NETWORK OPTIMIZATION

by

R. R. Meyer

Computer Sciences Technical Report #595

May 1985

NETWORK OPTIMIZATION*

R. R. Meyer
Computer Sciences Department and Mathematics Research Center
The University of Wisconsin-Madison
Madison, Wisconsin, USA 53706

ABSTRACT

Network optimization not only has a long and distinguished history within the field of mathematical programming, but also continues to be one of the principal ongoing areas of research in optimization . This is due in part to the abundance of network applications, and in part to the fortuitous confluence of mathematical theory and computer science research that makes possible the optimization of linear network problems involving millions of variables, while simultaneously posing the challenge of adapting these techniques to nonlinear networks with thousands of variables. This paper endeavors to survey recent theoretical and computational developments in deterministic network optimization, with emphasis on convex as opposed to nonconvex or combinatorial problems.

* Research supported in part by NSF grant MCS8200632 and ARO contract DAAG29-80-C-0041 . This paper was presented at the NATO Advanced Study Institute on Computational Mathematical Programming, Bad Windsheim, West Germany, 1984, and will appear in the proceedings of that conference.

1. Introduction

The earliest work in what may be regarded as modern network optimization coincides with the first modern paper in mathematical programming, namely that of [Kantorovich 1939] dealing with railway networks. It is worthwhile noting that both Kantorovich and Koopmans, who also did early work on the transportation problem [Koopmans 1949], received the Nobel Prize in 1975 in recognition of their contributions in the area of methods for the allocation of resources.

Other notable early achievements include Hitchcock's work on transportation [Hitchcock 1941] and the implementation of the stepping stone method on the National Bureau of Standards Eastern Automatic Computer in 1952, perhaps the earliest example of "production" grade mathematical programming software. After the initial dominance of primal simplex method approaches to linear network optimization, work by Ford and Fulkerson [Ford and Fulkerson 1962] led to the popularity of primal-dual methods during the 1960's. The computational pendulum drifted back to primal simplex methods again in the 1970's as a result of the development and exploitation of suitable computer science data structures by Johnson, Glover, Karney, Klingman, and others ([Johnson 1966], [Glover, Karney, and Klingman 1974]). At present the primal simplex method (specialized to take advantage of the tree structure of the basis) remains the method of choice for the general linear minimum cost network flow problem, although very promising results (particularly for transportation and assignment problems) have recently been reported in [Bertsekas and Tseng 1983] with a method based on some concepts from monotropic programming [Rockafellar 1984].

In section 2 of this paper we discuss some of the key concepts that have been used to develop efficient specializations of the primal simplex method to linear network optimization problems. The so-called network simplex method based on these techniques has been used to solve problems of up to 65,000,000 variables and 50,000 constraints [Barr and Turner 81]. These problems are perhaps the largest optimization problems ever solved. Multicommodity network techniques are described in section 3, along with extensions to networks with side constraints and variables, and so-called generalized networks, also known as networks with gains and losses. Finally section 4 surveys algorithms for networks with nonlinear convex objective functions, and notes the interrelationship between the use of network algorithms for computer network design and the use of local area computer networks (including the Crystal multicomputer at the Computer Sciences Department of the University of Wisconsin-Madison) to solve network optimization problems via parallel algorithms.

2. Linear Single-Commodity Network Optimization

The problem class to be considered in this section is

$$\begin{aligned} \min_x \ & \mathbf{cx} \\ s.t. \ & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \qquad \text{(NET)}$$

where $\mathbf{x}$ is the n-vector $(x_1, \ldots, x_n)^T$ in which x_i represents the flow on arc i; $\mathbf{c}$ is the vector of objective function coefficients (the juxtaposition of two vectors will be used to denote their inner product, so that $\mathbf{cx} = \sum_{i=1}^{n} c_i x_i$); $\mathbf{l}$ is a vector of lower bounds; $\mathbf{u}$ is a vector of upper bounds; A is the so-called *node-arc incidence matrix* of a directed graph or digraph with m nodes and n arcs, where if the kth arc goes from node i to node j (this arc will be denoted as (i, j))), then kth column of A has a $+1$ in row i and a -1 in row j (note that some authors use the opposite sign convention for these entries) and $0's$ elsewhere in the column; and $\mathbf{b}$ is the right-hand side (RHS) vector. When it is necessary to refer to the variable associated with an arc (i, j), this variable will be denoted as x_{ij}, and a similar convention will be used for cost coefficients. Each node is assumed to have at least one incident arc (i.e., an arc with that node as origin or destination) and each arc is assumed to have distinct origin and destination nodes (i.e., no self-loops are allowed). (Note that since $(1, 1, \ldots 1)A = 0$, a necessary condition for the feasibility of the system $Ax = b$ is that $\sum_{i=1}^{m} b_i = 0$. Without loss of generality we assume that this condition is satisfied as it is easily checked at input time.) Although for the sake of simplicity in presenting the key ideas of the network simplex method we massume that all variables have upper and lower bounds, the algorithm below may be augmented in a straightforward manner to handle unbounded variables and to detect unbounded problems. Also, for notational convenience we will assume that the lower bound has been translated to 0. With the sign convention on the entries of A, it is easily seen that a given row, say row i, of the product Ax corresponds to the sum of the flow variables for the arcs leaving node i minus the sum of the flow variables for the arcs entering node i. This quantity is often call the *divergence* of row i, and may be thought of as the *net* amount of flow leaving node i (if the node is a *conservation* node or a *supply* node) or the negative of the net amount remaining at node i if the node is a *demand* node. Thus, the three fundamental types of constraints occurring in the system $Ax = b$ are supply, demand, and conservation of flow constraints. Many other types of constraints can be converted to this network format, including supply and demand inequalities, bounds on the amount flowing through a node, bounds on the total

amount flowing out of a subset of nodes, etc. In cases in which the transformation to the standard network format is not obvious, there exist algorithms which determine if classes of linear transformation can be used to convert the problem to that format [Bixby 81].

Finally, the corresponding digraph is assumed to be *connected*, so that between every pair of nodes (i, j) there exists a *path* of ordered nodes and arcs such that node i is the initial node of the path, j is the last node of the path, and if i_ℓ and $i_{\ell+1}$ are consecutive nodes in the path, then the ℓ^{th} arc in the path is either $(i_\ell, i_{\ell+1})$ or $(i_{\ell+1}, i_\ell)$. (Thus, the orientation of the arcs in the path is arbitrary.) (If the digraph is not connected, it is easily seen that the problem (NET) may be decomposed into a set of separate optimization problems, each of which corresponds to a maximal connected component of the digraph).

It should be noted that in the (NET) problem format, the columns of the matrix A correspond to arcs of the network whereas the rows of A correspond to nodes. In stating optimality conditions, we will associate dual variables with the rows of A, so from a network viewpoint, these dual variables are associated with the corresponding nodes, while the primal variables are associated with the arcs.

Computationally it has been found efficient [Grigoriadis 82] to start the network simplex method with an all artificial basic feasible solution (BFS). Given a problem in the form (NET), the network structure may be maintained in spite of the addition of a full set of artificial variables by the following procedure: (1) to add the artificial variable x_{n+i}, add to the matrix A the i^{th} unit vector if $b_i \geq 0$, and the negative of the i^{th} unit vector otherwise; (2) form the negative of the sum of the rows of this augmented matrix, and then add the resulting row as an *additional row* of constraint coefficients with corresponding RHS value $b_{m+1} = -\sum_{i=1}^{m} b_i = 0$. It is easily seen that this additional row introduces a second non-zero of opposite sign in every artificial column, while adding only an additional zero to each column corresponding to an original variable. This new row may be thought of as corresponding to an *artificial node* $m+1$, to which each original node is connected by an arc of the appropriate direction. Computational experience [Grigoriadis 82] indicates that the best approach for handling the artificials is a gradual-penalty method in which a relatively small penalty coefficient (of the order of the objective function coefficient largest in optimal value) is first used for the artificials, followed by an increase in the penalty each time the optimal solution contains artificials at positive levels. If positive values of artificials persist even after the penalty has reached a sufficiently high level, then the original problem is declared infeasible. (It may be shown that the problem (NET) has the nice property that the penalty coefficient does not have to be raised above $mC/2$, where C is $\max_{1 \leq i \leq n} |c_i|$, assuming this is positive.)

A property of the node-arc incidence matrix A that results to a minor technical difficulty is the linear dependence of the rows of A. Since each column of A contains exactly two non-zeros, one $+1$ and one -1, it follows that the sum of the rows is the 0 vector. In order to be able to deal with basis matrices in a notationally simple manner, we will assume A has been augmented not only by the additional row and columns discussed above, but also by the column consisting of the $(m+1)st$ unit vector $(0,0,...,0,1)^T$, and that the corresponding variable, x_{root}, has been assigned lower and upper bounds of 0 and a 0 cost coefficient. Clearly, this does not affect the solution of the original problem (NET) in any way, but it does result in a constraint coefficient matrix of fall rank (see, e.g., [Kennington and Helgason 80]. From a graphical standpoint, x_{root} is considered to be the flow on a so-called *root arc* that originates at node $m+1$, but has no terminating node. (Node $m+1$ is thus termed the *root node* or simply the root). Without loss of generality we will assume that the problem (NET) also contains the variable x_{root} and its corresponding data as described above. (This variable actually plays no role in computation, since its value is always 0.) Since the root arc is included in every basis, the dual variables π associated with a basis B by the equation $\pi B = c_B$ have the property that $\pi_{m+1} = 0$. Moreover, since every other column has one $+1$ and one -1, these equations have the form

$$\pi_i - \pi_j = c_{ij} \qquad (PI)$$

for every other basic arc. The simple form of these equations helps to make the update of the dual variables after a pivot relatively easy, as we will see below. Because the initial basic arcs all begin or end at node $m+1$, the initial $\pi's$ (other than π_{m+1}) are simply $\pm K$, where K is the penalty cost associated with the artificial arcs.

Another key mathematical property of the problem (NET) that leads to enormous computational speedups in the application of the primal simplex method is the fact that there is a one-to-one correspondence between the $(m+1) \times (m+1)$ basis matrices of A and the *rooted spanning trees* of the corresponding network, where a rooted spanning tree is a connected subgraph that contains the root arc and all of the nodes of the original graph, but no cycles, a cycle being a non-trivial path whose origin and terminal nodes coincide (for a proof of this correspondence see, e.g., [Kennington and Helgason 80]). (An intuitive justification of this correspondence may be obtained by noting that connectedness is related to the *existence* of basic solutions for arbitrary RHS's - in particular for those RHS's corresponding to the selection of any given node as the sole supply node and any other node as the sole demand node, thereby necessitating a path between those nodes to correspond to the solution flow – whereas acyclicity is related to *uniqueness* of solutions, since the presence of a cycle would allow a feasible solution to be transformed into another

feasible solution by the superposition of flows on the cycle without any effect on the divergence equations. More will be said about cycle flows in the discussion of the pivot operation below). For simplicity, we will simply refer to rooted spanning trees as trees .

The *computational* utility of trees arises from the fact that the pivot operation of the network simplex method may be carried out by utilizing data structures (representing the tree) that make it relatively easy to identify the variable (i.e., arc) that leaves the basis and to carry out the updating of the primal and dual variables. In fact, it is the simplicity of the arithmetic of those operations that allows single-commodity network flow problems to be solved about 100 times faster than general LP's of the same size.

It is important to note that with this approach there are two aspects to the performance of the pivot operation: 1) the numerical update, in which changes to the values of the primal and dual variables are effected, and 2) the data structure update, in which the tree representation must be changed when the basis changes. As might be expected, there is an important trade-off between those two updates, since more complex tree representations reduce the effort required for numerical updates, but are themselves more difficult to store and update as the tree changes. Indeed, a major share of the research into computational aspects of network optimization has been devoted to the determination of data structures that reduce the total time for the two portions of the updating process. While there is no current agreement on optimal data structures, there are some key ingredients for superior performance , and we will provide a brief overview of these elements .

It is convenient for both notational and algorithmic purposes to assume that the nonbasics x_N have value 0. This is clearly the case in the initial iteration with the all-artificial basis, and at any iteration in which a nonbasic x_i reaches upper bound, a "reflection" of x_i is performed by replacing x_i by $x_i' = u_i - x_i$. Computationally, this requires only the reversal of the direction of the arc corresponding to x_i, a change in the sign of c_i, and a record that this reflection has been made. (Since the RHS b is not actually used in the algorithm after the initial basis, it need not be modified.) Under the assumption that $x_N = 0$, a sufficient optimality condition is that

$$c_{ij} - \pi_i + \pi_j \geq 0 \qquad (OPT)$$

for each nonbasic arc (i, j). Thus, the so-called "*pricing out*" operation for an arc requires only the subtraction and addition from the cost coefficient of the $\pi's$ for the nodes at the ends of the arc. (Just as with general linear programs, a variety of sophisticated strategies have been used wtih the goal of obtaining "good" entering nonbasics without having to price out a large number of nonbasics at each iteration. While so-called "candidate

list" strategies are effective, a simple "random" strategy has also been used successfully [Grigoriadis 82].)

Let us consider the operations that must be performed to accomplish a pivot in the network simplex method, given a nonbasic arc entering the current basis at lower bound. We will assume that the pivot is *non-degenerate*, i.e., that a flow increase is possible in the entering arc. (Degeneracy is actually fairly common in network optimization, particularly for special classes of problems such as the assignment problem, and special techniques have been devised [Barr, et al 77] to reduce the number of degenerate pivots. However, a discussion of these devices is beyond the scope of this paper, and for the general single-commodity network problem, the RNET code [Grigoriadis 82] to be discussed below has been found to be one of the most efficient of currently available network codes, yet contains no sophisticated anti-degeneracy features.)

The Network Simplex Pivot

(We assume that a BFS and a corresponding set of dual variables are available.)

1. If the optimality conditions (OPT) are satisfied, then quit with an optimal solution; else select an entering nonbasic arc (r, s) .

2. Increase the flow on arc (r, s) *until a) it reaches its bound or b) until one of the basic arcs is driven to a bound, whichever comes first.*

3. In case 2a), the primal variables are changed by appropriate amounts and we return to step 1, leaving the dual variables and tree unchanged.

4. In case 2b), the primal and dual variables and the tree representation are updated, then return to step 1 .

In order to determine how the flows in the basic arcs change as the flow in the entering arc increases, we take advantage of the fact that the addition of an entering arc (r, s) to the basis tree creates a graph with a *unique cycle* that consists of the path in the tree from r to s plus the new arc (r, s) (see Figure 1, where the arc orientation is shown only for cycle arcs). Since a flow change on the entering arc must yield a unique corresponding change in the basic flows, we will fully account for all changes in the basics by showing that appropriate flow changes on (only) the cycle arcs preserve feasibility. The orientation of the arcs of the cycle is determined by the orientation of the entering nonbasic arc (r, s), so that the arcs with positive orientation (i.e., in the same direction as (r, s)) are $(r, s), (p, q)$ and (t, p), whereas the arcs with negative orientation (i.e., opposite to (r, s)) are (r, q) and (t, s)). In both the example and the general case it is easily verified that feasibility (w.r.t. the divergence equations) is maintained if the flows on all positively oriented arcs are

increased by a given amount and the flows on all negatively oriented arcs are decreased by the same amount. Unit flow changes on the cycle of the example are illustrated by Figure 2. Thus, for each unit change in $x_{r,s}$, there is a unit change (of appropriate sign) in each cycle arc. It is therefore relatively easy to determine which of the cycle arcs will be the first to hit a bound as they increase or decrease in flow depending on their orientation, since the simplex ratio test thus reduces to a comparison of distances to bounds. If this arc is (r, s), then the solution update is fairly trivial since dual variable values remain the same, and only the flows on the cycle arcs need be modified. Let us assume then, that a basic arc is the first to reach a bound, so that a full update is required. In the example, suppose that (p, q) hits its bound first. Again the primal solution update is obtained by modifying the cycle flows, but now the duals and tree representation must also be updated. From a *visual* viewpoint, it is easy to see how to change the tree representation in the example since it is clear that what is required is the severing and removal of arc (p, q) and the "rehanging" of the subtree that lies below this arc from node s via the new arc (r, s) (see Figure 3). Computationally, however, tree updates require a data structure perspective, and this process has some subtleties that we will consider below. Finally, to accomplish an update of the dual variables, note from (PI) that the values of the dual variables may be computed by moving down the tree from node $m + 1$, where $\pi_{m+1} = 0$. Each tree arc corresponds to a dual equation, and following the paths down from $m + 1$ we can obtain values of successive duals by noting that only two duals are present in each equation and the one higher in the tree has already been determined. Since the only paths from node $m + 1$ that are modified by the basis change are those to the nodes in the subtree below the leaving arc, the only dual variables that change are those for the nodes of that subtree. Moreover, in the example, if Δ is the change in π_r, then all of the other dual variables of the rehung subtree also change by exactly Δ since the *difference* between the dual variables on opposite nodes of any subtree arc is unchanged (i.e., it remains the cost associated with the arc). This property holds for the dual update in general, keeping in mind that the rehung subtree will contain one of the nodes of the entering arc.

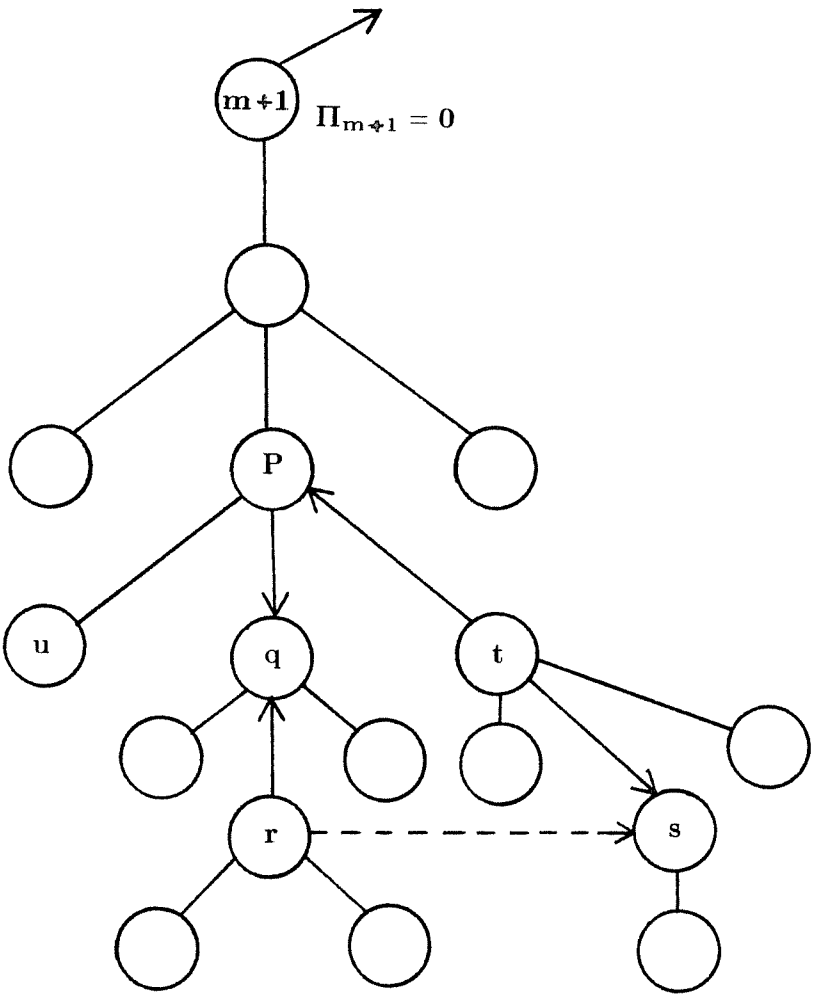

Figure 1. Tree augmented by entering arc (r,s)

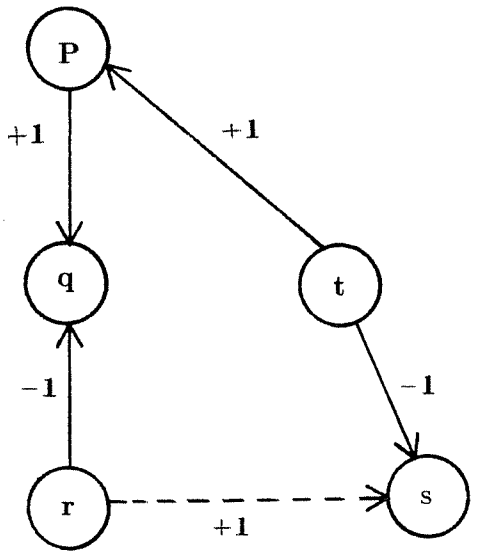

Figure 2. Flow changes on cycle

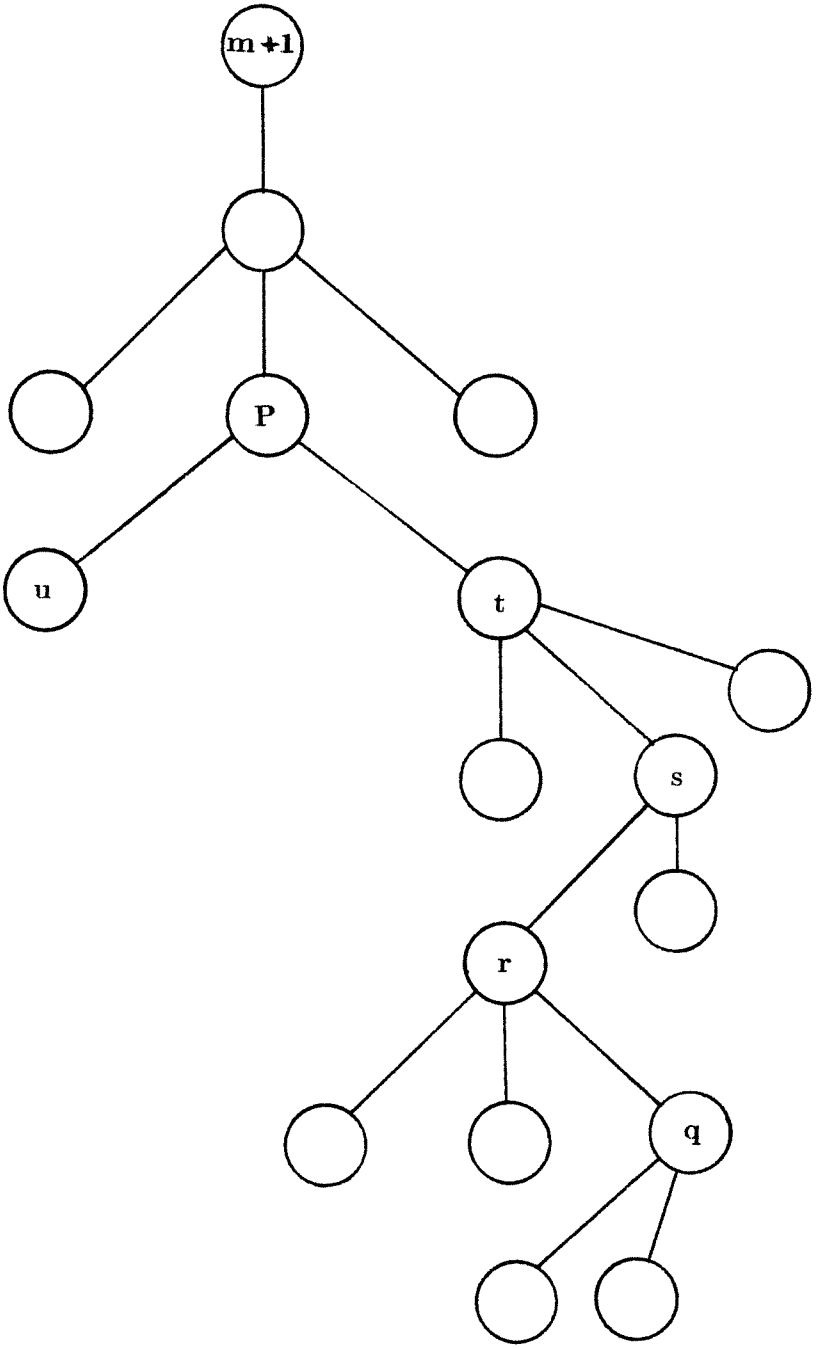

Figure 3. New basis tree

We will now define some concepts important for the description and update of basis trees. The unique path from a node i to the root node $m+1$ is called the *backpath* from i. The nodes on the backpath above i are the *predecessors* of i, and any node having i as predecessor is a *successor* of i. With this terminology, the determination of the cycle corresponding to an entering nonbasic (r, s) corresponds to generating the predecessors of both r and s and determining the first common predecessor (in the Figure 1 example, node p) . Conversely, the dual variables requiring update correspond to one of the nodes of the entering arc (the one on the rehung subtree) and its successors in the new tree. In summary, the primal update may be performed using predecessors, while the dual update involves successors.

Although the immediate predecessor of each node (other than the root) is uniquely determined, a node generally does not have a unique immediate successor and there are a variety of data structures suitable for representing the full set of successors of a node. The successor representation most commonly used in network optimization is the *preorder traversal* (or, simply, preorder) in which nodes are linearly ordered with the root being indexed first and the indexing proceeding to the leftmost immediate successor until this is no longer possible, then backtracking until a node with an unindexed successor is reached so that the indexing may be continued. The *initial* tree corresponding to the all-artificial basis has the nice property that no work is required to compute its corresponding immediate predecessor function p or the preorder traversal function s, since for $i \neq m+1$, $p(i) = m+1$ and $s(i) = i+1$ (for reasons to be noted below, it is useful to have node $m+1$ considered to be the "successor" of the "last" node, which in this case is node m).

Among contemporary network simplex algorithms, usage of only p and s would be considered to be a "minimal" collection of data structures for tree representation. Besides p and s, most algorithms use at least one additional node-length array to describe the tree in order to speed up the tree representation update and the determination of the intersection node of the backpaths of the nodes of the entering arc. For example, the set of successors of a node i may be generated by iterating the preorder function starting at i, and simultaneously checking the depth of the generated nodes, where the *depth* $d(i)$ of node i is the number of arcs in the backpath, since once a node j with $d(j) \leq d(i)$ is produced, the full set of successors of i is the set of previously generated nodes. Depth may also be used in cycle determination, since backtracking may be started from the node on the entering arc with greater depth and is continued simultaneously on both backpaths once the other depth is reached. Other additional data structures that have been used in the network simplex method include number of successors and "depth" with respect to the preorder index. Of course, all of the tree representation arrays must also be updated

each time the basis changes, and the efficiency of a collection of data structures is also dependent on this factor. The inverse preorder function is used in RNET, for example, to reduce the cost of the preorder update. A good recent survey of some of the choices used in this area is given in [Kennington and Helgason 80].

There has been extensive computational testing of network flow codes on a standardized set of 35 medium to large problems produced by the NETGEN test problem generator described in [Klingman, Napier, and Stutz 74]. These problems have up to 1500 nodes and 5730 arcs. Direct comparison of codes with the same machine and compiler has been infrequent, but some idea of the relative performance of contemporary approaches is given by a survey in [Glover and Klingman 82], which concluded that the RNET code of [Grigoriadis 82] ran approximately as fast as the ARC-II code of [Barr, Glover, and Klingman 79], and that these two codes were faster than other publically availables codes. The reported total time using a CDC 6600 with the MNF Fortran compiler on the full set of problems was 102.85 seconds for RNET and 103.12 records for ARC-II. The longest solution time for RNET for an individual problem of this set was 8.11 seconds. Total storage for RNET is 7m + 4n, as compared to 7m + 3n for ARC-II. RNET uses the depth array and an array corresponding to the inverse of s in addition to predecessor and successor arrays. The study of [Glover and Klingman 82] also notes the preliminary testing of their new Code ARCNET, whose performance on the standard test problem set is slightly better than RNET.

Space does not permit a discussion of methods for important special classes of single-commodity network flow problems such as shortest path, maximum flow, transportation, and assignment problems. While it is possible to efficiently solve such problems using a general network flow code, still faster solutions can generally be obtained by further exploiting special structure. For example, a primal-dual method of F. Glover, R. Glover, and Klingman for the *assignment* problem was found to be nearly an order of magnitude faster [Glover and Klingman 82] than the network simplex method. Solution time for their method on a typical 400 node, 3000 arc assignment problem is 0.1 seconds on a dual Cyber 170/175. [Glover and Klingman 82] is also a good survey of recent research on special methods for shortest path and max flow problems.

3. Network-Related Linear Programs

There are a number of important classes of linear programs that are closely related to ordinary single-commodity networks. These include multicommodity flows, networks with additional general side constraints and variables (also known as embedded networks), and so-called generalized networks. In this section we provide a brief review of procedures for extending algorithms for ordinary networks to these more general problem classes.

Linear Multicommodity Network Flow

In a wide variety of applications several different commodities simultaneously flow through a network, and each commodity has its distinct set of supply, demand, and conservation constraints. Coupling between commodities occurs typically as a result of the sharing of arc capacities, and takes the form of "hard" constraints on the total flow of all commodities through individual arcs or "soft" constraints that impose nonlinear objective function penalties based on the total flow through arcs. In this section we will give a brief description of techniques for the case of linear coupling constraints, and then in following section consider generalizing this model to allow arbitrary additional rows and columns to be added to the basic network constraint format. Nonlinear networks are discussed in section 4.

We will assume the problem format

$$\begin{aligned} \min_{x} \quad & \sum_{k} \mathbf{c_k x_k} \\ s.t. \quad & \mathbf{A x_k} = \mathbf{b_k} \quad (k = 1, \ldots, K) \\ & \sum_{k} \mathbf{x_k} \leq \mathbf{b} \\ & \mathbf{0} \leq \mathbf{x_k} \leq \mathbf{u_k} \quad (k = 1, \ldots, K) \end{aligned} \tag{MUL}$$

where A is a node-arc incidence matrix and $\mathbf{x_k}$ represents the set of flows corresponding to commodity k. (The techniques to be considered below may be extended in straightforward fashion to allow different networks constraints for each commodity as well as more general coupling constraints between commodities.)

Three fundamental approaches have been studied for this problem. These are Dantzig-Wolfe or price-directive decomposition, resource-directive decomposition, and primal partitioning. Dantzig-Wolfe decomposition uses an iterative process in which 1) the individual commodity constraints $\mathbf{A x_k} = \mathbf{b_k}, \mathbf{0} \leq \mathbf{x_k} \leq \mathbf{u_k}$ are approximated by convex combinations of extreme points, 2) these approximations are substituted in the coupling constraints $\sum \mathbf{x_k} \leq \mathbf{b}$ to obtain an approximation (the "master" problem) that is then optimized,

and 3) new extreme points of the single commodity problems are generated (assuming the optimality conditions are not satisfied) by optimizing via network techniques separate single commodity problems whose objective functions are determined by the optimal dual variables of the master problem. (In essence, this is the standard Dantzig-Wolfe method, except that network methods are used to obtain efficient solutions of the subproblems).

In resource-directive decomposition, the arc capacities are allocated among the commodities, the resulting single commodity subproblems are optimized, and the optimal dual variables are then used to determine a new allocation of capacities (assuming an optimal solution has not been obtained). [Kennington and Helgason 80] discuss tangential approximation and subgradient techniques for implementing resource-directive methods.

In primal partitioning, we take advantage of the fact that a basis for the multicommodity problem must contain bases for the single commodity problems. For notational purposes it is convenient to assume that slack variables y are added to the coupling constraints. Specifically, if $\bar{B}$ is a basis for (MUL), it may be partitioned as

$$\bar{B} = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$$

where B is a collection of bases for the single commodity problems, and $[D\ E]$ is the portion of the basis corresponding to the coupling constraints. It may be verified that

$$\bar{B}^{-1} = \begin{bmatrix} B^{-1} + B^{-1}CQ^{-1}DB^{-1} & -B^{-1}CQ^{-1} \\ -Q^{-1}DB^{-1} & Q^{-1} \end{bmatrix}$$

where $Q = E - DB^{-1}C$.

Note that in the limiting case in which all of the slack variables y are in the basis (so that nondegeneracy would imply that none of the joint capacity constraints were active), $C = 0$ and $E = I$, and the expression for $\bar{B}^{-1}$ reduces to

$$\bar{B}^{-1} = \begin{bmatrix} B^{-1} & 0 \\ -DB^{-1} & I \end{bmatrix}$$

More generally, it may be observed that relative simplicity of $\bar{B}^{-1}$ depends on the number of slack variables y in the basis, which in turn determine the complexity of E and the number of 0 columns of C. Of course, it is not necessary to explicitly compute $\bar{B}^{-1}$; instead we take advantage of the tree structure of the sub-bases of B in the computation of the solutions of the primal and dual systems of equations associated with $\bar{B}$. Details of the procedure may be found in [Kennington and Helgason 80]. Recent computational

experience reported in [Kennington and Patty 84] indicates that the efficiency of primal partitioning is closely related to the number of active generalized upper bounding (GUB) rows in the optimal solution. In the applications considered in that paper, the number of active GUB rows was quite small (typically, fewer than 10), and primal partitioning was found to yield solutions about 6 times faster than linear programming approaches that did not take advantage of the network structure.

Networks with Side Constraints and Variables

From a theoretical standpoint, a straightforward generalization of multicommodity networks is the problem in which additional constraints and variables are added to standard network constraints. Algebraically, the structure of the bases for this class is similar to that of the multicommodity case, except that the B part of $\bar{B}$ is generally only a basis of a submatrix of the network portion. Computational experience with primal partitioning for this class has been rather limited, but again it is the case that efficiency relative to the plain simplex method is highly dependent on the significance in the solution of the non-network portion of the problem. In fact, this problem class reaches to (and sometimes past) the limit of generalization within which it is still computationally worthwhile to exploit network substructure. Results reported in [Glover and Klingman 81] showed primal partitioning to be about 4 times faster than LP on some small problems and about 70 times faster on a problem with 1000 nodes, 5000 arcs, and 1 non-network row and column.

Generalized Networks

There are a great variety of additional applications that may be modelled by generalizing the network model to allow two arbitrary non-zero entries per column of the constraint matrix. (By using scaling, one of these may be assumed to be ± 1.) This class of problems is referred to as generalized networks or *"networks with gains and losses"*, since it can be thought of as allowing flows to increase or decrease as they traverse arcs. Physically, gains occur in such applications as financial models, in which interest is added as money flows from one time period to another, and losses occur in such phenomena as the transport of electricity and gas.

It may be shown that a basis for a network representation of the problem consists of a collection of one or more subgraphs, each of which consists of a rooted tree or a subgraph with exactly one cycle (such subgraphs are sometimes called one-trees). (Unlike ordinary networks, the presence of a cycle in a generalized network does not imply nonuniqueness of the solution.) By exploiting this basis structure using data structures that are extensions of those used for ordinary networks, it is possible to solve generalized networks about 50

times faster than by use of standard LP codes. In essence, cycles are dealt with as the roots of their components, and expressions are easily developed for the computation of primal and dual variables on the cycles. Details of the extension of the network simplex method to generalized networks may be found in [Kennington and Helgason 80]. A recent paper [Grigoriadis and Hsu 83] discusses some network analogs of partial pivoting that are useful for numerical stability during the pivot process, yet have very little computational overhead.

4. Nonlinear Networks

Nonlinear network optimization problems are of the form:

$$\begin{aligned} &\min f(\mathbf{x}) \\ &s.t.\ \mathbf{Ax} = \mathbf{b} \\ &\quad \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \qquad \text{(NLP)}$$

where f is a nonlinear function on the n-dimensional interval [**l**,**u**], and **Ax=b** represents a system of network constraints. Significant applications that give rise to this format include computer network design [Cantor and Gerla 74], [Gavish and Hantler 82], [Magnanti and Wong 84], traffic assignment [Bertsekas and Gafni 82], [Dafermos 80], [Dantzig, et al, 79], [Feijoo and Meyer 84a], [Lawphongpanich and Hearn 83], [Pang and Yu 82], hydroelectric power systems [Hanscom, et al, 80], [Rosenthal 81], [Escudero 84], telecommunications networks [McCallum 76], [Meyer 81], [Monma and Segal 82], and water supply systems [Beck, et al, 83], [Meyer 82,83]. These types of problems often have thousands of variables and are thus among the largest nonlinear programming problems that have ever been solved.

In the case in which f is *convex* and *separable*, i.e., of the form $f(\mathbf{x}) = \sum_{i=1}^{n} f_i(x_i)$, piecewise-linear approximation of the objective function has the nice property that the resulting approximating problems may be solved by linear network codes. [Kamesam and Meyer 82] showed that piecewise linear approximation of the f_i with iterative refinement of the grid yields a sequence of iterates converging to the optimal solution, and that Lipshitz continuity of the derivatives of the f_i implied a linear convergence rate of α^2 for the objective function, where α is the factor by which the grid size is multiplied at each iteration. Thus, for example, with $\alpha = 1/4$ (a typical value used in computation) the linear rate of convergence of the objective function will be $1/16$, in essence guaranteeing an additional figure of accurary with each iteration in the neighborhood of the optimal solution. The

key issue in this regard is the speed with which the successive linear programs may be generated and solved. By using an implicit grid approach in which linear approximating segments are generated only as needed and at most two segments are stored for any variable at any given time, the efficiency of the problem generation phase is ensured. Major areas in which further research appears to be needed in the solution phase are the specification of the initial basis and the pricing out strategy. [Dembo and Klincewicz 81] and [Klincewicz 83] discuss the use of quadratic approximation for separable nonlinear networks.

In the case that the objective function f is convex and *non-separable* , it was commonly held that piecewise-linear approximation was impractical because of the "curse of dimensionality", i.e., the need to deal with k^n grid points in order to have a uniform grid of k points for each variable . The theory and computational results of [Feijoo and Meyer 84a] showed that a convergent and efficient piecewise-linear approximation method could be achieved in the non-separable case by restricting the computation of approximating points to a small cluster situated on a set of axes translated to the current iterate. In essence, this is equivalent to using a locally-determined separable approximation at each major iteration, so that the techniques for the separable case are applicable. Moreover , this type of approximation applied to nonlinear networks has the advantage of preserving the network structure of the constraints.

Traffic equilibrium problems are an important class of nonlinear multi-commodity problems, i.e., they involve many different types of "commodities" flowing through the network, where depending on the application, a "commodity" will be associated with the traffic flowing out of a "source" node or between a designated origin-destination pair. Such problems arise in urban and regional transportation planning as well as computer network traffic models. Enormous problem sizes result from the fact that the number of variables and constraints is determined by the number of links and nodes multiplied by the number of commodities. These same features, however, also make these problems ideal candidates for solution via parallel algorithms, about which more will be said below. There are two fundamental types of traffic equilibrium problems: symmetric problems in which the congestion on a given link is determined by the total flow summed over all commodities in that link only, giving rise to a symmetric Jacobian matrix in the corresponding variational problem, and asymmetric problems in which the congestion on a link depends on the total flows in several links. It is well known [Steenbrink 74] that the former problem is equivalent to a convex optimization problem under relatively weak assumptions, whereas the latter gives rise to a variational inequality that has a more complex optimization formulation involving a non-differentiable objective function. In this paper we will concentrate on the former problem, although some of the decomposition ideas considered apply equally

well to both problem classes and indeed to large-scale optimization problems that have non-network constraints. This problem is of the form (NLP), where the objective function represents total congestion over all of the links of the network, and the constraints represent the supply, demand, and conservation constraints for each of the individual commodities flowing through the network. (Alternatively, it is possible to transform the problem into a separable problem by including constraints that sum the flows on each arc, but this destroys the network structure of the constraints.) the The number of commodities may be quite large because, depending on the formulation, there can be a commodity corresponding to each node or to each origin-destination pair. However, in the urban traffic assignment problem it is typically the case that the only coupling between different commodities occurs in the objective function.(For other problems in which there are explicit joint capacity constraints, Lagrangian and penalty approaches can frequently be exploited to move these constraints into the objective function.) Because of this coupling property, the approximation of the objective function by a separable function leads to a decomposition of the problem into separate optimization problems, one for each commodity. These single commodity problems may then be optimized in parallel (many other algorithms for the symmetric and asymmetric problems also contain a phase in which the objective function is replaced by an approximation that allows decomposition to be used).

For simplicity we will assume that a feasible solution $\mathbf{x}^i$ (one satisfying all of the constraints) is available at the start of the ith iteration of the problem (NLP). (At the initial iteration an arbitrary starting point within $[\mathbf{l}, \mathbf{u}]$ plays the same role.) To simplify notation below,consider the *shifted function* $f_{\mathbf{x}^i}(\mathbf{x}) := f(\mathbf{x}) - f(\mathbf{x}^i)$; it is clear that minimizing $f_{\mathbf{x}^i}$ is equivalent to minimizing f. A piecewise-linear separable approximation $\tilde{f}$ "centered" at $\mathbf{x}^i$ is then constructed as follows:

Consider the function

$$h : \mathbf{R}^n \longrightarrow \mathbf{R}$$

$$h(\mathbf{y}) = f_{\mathbf{x}^i}(\mathbf{y} + \mathbf{x}^i)$$

which is a translation of $f_{\mathbf{x}^i}$ to the origin, and let h_j be the restriction of h to the axis y_j, that is $h_j(y_j) = h(y_j \mathbf{e}^j)$ where $\mathbf{e}^j$ is the jth canonical unit vector. For a given *grid size* λ_j^i, generate a piecewise-linear approximation $\tilde{h}_j$ to h_j, with breakpoints at $\ldots, -2\lambda_j^i, -\lambda_j^i, 0, \lambda_j^i, 2\lambda_j^i, \ldots$ Defining $\tilde{f}_j(x_j) := \tilde{h}_j(x_j - x_j^i)$, the resulting piecewise-linear approximation to $f_{\mathbf{x}^i}$ is:

$$\tilde{f}(\mathbf{x}) = \sum_{j=1}^{n} \tilde{f}_j(x_j)$$

(Note that the computation of the $\tilde{f}_j$'s may be carried out *in parallel.*) The original

objective function is then replaced by the approximating function, and the resulting approximating problem is solved. This problem has the form:

$$\begin{aligned} &\min \tilde{f}(\mathbf{x}) \\ &s.t.\ \mathbf{A}\mathbf{x} = \mathbf{b} \\ &\quad \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \qquad (\text{PLP}(i))$$

(Note that the function $\tilde{f}$ depends on the base point $\mathbf{x}^i$ and consequently varies from iteration to iteration.)

The problem (PLP(i)) has two key features: 1) because the objective function is now separable, the problem may be decomposed into K separate optimization problems, where K is the number of commodities in the original problem; and 2) because of the convexity and piecewise-linearity of their objective functions, each of these new problems is equivalent to a linear network optimization problem.

Specifically, assume that $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_K)$, where $\mathbf{x}_k$ is the vector of variables corresponding to commodity k, and S_k is the correponding set of indices for those variables. Since there is no coupling between commodities in the constraints, the problem (PLP(i)) may be solved by solving *in parallel* (via very fast linear network codes) the set of problems:

$$\begin{aligned} &\min \sum_{j \in S_k} \tilde{f}_j(x_j) \\ &s.t.\ \mathbf{A}_k\mathbf{x}_k = \mathbf{b}_k \qquad\qquad k = 1, \ldots, K \\ &\quad \mathbf{l}_k \leq \mathbf{x}_k \leq \mathbf{u}_k \end{aligned}$$

where $\mathbf{A}_k$, $\mathbf{b}_k$, $\mathbf{l}_k$ and $\mathbf{u}_k$ are for commodity k the corresponding components of $\mathbf{A}, \mathbf{b}, \mathbf{l}$ and $\mathbf{u}$. (The effect of this approach is therefore similar to Dantzig-Wolfe decomposition, in that it results in the decoupling of the subproblems; the difference is that the coupling is removed here from the objective function rather than the constraints. Clearly, the data communications requirements for most decomposition approaches will be similar.)

Given the set of solutions of these problems $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_K$, if we let $\tilde{\mathbf{x}}^{i+1} = (\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_K)$, the next iterate $\mathbf{x}^{i+1}$ is obtained by solving the line search problem:

$$\begin{aligned} \min_{\theta}\ & f(\mathbf{x}^i + \theta(\tilde{\mathbf{x}}^{i+1} - \mathbf{x}^i)) \\ s.t.\ & \mathbf{l} \leq \mathbf{x}^i + \theta(\tilde{\mathbf{x}}^{i+1} - \mathbf{x}^i) \leq \mathbf{u} \\ & \theta \geq 0 \end{aligned}$$

Once the next iterate has been obtained, the procedure is repeated with a reduced grid size of $\lambda^{j+1} = \alpha\lambda^j$, where $\alpha \in (0,1)$ ($\alpha = 0.25$ has worked well on our tests). It has been

shown that this procedure will yield a sequence of iterates whose objective function values converge to the optimal value of the original problem (NLP).

Our preliminary computational results [Feijoo and Meyer 84b] with this approach have been very promising. The method was implemented on the University of Wisconsin CRYSTAL multicomputer [DeWitt, et al 84], a set of VAX 11/750 computers (currently there are 20) with 2 megabytes of memory each, connected by a 10 megabit/sec Proteon ProNet token ring and accessed via VAX 11/780 "host" machines. In a typical run, a 12-commodity, 432-variable traffic assignment problem was solved on a 12-machine partition of CRYSTAL in 23.5 seconds versus 112 seconds for solution via the sequential version of the algorithm on a single machine. Six seconds of the CRYSTAL time were used for communication by the master and an average of 0.3 seconds for communication by each slave . Similar computational results were obtained for related problems. Thus, communication between machines represents a relatively small percentage of the total computing time (summed over all of the machines), and this behavior should scale up favorably to yield nearly linear speedup for larger problems.

An important computational issue yet to be studied is the degree of optimization to be carried out in the optimization of the linear subproblems. In order to obtain a descent direction, these problems do not have to be fully solved, and it appears likely that, after a certain stage in the optimization, further pivots do not significantly improve the quality of the search direction. Research on an appropriate stopping rule for the individual subproblems should lead to significant efficiency gains, and also offers the possibility of improving the level of synchronization when problems are solved in parallel. Further study of this approach is needed in order to identify the convergence rate, which should be dependent on the relative magnitude of the off-diagonal terms of the Hessian matrix, and to estimate this rate in the case of problem classes such as traffic assignment optimization.

In the case of Crystal , in which the number of processors will be approximately 40, there will be a shortage of processors in the case of large traffic assignment problems, which may contain hundreds or thousands of commodities, each of which could be optimized at a particular iteration on a single processor. It will thus be appropriate to investigate the relative efficiencies of solving all of the single commodity problems at a given iteration by distributing them in fixed groups among the available processors versus having a dynamic allocation mechanism that assigns single commodity problems to newly idle processors versus solving only a proper subset of the single commodity problems and then doing the line search restricted to the corresponding subset of variables while the processors begin work on another subset of the single commodity subproblems. In a sense, this problem class is ideal for experiments in parallel computing since it involves large-scale problems

and allows the testing of a wide variety of strategies for mapping processes to processors.

Other algorithms for convex nonseparable nonlinear networks include the Frank-Wolfe method [Dantzig, et al 79], simplicial decomposition ([Lawphongpanich and Hearn 83] and [Pang and Yu 82]), projection methods [Bersekas and Gafni 82], and the reduced gradient method [Beck, et al 83]. All of these techniques exploit in some manner the underlying network structure.

References

R. S. Barr and J. S. Turner, "Microdata file merging through large-scale network technology", *Mathematical Programming Study 15* , 1-22 ,1981.

R. Barr, F. Glover and D. Klingman, "A new alternating basis algorithm for semi-assignment networks", in *Proceedings of the Bicentennial Conference on Mathematical Programming*, NBS, Gaithersburg, Maryland, USA ,1977.

R. Barr, F. Glover and D. Klingman, "Enhancements of spanning tree labelling procedures for network optimization", *INFOR*, 17, 16-34, 1979.

P. Beck, L. Lasdon and M. Engquist, "A reduced gradient algorithm for nonlinear network problems", *ACM Trans. on Math. Software*, 9, 57-70 ,1983..

D. P. Bertsekas and E. M. Gafni, "Projection methods for minimum cost network flow problems", *Mathematical Programming Study 17,* 1-22, 1981.

D. P. Bertsekas and P. Tseng, "Relaxation methods for minimum cost network flow problems", MIT Laboratory for Information and Decision Systems Report LIDS-P-1339 ,1983.

R.E.Bixby,"Hidden structure in linear programs", in *Computer- Assisted Analysis and Model Simplification* (H.J. Greenberg and J.S. Maybee , eds.), Academic Press, 1981.

D. G. Cantor and M. Gerla: "Optimal routing in packet switched computer networks", *IEEE Transactions on Computing* C-23, 1062-1068, 1974.

S. Dafermos,"Traffic equilibrium and variational inequalities", *Transportation Science*, 14, 42-54 ,1980.

G. Dantzig, R. Harvey, Z. Lansdowne, D. Robinson, and S. Maier: "Formulating and solving the network design problem by decomposition", *Transportation Research* 13B, 5-17, 1979.

R. Dembo and J. Klincewicz, "A scaled reduced gradient algorithm for network flow problems with convex separable costs", *Math. Prog. Study* 15, 125-147, 1981.

D. DeWitt, R. Finkel and M. Solomon, "The CRYSTAL multicomputer: design and implementation experience", Technical Report 553, Computer Sciences Department, The University of Wisconsin-Madison, September, 1984.

L.F.Escudero,"Using independent superbasic sets on nonlinear replicated networks with applications", paper presented at the NATO ASI on Computational Mathematical Programming , 1984.

B. Feijoo and R. R. Meyer, "Piecewise-linear approximation methods for nonseparable convex optimization", University of Wisconsin - Madison Computer Sciences Department Technical Report 521, 1984a.

B. Feijoo and R. R. Meyer, "Optimization on the CRYSTAL multicomputer", University of Wisconsin-Madison Computer Sciences Department Tech Report 562, 1984b.

L.Ford and D.Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, N.J., 1962.

B. Gavish and S. L. Hantler: "An algorithm for optimal route selection in SNA networks", Research Report RC 9549, IBM T. J. Watson Research Center, Yorktown Heights, N. Y., August, 1982.

F.Glover, D. Karney, and D. Klingman, "Implementation and computational comparisons of primal, dual, and primal-dual computer codes for minimum cost network flow problems," *Networks*, 4, 191-212, 1974.

F. Glover and D. Klingman, "The simplex SON algorithm for LP/embedded network problems", *Math. Prog. Study* 15, 148-176 ,1981.

F. Glover and D. Klingman, "Recent developments in computer implementation technology for network flow algorithms", The University of Texas at Austin, Center for Cybernetic Studies Report CCS 377, 1982.

M. D. Grigoriadis, "Minimum-cost network flows, part I: an implementation of the network simplex method", Rutgers University Laboratory for Computer Science Research Report LCSR-TR-37 ,1982.

M. D. Grigoriadis and T. Hsu, "Numerical methods for basic solutions of generalized flow networks", Rutger University Laboratory for Computer Science Research Report LCSR-TR-43 ,1983.

M. A. Hanscom, L. Lafond, L. Lasdon and G. Pronovost, "Modeling and resolution of the medium term energy planning problem for a large hydro-electric system", *Management Science* 26, 659-668, 1980.

F.Hitchcock, "The distribution of a product from to several sources to numerous localities", *Journal of Mathematics and Physics*, 20, 224-230 ,1941.

E.L. Johnson,"Networks and basic solutions", *Operations Research*, 14, 619-623 ,1966.

P.V.Kamesam and R.R.Meyer, "Multipoint methods for nonlinear networks", to appear in *Mathematical Programming Studies 22* ,1984.

L.Kantorovich, *Mathematical Methods in the Organization and Planning of Production*, Publication House of the Leningrad State University, 1939. Translated in *Management Science*, 6, 366-422, 1958.

J. L. Kennington and R. V. Helgason, *Algorithms for Network Programming*, John Wiley , 1980.

J. L. Kennington and B. W. Patty, "A computational comparison of specialized versus general codes for multi-commodity network problems", *COAL Newsletter* , 8-14, 1984.

J.G. Klincewicz, "A Newton method for convex separable network flow problems", *Networks*, 13, 427-442 ,1983.

D. Klingman, A. Napier and J. Stutz, "NETGEN - A program for generating large-scale (un)capacitated assignment, transportation, and minimum cost flow network problems", *Management Science*, 20, 814-821 ,1974.

T.C. Koopmans, "Optimum utilization of the transportation system", *Econometrica*, 17, 3-4 ,1949.

S. Lawphongpanich and D. W. Hearn. "Restricted simplicial decomposition with application to the traffic assignment problem", Research Report 83-8, Department of Industrial and Systems Engineering, University of Florida. Gainesville, Fl., September, 1983.

T. L. Magnanti and R. T. Wong: "Network design and transportation planning: models and algorithms", *Transportation Science* 18, 1-55, 1984.

C. J. McCallum, "A generalized upper bounding approach to communications network planning problems", *Transportation Science*, 18, 1-55, 1984.

R.R. Meyer,"A Theoretical and computational comparison of 'equivalent' mixed-integer formulations", *Naval Research Logistics Quarterly*, 28, 115-131 ,1981.

R.R. Meyer, "Recursive piecewise-linear approximation methods for nonlinear networks", in *Evaluating Mathematical Programming Techniques*, Springer-Verlag,1982.

R.R.Meyer, "Computational aspects of two-segment separable programming", *Mathematical Programming*, 26, 21-39 ,1983.

C. L. Monma and M. Segal, "A primal algorithm for finding minimum-cost flows in capacitated networks with applications", *The Bell System Technical Journal* 61, 949-968, 1982.

J. S. Pang and C. S. Yu, "Linearized simplicial decomposition methods for computing traffic equilibria on networks", Technical Report, University of Texas at Dallas, Richardson, Texas, 1982.

R.T. Rockafellar, *Network Flows and Monotropic Programming*, Wiley, 1984.

R. E. Rosenthal, "A nonlinear network flow algorithm for maximization of benefits in a hydroelectric power system", *Operations Research* 29, 763-786, 1981.

P. A. Steenbrink, *Optimization of Transport Networks,* Wiley, London, 1974.