Exceptional Paper

Design and Implementation of Large Scale Primal Transshipment Algorithms

Gordon H. Bradley; Gerald G. Brown; Glenn W. Graves

Management Science, Vol. 24, No. 1. (Sep., 1977), pp. 1-34.
Stable URL: http://links.jstor.org/sici?sici=0025-1909%28197709%2924%3A1%3C1%3ADAIOLS%3E2.0.CO%3B2-S

Abstract

A complete 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 transshipment problems solved is the most general of the minimum cost network flow models which include the capacitated and uncapacitated transportation problems and the classical assignment problem; these formulations are used for a large number of diverse applications to determine how (or at what rate) a good should flow through the arcs of a network to minimize total shipment costs. The presentation tailors the unified mathematical framework of linear programming to networks with special emphasis on data structures which are not only useful for basis representation, basis manipulation, and pricing mechanisms, but which also seem to be fundamental in general mathematical programming. A review of pertinent optimization literature accompanies computational testing of the most promising ideas. Tuning experiments for the network system, GNET, are reported along with important extensions such as exploitation of special problem structure, element generation techniques, postoptimality analysis, operation with problem generators and external problem files, and a simple noncycling pivot selection procedure which guarantees finiteness for the algorithm.

Design and Implementation of Large Scale Primal Transshipment Algorithms

Gordon H. Bradley; Gerald G. Brown; Glenn W. Graves

Management Science, Vol. 24, No. 1 (Sep., 1977), 1-34.

Stable URL:
http://links.jstor.org/sici?sici=0025-1909%28197709%2924%3A1%3C1%3ADAIOLS%3E2.0.CO%3B2-S

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.

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.

Management Science is published by INFORMS. Please contact the publisher for further permissions regarding the use of this work. Publisher contact information may be obtained at http://www.jstor.org/journals/informs.html.

Management Science
©1977 INFORMS

JSTOR and the JSTOR logo are trademarks of JSTOR, and are Registered in the U.S. Patent and Trademark Office. For more information on JSTOR contact jstor-info@umich.edu.

©2003 JSTOR

http://www.jstor.org/
Thu Sep 4 12:16:38 2003

MANAGEMENT SCIENCE
Vol. 24, No. 1, September 1977
Printed in U.S.A.

Exceptional Paper

DESIGN AND IMPLEMENTATION OF LARGE SCALE PRIMAL TRANSSHIPMENT ALGORITHMS*†

GORDON H. BRADLEY‡, GERALD G. BROWN‡ AND GLENN W. GRAVES§

A complete 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 transshipment problems solved is the most general of the minimum cost network flow models which include the capacitated and uncapacitated transportation problems and the classical assignment problem; these formulations are used for a large number of diverse applications to determine how (or at what rate) a good should flow through the arcs of a network to minimize total shipment costs. The presentation tailors the unified mathematical framework of linear programming to networks with special emphasis on data structures which are not only useful for basis representation, basis manipulation, and pricing mechanisms, but which also seem to be fundamental in general mathematical programming. A review of pertinent optimization literature accompanies computational testing of the most promising ideas. Tuning experiments for the network system, GNET, are reported along with important extensions such as exploitation of special problem structure, element generation techniques, postoptimality analysis, operation with problem generators and external problem files, and a simple noncycling pivot selection procedure which guarantees finiteness for the algorithm.

Introduction

This paper reports the development of a large scale primal network code that is possibly the fastest and most efficient program currently available for solving capacitated transshipment problems. The capacitated transshipment problem is the most general of the minimum cost flow models which include the capacitated and uncapacitated transportation problems and the personnel assignment problem. These models are used for a large number of diverse applications that include transportation of goods, design of communications and pipeline systems, assignment of men to jobs, bid evaluation and production planning. For further discussion of these applications see the survey articles of Bradley [5], Elmaghraby [18] and Fulkerson [22] and the textbooks of Busacker and Saaty [8], Charnes and Cooper [9], Dantzig [15] and Ford and Fulkerson [20].

The capacitated transshipment model and its specializations are minimum cost network flow problems. The goal is to determine how (or at what rate) a good should flow through the arcs of a network to minimize shipment costs. The network is a directed graph defined by a set of nodes, $\boldsymbol{N}$, and a set of arcs, $\boldsymbol{A}$, with ordered pairs of nodes (tail, head) as elements indexed by 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 capacity), u_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

* This paper has been refereed under guidelines for exceptional contributions.

† Accepted by Arthur M. Geoffrion; received November 10, 1976. This paper has been with the authors $1\frac{1}{2}$ months, for 1 revision.

‡ Naval Postgraduate School, Monterey.
§ University of California, Los Angeles.

Copyright © 1977, The Institute of Management Sciences

bounds and capacities and preserve the conservation of flow at each node:

$$\min \sum_{k \in A} c_k x_k$$

$$\text{s.t.} \sum_{k \in A \text{ with tail } i} x_k - \sum_{k \in A \text{ with head } i} x_k = b_i, \qquad i \in N, \tag{1}$$

$$l_k \leqslant x_k \leqslant u_k, \qquad k \in A,$$

where $b_i = \{$supply if i is a supply node; $-$demand if i is a demand node; 0 otherwise$\}$.

We choose this notation to emphasize that data will be stored only for arcs that are present in the network, and operations are defined only for use with these arcs. The usual textbook notation with, for instance, $\sum_i \sum_j x_{ij}$ is particularly inappropriate since for large practical problems it is seldom true that all node pairs are connected by an arc. Further, our notation allows multiple arcs (connecting any given pair of nodes) which are common in practical problems. This notation is also consistent with the input requirements of all contemporary large scale network optimization and linear programming codes.

These models are widely used because they accurately describe a variety of important applications. There has been a recent surge of interest in such models because more efficient computer programs have made possible the economic solution of problems with more variables than virtually any other optimization technique. Networks are also popular because they are more readily accepted by nonanalysts than perhaps any other class of operations research model.

These models can be solved as linear programming problems with a constraint for each node and a variable for each arc. For large scale problems, contemporary commercial linear programming codes require 50–200 times as much computer time and considerably more space for data storage than special purpose network flow algorithms.

Although many 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 GNET, an extremely efficient yet relatively simple code. An important objective of this paper is to make these new approaches easily accessible to a wide audience via a clear mathematical exposition and a concrete example of a highly efficient FORTRAN program. Further, the availability of the computer program will now make it possible for other investigators to reproduce and extend the experimental results.

The transportation model was originally proposed by Hitchcock [39] in 1941 and Koopmans [44] in 1946. Both presented computational methods that would now be called "primal simplex." Hitchcock shows that an optimal solution will be an extreme point solution and shows how to iteratively construct better extreme point solutions. He notes alternate optimal solutions and degeneracy (all in $6\frac{1}{4}$ pages). Koopmans develops simplex multipliers, or "node potentials" and the optimality criterion and he shows that an extreme point is equivalent to a tree. Dantzig [14] in 1951 showed how the transportation problem can be solved by his simplex algorithm; he also developed a special variant of the simplex algorithm for the transportation problem. Orden [52] shows that these results can be extended to the transshipment problem.

Approaches other than the primal simplex have been proposed including out-of-kilter (Fulkerson [21]), primal-dual (Ford and Fulkerson [19]), dual (Balas and

Hammer [1]), path (Busacker and Gowen [7]), negative cycle (Klein [42]) and scaling (Edmonds and Karp [17]). Also, special algorithms have been developed for the assignment, shortest route and maximum flow problems.

The contemporary implementations of the primal simplex network algorithm are based on compact representation of the basis, determination of the outgoing arc without trial and error and efficient techniques to update the simplex multipliers at each pivot. Some of these developments were suggested in the 1960's. Glicksman, L. Johnson and Eselson [27] describe a transportation problem with few sources and many sinks; their data structure may be viewed as storing the basis as an arborescence and using this structure to efficiently find the outgoing arc. E. Johnson [40] describes a triple label scheme that represents the basis as an arborescence and allows the simplex multipliers to be updated efficiently. Johnson describes his work as a modification of the proposal of Scoins [55].

The proposals of Glicksman, L. Johnson and Eselson [27], Scoins [55] and E. Johnson [40] for primal algorithms were not immediately pursued; the most widely known code of that decade is an out-of-kilter implementation by Clasen [12]. The contemporary work on network algorithms was begun in 1970 by Srinivasan and Thompson [59], [60] Glover, Karney and Klingman [28] and Glover, Karney, Klingman and Napier [29]. This work was a break with the past in that:

1. Primal algorithms were considered despite all the experiments in the 1950's and early 1960's that showed the apparent superiority of the out-of-kilter algorithm.
2. Contemporary computer science tools that had not been available a decade earlier were used.
3. Computer codes were developed for much larger problems.

Later and independently, McBride [48] and Graves and McBride [34] specialized their work on factorization of linear programs to transshipment problems. Although their development is quite different, the network specialization of their data structures is similar in many respects to data structures that evolved from a graph theoretic view of networks. Mulvey [49] has developed an efficient large scale primal code at TRW. Harris [36] has developed a primal algorithm for transportation problems with many sinks and few sources. Langley, Kennington and Shetty [46] have also developed a primal transshipment code.

A significant aspect of contemporary network research has been the computational testing of different algorithms on large standard test problems. One major topic has been primal algorithms versus out-of-kilter algorithms. Experiments in the 1950's and early 1960's convinced researchers that the out-of-kilter algorithm was superior, especially for transshipment problems. The most comprehensive recent comparison has been done by Glover, Karney and Klingman [28] and Barr, Glover and Klingman [2] who compare the algorithms on a diverse set of test problems [43]. Their primal code is from 30% (for transshipment) to 40% (for transportation) faster than their out-of-kilter code. The success of the primal algorithm has been independently verified by the experiments of others. Most researchers now believe that the primal algorithm is superior to others including the out-of-kilter algorithm (an exception is Hatch [38]). Current primal implementations are faster, require less storage, are more suitable when using secondary storage devices and are compatible as embedded parts of more general optimization systems.

The Approach

The approach we have used in this research is to study the data structures which seem to be most fundamental in the sense that they can be applied to many types of mathematical programming situations. In this context, we view the major advances over the last thirty years in efficient solution of large linear programming problems

(for example: revised simplex, product form inverse, bounded variables, generalized upper bounding, factorization, sparse matrix methods, etc.) as major changes in the representation of the data accompanied only by necessary modifications of the simplex procedure to accommodate these new data structures. The computational breakthroughs in primal network codes are also due to improvements in data representation and renew our interest in the subtle relationships between the algorithms and the data structures employed for implementation.

We view networks as an important special case of large scale linear programming. Historically this has been the perspective; however, in most recent literature, networks have been treated as a distinct research area with little reference to the notation and mathematical development of linear programming. This "network approach" is based on graph theory notation and results. We find this to be a confining view, as evidenced by the arduous nature of extensions of the graph theoretic basis tree results to more general mathematical programming models. Our approach is to develop the framework of a general linear programming problem and then to specialize it to primal network models. Therefore, it is necessary to build a sufficient mathematical foundation to answer the question: "What is the capacitated transshipment problem a specialization of?" rather than, "How can we generalize basis trees?" Thus we invite the reader to view this paper as drawing from a general theory of large scale mathematical programming for which the network algorithms are concrete realizations of a rich algebraic view of a problem with special structure. Although our approach is different from most recent work, we establish the graph theoretic interpretation for the pure network problem and thus can easily show the relationship of our work to that of others. For expository purposes we will draw from both linear algebra and graph theory using pictures and terminology consistent with past literature. A companion paper [32] takes a purely algebraic approach that shows via lattice theory the full generality of the basis permutation triangulation done here only for pure networks.

Many applications involve networks as an embedded part of a more general optimization problem. These models, which may involve additional constraints, nonlinear objective functions, decomposition, Lagrangian relaxation, etc., require careful coordination of the primal variables, dual variables, and bases of the network problem(s) with those of the general problem. A general linear programming framework and notation gives explicit relationships between the network and the remainder of the optimization problem. This approach facilitates development of new algorithms and also provides a unified framework for this work and the authors' parallel research on general large scale mathematical programming systems.

We continue with a brief algebraic description of the general bounded variable simplex algorithm and several commonly used implementation options. The algebraic specialization of the simplex method for pure network problems is then presented. After these necessary but somewhat mathematically involved sections, the specific design decisions and experiments carried out with GNET are described, including computational evidence which indicates that the code produced by this approach compares very favorably with other algorithms, primal or otherwise. Several extensions of GNET are presented, including a noncycling pivot selection procedure, codes tailored to capacitated and uncapacitated transportation problems, and other variants to exploit special network structure. Postoptimal and reoptimization procedures using GNET are discussed. Finally, a review of the literature traces the original contributions found to be of fundamental importance in this work.

Primal Simplex Algorithm

In this section we briefly review the mathematical underpinnings of the bounded variable revised simplex method. The notation and well-known results are necessary

for the network specialization that is described in the next section. In order to maintain a broad scope, the presentation intentionally avoids tangential issues of specific methods and implementations. Rather, the class of algebraic algorithms is characterized with only the briefest indication of options often employed for actual linear programming codes. Small italic letters will denote column vectors, and a prime indicates transpose. Large letters denote matrices; superscipts denote columns and subscripts denote rows.

Consider

$$\min c'x \text{ s.t. } Ax = b \quad \text{and} \quad 0 \leqslant x \leqslant u; \tag{2}$$

where A is viewed for the present as a matrix of technological coefficients.

As a practical matter, lower bounds on the variables in (1) have been eliminated by transformation.

The upper bounds, u, are most readily accommodated implicitly. Whenever x_k reaches its upper bound, it is logically replaced (*reflected*) by $u_k - x_k$; column k is logically treated as if its sign has changed and the explicit right-hand side is transformed to $b - A^k u_k$. If a record is kept of each variable that is at its upper bound, the original problem solution is easily recovered.

By construction (possibly including introduction of unit vectors representing slack and artificial variables) A may be partitioned $A = (B, N)$, with B an $m \times m$ matrix of linearly independent columns which is called a basis.

Given a feasible basis there always exists a unique $\hat{x}$ such that

$$B\hat{x} = b. \tag{3}$$

In terms of this $\hat{x}$ there is always a current basic solution $x^0 = \binom{\hat{x}}{0}$ and, upon partitioning c in the same manner as A, one has

$$c'x^0 = (c'_B, c'_N)\begin{pmatrix} \hat{x} \\ 0 \end{pmatrix} = c'_B\hat{x}. \tag{4}$$

Any generic solution x satisfying the constraints can be rewritten in terms of the basic solution:

$$Ax = (B, N)\begin{pmatrix} x_B \\ x_N \end{pmatrix} = Bx_B + Nx_N = b. \tag{5}$$

Further, since B is a basis, there exists a unique transformation Z such that

$$BZ = N. \tag{6}$$

Multiplying (6) by x_N and subtracting from (3) yields

$$B(\hat{x} - Zx_N) + Nx_N = b, \tag{7}$$

and subtracting (7) from (5) yields $B(x_B - [\hat{x} - Zx_n]) = 0$. Since the columns of B are linearly independent, $x_B = \hat{x} - Zx_N$ and the general solution becomes

$$x = \begin{pmatrix} x_B \\ x_N \end{pmatrix} = \begin{pmatrix} \hat{x} - Zx_N \\ x_N \end{pmatrix}. \tag{8}$$

With this form it is easy to compare the value of x to any current solution x^0 and identify an improved solution when one exists. The value of the generic solution (use (4) and (6)) is

$$\begin{aligned} c'x &= c'_B x_B + c'_N x_N = c'_B(\hat{x} - Zx_N) + c'_N x_N \\ &= c'_B\hat{x} + (c'_N - c'_B Z)x_N \\ &= c'_B\hat{x} + (c'_N - \mathrm{u}'N)x_N; \end{aligned} \tag{9}$$

where u (often called the dual solution or simplex multipliers) is the solution of

$$\mathrm{u}'B = c'_B. \tag{10}$$

From (9) it is clear (since $x_N \geqslant 0$) that a necessary condition for an improved solution is that there exist a column of N, N^k, such that

$$c_k - \mathrm{u}'N^k < 0. \tag{11}$$

With such a column consider a specific vector obtained from (8) by taking all components of x_N zero except x_k,

$$x' = (\hat{x} - Z^k x_k, 0, \ldots, x_k, \ldots, 0). \tag{12}$$

As a function of $x_k \geqslant 0$, this vector must by its derivation satisfy the explicit constraints, $Ax = b$; feasibility also requires satisfaction of the bounds, $0 \leqslant x \leqslant u$. For components $z_{ik} > 0$ this requires $\hat{x}_i - z_{ik}x_k \geqslant 0$, or

$$x_k \leqslant \hat{x}_i / z_{ik}. \tag{13}$$

For components $z_{ik} < 0$ this requires $\hat{x}_i - z_{ik}x_k \leqslant u_i$, or

$$x_k \leqslant (\hat{x}_i - u_i)/ - z_{ik}. \tag{14}$$

In addition, of course, $x_k \leqslant u_k$.

If the least bound on x_k is u_k, then x_k stays out of the basis but goes to its upper bound and (5) and (12) yield $\tilde{x} = \hat{x} - Z^k u_k$ as a new basic solution with $B\tilde{x} = b - N^k u_k$.

If (13) is the least bound on x_k, taking $x_k = \hat{x}_i / z_{ik}$ with $z_{ik} > 0$ in (12) leads to the exchange of B^i and N^k in the partition between basic and nonbasic columns and the new basic solution

$$\tilde{x}_r = \hat{x}_r - z_{rk}(\hat{x}_i / z_{ik}), \qquad r \neq i; \quad \text{and}$$
$$\tilde{x}_i = \hat{x}_i / z_{ik}.$$

If (14) is the least bound on x_k, taking $x_k = (u_i - \hat{x}_i)/ - z_{ik}$ with $z_{ik} < 0$ in (12) requires the basis exchange of B^i and N^k and yields $\tilde{x}_r = \hat{x}_r - z_{rk}(u_i - \hat{x}_i)/ - z_{ik}$, $r \neq i$, and $\tilde{x}_i = (u_i - \hat{x}_i)/ - z_{ik}$, as a new basis with x_i at its upper bound.

Assume that there is a current basis B, a current solution $\hat{x}$ to $B\hat{x} = b$, and a current solution u of $\mathrm{u}'B = c'_B$. An iteration of the simplex procedure is summarized:

Step S1, *Priceout*. Select a candidate variable to enter the basis with $(c_k - \mathrm{u}'N^k) < 0$.

Step S2, *Ratio Test*. Find the greatest bound such that (with $BZ^k = N^k$):

(a) $x_k \leqslant u_k$,

(b) $x_k \leqslant x_r / z_{rk}$ for $z_{rk} > 0$,

(c) $x_k \leqslant (u_r - x_r)/ - z_{rk}$ for $z_{rk} < 0$.

If the minimum ratio is determined by case (b) or (c), let i be a basis variable for which the minimum is achieved.

Step S3, *Pivot*. Update the primal solution: $\tilde{x} = \hat{x} - x_k N^k$. If x_k is bounded by case (a), reflect x_k and leave the basis and dual solution unchanged. Otherwise, change the basis by exchanging B^i and N^k, for case (c) reflect x_i. For case (b) or (c) find the new dual solution to $\tilde{\mathrm{u}}'\tilde{B} = \tilde{c}'_B$.

In executing the simplex algorithm a number of options have customarily been employed for generating the solution of the linear systems $BZ^k = N^k$ and $\mathrm{u}'B = c'_B$.

In general algorithms, the basis inverse $Q = B^{-1}$ is usually used, stored and updated in some form. Further, although there is no difficulty in deriving a new algebraic solution to (10), $\tilde{\mathrm{u}}'\tilde{B} = \tilde{c}'_B$, as a practical matter $\tilde{\mathrm{u}}$ may be directly achieved from u by simple transformation.

PROPOSITION. $\tilde{\mathrm{u}} = \mathrm{u} + \lambda Q_i$ *where* Q_i *is the ith row of the inverse of* B. (15)

PROOF. The new basic column N^k determines λ as $\tilde{u}N^k = uN^k + \lambda(Q_i N^k) = uN^k + \lambda(Q_i BZ^k) = uN^k + \lambda z_{ik} = c_k$, so that $\lambda = (c_k - uN^k)/z_{ik}$. Exclusive of the outgoing column B^i, $\tilde{u}'B^r = u'B^r + \lambda(Q_i B^r) = u'B^r = c_r$, $r \neq i$. Q.E.D.

The (pivotal) update of Q after exchange of B^i and N^k is easily derived. The most elementary and explicit procedure is to carry and update (by pivoting) a complete tableau

$$\begin{pmatrix} QN & Qb \\ c_N - u'N & \end{pmatrix}.$$

The revised simplex procedure generates these elements as needed by access to columns of N, c and b and use of Q. Most full scale systems employ an additional refinement by expressing Q as the product of elementary "eta" column vectors, each representing the pivotal transformations generating Q from an initial basis. Frequently the history of "eta" columns grows too long for reasonably efficient generation of Q, or numerical error is propagated and detected, forcing a reinversion with "eta" column representatives from only the current basis.

Other systems support the solution procedure by using combinations of features such as an "LU" decomposition of B [3], a Cholesky decomposition of $BB' = LL'$ [25], [53], [54], or list representation of nonzero elements of problem components and coefficient representation by pointers to a table of real values. Hybrid schemes factor B into partitions with special structure: Generalized Upper Bounding (GUB) identifies an inherent identity matrix for some rows of B [16]; a partial triangulation of B with an inverse for remaining columns and rows can be used [33], [34]. Whether systems solve (6) and (10) by triangular substitution, inverse transformation, or some combination, all are algebraically equivalent simplex implementations differing only in the structures chosen to support computation for the class of problems at hand.

Primal Network Specialization

A specialization of the simplex algorithm to the transportation problem was developed by Dantzig [14] in 1951. It is not surprising that the transportation algorithm was developed immediately after the simplex algorithm, because the works of Hitchcock [39], 1941, and Koopmans [44], 1946, on the transportation problem contain many concepts that presage the simplex algorithm. The interaction of general linear programming algorithms and transshipment algorithms has a long history that has enriched the study of both.

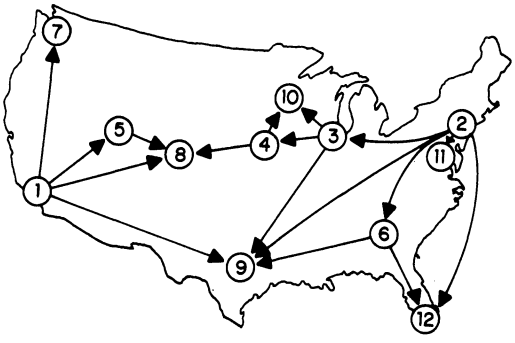
Here we establish explicitly the relationship between the general primal simplex algorithm and the modern implementations of the transshipment algorithms. Our goal is to understand the algebraic foundations of the modern transshipment implementations.

The fundamental fact that permits design of efficient primal transshipment algorithms is the well-known result that any transshipment basis can be put in (upper) triangular form by a simple permutation triangulation. This inherent triangularity can be exploited by network specializations of the simplex method by directly solving (6) by back substitution and (10) by forward substitution. Also, the triangulated basis simplex algorithms lead to much more efficient network solutions due to other fortunate simplifications. The most remarkable of these is that the solution update of Step S3 can be accompanied (in fact aided) by a very simple and efficient dynamic retriangulation of each new basis.

An initial transshipment basis with full row rank can always be constructed by introducing for each row in (1) a unit vector with sign matching that of the right-hand side (negative for demand nodes). With the addition of these artificial vectors, A has full row rank and each column of A has either a single nonzero element 1, or a single nonzero element -1, or two nonzero elements (a 1 and a -1).

THEOREM (E.G., DANTZIG [15]). *Any basis B extracted from A for a transshipment problem can be triangulated by rearranging rows and rearranging columns.*

PROOF. Let B have m rows. Locate a column with a single nonzero entry. Exchange rows and columns so that the nonzero entry is the first diagonal element. For the kth step of the construction, rows and columns have already been rearranged



City	NODE Node i	DATA Supply b_i	From i	To j	ARC Cost/ unit	DATA Lower Bound	Arc Capacity	A Basic Feasible Flow
Los Angeles	1	34	2	3	34	0	11	2
New York	2	56	3	4	23	0	6	4
Chicago	3	5	1	5	28	0	10	10
Omaha	4	0	2	6	45	5	25	25
Salt Lake City	5	−5	1	7	57	0	21	18
Atlanta	6	−9	5	8	24	0	5	5
Seattle	7	−18	1	8	56	0	7	6
Denver	8	−15	4	8	19	0	9	4
Austin	9	−8	1	9	61	0	5	0
Minneapolis	10	−3	2	9	99	0	12	8
Washington	11	−21	6	9	48	0	3	0
Miami	12	−16	3	9	53	0	24	0
			3	10	26	0	8	3
			4	10	20	0	2	0
			2	11	14	10	23	21
			6	12	34	0	16	16

FIGURE 1. A Single Commodity Transshipment Problem.

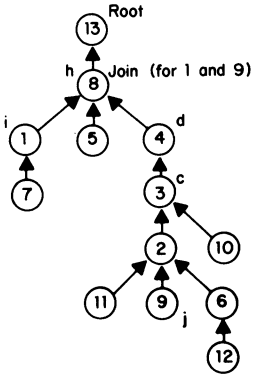


FIGURE 2. A Transshipment Basis (for Figure 1).

$$
\begin{array}{c}
\text{Row} \\
1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12
\end{array}
\left[\begin{array}{cccc|cccc|cccc}
 & 1 & & & & & & & & & 1 & \\
 & & & 1 & 1 & & 1 & & & & & 1 \\
1 & & & & & & & & & 1 & & -1 \\
-1 & & & 1 & & & & & & & & \\
\hline
 & & & & & & & & 1 & & & \\
 & & & & & & -1 & 1 & & & & \\
 & -1 & & & & & & & & & & \\
 & & & -1 & & & & -1 & -1 & & & -1 \\
\hline
 & & -1 & & & & & & & & & \\
 & & & & & & & & & -1 & & \\
 & & & & -1 & & & & & & & \\
 & & & & & & & & & & & -1
\end{array}\right]
$$

Transshipment Basis Matrix

$$
\begin{array}{c}
I \\
8 \\ 1 \\ 7 \\ 5 \\ 4 \\ 3 \\ 2 \\ 11 \\ 9 \\ 6 \\ 12 \\ 10
\end{array}
\left[\begin{array}{cccc|cccc|cccc}
-1 & -1 & & -1 & -1 & & & & & & & \\
 & 1 & 1 & & & & & & & & & \\
 & & -1 & & & & & & & & & \\
 & & & 1 & & & & & & & & \\
\hline
 & & & & 1 & -1 & & & & & & \\
 & & & & & 1 & -1 & & & & & 1 \\
 & & & & & & 1 & 1 & 1 & 1 & & \\
 & & & & & & & -1 & & & & \\
\hline
 & & & & & & & & -1 & & & \\
 & & & & & & & & & -1 & 1 & \\
 & & & & & & & & & & -1 & \\
 & & & & & & & & & & & -1
\end{array}\right]
$$

A Preorder Triangulation

$$P(i_k) = [8 \quad 3 \quad 4 \quad 8 \quad 8 \quad 2 \quad 1 \quad 13 \quad 2 \quad 3 \quad 2 \quad 6]$$

The Basis Predecessor Function

FIGURE 2 (*continued*)

so that columns $1, 2, \ldots, k-1$ have only zeros below the diagonal. Select a column with a single nonzero entry in rows $k, \ldots, m$. Exchange rows and columns so that the nonzero element is the kth diagonal element. This construction is well defined only if there is a column with a single nonzero element in rows $k, \ldots, m$. There must be such a column, for otherwise each column would have no nonzero elements, or exactly one $+1$ and one -1 in rows $k, \ldots, m$, and the sum of rows $k, \ldots, m$ would be the zero row, which would contradict the assumption that B is a basis. Q.E.D.

A graph can be defined that represents the transshipment basis. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a row ordering corresponding to a triangulated B. Associate with each node i_k the row number $p(i_k)$ of the offdiagonal element in the kth column of the triangulated basis; if there is no nonzero offdiagonal element, set $p(i_k)$ to $m+1$. Define a graph with nodes $1, 2, \ldots, m+1$ and let $(i_k, p(i_k))$, $k = 1, 2, \ldots, m$, be directed arcs from i_k to $p(i_k)$. Since each node i_k is connected to a node $p(i_k) = i_h$ with $h < k$ or to node $m+1$, there is a directed path from each node to node $m+1$. The graph is called a *rooted arborescence* [4] with node $m+1$ the *root*. Ignoring the orientation of the arcs, since the graph is connected and has $m+1$ nodes and m arcs it is a tree (it can be shown [4] that this definition is equivalent to the usual definition of a tree as a connected graph with no cycles). In the computer science literature (e.g., [45]) the term tree is often used instead of rooted arborescence. Figure 1 is an example of a transshipment problem with a basic feasible solution specified and Figure 2 has the basis and the associated arborescence. Our pictorial representation with the "root" at the top rather than at the bottom is fairly standard.

For node i, $p(i)$ is called the *predecessor* of i and the rooted arborescence is called the *predecessor graph*. The predecessor graph is closely related to, but not identical with, the classical result of Koopmans [44] that the arcs of a transshipment basis form a tree over the nodes of the problem. The classical tree preserves the orientation of arcs in the original network and does not include node $m+1$.

The predecessor function $p(\ )$ is a well-known compact way to represent trees and rooted arborescences and has been used in network algorithms for at least 15 years. The predecessor graph has often been used interchangeably with the classical basis tree. It is important to distinguish between them because the predecessor graph is a data structure that supports the computation of the algorithm and the orientation of the arcs indicates the unique direction to the root rather than a direction in the network. Furthermore, the predecessor graph can be extended to triangular bases that have no underlying network [32].

The predecessor function $p(\ )$ can be iterated to construct the unique path from any node to the root; this path is called a *backpath*. 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 the *successors*. Another characterization of the successors of node i is that they are all nodes whose backpath to the root includes node i. A tree can also be represented by the immediate successors of each node; however, since the class of trees that arise in network problems is m-ary, the (0 to m) immediate successors of each node are more difficult to maintain dynamically than the predecessor function which always has a single unique value for each node except the root.

In general, there are many different triangulations for any given transshipment basis. (Note that at each step in the construction there may be several choices for the next column.) However, all such triangulations yield the same predecessor function and graph (where the ordering of successors right to left for any node is immaterial). Thus, the predecessor graph does not completely represent a triangulation without additional information, namely an ordering of the rows. A mathematical development of these properties and their implications is given in the companion paper [32].

The relationship between the algebraic view of the simplex algorithm and the graph theoretic view of much of the network literature can be shown by describing the operations of the simplex algorithm and the triangulation in terms of the predecessor graph. A graph theoretic proof of the triangulation theorem identifies a node with each row and an arc with each column with two nonzero elements. Also included is the root node; columns with a single -1 (or 1) are represented as an arc to (from) the root node. A triangulation and the predecessor graph are constructed by first selecting the root node. Select any arc with one end the root and add the new node and arc orienting the arc toward the root. For the kth step of the construction k nodes and $k-1$ arcs are already in the graph. Select an arc with exactly one node already in the graph, add the new node and the arc to the graph orienting the arc toward the old node. The resulting triangulation is characterized by the order in which the nodes and arcs are introduced.

Note that we have used an additional node (the root) to allow a unified treatment of slack, surplus and artificial vectors. Some other researchers have designated one of the nodes of the original problem as a "root" and then defined a predecessor function; the predecessor function is then unique only for a given choice of the "root" node.

For the network linear program each node is identified with an equality constraint, and each arc k from node i to node j is identified with a variable with column N^k having a 1 in row i, a -1 in row j and 0's elsewhere. For the discussion below it is assumed that the basis B is in triangular form and to simplify notation it is momentarily assumed that row i of B corresponds to node i for all $i \in N$. (Renumber the nodes if necessary so that $I = \{1, 2, \ldots, m\}$.) It is further assumed that all the diagonal elements of B are 1; if B initially has a -1 on the diagonal, the reflection $u_k - x_k$ transforms it to a 1. This may be viewed as transforming the basis variables to make the orientation of arcs in the predecessor graph the same as the orientation in the Koopmans basis tree.

Step S1, *Priceout*. Given the vector of simplex multipliers u, the priceout formula (11) for nonbasic arcs (step S1) simplifies to $c_k - \mathrm{u}_i + \mathrm{u}_j$. Thus the priceout involves only addition operations.

Step S2, *Ratio Test*. For the determination of the arc to leave the basis in step S2 the system of equations $BZ^k = N^k$ must be solved for Z^k. This calculation is described by showing how to solve $BQ^j = e_j$ for Q^j (e_j is the jth unit vector and Q^j is thus the jth column of the inverse of B). Since B is triangular, Q^j can be obtained by simple backward solution: the $m, m-1, \ldots, j+1$ elements of Q^j are seen to be 0, the jth element is 1. Setting the jth element in the modified right-hand side equal to 0 puts a 1 in the row corresponding to the offdiagonal -1 (if any) in the jth column of B, this row is $p(j)$. The procedure is then continued until a column of B with no nonzero offdiagonal is encountered. The resulting Q^j is all zeros except for ones in rows j, $p(j)$, $p(p(j))$, etc.

Thus, in terms of the predecessor graph, all arcs traversed on the backpath from node j to the root have an element 1 in Q^j; all other elements are 0. Therefore, Q^j can be generated directly from the precedence function $p(\ )$, which (with I) gives the substitution rules for the back solution.

The calculation of Z^k follows immediately since $Z^k = Q^i - Q^j$. The element in a row of Z^k is 1 for all rows with a 1 in Q^i alone, -1 for all rows with 1 in Q^j alone and 0 for all rows with 0 in both or 1 in both. Since the nonzero elements of Z^k are 1 or -1, the calculations in step S2 involve only addition and subtraction. Further, the calculation is usually reduced enormously by the extinguishment of the elements common to both i and j backpaths.

As an example, consider the triangulated basis in Figure 2 (noting that the nodes

have not been renumbered) with column N^k associated with arc (1, 9), $BQ^9 = e^9$, and $BQ^1 = e^1$.

$$
\begin{aligned}
I &= \{8 \quad 1 \quad 7 \quad 5 \quad 4 \quad 3 \quad 2 \quad 11 \quad 9 \quad 6 \quad 12 \quad 10\},\\
p(\ ) &= \{8 \quad 3 \quad 4 \quad 8 \quad 8 \quad 2 \quad 1 \quad 13 \quad 2 \quad 3 \quad 2 \quad 6\},\\
Q^1 &= [1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]',\\
Q^9 &= [1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0]',\\
Z^k &= [0 \quad 1 \quad 0 \quad 0 \quad -1 \quad -1 \quad -1 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0]'.
\end{aligned}
$$

Step S3, *Pivot*. In step S3, the primal solution is updated using Z^k. Also, as shown by (15), the simplex multipliers, u, can be updated rather than calculated at each step. The algebraic characterization of the update applies the lth row of the inverse of B, Q_l, where the outgoing arc is the lth column of B. The characterization of Q^j above showed that an element of Q^j is nonzero and equal to 1 only if the corresponding basic arc is traversed on the backpath from node j to the root. It follows that Q_l is all 0's except for 1 in the lth element and a 1 for any element j such that the outgoing arc is on the backpath from j to the root. In terms of the predecessor graph, Q_l is 1 for node l and each of its successors and 0 elsewhere. Since the nonzero elements of Z^k are 1 and -1, λ in (15) is either plus or minus $c_k - u_i + u_j$ where k is the incoming arc. Since Q_l is all 0's and 1's, the update of u is accomplished by adding λ to u_l and to the u's of (only) the successors of node l.

As will be shown in the next section, the simple additive updates of the primal solution and dual multipliers in step S3 are actually accomplished in a single integrated process. We will refer to this process as the "pivot" even though in addition to the usual pivot it simultaneously updates the primal solution and dual multipliers and retriangulates the new basis.

Implementation

The design of large scale programming codes necessarily involves many significant decisions which have major impact. The following fundamental principles were used to resolve design questions in the development of the code reported here.

1. The code should be designed for large scale problems. Even though experimental testing will be confined by economic considerations to problems with some arbitrary maximum size (say, 10,000 equations), the design decisions should lead to a code with superior large scale performance.

2. The code should solve the most general capacitated transshipment problem. While problems with specializations (e.g., uncapacitated, transportation, assignment) must be solved, the basic code will not be tailored to these special features. In addition, no special numbering of nodes, extensive preprocessing, or other design specificity will be required that will limit the capability of the code. Efficient solution of problems should not require detailed advance knowledge of problem structure (for instance, a feasible initial solution). Problems with multiple arcs will be accommodated.

3. For practical problems the number of arcs is, in general, much greater than the number of nodes, m. However, the problems are usually sparse in the sense that the number of arcs seldom approaches the maximum number $m(m-1)$ of oriented node pairs. Thus, it is significantly more expensive to store information associated with each arc than it is for each node and prohibitively expensive to store a node-arc incidence matrix. Practical general minimum cost network flow problems are always heavily capacitated.

4. The code should be machine independent as well as efficient. For example, machine specific features such as assembly language, use of particular offline mass storage devices, storage of data using bit string logical vectors, use of other architectural curiosities on particular machines or nonstandard language features will be avoided. The language used is elementary FORTRAN.

5. Where feasible, speed of execution should be given preference over economy of space for data storage.

6. Since the program will be used for comparisons of various data structures on a wide variety of network problems, it should be equipped with effective external tuning parameters. While some tuning is possible for the pivot mechanism, the pricing scheme invites especially close scrutiny for tuning purposes.

Once the design of an efficient network code is chosen, consideration will also be given to addition of other advanced features such as "in-core/out-of-core" operation, implicit arc generators, crashed bases, nonlinear costs, postoptimal analysis, and so forth. These extensions will not be allowed to interfere with the basic design goals, but they should not be precluded by the basic design decisions.

The description of GNET begins with discussion of arc and node storage representation. Next, the ratio test is described. The pivot is explained functionally (with a more extensive algebraic derivation left for the companion paper). Finally, the pricing mechanism is examined. This order is chosen (steps S2 and S3 followed by S1) to report faithfully the history of our implementation and to lead smoothly to the computational performance tests. Hereafter, notation with upper-case Roman letters indicates a program variable and addition of parentheses denotes an array. For instance, the predecessor *array* is referred to as P( ).

Since there will be many arcs, it is critical to minimize the stored data describing each arc. A typical input format (for example, SHARE) for each arc is tail, head, cost, upper bound and lower bound on flow. The lower bounds are removed by transformation. If the arcs are sorted so that all arcs with the same head are stored in contiguous space, the list of heads can be replaced by a node-length array whose jth element is the location of the first arc with head j. Since network models have many more arcs than nodes, this reduces the storage requirements for the algorithm. Thus, the network is stored as three arc-length arrays: the tails T( ), the costs C( ) and the upper bounds (capacities) CP( ); also, one node-length array is used, the head entries H( ) into T( ). Positive capacities are required after transformation of lower bounds for all arcs—uncapacitated arcs have capacity set to some value greater than the total supply. Arcs out of the basis at their upper bounds are marked with a sign bit on the capacity (−CP( )); alternatively, the tails could be used for this purpose.

It is natural to associate with each node i (except the root) the unique basic arc that connects i to its predecessor. It is convenient to have the basic arcs oriented in the same way as in the predecessor graph, that is, from a node to its predecessor. This is accomplished by reflecting arcs as necessary. The predecessor array P( ) is marked with a minus sign for all arcs that have not been reflected. (Subsequently, when using P( ) we will assume for simplicity that it is positive.) The flow on arc (i, P(i)) is stored in X(i).

In step S2 of the simplex algorithm, it is necessary to compute capacity minus flow for basic arcs with $z_{ij} < 0$. It is convenient to use a node-length array to speed up this calculation. One approach is to store for each basic arc a pointer to its capacity in the CP( ) array. Another technique is to store the capacity rather than the pointer. A third method is used in GNET. The capacity minus flow is stored in a node-length array CPX( ).

The simplex multipliers are stored in a node-length array U( ). Figure 3 shows these arrays for the basis in Figure 2 (the IT( ) array shown in Figure 3 is defined below).

After the incoming arc has been chosen in step S1, the outgoing arc is determined in step S2 and the data structures are updated in step S3. Let k be the incoming arc going from node i to node j. The possible outgoing arcs correspond to the nonzero entries in Z^k. We have already seen that Z^k is zero for all nodes common to both i and j backpaths. Let the *join* be the first node on the backpath from i to the root that is on the backpath from j to the root (the join is the backpath node beyond which all terms of Z^k are zero). The possible outgoing arcs are the arcs on the backpath from i to the join and the arcs on the backpath from j to the join.

It is critical to identify the backpaths from i and j to the join efficiently. The trial and error of the classical "stepping stone" methods of most textbooks will clearly not suffice for any but trivial problems. We discuss four methods to identify the backpaths from i and j to the join node. Note that only predecessor information is available to the program at each node and that our data structure has no global view of the arborescence as does the reader of Figure 2.

The most naive method is to mark in some way all the nodes on the backpath from i to the root. Then, the first marked node encountered on the backpath from j to the root is the join. This method is satisfactory for smaller problems, but for larger problems it is more efficient to avoid the iteration along one complete backpath all the way to the root by keeping additional information about the tree. Also, in this way it will not be necessary to unmark the marked nodes before performing the next simplex pivot.

Arc:	k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Tail:	T	2	3	1	2	1	5	1	4	1	2	6	3	3	4	2	6
Cost:	C	34	23	28	45	57	24	56	19	61	99	48	53	26	20	14	34
Capacity:	CP	11	6	−10	20	21	5	7	9	5	12	3	24	8	2	13	16

Arc-length arrays

{"−" indicates arc out of basis at capacity

Node:	i	1	2	3	4	5	6	7	8	9	10	11	12	13
Heads:	H	0	0	1	2	3	4	5	6	9	13	15	16	17
Predecessor:	P	−8	−3	−4	−8	−8	2	1	13	2	3	2	6	
Traversal:	IT	7	11	2	3	4	12	5	1	6	13	9	10	8
Flow:	X	6	2	4	4	5	0	3	96	4	5	2	0	
	CPX	1	9	2	5	0	20	18	0	8	3	11	16	
Depth:	D	2	4	3	2	2	5	3	1	5	4	5	6	0
Multiplier:	U	56	76	42	19	24	31	−1	0	−23	16	62	−3	

Node-length arrays, GNET/Depth

{First arc with head i located at $k = H(i)$
{"−" gives arc orientation above node i

$P(i) < 0 \rightarrow$ flow is $X(i)$, otherwise flow is $CPX(i)$

Successors:	NS	1	4	6	7	0	1	0	11	0	0	0	0	12
Preorder:	PD	2	7	6	5	4	10	3	1	9	12	8	11	0

Arrays used by other GNET versions (in lieu of depth)

$$\begin{aligned}\text{Priceout arc } 1 \rightarrow 9:\quad DF &= C(k = \text{arc}) - U(i = \text{tail}) + U(j = \text{head}) \\ &= C(9) - U(1) + U(9) \\ &= 61 - 56 - 23 \\ &= -18\end{aligned}$$

$$\text{Ratio Test:}\quad \min\left\{\begin{array}{ll} CP(9): & 5 \\ X(\ ) \text{ for } 1-\text{backpath}: & 6 \\ CPX(\ ) \text{ for } 9-\text{backpath}: & 8, 9, 2, 5 \end{array}\right\} = 2 \text{ for arc } (3, 4)$$

FIGURE 3. GNET Arrays (for Basis in Figure 2).†

† Two candidate queue arrays also used by GNET are omitted for clarity. Actual simplex multipliers in GNET would all be exactly 1,188 units smaller than shown. This difference is the high cost associated with artificial demand arc $13 \rightarrow 8$, initially computed as BM = Nodes × maximum|cost|.

One efficient method is to store for each node the number of nodes on the backpath to the root, called the *depth* of the node, D( ), in the tree. With depth information available it is possible to iterate the backpaths synchronously from i and j to identify the join without iterating either backpath past the join. Depth is used to indicate which backpath node is deeper in the tree and should be iterated. When both backpath nodes have matching depths, the nodes are compared for equality. A match indicates the join, and a mismatch indicates that both backpaths should be iterated for another comparison.

Another efficient method similar to the depth approach is to store for each node the number of nodes in its subtree, called the *number of successors*, NS( ). Starting with nodes i and j, the backpath node with strictly fewer successors is iterated. When both backpath nodes have the same number of successors, a match indicates the join and a mismatch forces iteration of both backpaths. The fourth method will be discussed below.

The latter three methods for locating the join look only at arcs that are on the backpaths to the join, thus it is possible to determine the outgoing arc while searching for the join. As noted above, all the arcs on the backpath from i to the join are the $+1$ elements of Z^k and all the arcs on the backpath from j to the join are the -1 elements. The ratio test step S2 is then simply

$$\min\begin{cases} \mathrm{CP}(k) & \text{the capacity of the incoming arc,} \\ \mathrm{X}(\ ) & \text{for arcs on the backpath from } i \text{ to the join,} \\ \mathrm{CPX}(\ ) & \text{for arcs on the backpath from } j \text{ to the join.} \end{cases} \tag{16}$$

The computational simplicity of this ratio test is the rationale for the reflection of basic arcs and the adoption of CPX( ).

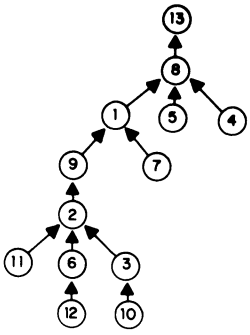
If the incoming arc is out of the basis at its capacity, then step S2 may be viewed as increasing flow in a fictitious arc from node j to node i with the same capacity.

A major proportion of the work of each simplex step is S3, the pivot. In this step, the entering and leaving arcs are exchanged, the flows, X( ) and CPX( ), are updated, the simplex multipliers, U( ), are changed, and the simplex data arrays P( ) and D( ) (using the depth mechanism for example) are modified.

Within this step the coordination and sequencing of operations are critical to the efficiency of the network algorithm because the manipulation of many nodes and heavy use of the simplex data structure are involved. If properly done, this step is the elegant central part of the code that can be executed by a computer quickly; however, the explanation will be somewhat intricate—this part of the algorithm is considered by many to be the "secret" part of the fastest codes.

To illustrate a typical pivot, Figure 2 shows the entering arc (i, j), join and leaving arc (c, d). Call the backpath from j to c the *pivot stem*. Figure 4 includes the predecessor graph after the pivot. Notice that the subarborescence with root c is "rehung" from node i, and that nodes on the pivot stem have their predecessor relationships reversed. The flows, X( ) and CPX( ), change only on the backpaths from i and j to the join. The simplex multipliers, U( ), and the depth, D( ), are recomputed for all nodes in the subarborescence with root c.

The most expensive part of the pivot is the update of the simplex multipliers by the addition of λ (15) to the U( )'s of node c and all nodes in the subarborescence with root c. With the data structure presented so far, it is not easy to identify all the (successor) nodes in a subarborescence. The identification of these nodes can be facilitated by a *traversal* data structure that begins at the root of the predecessor graph and exhaustively "walks" through all the nodes in the same sequence that the nodes occur in a triangulation of the basis. This is done with a node-length array



Node:	i	1	2	3	4	5	6	7	8	9	10	11	12	13
Predecessor:	P	−8	*−9*	*2*	−8	−8	2	1	13	*1*	*3*	2	6	
Traversal:	IT	*9*	11	*10*	*13*	4	12	5	1	*2*	*7*	*6*	*3*	8
Flow:	X	*4*	*6*	*7*	*6*	5	0	3	96	*3*	5	2	0	
	CPX	*3*	*6*	*4*	*3*	0	20	18	0	*2*	3	11	16	
Depth:	D	2	*4*	*5*	2	2	*5*	3	1	*3*	*6*	*5*	*6*	0
Multiplier:	U	56	*94*	*60*	19	24	*49*	−1	0	*−5*	*86*	*80*	*15*	

GNET/Depth arrays changed by pivot

(Recomputed elements in italics)

Successors:	NS	*8*	*5*	*1*	*0*	0	1	0	11	6	0	0	0	12
Preorder:	PD	2	*4*	*8*	*12*	*11*	*6*	*10*	1	*3*	*9*	*5*	*7*	0

Alternate arrays (for D) after pivot

(Since outgoing arc (3, 4) left basis at its capacity, CP(2) will be marked "−" also.)

$$
\begin{array}{c}
I \\
8 \\ 1 \\ 9 \\ 2 \\ 11 \\ 6 \\ 12 \\ 3 \\ 10 \\ 7 \\ 5 \\ 4
\end{array}
\left[\begin{array}{cccc|cccc|cccccc}
-1 & -1 & & & & & & & & & & -1 & -1 \\
 & 1 & 1 & & & & & & & 1 & & & \\
 & & -1 & -1 & & & & & & & & & \\
 & & & 1 & 1 & & 1 & & 1 & & & & \\
\hline
 & & & & -1 & & & & & & & & \\
 & & & & & -1 & & 1 & & & & & \\
 & & & & & & -1 & & & & & & \\
 & & & & & & & -1 & 1 & & & & \\
\hline
 & & & & & & & & -1 & & & & \\
 & & & & & & & & & -1 & & & \\
 & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & 1
\end{array}\right]
$$

Retriangulated Basis

FIGURE 4. GNET Arrays and Basis after Pivot (Figure 2).

IT( ) whose ith element is the next node to visit from node i. IT( ) is thus a different way to represent the information in I. It is convenient to make this a circular list by setting the IT( ) of the last node in the triangulation equal to $m + 1$.

From the triangulation construction it is clear that a node is added to the predecessor graph before any of its successors. The construction of the triangulation can always be modified to add all the successors (if any) of a node before any other node is considered. For this restricted class of triangulations the corresponding traversal IT( ) will visit all the nodes of a subarborescence contiguously, precisely as is required in the update of the simplex multipliers. This traversal is the obvious extension to the m-ary case of the *preorder* (or, equivalently, dynastic order) traversal in computer science literature. The recursive definition of the preorder traversal given in the computer science literature [45] reveals precisely its value in updating the simplex multipliers.

Let us look at the work that must be done in the pivot. The algorithm visits each node of the subarborescence with root c exactly once. It proceeds up the pivot stem one node at a time. At each stem node, the successors of the next lower stem node have already been visited. The unvistited successors of the current stem node can be divided into two groups: the nodes visited in preorder (by iterating IT( )) from the stem node until the next lower stem node is encountered, called the left successors of the stem node, and the remaining unvisited nodes in preorder, called the right successors of the stem node. For example, in Figure 2 the left successor of stem node 2 is 11, and the right successors are 6 and 12. Figure 5 gives a general description of these operations for the pivot.

We have experimented with three different data arrays that will support the *one-pass pivot* computation: depth D( ), number of successors NS( ), and an addi-

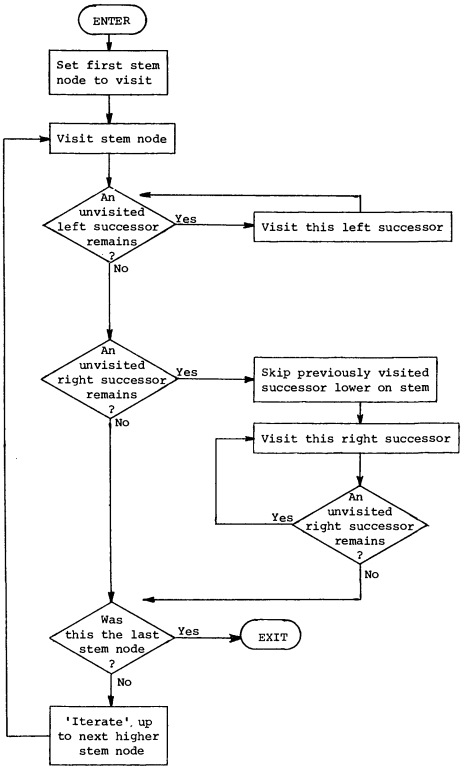


FIGURE 5. Pivot Traversal Scheme.

tional structure PD( ) to be discussed below. We did not compare another, less efficient two-pass method which marks nodes with sign bits, and later unmarks them, much like the procedure described for locating the join node. These arrays are used in the pivot solely to answer the question (Figure 5), "An unvisited right successor remains?" This question is nontrivial because the node information now available to us is local in nature.

Figure 6 shows a detailed description of the pivot traversal for the pivot stem and its successors using depth. The primary purpose of visiting each of these nodes is to update U( ). The pivot also updates the arrays D( ) and IT( ); the update of D( ) is omitted for clarity. The arrays P( ), X( ) and CPX( ) are recomputed for the pivot stem only. (The straightforward X( ) and CPX( ) flow update for backpath nodes not on the pivot stem is not shown.) The update of IT( ) is easy because IT( ) changes only for the nodes on the pivot stem as well as for the last left successor (if any) and the last right successor (if any) of each pivot stem node. For nodes on the pivot stem, D( ) is updated as the pivot moves up the stem. The right and left successors of each pivot stem node "inherit" their depth from the pivot stem node—if the depth of the pivot stem node changes by DADJ, then so do the depths of its right and left successors.

The number of successors NS( ), can also be used as illustrated in Figure 7. The updating of the number of successors is not shown in the figure. The number of successors is easy to update because it changes only for the nodes on the backpaths from i and j to the join; thus, this update can also be performed for stem nodes during the traversal and for the other backpath nodes (beginning with i and d) with the X( ) and CPX( ) flow update.

Degeneracy is a critical issue in transshipment problems. In some of our test problems more than 90% of the (tens of thousands of) pivots are degenerate. The search for the join may be aborted when degeneracy is encountered since the only

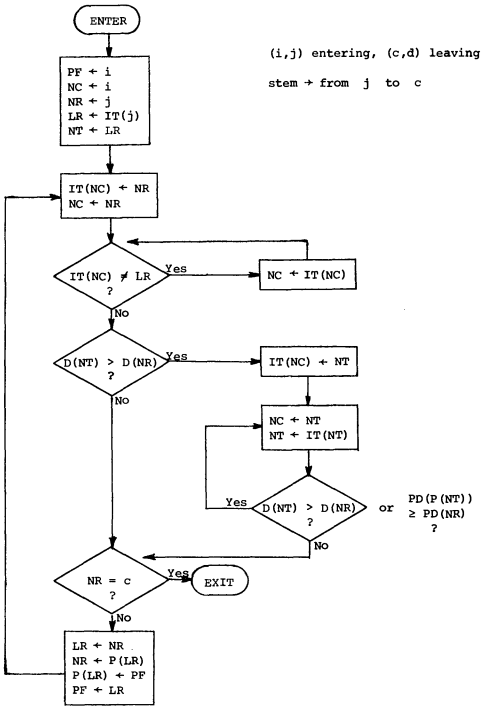


FIGURE 6. Pivot (Retriangulation) Segment Using Depth (or Preorder Distance).

purpose of the search is identification of the leaving arc and backpaths for flow change (zero in the case of degeneracy). Stopping short also tends to make the number of nodes visited by the pivot smaller. In the depth version, the lowest degenerate arc can be chosen to leave, while in the version with number of successors the smallest number of nodes can be chosen for the pivot. But the successors version must still locate the join to update the number of successors on each backpath (a relatively easy process) while the depth version requires no further backpath search.

Another data array that can be used in place of depth or number of successors is position in the triangulation, or equivalently position in IT( ). For each node i define *preorder distance* PD(i) to be the row number in the triangulation of equation i or equivalently to be the number of iterations of IT( ) beginning with the root to get node i. Set PD(root) = 0.

The search for the join is particularly easy with preorder distance. The following proposition gives a simple construction that determines the join.

PROPOSITION (MCBRIDE [48], GRAVES AND MCBRIDE [34]). *Given a basis with a preorder triangulation, for any two nodes i and j such that* PD(j) > PD(i) *the first node h on the backpath from j with* PD(h) $\leqslant$ PD(i) *is the join.*

PROOF. The construction always determines a node since PD(root) = 0. The join is the first node on the backpath from one node, say j, that is also on the backpath from the other node i. Thus we need to show that i is either h or a successor of h and that i is not a successor of any other node on the backpath from j to h. In any triangulation PD(k) > PD(P(k)) for any node k, thus by construction h is the only node on the backpath from j to h that could be the join. In a preorder triangulation, the successors of any node k have contiguous PD( ) numbers beginning with PD(k) + 1. Since PD(h) $\leqslant$ PD(i) < PD(j), i either equals h or is a successor of h. Therefore, h is the join of i and j. Q.E.D.

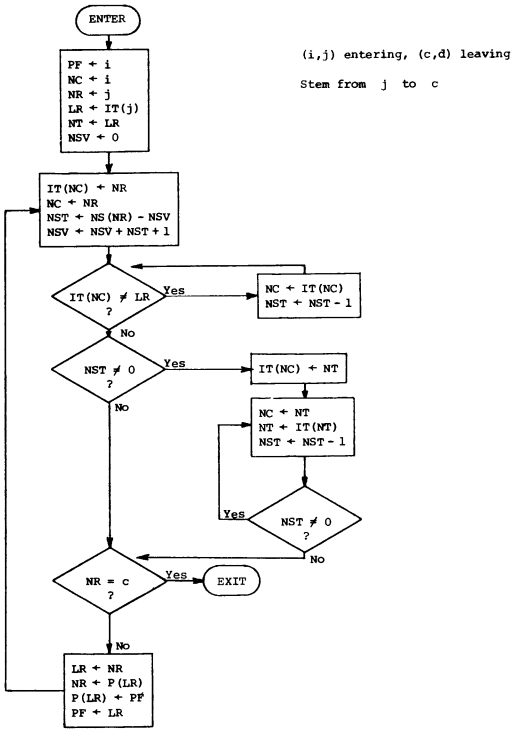


FIGURE 7. Pivot (Retriangulation) Segment Using Number of Successoı

Using PD( ) the outgoing arc can be determined simultaneously with the search for the join, and if degeneracy is discovered, the search for the join can be stopped.

The question in the pivot, "An unvisited right successor remains?" in Figure 5 is equivalent to "PD(P(NT)) $\geqslant$ PD(NR)?" and the flowchart for preorder distance is equivalent to Figure 6 with "D(NT) > D(NR)" replaced. PD( ) is recomputed for all nodes in the subarborescence with root c and some additional nodes.

Pricing Mechanism

The pricing operation of step S1 in the simplex algorithm requires a great deal of computational effort. As in other large scale mathematical programming problems, network codes can spend more than half of their execution time selecting incoming variables by pricing. Thus, the pricing mechanism is crucial to overall performance and must also provide the flexible, broad and effective external controls necessary to permit tuning of the code for network problems with special or even bizarre structure.

Choice of pricing strategy is truly an art for large scale mathematical programming codes. It is based on intuition, experience and empirical experiments with the class of problems to be solved. Although simplex pricing has received very little exclusive attention in the literature, fast primal codes all employ some form of multiple or partial pricing of subsets of the variables at each pivot [51].

Examination of many problem trajectories suggests that pricing be performed in three major phases—an opening gambit, a middle game and an end game. Initially, a feasible solution is being constructed and pricing must select incoming variables carefully among many available choices with the view of satisfying violated constraints. Later, normal gains toward optimality seldom justify extensive competition among candidates. Finally, a favorable candidate becomes quite rare and thus has considerable value since eventually all variables must be exhaustively priced to verify optimality.

GNET performs pricing by selectively using a *general scan* mechanism and keeps a set of good pivot information in a *candidate queue*. For a given head node, the scan selects the single incident arc (if any) pricing most favorably (for each tail examined incident to a particular head node only one addition and one comparison are required) and places it on the candidate queue. The candidate queue is a varying length cyclic list of "interesting node" and "good arc" information. Each entry includes a head node and either a tail pointer specifying the location of a good candidate arc or a null pointer indicating an interesting node which has not yet been priced. The candidate queue mechanism provides for user control of network solutions. Although there are many special uses for these controls, for example, basis crashing, it is even more important to provide robust automatic procedures for initialization, selection of incoming arcs, and queue maintenance for the most general class of transshipment problems.

Since GNET is designed for general capacitated transshipment problems, heuristics for advanced feasible starting solutions are not employed. These heuristics are usually designed for problem specializations, and we do not find the cost of their use to be clearly justified for large problems. GNET uses an initial basis of artificial arcs connected to the root node with initial flows equal to supplies and demands and a high cost attached to each artificial arc incident to a demand node. The candidate queue is initially loaded with all these demand nodes.

For each pivot, the incoming arc is selected from the candidate queue by examining entries in a block of specified size ("number examined" NNE): each candidate arc encountered is individually repriced; each interesting node is priced out by the general scan. The most favorable arc found in the block (if any) is chosen to enter the basis. Also, all arcs pricing favorably are returned to the candidate queue.

The opening gambit is designed to accelerate the achievement of feasibility. For this sequence of ("number starting" NNS) pivots the high cost of infeasibility is likely to be the cause for arcs to price favorably. These pivots essentially build chains from demand nodes to supply nodes. Accordingly, GNET stimulates such early chain-building by placing on the candidate queue the head and the tail of each incoming basic arc. Thus, the relatively costly general scan mechanism is subsequently directed to price out arcs connected to these two interesting nodes probably representing infeasible problem constraints. This "demand driven" scheme works particularly well on problems with relatively large numbers of demand nodes and actual structural chains. It also motivates the arc list organization by head node rather than the usual tail ordering.

During the opening gambit, if no favorable arc is discovered after examining a block on the candidate queue, another block of entries is accessed, and so forth, until an incoming arc is discovered or the candidate queue is completely emptied. An exhausted queue is refreshed by directing the general scan to a specified number ("page" IPG) of head nodes.

After the opening gambit, the nodes of each incoming arc are no longer inserted in the candidate queue. After each complete cycle around the queue it is refreshed by a general scan of IPG nodes. Each of these scans begins where the previous scan has ended and thus moves cyclically down through the arc list, pricing arcs in contiguous storage locations. When managed by these procedures, the candidate queue generally grows during and just after the opening gambit and then shrinks finally to a few good candidates. The end game is played by concentrating on these last few candidate arcs.

A major portion of the experimentation with sample problems has been devoted to tuning the three pricing parameters NNE, NNS, and IPG with the objective of estimating optimal settings as simple functions of solution progress and easily measured problem characteristics such as number of nodes, arcs, supply and demand nodes, degree of capacitation, cost range, and so forth. Also, the sensitivity of performance to problem structure has been investigated.

High resolution internal computation timing is often difficult to carry out on contemporary multiprogramming computer systems. The computer timing routines often take longer to execute than the network algorithm coding segments under study. Therefore, in order to isolate times of pricing and pivots, a special experiment was designed to minimize relative timing error. Sample problems were run to completion by GNET under fixed parameter settings, each producing a single gross solution time and number of pivots. Next, histories of the entering arcs for the solutions were fed to a specially modified code, essentially eliminating the pricing mechanism; these solution times represent almost exclusively pivot time. Also, nontiming runs were made with another code with extensive internal data collection for detailed performance analysis.

Pricing schemes were tried ranging from "first negative" (terrible) to "most negative" (worse). Optimal intermediate settings of NNE and IPG lead to a nearly equal distribution of time between pricing and pivoting. As the length of the opening gambit, NNS, is increased, solution time is greatly reduced; however, beyond some point, times again go up as the candidate queue becomes clogged with bad nodes inserted by the pivots. The size of the candidate queue swells, and then declines as predicted, with the maximum size a complicated function of all three parameters. As expected, several complete cyclic sweeps of the entire arc list in blocks of IPG head entries always occur late in the end game, but the candidate queue focuses attention on the best remaining candidates. The total number of pivots is very sensitive to NNE. However, total solution time is relatively stable as NNE and the other tuning parameters are varied from reasonably good default settings. Experiments to change

NNE dynamically during solution progress have not improved performance. In fact, many dynamic tuning schemes have been tried without much success.

Hundreds of such calibration runs have been performed in an attempt to determine empirically the form of the response surface of execution time as a function of the pricing parameters. To date, the best general automatic default settings for transshipment problems are

NNE = 32	entries examined in each candidate queue block,
NNS = $3m/4$	pivots in the opening gambit,
IPG = $m/10 + 1$	head nodes priced to refresh the candidate queue in each page of the general scan.

These suggested settings are surprisingly robust for a wide variety of problems. However, for particular classes of problems sharing uncommon structure, specific tuning can achieve remarkable further improvements.

Computational Experience

The family of GNET codes first presented at the Spring 1975 ORSA/TIMS meeting represents the state of the art in fast large scale minimum cost network flow systems. Tables 1 and 6 report GNET solution times and numbers of pivots for a set of standard test problems [from NETGEN, 43] that have also been solved by other contemporary codes. The additional problems labelled NPS and XN1 are generated (using the same random number seed) to exhibit structure common to many practical applications—all arcs are capacitated, and there are few sources and many sinks. These solution times are achieved by GNET with the pricing parameters set at their default values. Thus, our experiments are completely reproducible by other researchers. Benchmarks on various machines (see Table 2, for instance) generally agree

TABLE 1

GNET/Depth Performance on Several Transshipment Examples (Default·Tuning)

Problem	Nodes	Sources	Sinks	Arcs	Percent Cap'd	IBM 360/67 Seconds	Pivots
NG27	400	4	12	2,676	80	2.3	607
NG36	8,000	200	1,000	15,000	0	212	13,012
NG37	5,000	150	800	23,000	0	138	11,610
NG38	3,000	125	500	35,000	0	97	10,637
NG39	5,000	180	700	15,000	0.7	113	9,553
NG40	3,000	100	300	23,000	0.7	67	6,409
NPS	10,000	50	5,000	21,000	100	441	22,153
XN1	5,000	100	4,800	40,000	100	290	12,111

Generated with NETGEN [43]; cost ranges are 1–1000 (XN1) and 1–100 (others).

TABLE 2

GNET/Depth Performance on "NG27"—Calibration with a Highly Capacitated Transshipment Problem

Machine	Solution Seconds
CDC 7600	0.3
IBM 370/168	0.5
CDC 6600	0.6
TI ASC	0.7
IBM 360/91	0.8
UNI 1108	2.2
IBM 360/67	2.3

with standard hardware comparisons of computer performance and show that the times in Tables 1 and 6 are superior to all published times of which we are aware. However, we cannot verify published reports of machine, compiler and installation performance variations. In any case, future codes produced by incorporating fresh research ideas will undoubtedly make current performance records obsolete. We do not hold that primal network codes are anywhere near an asymptotic efficiency level.

The GNET family presently includes GNET/Depth, GNET/Successors and GNET/Preorder-distance as described above, as well as other variants to be discussed below. The three basic versions of GNET have been tested on a suite of randomly generated problems and real formulations to determine which version is best. Experiments on smaller problems (less than 500 nodes and 5,000 arcs) show that all three are remarkably close with the depth version a narrow favorite for sheer speed. Although preorder distance may be preferable for mathematical reasons when extending the data structures to nonnetwork problems, large scale testing has been limited to depth and number of successors versions. GNET/Successors execution times differ from the GNET/Depth times by at most 2 percent on the problems in Table 1.

All versions have a read and edit routine and an arc list sort to create the compressed arc arrays; supplies and demands are determined for each node and the candidate queue is initially loaded with the demand nodes. Simplex pricing via the candidate queue mechanism is used to reduce artificial flows to zero rather than a Phase I–Phase II approach (used in earlier GNET versions). The high cost for artificial arcs is computed as the product of the number of nodes (maximum path length) and the maximum absolute arc cost, thus guaranteeing that a feasible problem will have no flow on artificial arcs in a final basis. The cost is attached only to artificial arcs incident to demand nodes; experiments attaching the cost to arcs incident to supply nodes alone, and to both supplies and demands have not improved preformance. GNET is tuned to exploit a node numbering convention (which is not a strict requirement) ascending from supply through transshipment to demand nodes. A related underlying assumption is that problems will normally have many more sinks than sources. Problems with more sources than sinks or erratic node numbering can usually be easily reformulated to our preference if solution speed is of prime importance; with large scale models this is a minor undertaking in the problem generator software.

Use of random test problems in tuning network codes has its pitfalls. GNET solves real network models much faster than random NETGEN problems of nominally comparable size and structure; this suggests that much remains to be learned from further investigation of special problem structure. Large random test problems are very expensive to generate with NETGEN, requiring about five times more computer time than the associated GNET optimization (but much less region). However, the reproducibility of the experiments and results is so important that it justifies the expense. Also, the cost and awkwardness of distributing and using magnetic tapes with standard problem libraries is avoided. Unfortunately, large NETGEN problems can vary slightly between computers with the length of the real mantissa (this is due to a random number generator which simulates 35 bit integer arithmetic, normalization of the integer result to a real 0-1 variable on the host machine and subsequent transformation back to a uniform integer with the desired range). NETGEN puts capacities of +1 on all arcs in assignment problems, needlessly complicating their solution by a general capacitated network code. Negative demands can also be generated. These and other minor NETGEN problems are overcome by a few program modifications. It also would be worthwhile to add the capability to generate multiple arcs and special problem structures.

GNET execution times do not show much sensitivity to cost ranges, contrary to past reports [60]. Experiments have been performed in which individual problems are solved repeatedly with only the costs modified either by low order digit truncation or addition of uniform random low order digits as necessary to provide the desired cost range. Solution times are very insensitive to cost ranges thus produced, seldom varying by more than 15 percent. The outcome of one such experiment is shown in Table 3. This surprising result appears to be attributable to the candidate queue pricing mechanism.

TABLE 3

GNET/Depth Performance for "NG40" *with Varied Ranges for Random Uniform Costs (Original Range* 1–100)

Cost Range	IBM 360/67 Seconds	Pivots
1–10	84	6,726*
1–100	76	6,933
1–1,000	74	6,921†
1–10,000	74	6,958†

* Low order digit truncated from original problem costs.

† Low order digits generated by RANDU and concatenated with original problem costs.

Uncapacitated transshipment problems are presented to GNET with upper bounds on each arc exceeding total problem supplies. Minor modifications of GNET can reduce solution times by about 10 percent for strictly uncapacitated problems; a more important issue is the potential elimination of the arc-length array of capacities. This space economy may also be realized on lightly capacitated problems by modification of the capacity array or by problem reformulation. One section of a report by Cheong [11] includes a catalog of problem transformations useful in network models and gives results of experiments with several large, lightly capacitated transshipment problems. Two special versions of GNET/Depth were prepared which utilize a well-known transformation to replace each capacitated arc by a pair of uncapacitated arcs and a new node. One modification performs the transformation to the arc list before solution. The other carries out the transformation "on-the-fly" as arcs with capacities are introduced into the basis. Table 4 gives an example of performance for the codes, with the modifications each using one third less space for the arc arrays with little speed degradation. As the proportion of capacitated arcs increases, the space-time tradeoff becomes much less favorable.

TABLE 4

Lightly Capacitated Transshipment Problems—Performance of GNET/Depth and Modifications Solving the Equivalent, Reformulated Uncapacitated Problems [11]

	NT39 IBM 360/67 Seconds	Pivots	NT40 IBM 360/67 Seconds	Pivots
GNET (depth)	113	9,553	67	6,409
Preprocess	113	9,615	71	7,511
Transform on-the-fly	114	9,652	65	6,733

Recently, Cunningham [13] has developed a rule to break ties in the ratio test that guarantees finite convergence for the primal capacitated transshipment problem. For

each pivot either the value of the objective function strictly decreases or the λ (15) is strictly negative, causing a strict decrease in the sum of the dual variables. Thus no basis can repeat. Because of our convention orienting the arcs toward the root, the two cases in [13] reduce to one. Ties in the "min" operation of (16) are resolved in the following order (a) if CP(k) is involved in the tie, the incoming arc is chosen to be the outgoing arc, (b) if any arcs on the X( ) backpath are involved in the tie, the deepest is chosen as the outgoing arc, (c) otherwise, the tied arc with least depth from the CPX( ) backpath to the join is chosen as the outgoing arc. Experiments with a modified GNET/Depth (2 FORTRAN statements changed) on the large test problems show different pivot trajectories but virtually equal times and numbers of pivots. Further analysis reveals that the unmodified GNET has a negative λ on over 90% of the degenerate pivots; this helps explain why the unmodified GNET has never cycled and why Cunningham's modification does not degrade solution times. Since GNET uses exact integer arithmetic, the noncycling modification guarantees finiteness.

Postoptimal Analysis and Reoptimization

In some applications, analysis of the sensitivity of the optimal solution to modifications in the supplies, demands and cost coefficients is desirable. All the postoptimal analysis for linear programming problems can be done with primal transshipment codes. The data structures described above support efficient techniques for this analysis. Ranging [51] of problem coefficients traditionally requires information from the inverse of the optimal basis; columns are required for ranging of supplies and demands, rows are required for ranging of costs of basic arcs. As described above, the predecessor function can generate columns of the inverse and the traversal mechanism can generate rows. Note that while the ranging of cost coefficients of basic arcs is simple in principle and much faster than for linear programming systems, for applications with many arcs it can be time consuming relative to the time to construct the optimal solution.

A more important issue, especially for applications that have a network embedded in a larger problem, is the reoptimization of a problem after modifications to the problem coefficients. The case for primal-dual or dual algorithms is often based on the reputed ease of reoptimization; although reoptimization is conceptually easier than for primal algorithms, this does not imply that primal-dual or dual algorithms reoptimize more quickly. There have not been comparisons of primal and nonprimal algorithms for reoptimization of large scale problems (indeed, it is not clear what types of problems would constitute a fair comparison). In any case, the computer storage advantage of primal algorithms is still maintained since reoptimization requires no additional data arrays.

The following design considerations were used for a GNET reoptimization procedure: (1) there would be no modifications to the primal optimization segments of GNET; (2) no new arcs would be inserted into the arc list; and (3) the initial basis for the reoptimization would include as much of the previous optimal basis as possible. Algebraically, the changes to the supplies, demands, lower bounds and upper bounds are translated into a vector of changes, d, to be added to the original right-hand side vector b. Using the previous optimal basis B, the set of equations $B\hat{x} = b + d$ is solved with artificial arcs introduced into the basis as necessary to avoid an infeasible $\hat{x}$. Since the previous optimal solution $\bar{x}$ is available (b is not), $B\tilde{x} = d$ is solved and then added to $\bar{x}$ to get $\hat{x}$.

The vector d originates from coefficient modifications by: (1) changes in supplies and demands, (2) a change of Δ in the upper bound of an arc (i, j) out of the basis at this upper bound (Δ is added to d_i and subtracted from d_j), and (3) a change of Δ in the lower bound of an arc (i, j) out of the basis at zero (Δ is subtracted from d_i and

added to d_j). A traversal mechanism for back substitution of the basis is provided by a single pass through the IT( ) array. The vector $\tilde{x}$ is constructed by backsolving with the predecessor array P( ), at each step $\hat{x}_i$ is computed as $\bar{x}_i + \tilde{x}_i$. If $\hat{x}_i$ is nonnegative and less than or equal to its capacity, the backsolving continues. Otherwise, an artificial vector from the node to the root is introduced to replace the arc which leaves the basis at zero (if $\tilde{x}_i < 0$) or its capacity (if $\tilde{x}_i >$ capacity). The artificial arc is oriented so that its flow is positive and it is assigned the large cost. The exchange of an arc and an artificial arc involves "rehanging" the node and all its successors; this is accomplished with only three changes in the preorder traversal IT( ) by inserting these nodes at the end of IT( ). Arcs removed from the basis are placed in the candidate queue. The backsolving continues until the whole basis is restored. If any artificial arcs have been introduced it is necessary to recompute the simplex multipliers. The problem is then reoptimized by GNET.

If cost coefficient changes involve only nonbasic arcs, reoptmization is necessary only if one or more of these arcs now price favorably (such arcs are added to the candidate queue). If cost coefficients change for basic arcs it is necessary to recompute the simplex multipliers and then reoptimize with GNET.

Since GNET is so fast, experience indicates that if there are many coefficient changes it may be more efficient to begin with an all artificial basis with the previous optimal basis arcs preloaded in the candidate queue. On the other hand, applications often require many particular, recurring reoptimizations of some special type which permit more efficient (and problem specific) methods to be applied.

Further Refinements and Applications

One of the major ideas of mathematical programming is that elements of the simplex tableau may be generated as needed rather than stored and updated at each pivot. For example, as described above, in GNET the columns of the basis inverse are generated when needed to determine the outgoing arc by iterating the predecessor function. However, the basic flows and simplex multipliers are explicitly stored and those that change are updated during each pivot. Analysis of computational experiments shows that the major portion of the calculations in the pivot (step S3) is to update the simplex multipliers U( ), depth D( ) and the preorder traversal array IT( ) (note that IT( ) is maintained solely to allow efficient update of the multipliers). The authors were thus led to consider modifications to GNET that would allow some or all of the multipliers to be generated as needed.

It follows immediately from (10) that for basis arc k from node i to j: $C(k) - U(i) + U(j) = 0$. Arbitrarily setting $U(\text{root}) = 0$, it is possible to solve for all the U( )'s by forward substitution with the triangulated basis, that is, solve for U( )'s in the order IT(root), IT(IT(root)), etc. To compute a particular $U(h)$ it is necessary only to calculate some of the other U( )'s, namely those for the nodes on the backpath from h to the root. If k is the arc joining node h to its predecessor $P(h)$, then $U(h) = U(P(h)) \pm C(k)$ with "+" if the arc is oriented from h to $P(h)$ and "−" otherwise. Since there are many nodes in large problems and only a relatively few multipliers change from pivot to pivot, it does not seem worthwhile to generate all multipliers at each pivot. Rather, we choose to store explicitly enough of the multipliers so that for each node i either $U(i)$ is stored explicitly or $U(P(i))$ is stored explicitly. In the latter case a single addition generates the multiplier only when it is needed.

The impetus for this approach comes from consideration of the capacitated transportation problem with many demand nodes (sinks) and relatively few supply nodes (sources). This is an important special case of the minimum cost network model that

has many applications. A typical model is a distribution system with a few plants and many warehouses or a few centralized warehouses and many customers. This is the type of problem most often encountered in multicommodity distribution problems, e.g. [23]. Another application that requires the repeated solution of many such problems is the traffic assignment problem.

A special version of GNET/Depth called TNET has been developed for this problem. The multipliers are explicitly kept only for the relatively few sources of the problem. Since in the (bipartite) transportation problem arcs only join sources to sinks, the predecessor of each sink node must be a source and thus the multiplier for each sink is computed by a single addition (also, the pricing mechanism loads only sinks as interesting nodes during the opening gambit). The GNET arc storage is ideal for this calculation since all the arcs with sink node j are stored contiguously; thus in the general scan all are priced out together and only a single calculation of the sink multiplier is required.

Since the traversal array IT( ) is maintained solely for the purpose of updating the multipliers, IT( ) and D( ) need to be maintained only for the relatively few source nodes. It is also convenient to include in IT( ) the relatively few sinks that have a successor. An alternate description is that IT( ) is maintained only for the smallest possible subarborescence that includes all the sources; this is easy to do since at each pivot the only possible nodes that can join the subarborescence are the two ends of the incoming arc, and the only possible nodes to leave are the two ends of the outgoing arc. No additional storage is required, the U(i) for sink i is used to store the cost of the arc from P(i) to i, and the sinks not in the subarborescence are marked with 0 in D( ). The predecessor array P( ) is maintained exactly as before and the determination of the backpaths and outgoing arc is the same.

Experiments (see Table 5) show that TNET is significantly faster than GNET for transportation problems with many more sinks than sources. The pivot choice is the same in both the original and modified versions so that with the same value of the pricing parameters the same sequence of pivots is generated. For NNE = 32, the reduction in time for TNET is due solely to savings in the update of U( ), IT( ), and D( ) in step S3 (step S2 is identical and step S1 is slightly slower for TNET). Experiments with six smaller problems show that GNET spends roughly half its time in step S1; with TNET there is 80% less time for S3 and 5% more time in S1. Since the pivot step S3 is so much faster in the modification it is not worthwhile to select the incoming arc as carefully as in GNET; experiments show that NNE = 8 is a good setting for TNET. Table 5 shows that with NNE = 8 there are significantly more pivots than with NNE = 32, but total solution time is reduced.

TABLE 5

Transportation Problems with Relatively Few Sources

Problem	Sources	Sinks	Arcs	NNE	IBM 360/67 Seconds TNET	GNET	Pivots
TN7	100	4,900	40,000	32	127	274	11,909
				8	99		17,583
TN8	250	4,750	40,000	32	142	285	12,910
				8	135		19,759
TN9	10	4,990	40,000	32	104	261	9,574
				8	79		12,327

Generated with NETGEN [43]; all arcs are capacitated; cost ranges are 1–100 (TN7) and 1–1000 (TN8, TN9).

Although TNET is designed for transportation problems with many sinks and relatively few sources, it does well (about the same as GNET) on problems with an equal number of sources and sinks. TNET handles problems with many sources and few sinks by a minor modification in the input that reverses the orientation of the arcs.

The same idea is extended to the general minimum cost network problem in a program called XNET. The multipliers are explicitly kept only for nodes that have successors in the predecessor graph. For example, in Figure 2 IT( ) is maintained only for nodes 13, 8, 1, 4, 3, 2, and 6, in that order. The U( ) and D( ) arrays for nodes with no successors are used as in TNET. In order to (dynamically) keep the multipliers only for the nodes with successors, it is necessary in these (nonbipartite) problems to maintain an array that records for a node the number of its successors that do not currently have explicit multipliers ("aggregated successors" A( )). In this way, one can distinguish between a node with no successors and a node with none of its successors currently in the IT( ) array.

The results in Table 6 indicate that XNET is faster than GNET on all problems and significantly faster on problems with relatively many sinks. XNET is only slightly slower than TNET on the problems that TNET is tailored for, but the loss on these problems is balanced by its generality and by its dominance of GNET. XNET is successful because the predecessor graphs have many nodes with no successors. In many practical problems known to the authors, most nodes are pure sinks; since the successor in the predecessor graph of a pure sink must be a node that is not a pure sink, many pure sink nodes have no successors.

TABLE 6

XNET/Depth Performance with NNE = 8

Problem	IBM 360/67 Seconds	Pivots	Problem	IBM 360/67 Seconds	Pivots
NG27	2.7	964	NPS	265	29,045
NG36	111	17,993	XN1	136	19,726
NG37	110	18,195			
NG38	94	13,124	TN7	112	17,583
NG39	76	14,809	TN8	148	19,759
NG40	59	11,002	TN9	92	12,327

TNET and XNET are refinements of GNET/Depth; number of successors or preorder distance can also be used for such modifications.

A modified version of XNET is also being used on a set of large, 100-percent dense problems. The network models are uncapacitated single commodity transportation problems embedded in a recent implementation of a multicommodity, multiple time period econometric model described in [37]. A prototype problem size is 200 sources and 300 sinks (and thus 60,000 arcs) with a matrix of region-to-region bulk transport costs. The XNET modification is stripped of arc-length arrays, list references are modified for the cost matrix and several capacitated features removed. Also, the candidate queue is modified to access only sink nodes and to price a restricted menu of the few (ultimately 5) cheapest sources for each sink during the opening gambit. Optimization from a cold start requires 8.6 seconds (2350 pivots) on an IBM 370/168. Tuning of pricing parameters produces surprisingly little improvement. Reoptimization procedures are employed to exploit period-to-period similarity of optimal bases (despite significant temporal variations in problem structure). A typical reoptimization time from a crashed basis is 1.2 seconds.

A similar modification of XNET is imbedded in a ship scheduling system described by Sibre [56]. The system is an enhanced large scale implementation of the production scheduling model of Geoffrion and Graves [24] which requires solution of a large scale (Koopmans-Beckman) quadratic assignment problem. Sibre develops a new quadratic assignment method which builds upon the work of Graves and Whinston [35] by using as an advanced starting solution a large embedded network formulation of the fixed charge portion of the problem.

A typical prototype assignment problem for this scheduling model has 1,720 nodes (860 sources and sinks) and 739,600 arcs. Full optimization requires 9,602 pivots in 121.3 compute seconds (2.1 minutes) on an IBM 370/168. Since the ratio tests and pivoting account for less than 5 percent of this solution time, no modifications have been made to the code for exploiting the special (bipartite) basis structure of the assignment problem. Based on benchmark tests, the original Graves-Whinston method is estimated to require about 5,143 seconds (1.4 hours) for this portion of the problem, and is not expected to produce as good a global solution.

Historical Perspective

Although the authors have not had access to any other primal transshipment computer programs, it is possible to identify some of the major ideas from papers and presentations [10], [28]–[31], [34], [49], [50], [59], [60]. The major design decisions and important coding specifics may vary widely in these systems, but all the successful contemporary large scale codes seem to be based on a few key ideas. Notwithstanding our limited knowledge of specific techniques used by others, we trace the development of these ideas as best we can.

The major ideas underlying all contemporary primal network codes are the representation of the basis as a predecessor graph, a traversal mechanism to update simplex multipliers, the use of depth or number of successors or preorder distance to determine the outgoing arc and to facilitate the update of the simplex multipliers, the use of a pricing mechanism such as the candidate queue, and generation of simplex multipliers.

The predecessor array is used in all the codes. This construction in primal network codes goes back at least to Glicksman, L. Johnson and Eselson [27], 1960, and goes back further in other disciplines. It is a standard approach in many research areas.

Srinivasan and Thompson [59], 1972, have proposed the use of depth to identify the backpaths and to determine the outgoing arc. This has been adopted by Glover, Karney and Klingman [28], 1974, Mulvey [49], 1974, [50], 1975, and GNET/Depth [6], 1975. The method [59] for identifying the join of the backpaths is exactly as described for GNET; however, the update of depth at each pivot in that paper is a more involved approach that does not aid in the pivot as described here.

Preorder traversal has been used in transshipment codes by Glover, Klingman and Stutz [31], 1974, who call it the augmented threaded index method. They show that it is more efficient than the triple labeling scheme of E. Johnson [40]. Independently, McBride [48], 1973, and Graves and McBride [34], 1973, have developed the zero-to-right triangulation of network bases which has since been shown to be equivalent to preorder. Preorder has been used in the computer science literature for at least 20 years and has been used in contemporary shortest path algorithms [26], 1973. Preorder has been adopted by Srinivasan and Thompson and Mulvey, and it is used in all versions of GNET. As discussed above and shown in Figure 6, depth makes possible a one-pass update of the simplex multipliers that simultaneously updates the preorder and depth (or number of successors, or preorder distance). Since the pivot takes much more time than the determination of the outgoing arc, the use of depth in the update of the simplex multipliers is more critical to the success of GNET than its use to find

the outgoing arc. We do not know the pivot details or the use of depth (if any) in other codes nor do we know if any others have a one-pass update.

Srinivasan and Thompson [59], 1972, have proposed number of successors to be used with depth to determine how many nodes are in the subarborescence that is rehung in the pivot. If the subarborescence has less than half the nodes of the problem, they propose moving the root and performing the pivot on the smaller part of the predecessor graph. Our experiments show that the subarborescence can always be expected to have significantly less than half the nodes, so we have rejected the idea of moving the root. However, we have discovered that number of successors can be used in a way not envisaged by Srinivasan and Thompson. As described above it can replace depth to support the determination of the outgoing arc and the one-pass update of the simplex multipliers. We have also noted that the number of successors for nodes in the pivot subarborescence changes only for nodes on the pivot stem avoiding the update indicated in [59]. Subsequent to our presentation of GNET/Successors, Glover and Klingman [30], 1975, have proposed a number-of-successors version of their algorithm. They describe a one-pass update of the simplex multipliers that uses an additional node length array to store for each node the last successor ranked by preorder. They conjecture [30, p. 7] that a code based on this approach will dominate their previous codes. This may be true for their codes, but as discussed above this is not our experience in comparing our one-pass depth version against our one-pass number of successors version. Also, as indicated in Figure 7, GNET/Successors accomplishes a one-pass update without an additional array to store and manipulate.

Preorder distance is used by McBride [48], 1973, and Graves and McBride [34], 1973. They develop the zero-to-right property and establish the proposition stated above to find the join.

Mulvey [49] uses a candidate list of arcs that corresponds to the "partial suboptimization" that is used in commercial linear programming systems [51]. His candidate list is controlled by two parameters: x_1 and x_2. In our terminology, general scans of nodes are done until there are x_2 arcs on the candidate list, then up to x_1 pivots are performed by choosing arcs from the candidate list. The candidate list is then discarded and the process begins again. As noted in [10], Mulvey's approach has been adopted by Glover and Klingman.

The candidate queue used in GNET has evolved from similar mechanisms developed by Graves for more general mathematical programming systems. It is unique in that it contains both nodes and arcs, it is used cyclically and arcs are never removed as long as they continue to price out favorably.

The Srinivasan and Thompson code [60] and the Harris code [36] are for dense uncapacitated transportation problems; all other contemporary codes known to the authors solve the capacitated transshipment problem with a sparse representation of the network similar to that described here.

Development of a version with an in-core, out-of-core arc list is reported by Karney and Klingman [41].

Harris [36], 1976, describes a code for dense uncapacitated transportation problems that have few sources and many sinks. The brief description indicates that the simplex multipliers are not stored for the sinks but there are no details on the handling of the pivot nor on the use of a traversal mechanism. (In a recent private communication, Harris indicates that his code uses depth and a preorder traversal.) Independently, the authors have developed TNET for the sparse capacitated transportation problem and have subsequently developed XNET for the general capacitated transshipment problem. TNET and XNET are specific examples of the generation of simplex multipliers; other refinements that generate more (or all) the multipliers are possible.

There is considerable literature on postoptimal analysis and parametric programming for transportation and transshipment problems (e.g. [9], [57], [58]).

Conclusion

The GNET programs are small, fast and easy to modify. They integrate many well-known techniques from mathematical optimization and computer science. It is important, however, to discriminate carefully between the underlying ideas of mathematical programming and the computer science topics applied to their implementation. The computer science literature has contributed a great deal to the local representation of global information in trees and graphs, and has given valuable recursive methods for manipulating data structures; many of these techniques can be applied to the basic arborescence. But some, such as rerooting, pruning, balancing and even conversion to an equivalent binary tree with extra dummy nodes and arcs, do not work well for many network problems. In these cases, the mathematical interpretation implies (as does the experimental evidence) that these devices are needless complications that increase solution expense or introduce superfluous equations and variables.

In our experience large scale problems are always created from source data by problem generator programs. The problem generator may occupy significantly less computer storage than the file of coefficients produced. Thus, is can be worthwhile to generate the data as needed rather than store it explicitly. Such generators are easily incorporated in GNET. (This approach does not entirely avoid arc length information for capacitated problems since a record must be maintained for each arc out of the basis at its upper bound.) These advantages also invite development and use of a random problem generator subroutine to replace the cumbersome problem files and cost of using NETGEN [43] to generate very large problems. The design of GNET is also consistent with the use of explicit arc arrays stored in peripheral devices.

Perhaps the most important potential for the pure network solution speed of GNET lies in more general large scale models with embedded networks in their structure. The multicommodity distribution system design code of Geoffrion and Graves [23] has been revised to incorporate GNET to repeatedly solve the (numerous) network subproblems. A goal programming code for network problems with quadratic objective functions has been successfully built by the authors, with GNET used as the key subroutine.

The theme of GNET is replacement of arithmetic primal simplex computations by simple but equivalent logical tests. This is reminiscent of the motives for generalized upper bounding and suggests that network factorization may prove to be a viable competitor for GUB in general linear programming systems [33], [34], [48].

Lee [47] indicates that truly huge models may be solved by mathematical aggregation and develops a wide class of network aggregation methods producing surrogate problems that are pure networks which are smaller in size and which preserve and exploit special global structure in the original problem. This approach is intriguing due to the curious general improvement in performance encountered when solving real models rather than random test problems of equivalent size.

The FORTRAN program GNET/Depth [6], 1975, is distributed to researchers for a nominal handling charge on an exclusive use basis. For further information write Professor Glenn Graves at the Western Management Science Institute, Graduate School of Management, University of California, Los Angeles, California 90024.[1]

[1] The research for this Exceptional Paper has been partially supported by the National Science Foundation and the Office of Naval Research.

References

1. BALAS, E. AND HAMMER, P. L., "On the Transportation Problem—Part I," *Cahiers du Centre d'Etudes de Recherche Operationelle*, Vol. 4, No. 2 (1962), p. 98.
2. BARR, R. S., GLOVER, F. AND KLINGMAN, D., "An Improved Version of the Out-of-kilter Method and a Comparative Study of Computer Codes," *Mathematical Programming*, Vol. 7, No. 1 (August 1974), p. 60.
3. BARTELS, R. H. AND GOLUB, G. H., "The Simplex Method of Linear Programming Using LU Decomposition," *Communications of the Association for Computing Machinery*, Vol. 12, No. 5 (May 1970), p. 266.
4. BERGE, C., *The Theory of Graphs and its Applications*, translated by A. Doig, John Wiley and Sons, New York, 1962.
5. BRADLEY, G. H., "Survey of Deterministic Networks," *AIIE Transactions*, Vol. 7, No. 3 (September 1975), p. 222.
6. ———, BROWN, G. G. AND GRAVES, G. W., "GNET, A Primal Capacitated Network Program," Copyright 1975, 1977.
7. BUSACKER, R. G. AND GOWEN, P. J., "A Procedure for Determining a Family of Minimum-Cost Network Flow Patterns," ORO Technical Report 15, Operations Research Office, Johns Hopkins University (1961).
8. ——— AND SAATY, T., *Finite Graphs and Networks*, McGraw-Hill, New York, 1965.
9. CHARNES, A. AND COOPER, W. W., *Management Models and Industrial Applications of Linear Programming*, Volumes I and II, John Wiley and Sons, New York, 1961.
10. ———, GLOVER, F., KARNEY, D., KLINGMAN, D. AND STUTZ, J., "Past, Present and Future of Large Scale Transshipment Computer Codes and Applications," Research Report CS 131, Center for Cybernetic Studies, The University of Texas, Austin (July 1973, revised October 1974).
11. CHEONG, Y. P., "Network Transformations and Some Applications," MS Thesis, Naval Postgraduate School (December 1975).
12. CLASEN, R. J., "The Numerical Solution of Network Problems Using the Out-of-kilter Algorithm," RAND Memorandum RM-5456-PR, Santa Monica, California (March 1968).
13. CUNNINGHAM, W. H., "A Network Simplex Method," *Mathematical Programming*, Vol. 11, No. 2 (October 1976), p. 105.
14. DANTZIG, G. B., "Application of the Simplex Method to a Transportation Problem," in *Activity Analysis of Production and Allocation*, T. C. Koopmans, ed., John Wiley and Sons, New York, 1951.
15. ———, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.
16. ——— AND VAN SLYKE, R. M., "Generalized Upper Bounding Techniques for Linear Programming," *Journal of Computer and System Sciences*, Vol. 1, No. 3 (October 1967), p. 213.
17. EDMONDS, J. AND KARP, R. M., "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," *Journal of the Association for Computing Machinery*, Vol. 19, No. 2 (April 1972), p. 248.
18. ELMAGHRABY, S., *Some Network Models in Operations Research*, Springer-Verlag, New York, 1970.
19. FORD, L. R. AND FULKERSON, D. R., "A Primal-Dual Algorithm for the Capacitated Hitchcock Problem," *Naval Research Logistics Quarterly*, Vol. 4, No. 1 (March 1957), p. 47.
20. ——— AND ———, *Flows in Networks*, Princeton University Press, Princeton, New Jersey, 1962.
21. FULKERSON, D. R., "An Out-of-Kilter Method for Minimal-Cost Flow Problems," *SIAM Journal of Applied Mathematics*, Vol. 9, No. 1 (March 1961), p. 18.
22. ———, "Flow Networks and Combinatorial Operations Research," *American Mathematical Monthly*, Vol. 73, No. 2 (February 1966), p. 115.
23. GEOFFRION, A. M. AND GRAVES, G. W., "Multicommodity Distribution System Design by Benders Decomposition," *Management Science*, Vol. 20, No. 5 (January 1974), p. 822.
24. ——— AND ———, "Scheduling Parallel Production Lines with Changeover Costs: Practical Application of a Quadratic Assignment/LP Approach," *Operations Research*, Vol. 24, No. 4 (July 1976), p. 595.
25. GILL, P. E. AND MURRAY, W., "A Numerically Stable Form of the Simplex Algorithm," *Journal of Linear Algebra and its Applications*, Vol. 6 (1973), p. 99.
26. GILSINN, J. AND WITZGALL, C., "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees," Technical Note 772, National Bureau of Standards, Washington, D.C. (May 1973).
27. GLICKSMAN, S., JOHNSON, L. AND ESELSON, L., "Coding the Transportation Problem," *Naval Research Logistics Quarterly*, Vol. 7, No. 2 (June 1960), p. 169.
28. GLOVER, F., KARNEY, D. AND KLINGMAN, D., "Implementation and Computational Comparisons of Primal, Dual and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems," *Networks*, Vol. 4, No. 3 (1974), p. 191.

29. ———, ———, ——— AND NAPIER, A., "A Computational Study on Start Procedures, Basis Change Criteria and Solution Algorithms for Transportation Problems," *Management Science*, Vol. 20, No. 5 (January 1974), p. 793.
30. ——— AND KLINGMAN, D., "Improved Labeling of L. P. Bases in Networks," Research Report CCS 218, Center for Cybernetic Studies, The University of Texas, Austin (August 1974, revised October 1975).
31. ———, ——— AND STUTZ, J., "Augmented Threaded Index Method for Network Optimization," *INFOR*, Vol. 12, No. 3 (October 1974), p. 293.
32. GRAVES, G. W., "Theory of Permutation Triangulation with Application to Network Flow Problems," Working Paper No. 267, Western Management Science Institute, UCLA (May 1977).
33. ——— AND MCBRIDE, R. D., "The Factorization Approach to Large-Scale Linear Programming," Eighth International Symposium on Mathematical Programming, Stanford University, August 1973 (also in *Mathematical Programming*, Vol. 10, No. 1 (February 1976) p. 91).
34. ——— AND ———, "A Factorization Algorithm for Network Problems with Side Constraints," 44th National ORSA Meeting, San Diego (Fall 1973).
35. ——— AND WHINSTON, A. B., "An Algorithm for the Quadratic Assignment Problem," *Management Science*, Vol. 17, No. 7 (March 1970), p. 453.
36. HARRIS, B., "A Code for the Transportation Problem of Linear Programming," *Journal of the Association for Computing Machinery*, Vol. 23, No. 1 (January 1976), p. 155.
37. HARRIS, C. C. AND HOPKINS, F. E., *Locational Analysis—An Interregional Econometric Model of Agriculture, Mining, Manufacturing, and Services*, Lexington Books, Lexington, Massachusetts, 1972.
38. HATCH, R. S., "Bench Marks Comparing Transportation Codes based on Primal Simplex and Primal-Dual Algorithms," *Operations Research*, Vol. 23, No. 6 (November 1975), p. 1167.
39. HITCHCOCK, F. L., "The Distribution of a Product from Several Sources to Numerous Localities," *Journal of Mathematics and Physics*, Vol. 20, No. 2 (April 1941), p. 224.
40. JOHNSON, E. L., "Networks and Basic Solutions," *Operations Research*, Vol. 14, No. 4 (August 1966), p. 619.
41. KARNEY, D. AND KLINGMAN, D., "Implementation and Computational Study on an In-Core, Out-of-Core Primal Network Code," *Operations Research*, Vol. 24, No. 6 (December 1976), p. 1056.
42. KLEIN, M., "A Primal Method for Minimal Cost Flows with Application to the Assignment and Transportation Problems," *Management Science*, Vol. 14, No. 3 (November 1967), p. 205.
43. KLINGMAN, D., NAPIER, A. AND STUTZ, J., "NETGEN—A Program for Generating Large Scale (Un) Capacitated Assignment, Transportation and Minimum Cost Flow Network Problems," *Management Science*, Vol. 20, No. 5 (January 1974), p. 814.
44. KOOPMANS, T. C., "Optimum Utilization of the Transportation System," *Proceedings of the International Statistical Conferences*, Washington, D.C. (1947), published in Volume 5 (1949), p. 136. (Also in *Scientific Papers of Tjalling C. Koopmans*, Springer-Verlag, New York (1970), p. 184.)
45. KNUTH, D. E., *The Art of Computer Programming, Volume 1 (Fundamental Algorithms)*, Addison-Wesley, Reading, Massachusetts, 1968.
46. LANGLEY, R. W., KENNINGTON, J. AND SHETTY, C. M., "Efficient Computational Devices for the Capacitated Transportation Problem," *Naval Research Logistics Quarterly*, Vol. 21, No. 4 (December 1974), p. 637.
47. LEE, S., "Surrogate Programming by Aggregation," Ph.D. dissertation, UCLA (September 1975).
48. MCBRIDE, R., "Factorization in Large-Scale Linear Programming," Working Paper No. 200 (also Ph.D. dissertation), Western Management Science Institute, UCLA (June 1973).
49. MULVEY, J., "Column Weighting Factors and Other Enhancements to the Augmented Threaded Index Method for Network Optimization," Joint National Meeting of ORSA/TIMS, San Juan, Puerto Rico (Fall 1974).
50. ———, "Special Structure in Network Models and Associated Applications," Ph.D. dissertation, UCLA (August 1975).
51. ORCHARD-HAYS, W., *Advanced Linear-Programming Computing Techniques*, McGraw-Hill, New York, 1968.
52. ORDEN, A., "The Transshipment Problem," *Management Science*, Vol. 2, No. 3 (April 1956).
53. SAUNDERS, M. A., "Large Scale Linear Programming using the Cholesky Factorization," Technical Report CS-72-252, Computer Science Department, Stanford University (1972).
54. ———, "Product Form of the Cholesky Factorization for Large-Scale Linear Programming," Technical Report CS-72-301, Computer Science Department, Stanford University (1972).
55. SCOINS, H. I., "The Compact Representation of a Rooted Tree and the Transportation Problem," International Symposium on Mathematical Programming, London (1964).
56. SIBRE, C. E., "A Ship Scheduling Model for the United States Coast Guard: a Quadratic Assignment/Linear Programming Approach," MS Thesis, Naval Postgraduate School (June 1977).

57. SRINIVASAN, V. AND THOMPSON, G. L., "An Operator Theory for the Transportation Problem—I," *Naval Research Logistics Quarterly*, Vol. 19, No. 2 (June 1972), p. 205.
58. ——— AND ———, "An Operator Theory for the Transportation Problem—II," *Naval Research Logistics Quarterly*, Vol. 19, No. 2 (June 1972), p. 227.
59. ——— AND ———, "Accelerated Algorithms for Labeling and Relabeling of Trees with Application for Distribution Problems," *Journal of the Association for Computing Machinery*, Vol. 19, No. 4 (October 1972), p. 712.
60. ——— AND ———, "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," *Journal of the Association for Computing Machinery*, Vol. 20, No. 2 (April 1973), p. 194.