
 & & & & & & & & a_r & & & & & & & & \vdots \\
 & & & & & & & & & \ddots & & & & & & & \\
 & & & & & & & & & & a_{\underline{d}-1} & b_{\underline{d}} & & & & & 0 \\
 & & & & & & & & & & & a_{\underline{d}} & & & & & b_k \\
 & & & & & & & & & & & & \ddots & & & & 0
\end{array}$$

By back substitution, its solution is:

$$q_{\underline{d}} = b_k / a_{\underline{d}} \qquad (\underline{d} \text{ begins } t_k \text{ backpath sequence}),$$

$$q_\delta = \frac{-b_{\delta+1} q_{\delta+1}}{a_\delta} \qquad \text{for} \quad \delta = \underline{d}, \underline{d} - 1, \ldots, r,$$

$$q_d = a_k / a_d \qquad (d \text{ begins } f_k \text{ backpath sequence}),$$

$$q_\delta = \frac{-b_{\delta+1} q_{\delta+1}}{a_\delta} \qquad \text{for} \quad \delta = d, d-1, \ldots, w,$$

$$q_{l-1} = \frac{-(b_w q_w + b_r q_r)}{a_{l-1}},$$

$$q_\delta = \frac{-b_{\delta+1} q_{\delta+1}}{a_\delta} \qquad \text{for} \quad \delta = l-2, l-3, \ldots, 0.$$

Suppose that entering arc (f_k, t_k) with coefficients (a_k, b_k) enters and that the backpaths converge on a cycle. The corresponding system is:

$$
\left|\begin{array}{cccccccccccccccccccccccc}
a_{-\rho} & b_{-\rho+1} \\
& a_{-\rho+1} \\
& & \ddots \\
& & & a_{-l-1} & b_{-l} & & & & & & & & b_w \\
& & & & a_{-l} \\
& & & & & \ddots \\
& & & & & & a_{-s-1} & b_{-s} & & & & & & & & & & & & b_r \\
& & & & & & & a_{-s} \\
& & & & & & & & \ddots \\
& & & & & & & & & a_{-2} & b_{-1} \\
b_{-\rho} & & & & & & & & & & a_{-1} \\
& & & & & & & & & & & \ddots \\
& & & & & & & & & & & & a_w & b_{w+1} \\
& & & & & & & & & & & & & a_{w+1} \\
& & & & & & & & & & & & & & \ddots \\
& & & & & & & & & & & & & & & a_{d-1} & b_d \\
& & & & & & & & & & & & & & & & a_d \\
& & & & & & & & & & & & & & & & & \ddots \\
& & & & & & & & & & & & & & & & & & a_r & b_{r+1} \\
& & & & & & & & & & & & & & & & & & & a_{r+1} \\
& & & & & & & & & & & & & & & & & & & & \ddots \\
& & & & & & & & & & & & & & & & & & & & & a_{\underline{d}-1} & b_{\underline{d}} \\
& & & & & & & & & & & & & & & & & & & & & & a_{\underline{d}} \\
& & & & & & & & & & & & & & & & & & & & & & & \ddots
\end{array}\right| \quad Q^{t_k} = \left|\begin{array}{c} 0 \\ \vdots \\ 0 \\ a_k \\ 0 \\ \vdots \\ 0 \\ b_k \\ 0 \end{array}\right|
$$

By modified back substitution, its solution is:

$$q_{\underline{d}} = b_k / a_{\underline{d}} \qquad (\underline{d} \text{ begins } t_k \text{ backpath sequence}),$$

$$q_\delta = \frac{-b_{\delta+1} q_{\delta+1}}{a_\delta} \qquad \text{for } \delta = \underline{d}, \underline{d}-1, \ldots, r,$$

$$\bar{q}_{-s-1} = \frac{-b_r q_r}{a_{-s-1}} \times \frac{1}{f},$$

$$\bar{q}_\delta = \frac{-b_{\delta+1} \bar{q}_{\delta+1}}{a_\delta} \qquad \text{for } \delta = -s-2, \ldots, -l, -l-1, \ldots, -s,$$

$$q_d = \frac{a_k}{a_d} \qquad (d \text{ begins } f_k \text{ backpath sequence}),$$

$$q_\delta = \frac{-b_{\delta+1} q_{\delta+1}}{a_\delta} \qquad \text{for } \delta = d, \ldots, w,$$

$$\hat{q}_{l-1} = \frac{-b_w q_w}{a_{-l-1}} \times \frac{1}{f},$$

$$\hat{q}_\delta = \frac{b_{\delta+1} \hat{q}_{\delta+1}}{a_\delta} \qquad \text{for } \delta = -l-2, \ldots, -\rho, -1, \ldots, -s, \ldots, -l,$$

$$q_\delta = \bar{q}_\delta + \hat{q}_\delta \qquad \text{for } \delta = -1, \ldots, -\rho.$$

For the cycle in Figure 2,

$$f = 1 - \frac{-(-1)}{1} \times \frac{-(1)}{-0.79} \times \frac{-(-1.17)}{1} = 0.48101.$$

By now it should be apparent that one composite back substitution scheme will suffice for all cases. The cycle factor is applied once when, and if, a cycle is encountered on a backpath.

If the backpaths of f_k and t_k converge, let *join* be the first node on the t_k backpath that is also on the f_k backpath. If the backpaths converge on a cycle and if the leaving arc precedes the *join* on the f_k backpath, define join as the first cycle node encountered on the f_k backpath. If the backpaths do not converge, *join* = Ø.

Several schemes are available for identifying the join efficiently (Bradley et al. 1977). The depth (or number of successors, or preorder distance) of nodes on the backpaths can be used to avoid iterating either backpath past the join. Depth, the number of nodes on the backpath until a cycle is encountered, can be used to indicate which backpath node is deeper and should be iterated. When both backpath nodes have matching depths, zero depths indicate that each backpath is on a root cycle. By remembering the first root cycle node on each backpath, further iteration will either reveal the root cycles to be distinct with *join* = Ø, or coincident with join defined as above. When both backpath nodes have matching depths greater than zero, the nodes are compared for equality. A match indicates the join, and a mismatch indicates that both backpaths should be iterated for another comparison.

If a join is encountered, the backpaths have merged and either one can be used to complete the ratio test (GENNET continues the t_k backpath). When a cycle is encountered the $\hat{q}$ values are computed on the first pass around the cycle. On the second pass around the cycle the $\bar{q}$ values are computed and the ratio test is completed.

As the backpaths are iterated, the column transformation is applied and the

resulting terms of transformed column, Z^k, are used in the *ratio test*, seeking the *minimum ratio*:

$$\text{MIN}\begin{cases} \text{CP}(k) & \text{the capacity of the incoming arc,} \\ \dfrac{\text{X}(l)}{z_l^k} & \text{for } z_l^k > 0, \text{ node } l, \\ \dfrac{\text{CP}(\text{IVAR}(l)) - \text{X}(l)}{-z_l^k} & \text{for } z_l^k < 0. \end{cases}$$

If a zero ratio is encountered during this process, the ratio test may be preemptively terminated.

Step S3, Pivot

If CP(k) is selected as the minimum ratio, then the entering variable remains nonbasic and is reflected to its opposite bound. Only the flows X( ) need be updated. To do this, the backpaths are iterated again and for each node l encountered, X( ) is reduced by $z_l^k \times \text{CP}(k)$.

If a basis exchange is required, an efficient update of the basis representation must preserve the rooted cycle orientation, updating some entries in P( ) and D( ). Also, some flows X( ), and some dual variables U( ), must be changed. FAC( ) must be established for nodes on newly created cycles. Some bookkeeping in IVAR( ) and CP( ) may also be required.

The apparent intricacy of our task is deceptive. Careful analysis yields an elegant solution. However, the supporting arguments require close attention.

To simplify the explanation, reorient the incoming arc (f_k, t_k) to (i, j) or (j, i) (if necessary) so that the minimum ratio is on the j backpath. Let the entering arc (i, j) have the outgoing arc (c, d) on its j-backpath. Also, reorient the outgoing arc (if necessary) so that the first node encountered on the j backpath is c. Figure 2 shows a case for which both reorientations are necessary.

In our example, arc 20, oriented from node 2 to node 11, leaves the basis and arc 27, oriented from node 5 to node 13 enters. We call the backpath segment from j to arc (c, d) the *j-stem*. In Figure 2, i, j, c, and d are shown. The j-stem is composed of nodes 5, 2, 3, and 11. The skeletal update:

a. Reverses the orientation of arcs within the j-stem; and

b. Orients the entering arc so that it precedes this redirected path with the same orientation.

If the i and j backpaths merge, the node where they merge is called the join. The join is node 3 in Figure 2.

If the leaving arc lies beyond the join on the backpaths a new cycle is created in the basis exchange. In this case the portion of the i backpath from node i to the node with the join as its predecessor is called the i-stem. When node i is on the j backpath the i-stem is null. In Figure 2, the i-stem is node 13.

Figure 4 displays the pivot logic to be applied. The algorithm visits each node affected by the basis update *exactly once*. It proceeds up the j-stem one node at a time visiting the preorder successors of each stem node via IT( ). If a join is encountered it switches to proceed up the i-stem one node at a time and then returns to the j-stem. At each j-stem node, the successors of the next lower stem node have already been visited. The unvisited successors of the current stem node can be divided into two groups: the *left successors* are the nodes visited in preorder by iterating IT( ) from the current stem node until the next lower stem node is encountered, and the *right successors* of the stem node are the remaining unvisited nodes reached by further iteration of IT( ). In Figure 2, nodes 8, 14, 12, and 7 are right successors of node 2 and nodes 10, 13, 1, 6,

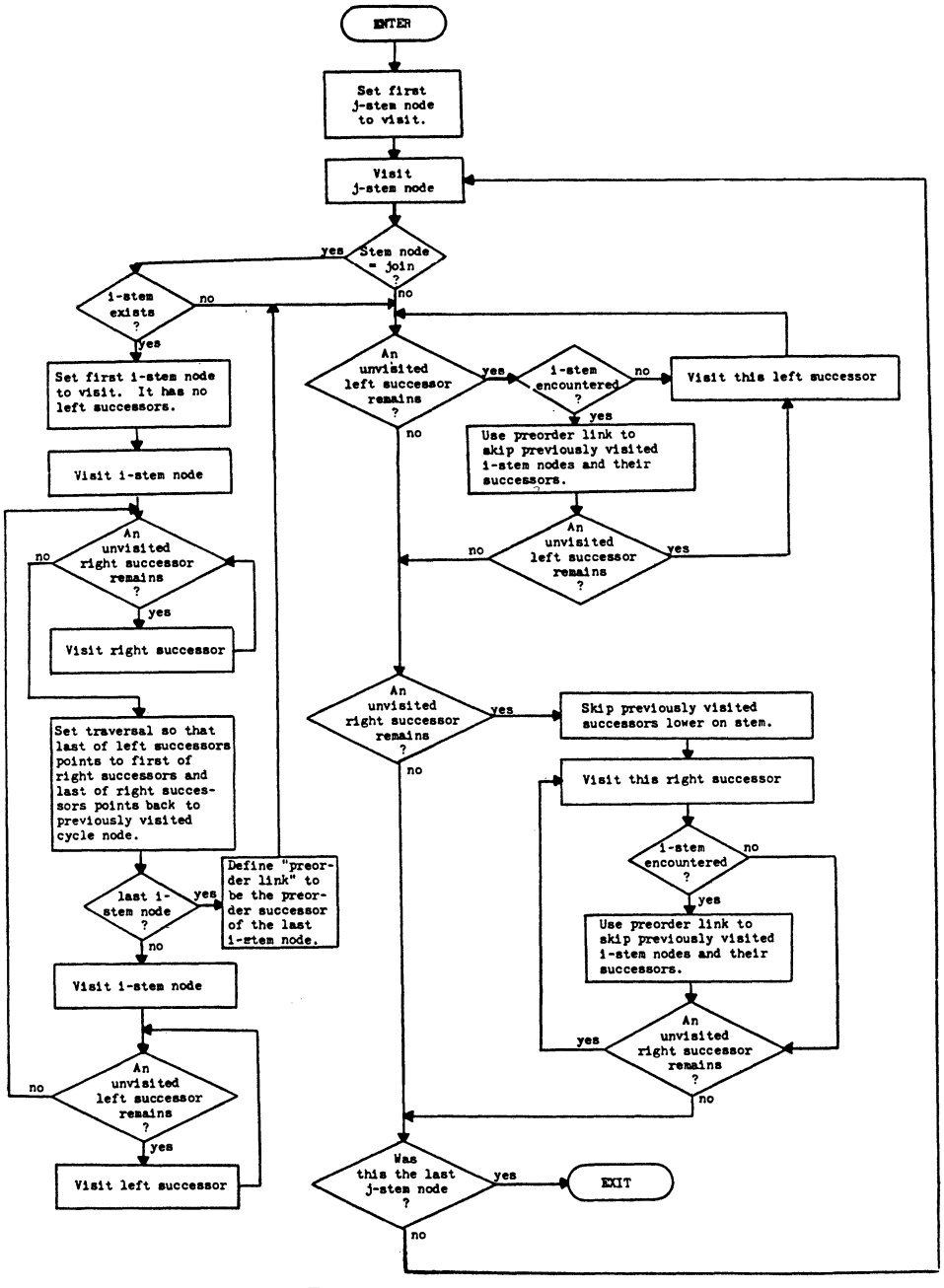


FIGURE 4. Pivot Traversal Scheme.

15, 9, and 4 are left successors of node 3. As we climb the *i*-stem the traversal IT( ) is modified so that the last of the left successors (if any) points to the first of the right successors and the last of the right successors (if any) points to the previous stem node. As we climb the *j*-stem, the traversal is modified so that the last of the left successors (if any) points to the first of the right successors and the last of the right successors (if any) points to the next node up the stem (because the update reverses predecessor orientation for the *j*-stem).

The immediate switch to the *i*-stem upon encountering the join during the *j*-stem iteration is motivated by a subtle complication: while visiting the left and right successors of the *j*-stem nodes, the nodes on the *i*-stem and their successors must be

skipped if encountered. Because the i-stem nodes are successors of the join, visiting the i-stem as soon as the join is encountered (if one exists) on the j-stem leaves us with the preorder successor of the last i-stem node visited. This valuable artifact enables the subsequent j-stem iteration to immediately skip all i-stem nodes and their successors should they be encountered. This is the key step preserving an efficient one-pass basis update. In Figure 2, i-stem node 13 and its successor node 1 are successors of the join, node 3. The preorder successor of the last i-stem node (called the *preorder link* in Figure 4) is node 6.

A stem node may have a right successor which is on the root cycle (with depth zero). The preorder traversal array is organized so that all successors of a cycle node are encountered before the next cycle node. This implies that a cycle being broken by a leaving arc will always be encountered as a right successor of a stem node. In Figure 2, the broken cycle is encountered as a right successor of node 2; if arc (2, 11) were not the leaving arc, then node 11 would be a preorder successor of node 2.

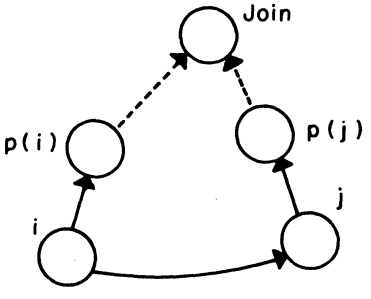
The basic arc flows, X( ), are changed as each arc is visited on the j-stem, and (if a new cycle is created) on the i-stem. If no cycle is created, X( ) is changed *only* if the minimum ratio is nonzero, and then only on the arcs visited on the backpaths.

During the update, the dual variables must be changed so that $c_k = u_{f_k} - \underline{m}_k u_{t_k}$, for every basic arc k oriented from node t_k to node f_k. With incoming arc (i, j) (reoriented as in Figure 2 so that the outgoing arc is on the j-stem), this relationship is retained for all nodes except for those which the update changes to be successors of i: (i.e. the nodes on the j-stem and their successors).

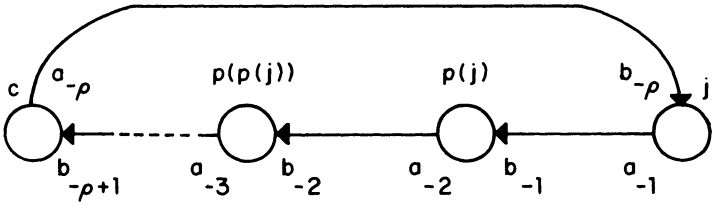
If a cycle is not created, this update proceeds for each j-stem node and its successors as these nodes are visited in preorder. For node s, and associated basic arc l oriented from s to $p(s)$, $U(s) = C(l) + MUL(l)*U(P(s))$, while the reverse orientation $-P(s)$ to s yields $U(s) = (C(l) - U(-P(s)))/(-MUL(l))$. As in pure networks, the preorder traversal assures that a value of the dual variable of a predecessor node is always determined prior to its use by any immediate successor nodes.

When a cycle is created, the dual variable must be determined for one of the cycle nodes. Then, the dual variables of the remaining cycle nodes and their descendants can be found one at a time in preorder traversal.

This key dual variable is computed immediately after the ratio test predicts creation of a new cycle (the leaving arc lies beyond the join on the ratio backpaths). Consider the current context for a new cycle:



from which the new cycle will be formed (with modified predecessor function p( )):



The corresponding near-triangular system is:

$$(u_{-\rho}, u_{-\rho+1}, \ldots, u_{-1}) \begin{bmatrix} a_{-\rho} & b_{-\rho+1} & & & & & \\ & a_{-\rho+1} & & & & & \\ & & \ddots & & & & \\ & & & a_{-3} & b_{-2} & & \\ & & & & a_{-2} & b_{-1} & \\ b_{-\rho} & & & & & a_{-1} & \end{bmatrix} = (c_{-\rho}, c_{-\rho+1}, \ldots, c_{-1})$$

where the indexing of c is understood to yield basis arc costs (accomplished by indexing with IVAR( )).

The determination of one term (say, u_{-1}) of the solution of this system is induced by modified forward substitution to be (e.g., Dantzig 1963): $u_{-1} = u'_{-1} \times 1/f$, where (the cycle factor) f is defined (again) as

$$f = 1 - \prod_{\delta=-1}^{-\rho} \frac{-b_\delta}{a_\delta},$$

and u'_{-1} is the corresponding term of the solution of the strictly upper triangular system (omitting the cyclic coefficient $b_{-\rho}$, and using forward substitution):

$$u'_{-\rho} = \frac{c_{-\rho}}{a_{-\rho}}, \qquad u'_\delta = \frac{c_\delta - b_\delta u'_{\delta-1}}{a_\delta} \qquad \text{for} \quad \delta = -\rho+1, \ldots, 1.$$

Note that computation of u'_{-1} requires that we traverse the new cycle in a direction *opposite* to its predecessor orientation. However, *before* the update creating the new cycle, the j-stem exhibits proper orientation for at least part of our work. Thus, we can complete the first portion of the forward substitution for u'_{-1} and accumulate the associated partial product component of f while iterating the j-stem before the update.

The remainder of the new cycle is accessed by iterating the i-stem. As we proceed up the i-stem the remaining product terms of f are accumulated, and the reverse i-stem path is stored (e.g., using U( ) locations, which contain obsolete dual values to be replaced during the imminent update). Reaching the join, this stored reverse i-stem path is then accessed to complete computation of u'_{-1}.

The newly created cycle in Figure 5 is composed of j-stem nodes 5, 2, and 3, and i-stem node 13. The dual system becomes

$$\begin{aligned} -u_{13} + u_3 &= 45.60, \\ -u_3 + u_2 &= 15.47, \\ u_2 - 0.74u_5 &= 26.76, \\ -0.88\,u_{13} + u_5 &= 20.67. \end{aligned}$$

which yields

$$u'_{13} = -19.0142, \qquad f = 0.3488 \qquad \text{(the new cycle factor), and}$$

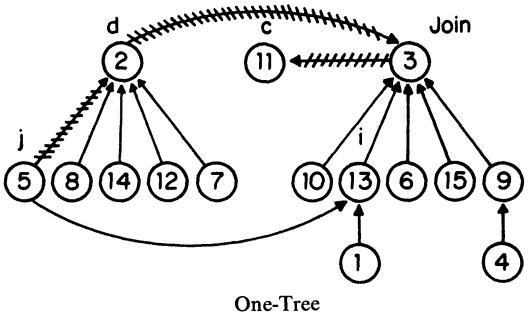
$$u_{13} = -54.5132 \qquad \text{(the new dual for node 13).}$$

Thus, when a new cycle is to be formed, the new cycle nodes must be visited once (after the ratio test and prior to the pivotal update).

The one-pass preorder traversal update can now proceed as presented in Figure 4. The basis representation arrays are all modified on-the-fly during this traversal. The update of nodes on a newly created cycle (if any) includes establishing the new cycle factor FAC( ). Changes for P( ), D( ), IT( ), IVAR( ), X( ), and U( ) proceed analogously to the pure network case (e.g., GNET Bradley et al. 1977) with simple

Row \ Column	2	4	12	29	23	10	27	21	17	26	24	7	30	14	13
2	1	1	1	1	1	1									
5		-.74					*1*								
8			-.92												
14				-.98											
12					-1.18										
7						-1.06									
11								-1.17							
3	-1							1	1	1		1	1	1	
10									-1.07						
13							*-.88*			-1.0	-.83				
1											1				
6												-.91			
15													-1.0		
9														-1.0	-1.0
4															1

(The entering column is *italicized*.)

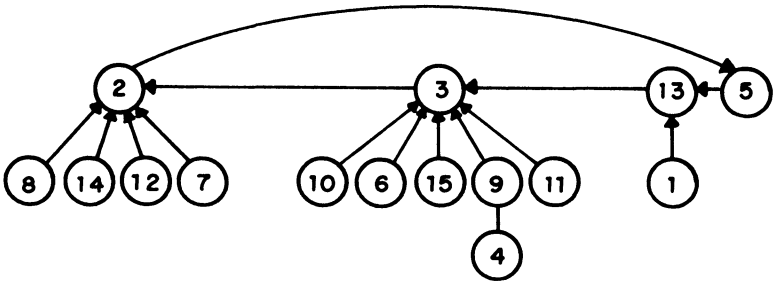


One-Tree

FIGURE 5. New Basis before Restoring Near-Triangulation (Entering Column 27, Arc (5, 13)).

Row \ Column	4	12	29	23	10	2	17	7	30	14	13	21	26	24	27
2	1	1	1	1	1	1									
8		-.92													
14			-.98												
12				-1.18											
7					-1.06										
3						-1.0	1	1	1	1		1	1		
10							-1.07								
6								-.91							
15									-1.0						
9										-1.0	-1.0				
4											1				
11												-1.17			
13													-1.0	-.83	-.88
1														1	
5	-.74														1

A Preorder Near-Triangulation.



One-Tree

FIGURE 6. New Basis Near-Triangulated.

Node	Predecessor	Depth	Traversal	Basic Variable		Dual	Cycle Factor
	P( )	D( )	IT( )	IVAR( )	X( )	U( )	FAC( )
1	13	*1*	*5*	24	22.860	*−17.026*	*
2	*5*	0	*8*	*4*	*3.883*	*6.557*	*0.3488*
3	*−2*	0	10	*2*	*118.456*	*−8.913*	*0.3488*
4	9	2	*11*	13	0.0	*−35.173*	*
5	*13*	*0*	*2*	*27*	*2.872*	*−27.302*	*0.3488*
6	−3	1	*15*	7	21.308	*−48.388*	*
7	−2	1	*3*	10	3.434	*−77.192*	*
8	−2	1	14	12	27.087	*−16.634*	*
9	−3	1	4	14	9.380	*−38.633*	*
10	−3	1	*6*	17	13.150	*−9.330*	*
11	*−3*	*1*	*13*	*21*	*48.641*	*−50.746*	*
12	−2	1	7	23	2.076	*−1.969*	*
13	−3	*0*	1	26	*9.428*	*−54.513*	*0.3488*
14	−2	1	12	29	22.204	*−11.834*	*
15	−3	1	9	20	16.550	*−76.533*	*

FIGURE 7. GENNET Basis Representation Arrays (for Basis in Figure 5).

modification of generators for X( ) and U( ) to accommodate generalized network coefficients for basic arcs.

Figure 5 shows the new basis (derived from the example in Figure 2) *before* restoring near-triangulation with the update. A new cycle is to be created and the new cycle factor and a dual solution (for node 13 on the *i*-stem) are found at this stage.

Figure 6 displays the new basis restored to near-triangular form. At this point, all basis representation arrays are updated with the values shown in Figure 7 (data in *italics* have been changed by the update).

GENNET is designed to exploit intrinsic pure network structure commonly found embedded in generalized network problems. Note that when this algorithm is applied to pure network problems, *it automatically adapts to a minor variant of GNET*. Of course, GENNET uses floating point arithmetic operations which are intrinsically slower on most computers than the pure additive integer arithmetic of GNET (also, floating point arithmetic requires some extra editing for mantissa truncation errors).

To mitigate this disadvantage, GENNET can test logically for pure network arcs (with unit multipliers) and avoid unnecessary floating point multiplication and division operations.

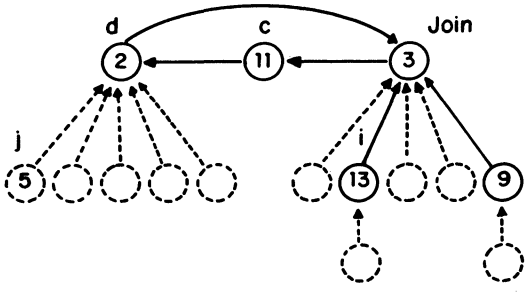
GENNET also employs an automatic dual basis aggregation refinement (Bradley et al. 1977, p. 26 ff). Explicit values for D( ), U( ) and IT( ) are maintained *only* for nodes with successors. An array, A( ), records for each node the current number of its *aggregated successors*. When an aggregated node is encountered in the priceout, its dual is generated from that of the immediate predecessor of the node. When a backpath of an entering arc begins with an aggregated node, it is disaggregated, and when the leaving arc isolates nodes with no successors, they are aggregated.

Figure 8 shows the arrays affected by the dual aggregation scheme for the basis in Figures 2 and 3. IT( ) indicates aggregated nodes with 0 entries, and these nodes have broken outlines in the one-tree depiction. The priceout of arc (5, 13) requires that U(5) be generated using the predecessor dual U(2). Node 5 subsequently starts the *j*-backpath and is disaggregated. The update leaves node 11 with no successors, and thus aggregated.

5. Computational Experience

Significant design alternatives for GENNET have each been evaluated by extensive experimentation at large scale. Illustrative computational experience is abstracted in

Node	Depth D( )	Traversal IT( )	Dual U( )	Aggregated Successors A( )
1		0		0
2	0	3	47.148	5
3	0	13	31.678	3
4		0		0
5		0		0
6		0		0
7		0		0
8		0		0
9	1	11	9.380	1
10		0		0
11	0	2	4.170	0
12		0		0
13	1	9	11.956	1
14		0		0
15		0		0



Aggregated One-Tree

FIGURE 8. Aggregated Basis Representation (for Figures 2 and 3).

this section for some of the prototypic systems tested. Departing somewhat from the style of the paper documenting such work for GNET (Bradley et al. 1977), relative performance is reported even for some competitive design features subsequently rejected for adoption (some readers of Bradley et al. 1977 have concluded, quite incorrectly, that only those features reported for GNET were tested). This should help other researchers avoid our mistakes, and may even change some widely held misconceptions and correct a few translation errors in textbooks.

Among the key issues to be resolved are:

(a) *Static Storage of Arcs*. The arc lists can be stored in arbitrary sequence, or, to save space, arcs can be stored contiguously by tail node, or by head node, thereby replacing an arc-length index array by a (presumably much shorter) hierarchical node-length entry point array.

(b) *Preorder Manipulation of Basis*. The triple label (augmented predecessor index) method (Glover et al. 1972, 1973) will be presented and compared with the preorder traversal method.

(c) *Basis Aggregation*. An aggregated basis representation will compete with an explicit representation.

(d) *Pricing Schemes*. Candidate list schemes and explicit arc pricing mechanisms widely used in general linear programming systems will vie with dynamic candidate queue disciplines.

Problem	Nodes	Sources	Sinks	Arcs	Percent Capacitated
NETGEN				(Pure)	
NG15	400	200	200	4,500	0
NG18	400	8	60	1,306	20
NG19	400	8	60	2,443	20
NG22	400	8	60	1,416	40
NG23	400	8	60	2,836	40
NG26	400	4	12	1,382	80
NG27*	400	4	12	2,676	80
NG28	1,000	50	50	2,900	0
NG29	1,000	50	50	3,400	0
NG30	1,000	50	50	4,400	0
NG31	1,000	50	50	4,800	0
NG32	1,500	75	75	4,342	0
NG33	1,500	75	75	4,385	0
NG34	1,500	75	75	5,107	0
NG35	1,500	75	75	5,730	0
NETGENG				(Generalized)	
GT01	200	100	100	1,500	0
GT02	200	100	100	2,000	100
GT07	300	135	115	4,000	0
GT12	400	20	100	5,000	0
GT15	1,000	5	995	4,000	100
GT16	1,000	20	100	6,000	100
GT18	1,000	30	400	7,000	0

FIGURE 9. Some Benchmark Problems (Problem NG27 is extensively studied in Bradley et al. 1977).

(e) *Pure Network Specialization.* Generalized network algorithms would ideally adapt to pure networks with efficiency comparable to pure network codes.

(f) *Starting, Tuning and Tailoring.* Which algorithm parameters and settings lead to high efficiency for interesting classes of problems? Are heuristics for advanced starting solutions worthwhile?

(g) *Generalizations.* Advanced features and generalizations will be suggested.

Computational tests have been made with many problems, including a benchmark suite of pure network problems generated with NETGEN (Klingman et al. 1974), and generalized network problems generated by NETGENG (Glover et al. 1977, 1978). Figure 9 gives some problem characteristics.

Static arc storage has been implemented in three ways:

- Contiguous by head node with hierarchical node-length entry point array,
- Contiguous by tail node, with hierarchical node-length entry point array, and
- Explicit arcs in arbitrary order.

In competition with our basis manipulation using preorder traversal, a triple label representation originally suggested by E. Johnson (1966) has been implemented for pure networks and called the augmented predecessor indexing method by Glover, Karney and Klingman (1972). Glover, Klingman and Stutz (1973) report that the method has been extended to generalized networks, but reveal no details.

We have implemented our own efficient version of the triple label scheme.

The triple label representation uses predecessor, successor, and brother pointers for each node. Figure 10 shows these arrays for the basis in Figure 2.

To briefly illustrate the triple label scheme for generalized networks, let our situation be the same as for the preorder example (the j-backpath of the incoming arc is arranged to include the outgoing arc and to encounter node c first on that backpath). Let $v_{\underline{d}+1} = i$, and the backpath from j to c, $v_{\underline{d}}, v_{\underline{d}-1}, \ldots, c$. The skeletal update

Node	Predecessor P( )	Successor S( )	Brother B( )
1	13	0	0
2	3	11	10
3	11	2	0
4	9	0	0
5	−2	0	8
6	−3	0	15
7	−2	0	0
8	−2	0	14
9	−3	4	0
10	−3	0	13
11	−2	3	5
12	−2	0	7
13	−3	1	6
14	−2	0	12
15	−3	0	9

FIGURE 10. Triple Label Basis Representation (for Figure 2).

scheme is:

1. Set $\delta = \underline{d}$.
2. If $S(v_{\delta-1}) = v_\delta$, *go to Step 3.*

Otherwise, if possible, find a node v^* such that $P(v^*) = v_{\delta-1}$ *and* $B(v^*) = v_\delta$, and set $B(v^*) \leftarrow B(v_\delta)$, *Go to Step 4.*

3. Set $S(v_{\delta-1}) \leftarrow B(v_\delta)$.
4. Set $P(v_\delta) \leftarrow v_{\delta+1}$, $B(v_\delta) \leftarrow S(v_{\delta+1})$, $S(v_{\delta+1}) \leftarrow v_\delta$.
5. Set $\delta \leftarrow \delta - 1$. Then, if $v_{\delta+1} \neq c$, *Go to Step 2.* Otherwise, *Stop.*

(Step 2 exhibits the key extension for generalized networks of the pure network triple label scheme.)

These five steps reverse the orientation of all arcs on the j-stem and orient $(v_{\underline{d}+1}, v_{\underline{d}})$ so that it begins this redirected path. All other triple label operations are obvious alterations of the preorder traversal procedure.

A static *candidate list* pricing strategy (e.g., Glover et al. 1977, 1978, Mulvey 1978) an *explicit arc pricing* method reminiscent of general linear programming systems, and a dynamic *candidate queue* (Bradley et al. 1977) have been tested.

The (L_1, L_2) candidate list procedure is a simple strategy. Nodes with leaving arcs (or entering arcs for contiguous head node arc lists) are sequentially priced, placing the most negative candidate (if any) from each node on the candidate list until L_2 candidates have been located (arbitrarily organized arc lists require explicit arc pricing). Entering arcs are chosen from the candidate list by most negative reduced cost until L_1 iterations have been carried out, or until all list entries have nonnegative reduced costs. Arcs with nonnegative reduced costs are dropped from the list and replaced by continuing to scan the nodes until L_2 candidates have been found. If an exhaustive pass through the nodes results in less than L_2 candidates, then an optional closing gambit sets L_2 equal to the actual candidates found, and reduces L_1 by half unless L_1 equals one. Glover et al. (1977, 1978) report (L_1, L_2) of (5, 10) to be best in their work.

The candidate queue is a dynamic list of interesting arcs *and* nodes, scanned in a cyclic manner. The entering arc is selected from the queue by pricing NNE entries; if an interesting node is encountered it is replaced by its best-priced entering arc (or leaving arc for contiguous tail node arc lists). Arcs pricing favorably are retained in the queue. When the end of queue is encountered, the queue is refreshed by pricing IPG nodes in a cyclic general arc scan. During an opening gambit of NNS pivots, the nodes incident to the entering basic arc are added to the queue. There is no closing gambit,

Problem NETGEN	(start) (5, 10) TLA	Generalized Network Codes NNE = 32 TQA	NNE = 32 HQA	NNE = 32 HQP	NNE = 8 HQPX	(Big-M)	Pure Network Codes GNET NNE = 16 HQP	GNET NNE = 8 HQPX	SUPERK
NG15	2.19	2.17	3.10	2.50	2.07	1.5	1.37	1.17	2.15
NG18	0.83	0.60	0.63	0.63	0.44	2.5	0.42	0.39	1.30
NG19	1.71	1.10	0.90	0.79	0.70	2.0	0.67	0.56	2.36
NG22	0.97	0.61	0.57	0.64	0.46	2.0	0.43	0.38	1.54
NG23	1.60	0.76	0.84	0.63	0.71	1.5	0.48	0.43	2.11
NG26	1.03	0.48	0.54	0.52	0.45	2.0	0.36	0.30	0.93
NG27	1.11	0.74	0.78	0.71	0.59	1.5	0.48	0.47	1.17
NG28	4.42	2.20	2.71	2.13	1.67	2.5	1.41	1.36	4.53
NG29	5.96	2.32	3.12	1.96	1.95	2.0	1.51	1.40	4.42
NG30	4.52	2.58	3.00	2.27	2.30	1.5	1.67	1.45	5.58
NG31	5.46	2.58	3.40	2.27	2.53	1.5	1.69	1.58	5.47
NG32	9.79	3.32	4.42	3.32	3.12	2.0	2.56	2.28	7.92
NG33	9.89	4.18	4.46	3.15	3.49	2.5	2.61	2.58	6.83
NG34	9.22	4.24	4.83	3.47	3.53	1.5	2.94	2.44	8.94
NG35	10.18	3.83	5.67	3.59	3.90	1.5	2.77	3.00	9.32
Seconds:	68.88	31.71	38.97	28.58	27.91		21.37	19.79	64.57
Rank:	5	3	4	2	1				

Legend: Head/Tail arc organization
List/Queue pricing mechanism
API triple label/Preorder traversal
X aggregated successors

FIGURE 11. Pure Network Test Problems.

Problem NETGENG	(start) (5, 10) TLA	(5, 10) TLA	NNE = 32 TQA	(5, 10) ELPX	(start) NNE = 32 HQA	NNE = 32 HQA	NNE = 32 HQP	GENNET NNE = 16 HQPX	(Big-M)
GT01	1.48	1.65	2.08	2.73	1.52	2.17	2.00	2.14	1.5
GT02	3.96	4.12	3.80	6.41	3.61	3.75	4.17	3.99	1.5
GT07	2.30	2.91	3.03	0.78	1.63	0.94	0.82	0.56	2
GT12	3.62	4.59	3.34	4.01	3.66	3.49	3.02	2.77	1.5
GT15	13.41	24.14	28.99	6.46	9.92	8.43	7.94	3.14	1.5
GT16	9.98	17.84	7.47	12.08	12.16	9.84	5.77	5.83	1.5
GT18	12.71	19.19	13.63	17.89	13.89	13.05	10.46	9.72	1.5
Seconds:	47.46	74.44	62.34	50.36	46.39	41.67	34.18	28.15	
Rank:	5	8	7	6	4	3	2	1	

Legend: Head/Tail/Explicit arc organization
List/Queue pricing mechanism
API triple label/Preorder traversal
X aggregated successors

FIGURE 12. Generalized Transshipment (GT) Test Problems.

since the queue automatically shrinks and finally collapses at optimality. Bradley, Brown, and Graves (1977) suggest NNE = 32, NNS = $3m/4$ and IPG = $m/10 + 1$ for pure network problems with m nodes.

A rule to break ties in the ratio test which guarantees finite convergence for pure transshipment problems has been developed by Cunningham (1976). Bradley, Brown and Graves (1977) show that the conditions necessary for finite convergence are naturally satisfied by GNET on over 90% of its degenerate pivots. Elam, Glover and Klingman (1979) have observed that the results of Cunningham can only be extended to the generalized network case when the multipliers are positive. We have not used Cunningham's modification.

Bland (1977) presents a class of restrictions of pricing and ratio tests for general linear programs which relies exclusively on primal simplex representation and guarantees finite convergence. These rules are easily modified to produce an efficient finite simplex algorithm. The modification interferes with effective pricing strategies only during degenerate pivot sequences, and the restrictions increase in severity only with the number of pivots in that sequence. However, during a degenerate pivot sequence restriction records must be accumulated (e.g., a list with each incoming variable in one of our schemes). This record is naturally accommodated by the dynamic candidate queue, but not by a static candidate list or explicit pricing. No purpose is served by reporting such unbalanced competition.

A *starting strategy* has also been tested in conjunction with pricing alternatives. A straightforward starting method examines each node with supply (or demand for contiguous arc storage by head node) and assigns as much flow as possible to its least cost leaving (entering) arc. The procedure stops when an exhaustive pass of the nodes makes no additional flow assignments. The starting solution achieved is not necessarily feasible (e.g., "exhaustive pass sequential source minimum start" Glover et al. 1978).

Artificial arcs are driven from the basis in all *experiments* using a Big-M method (e.g., Bradley et al. 1977). This choice is principally motivated by the comparability of competitive tests between pure and generalized network codes on pure and generalized network problems. (A two-phase method is employed in production use.)

Choosing the best Big-M value is a bit tricky. The smallest Big-M value which yields a feasible optimal solution (if one exists) is best in our experience. Small Big-M values may fail to produce feasibility, and large values inflict numerical difficulties. In practice, a default value is used and an automatic restart recovery is applied if an infeasible solution persists. If a restart with a higher Big-M value fails to reduce the total infeasibility, a terminally infeasible solution is declared. Figures 11 and 12 indicate the multiple of maximum absolute arc cost used for Big-M in each problem.

Computational tests have been performed on various computer systems. The times reported here are accurate to the precision displayed for IBM 370/168-3 using the FORTRAN H compiler with OPT(2). Solution times exclude input/output overhead. GENNET uses double precision (IBM REAL*8) arithmetic for floating point operations and storage.

Solution times are given in Figure 11 for the pure network test problems. Performance is given for three pure network codes (two versions of GNET and SUPERK) as well as for several representative generalized network prototypic systems. We can thus compare the best generalized network scheme with the best pure network code exhibiting equivalent features (GNET, or its aggregated successor variant (Bradley et al. 1977).

The times for SUPERK also provide the only available objective means for comparison of our implementations to that of Glover, Hultz, Klingman and Stutz (1973); they report that their generalized network code, NETG, is about as fast as SUPERK (a fast out-of-kilter code for pure networks) when both are used to solve

Problem	Nodes	Arcs	Seconds	Pivots
AIRLP	170	3,040	2.62	420
COAL	170	3,923	1.80	471
STEEL	422	1,279	0.39	499
FOAM	951	4,953	3.74	1,258
ODSAS(GN)	1,431	4,615	3.22	1,427
ALUMINUM(GN)	2,178	7,216	3.57	2,794
REFINE(GN)	3,110	6,617	4.72	3,322
FOOD(GN)	3,716	13,907	12.11	7,004

(Big-M = 10 × largest cost coefficient)

FIGURE 13. (GN) Test Problems (GN rows extracted from real-world LP/MIP models Brown, McBride and Wood to appear with null columns deleted and slack arcs added).

pure networks. Using *their* version of SUPERK on *our* computer we have shown that *our* implementation of NETG (called TLA in our nomenclature) is at least as efficient as their claims for NETG.

However, in these tests TLA is substantially outperformed by the alternate systems. For the pure network problems, best performance is achieved by (Figure 11):

Contiguous arcs by head node,
Candidate queue pricing,
Preorder traversal, and
Aggregated successors.

TLA does not incorporate any of these features, and is generally less than half as fast as competitors.

Although GENNET (HQPX) should in theory rival GNET (HQPX) with pure network problems, the overhead of testing in GENNET for more general basis structure and the additional computational burden of floating point arithmetic exact a performance penalty of about 15 percent.

Figure 12 shows solution times for the generalized transshipment network problems. Note that the starting strategy helps candidate list performance and hinders the candidate queue. Arranging arcs contiguously by head node dominates both tail node and explicit arc list designs. The candidate queue provides good performance *if* accompanied with contiguous arcs by head node. Preorder traversal continues to provide better performance than triple label representation in all design contexts. Aggregated successors offer a pronounced advantage.

GENNET (HQPX) provides best overall performance. It offers a decided advantage on problems with many more sinks than sources, a situation common in real life.

Figure 13 displays performance of GENNET (GN, HPQX) applied to a set of (GN) problems extracted from a collection of real-world LP/MIP models (Brown, McBride and Wood to appear). Despite the slight additional floating point arithmetic required to solve (GN), GENNET solves these problems much faster than would be predicted by experience with randomly generated (GT) problems. Tuning of the pricing mechanism greatly enhances this difference.

Close scrutiny of solution trajectories lends some insight into GENNET's good performance. GENNET has a one-pass iteration *unless* a cycle is formed; a cycle is formed on only 5-to-24 percent of all iterations for these problem sets—5-to-10 percent for most problems. Also, the explicit (nonaggregated) subset of the nodes is remarkably small, seldom numbering much more than the number of source nodes. Finally, the length of backpaths is quite short, averaging about the number of echelons (path length from sources to sinks) in the model, or just more than 2 in these problems.

6. Conclusion

The generalized network system GENNET is small, fast and easy to modify. Adaptations have already included using GENNET in a system to solve generalized networks with complicating side constraints

and/or complicating variables (McBride to appear). GENNET has also been incorporated in a powerful microcomputer-based network optimization system by Brown, Duff and Finley (1982) and Finley (1982) using an APPLE-II host and PASCAL implementation language. Modifications for mixed integer generalized networks have also been tested (though not with care sufficient to warrant publication at this time). GENNET has proven to be a worthy successor of GNET (Bradley et al. 1977).

Preorder traversal is appealing for its mathematical and implementation elegance, and has proven to be efficient and flexible for generalized networks (as it was for pure networks). (Adolphson and Heum 1981 have also suspected this and have independently pursued this avenue.)

Experience shows that the GENNET design performs much more efficiently on real models than on randomly generated test problems of nominally equivalent size; this design is also technically and philosophically compatible with the various systems we have devised for solving other more general classes of optimization models.

The FORTRAN programs GENNET—(GT) and (GN) versions—(Copyright 1984) are licensed to researchers for a nominal charge on an exclusive use basis. For further information write the authors via P.O. Box 1832, Alexandria, Virginia 22314, USA.

Acknowledgement

Gordon Bradley and Glenn Graves have always offered us complete support in our work: we gratefully acknowledge their help and encouragement. Rick Rosenthal contributed many valuable suggestions to clarify the presentation. John Tomlin has discovered some subtle imprecisions in our description. Kevin Wood has taught from early manuscripts and suffered their shortcomings nobly.

References

ADOLPHSON, D. AND L. HEUM, "Computational Experiments on a Threaded Index Generalized Network Code," presented at the Houston ORSA/TIMS Meeting, October 14, 1981.

BALACHANDRAN, V., "An Integer Generalized Transportation Model for Optimal Job Assignment in Computer Networks," *Oper. Res.*, 24, 4 (1976), 742–759.

BARR, R., F. GLOVER, AND D. KLINGMAN, "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," *Math. Programming*, 7, 1 (1974), 60–87.

BLAND, R., "New Finite Pivoting Rules for the Simplex Method," *Math. Oper. Res.*, 2, 2 (1977), 103–107.

BRADLEY, G., "Survey of Deterministic Networks," *AIIE Trans.*, 7, 3 (1975), 222–234.

———, G. BROWN AND G. GRAVES, "Design and Implementation of Large Scale Primal Transshipment Algorithms," *Management Sci.*, 24, 1 (1977), 1–34.

BROWN, G., R. DUFF AND M. FINLEY, "Design and Demonstration of a Microcomputer-Based Network Optimization System," presented and demonstrated at the Detroit ORSA/TIMS Meeting, April 19, 1982.

——— AND G. GRAVES, "Design and Implementation of a Large Scale (Mixed Integer, Nonlinear) Optimization System," paper presented at the Las Vegas ORSA/TIMS Meeting, November 1975.

——— AND R. MCBRIDE, "Exploiting Large-Scale Networks with Gains," paper presented at the Houston ORSA/TIMS Meeting, October 14, 1981.

———, ——— AND K. WOOD, "Extracting Embedded Generalized Networks from Linear Programming Problems," *Math. Programming* (to appear).

——— AND W. WRIGHT, "Automatic Identification of Embedded Network Rows in Large-Scale Optimization Models," *Math. Programming*, 29 (1984), 41–56.

CUNNINGHAM, W., "A Network Simplex Method," *Math. Programming*, 2, 2 (1976), 105–116.

DANTZIG, G., *Linear Programming and Extensions*, Princeton University Press, Princeton, N.J., 1963.

EISEMANN, D., "The Generalized Stepping Stone Method for the Machine Loading Model," *Management Sci.*, 11, 1 (1964), 154–177.

ELAM, J., F. GLOVER AND D. KLINGMAN, "A Strongly Convergent Primal Simplex Algorithm for Generalized Networks," *Math. Oper. Res.*, 4, 1 (1979), 39–59.

FINLEY, M., "An Extended Microcomputer-Based Network Optimization Package," MS Thesis, Naval Postgraduate School, Monterey, September 1982.

GLOVER, F., J. HULTZ, D. KLINGMAN, AND J. STUTZ, "A New Computer-Based Planning Tool," Research Report CCS 289, Center for Cybernetic Studies, University of Texas at Austin, 1977.

———, ———, ——— AND ———, "Generalized Networks: A Fundamental Computer-Based Planning Tool," *Management Sci.*, 24, 12 (1978), 1209–1220.

———, D. KLINGMAN, AND J. STUTZ, "Augmented Threaded Index Method for Network Optimization," *INFOR*, 12, 3 (1974), 293–298.

———, D. KARNEY AND D. KLINGMAN, "The Augmented Predecessor Index Method for Locating Stepping Stone Paths and Assigning Dual Prices in Distribution Problems," *Transportation Sci.*, 6, 2 (1972), 171–179.

——— AND D. KLINGMAN, "A Note on Computational Simplifications in Solving Generalized Transportation Problems," *Transportation Sci.*, 7, 4 (1973), 351–361.

———, ———, AND J. STUTZ, "Extensions of the Augmented Predecessor Index Method to Generalized Network Problems," *Transportation Sci.*, 7, 4 (1973), 377–384.

JENSEN, P. AND J. BARNES, *Network Flow Programming*, John Wiley and Sons, New York, 1980.

——— AND G. BHAUMIK, "A Flow Augmentation Approach to the Network With Gains Minimum Cost Flow Problem," *Management Sci.*, 23, 6 (1977), 631–643.

JEWELL, W., "Optimal Flow Through Networks with Gains," *Oper. Res.*, 10 (1962), 476–499.

JOHNSON, E., "Networks and Basic Solutions," *Oper. Res.*, 14, 4 (1966), 619–623.

KENNINGTON, J. AND R. HELGASON, *Algorithms for Network Programming*, John Wiley & Sons, New York, 1980.

KLINGMAN, D., A. NAPIER, AND J. STUTZ, "NETGEN—A Program for Generating Large Scale (Un) Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems," *Management Sci.*, 20, 5 (1974), 814–821.

MAURRAS, J., "Optimization of the Flow Through Networks with Gains," *Math. Programming*, 3 (1972), 135–144.

MCBRIDE, R., "Solving Embedded Generalized Network Problems," *European J. Oper. Res.* (to appear).

MULVEY, J., "Pivot Strategies for Primal-Simplex Network Codes," *J. Assoc. Comput. Mach.*, 25 (1978), 266–270.