12
NW

CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712

Research Report CCS 306

NETWORK VERSUS LINEAR PROGRAMMING ALGORITHMS AND IMPLEMENTATIONS

by

Fred Glover*

John Hultz**

Darwin Klingman***

September 1977

* Professor of Management Science, University of Colorado, Boulder, CO 80302

** Senior Systems Analyst, Analysis, Research, and Computation, Inc., P.O. Box 4067, Austin, TX 78765

*** Professor of Operations Research and Computer Sciences and Director of Computer Science Research Center for Cybernetic Studies, University of Texas, BEB 608, Austin, TX 78712.

This research was partly supported by Project No. NR047-021, ONR Contracts N00014-75-C-0569 and N00014-75-C-0616 with the Center for Cybernetic Studies, The University of Texas. Reproduction in whole or in part is permitted for any purpose of the United States Government.

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Business-Economics Building, 203E
The University of Texas
Austin, Texas 78712
(512) 471-1821

ABSTRACT

Over the years modelers and practitioners have become so adept at designing large scale linear programming problems that in some cases the complexity is staggering. However, such problems place a tremendous burden on current standard linear programming systems and computer resources. This has led to a demand for alternative design and solution procedures.

Noting that the vast majority of linear programming problems contain at least some embedded network structure, researchers in the network area have made many advances in the past few years. In fact, these advances have led to the important new area of *network computer implementation technology*.

This paper provides insights as to why network techniques are having such a large impact on modelers and why network solution procedures are being selected over standard linear programming systems for use in many real world applications. As dramatic proof of the savings offered by network solution procedures, various network computer codes are compared with the standard linear programming system APEX-III. Results indicate that the network techniques are 200 times faster on highly structured problems and as much as 25 times faster on more complex embedded network problems.

1. INTRODUCTION

The growth of the computer industry has had a profound influence on many areas, perhaps affecting none more dramatically than the area of mathematical optimization.

The techniques for building, solving, refining, and analyzing mathematical optimization models have undergone a steady evolutionary development as computer hardware has changed. This evolution has led to the important new area of *network computer implementation technology.*

This technology has emerged from recent research on new solution algorithms and implementation techniques for solving network problems [1, 2, 3, 6, 9, 10, 13, 15, 17]. The fruits of this research have dramatically reduced the cost of solving linear and mixed integer network type problems. This cost reduction, significantly, has been entirely above and beyond any reductions afforded by changes in computer hardware or compilers. For example, the cost of solving network problems with 2400 equations and 500,000 arcs on an IBM 360/65 has been reduced from a conservative estimate of $10,000 in 1968 to $300 in 1976 by these advances. In addition these advances have stimulated the development of new modeling techniques for handling a multitude of problems that arise in applications of scheduling, routing, resource allocation, production, inventory management, facilities location, distribution planning, and other areas [8, 11, 12, 14].

Yet the full scope and significance of these developments remains to be fully appreciated by members of the mathematical and computer optimization community. For example, although the producers of large LP systems have always recognized that specialized computer codes could be better than standard codes they strongly believed that the specialized codes would be only 5 - 10 times faster at best. This belief stemmed in part from the fact that large sums of money and many hundreds of man years have been invested to produce top notch LP systems. In addition, LP has been an omnipresent and often dominant topic in management science textbooks, whereas networks are often barely touched upon.

In light of the recent advances in network computer implementation technology, we decided to compare the best network software against one of the best commercially available LP systems. The results have completely overturned the traditional view of the relative efficiencies of LP and network systems. In fact, some LP producers whom we have made aware of these results have come to recommend special purpose codes to users who have network type models. The large numbers of practical applications in which network models play a role makes it important that the mathematical optimization community at large become acquainted with the solution power of advanced network computer codes, the theory behind network solution algorithms, and the implications of latest developments for the future of optimization software.

To achieve these goals, this paper has been divided into three sections. The first presents a detailed comparison of current network codes with the state-of-the-art LP code APEX-III. The second section briefly compares and contrasts the implementation logic of network codes and LP codes to provide a more complete understanding of the results in the first section. The final section discusses how the network and LP procedures can be merged to provide a new generation of LP codes.

The discussion in each of these sections is devoted to sketching the concepts and innovations that represent the highlights of the field. A minutely detailed coverage of these topics would easily fill a book. Thus the intent of each section is to provide the reader with an executive summary rather than a microscopically focused account of these topics.

2. COMPARISON RESULTS

In this Section we examine the consequences of advances in network computer implementation technology. In particular we provide fuller insight into the practical significance of these methods by presenting the results of empirical tests against a leading example of the state-of-the-art in solution systems that does not incorporate the network technology. In particular, we report computational comparisons of the new network codes against APEX-III on a wide array of network problems of varying structures. We follow the customary classifications of network problem types (e.g., assignment transportation, transshipment, generalized network) of the mathematical programming literature [4, 5, 7, 18, 19].

The test results are not biased by variations in computer hardware: all problems were solved on the same machine. Further, an attempt was made to execute the codes when the computer had the same type of job load. Even with these safeguards, minor differences between two solution times should be statistically ignored and the focus should be on order of magnitude differences. For this reason, the times reported are for large problems so that timing variations become less significant.

Table I contains solution times on 15 network problems using APEX-III on a CDC CYBER-74. The first set of problems consists of assignment problems and the reported network solution times were obtained using the AP-AB code of [2]. The solution times indicate that the AP-AB code is roughly 200 times faster than APEX-III on assignment problems.

The network code times reported on the transportation and transshipment problems were obtained using the ARC-II code of [1]. Again the network solution times are substantially superior (on the order of 130 times faster than APEX-III). The fourth set of solution times are for generalized network (GN) problems. The network code times refer to the NETG code [8].

The relative superiority of network code times to APEX-III is smaller for generalized networks than for pure networks. The code NETG is on the order of 50 times faster than APEX-III on generalized networks; nevertheless, this superiority is dramatic, especially in terms of computer costs for solving such problems.

The final set of test results are for LP problems composed of a GN problem augmented by an arbitrary linear constraint. This problem-type is called the *singularly constrained generalized* network problem. The network code times refer to the NETSG [16] code and the results even for this class of constrained generalized networks are *more than an order of magnitude* faster than available with the advanced LP system.

In addition to improving solution speed, the network processing techniques have the noteworthy advantage of requiring less computer memory to solve a problem. This allows larger problems to be solved without resorting to external storage devices, which can incur significant cost increases due to lengthened computer run times. Further, the reduced memory requirements enable many computer-based

TABLE I

(times are in billing units)

PROBLEM TYPE	no. of equations	no. of variables	APEX III solution times	cost	Network Code solution times	cost
Assignment	400	1500	231.85	$ 41.73	1.16	$ .21
	400	2250	336.37	$ 60.55	1.34	$ .24
Transportation	200	1300	105.68	$ 19.02	.94	$ .17
	200	1500	124.53	$ 22.42	1.07	$ .19
	200	2000	164.94	$ 29.69	1.21	$ .22
Transshipment	400	1306	174.83	$ 31.47	1.51	$ .27
	1000	2900	833.63	$150.05	5.28	$ .95
Generalized networks	250	4000	453.02	$ 81.54	16.65	$ 3.00
	250	4000	742.61	$133.67	14.74	$ 2.65
	500	5000	1044.34*	$187.98	22.55	$ 4.06
	1000	6000	1633.64*	$294.06	50.22	$ 9.04
Singularly constrained generalized networks	200	2000	205.87	$ 37.06	16.10	$ 2.90
	200	1000	130.18	$ 23.43	11.38	$ 2.05
	500	4000	943.25	$169.79	32.72	$ 5.89
	1000	6000	1875.55*	$337.60	83.13	$14.96

* Not optimal after 10,000 iterations.

decision systems to be used in an interactive real-rime processing environment.

Yet another important advantage of the network codes is their portability. All of these codes are written in standard FORTRAN IV. Several beneficial consequences result. For example, this portability feature allows easy transfer of the network component of a computer-based decision system to a new computer. It also greatly facilitates imbedding the code as a subroutine within a larger system.

A final computational advantage, which will be discussed in the next section, is reduced round-off error. Taken together, the impressive array of advantages of the network solution codes makes it clear why their use in industry and government applications is rapidly increasing [11].

3. IMPLEMENTATION APPROACHES

Linear network problems, as previously pointed out, are special types of LP problems and can thus be solved using any standard LP solution technique. Improvements in LP inversion and reinversion processes, data compactification, and pivot strategies have materially improved the efficiency of primal simplex LP computer codes in recent years. In many cases, special structures such as GUB constraints (which are embodied within network problems) are detected by current LP codes. This information is then used to reduce storage requirements and to simplify operations. Despite these improvements, we saw in the preceding section that the special purpose network solution systems dwarf the LP systems in their efficiency. In this section we examine some of the reasons for this, and undertake to trace their more significant implications.

Undoubtedly, the primary reason for the superiority of network codes over LP codes is the fact that the latter, with the exception of the GUB feature, are based primarily on algebraic or arithmetic processing. That is, these codes maintain and update a basis inverse by manipulating numbers. Further, the representation of a variable to enter the basis is computed by matrix multiplication.

By contrast, the most efficient methods for solving network problems are based on replacing arithmetic operations with "logical" operations. More precisely, these solution procedures are based on viewing the problem in a graphical context (just as in the case of the network modeling ideas discussed previously). In particular, the network codes AP-AB, ARC-II, NETG, and NETSG (for assignment, transshipment, generalized, and singularly constrained generalized networks, respectively) all store the coefficient matrix and basis matrix as graphs using computer list structures.

The use of such computer list structures reduces both the amount of work needed to perform the algorithmic steps and the amount of computer memory required to store essential data. For example, network codes normally store only the cost coefficient, the upper bound, and the "to" node for each column of the coefficient matrix. (In the case of a GN problem, the multiplier is also stored.) In this way, problem data can often be resident in central memory even for large-scale problems.

To illustrate, a popular way of storing a network is to use linked-list structures. In this method, all of the arcs that begin at the same node are stored "together" under the name of their "from" node and each is represented (distinguished from the others) by recording only its "to" node, cost, and upper bound. A pointer is then kept for each node which indicates the block of computer memory locations for the arcs beginning at this node. The set of arcs emanating from node u is called the forward star of node u [6]. If the nodes are numbered sequentially from 1 to the number of nodes, and the arcs are stored consecutively in memory such that the arcs in the forward star of node i appear immediately after the arcs in the forward star of node i-1, then this method, called the forward star form, requires only |N| + 2|A| units of memory for pure uncapacitated problems, |N| + 3|A| for pure capacitated problems, and |N| + 4|A| for capacitated generalized network problems where |N| denotes the number of nodes and |A| the number of arcs. Figure 1 illustrates the storage of a pure network in forward star form. The number in the square attached to an arc of the network diagram is the cost. It is not necessary to store the NODE POINTER array when random access of forward stars is not required; in fact, the AP-AB and ARC-II codes do not use this array.

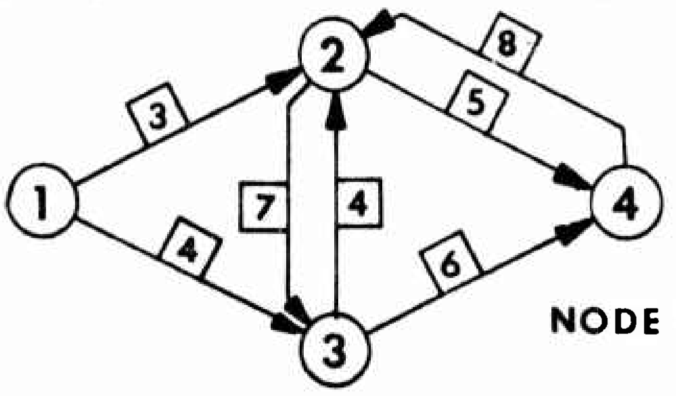
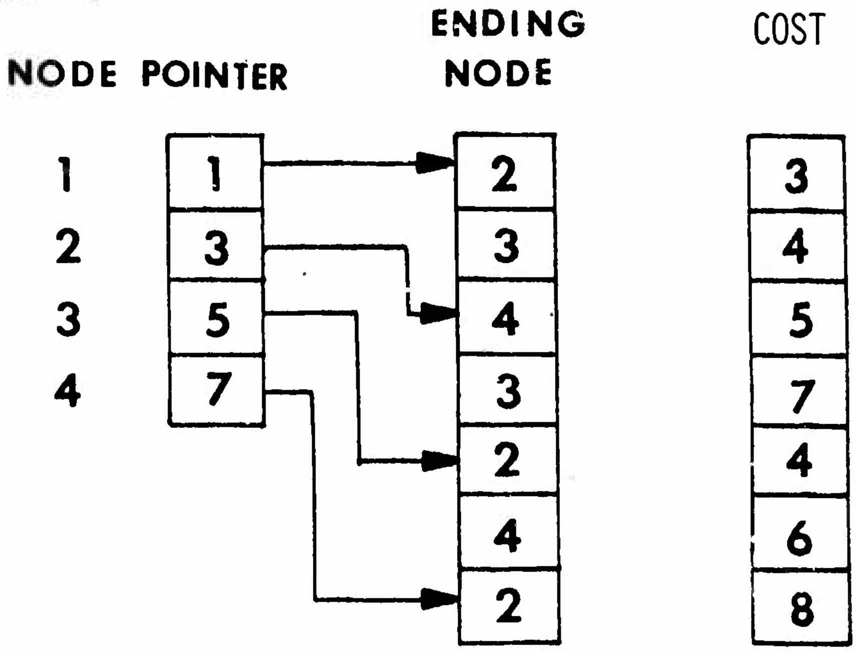


FIGURE 1 - FORWARD STAR FORM



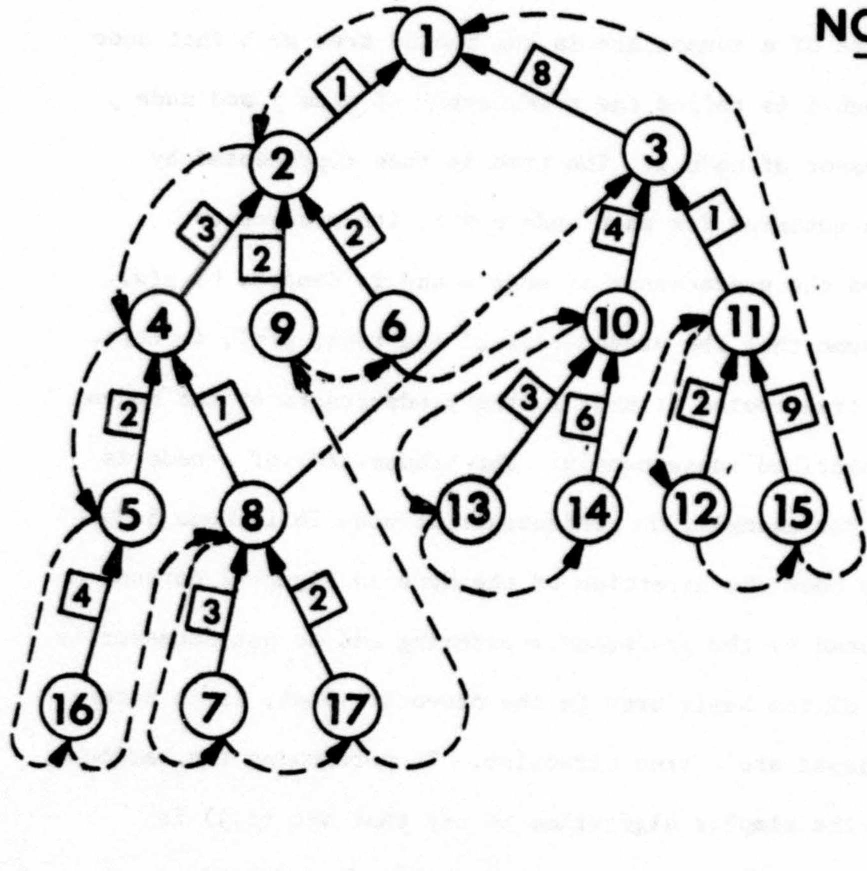
Network basis matrices, like network coefficient matrices, have a graph structure. Moreover, the graph structure of a basis matrix is very special, consisting either of disjoint trees or of trees coupled with one additional arc (called *quasi-trees* [8]). The basis of a pure network consists of a single tree and the basis of a generalized network problem consists of one or more trees and quasi-trees. These trees and quasi-trees can be stored and updated with remarkable efficiency by special linked list and labeling procedures that have been developed in the past few years [1, 2, 9, 13, 17].

A common way of representing a tree in a computer is to think of the root node r as the highest in the rooted tree with all other nodes hanging below it. If nodes i and j are endpoints of a common arc in the rooted tree such that node i is closest to the root, then i is called the *predecessor* of node j and node j is called an *immediate successor* of node i. The tree is then represented by keeping a pointer list which contains for each node $w \neq r$, its predecessor. This upward pointer is called the predecessor of node w and is denoted by p(w). For convenience, we will assume that the predecessor of the root, p(r), is zero.

Figure 2 illustrates a tree rooted at node 1, the predecessors of the nodes, and other functions to be described subsequently. The predecessor of a node is identified in the p array. For example, the predecessor of node 16 is node 5.

It is important to note that the direction of the arcs in Figure 2 correspond to the orientation induced by the predecessor ordering and do <u>not</u> necessarily correspond to the direction of the basic arcs in the network. Thus, it is necessary to keep track of each basic arc's true direction. In performing the various tree traversals required by the simplex algorithm, we say that arc (i,j) is traversed in the *forward direction* if it is traversed from node i to node j

Predecessor	**p(x)**
Node potential	**d(x)**
Thread	**t(x)**
Reverse thread	**rt(x)**
Depth	**dh(x)**
Cardinality	**c(x)**
Last node in subtree	**f(x)**



NODE	p	d	t	rt	dh	c	f
1	0	0	2	15	0	17	15
2	1	1	4	1	1	9	6
3	1	8	10	6	1	7	15
4	2	4	5	2	2	6	17
5	4	6	16	4	3	2	16
6	2	3	3	9	2	1	6
7	8	8	17	8	4	1	7
8	4	5	7	16	3	3	17
9	2	3	6	17	2	1	9
10	3	12	13	3	2	3	14
11	3	9	12	14	2	3	15
12	11	11	15	11	3	1	12
13	10	15	14	10	3	1	13
14	10	18	11	13	3	1	14
15	11	18	1	12	3	1	15
16	5	10	8	5	4	1	16
17	8	7	9	7	4	1	17

FIG. 2 - TREE LABELING TECHNIQUES

(i.e., in accordance with its network direction) and is traversed in the *reverse direction* otherwise.

Most primal simplex codes keep another list indexed by the node numbers associated with the tree. This list contains for each node v a label d(v), whose value is the current dual vari ble value of this node. Henceforth d(v) will be called the *node potential* of node v. The root r has a node potential of zero.

In Figure 2 the number in the square on each arc indicates the cost of the arc. The entries in the d array identify dual variable values which satisfy complementary slackness for the basic variables. Representing the cost on arc (i,j) by c_{ij} and supply (demand) as a negative (positive) quantity, this condition for each basic arc (i,j) will be $-d(i) + d(j) = c_{ij}$.

Figure 2 illustrates additional tree information expressed as node functions, which <u>can</u> be used in the computer implementation procedures.

The first of these functions, the *thread* function [1, 13], is denoted by t(x). This function is a downward pointer through the tree in the sense that it is used to locate all nodes lying below a given node. As illustrated in Figure 2 by the small dashed line, function t may be thought of as a connecting link (thread) which passes through each node exactly once in a top to bottom, left to right sequence, starting from the root node. For example, in Figure 2, t(1) = 2, t(2) = 4, t(4) = 5, t(5) = 16, t(16) = 8, etc.

Letting n denote the number of nodes in the tree, the function t satisfies the following inductive characteristics:

a) The set $\{r, t(r), t^2(r), \ldots, t^{n-1}(r)\}$ is precisely the set of nodes of the rooted tree, where by convention $t^2(r) = t(t(r))$, $t^3 = t(t^2(r))$, etc. The nodes r, t(r),..., $t^{k-1}(r)$ will be called the *antecedents* of node $t^k(r)$.

b) For each node i other than node $t^{n-1}(r)$, t(i) is one of the nodes such that p(t(i)) = i, if such nodes exist. Otherwise, let x denote the first node in the predecessor path of i to the root which has an immediate successor y and y is not an antecedent of node i. In this case, t(i) = y.

c) $t^n(r) = r$; that is, the "last node" of the tree threads back to the root node.

The *reverse thread* function, rt(x), is simply a pointer which points in the reverse order of the thread. That is, if t(x) = y, then rt(y) = x. Figure 2 also lists the reverse thread function values.

The *depth* function [17], dh(x), indicates the number of nodes in the predecessor path of node x to the root, not counting the root node itself. If one visualizes the nodes in the tree as arranged in levels--where the root is at level zero, and all nodes "one node away from" the root are at level one, etc.--then the depth function simply indicates the level of a node in the tree. (See Figure 2.)

The *cardinality* function, c(x), specifies the number of nodes contained in the subtree associated with node x in the tree. By the nodes in the subtree associated with node x, we mean the set of all nodes w such that the predecessor path from w to the root contains x. (See Figure 2.)

The *last node in a subtree* function, f(x), specifies that last node in the subtree of x that is encountered when traversing the nodes of this subtree "in thread order." More precisely, f(x) = y where y is the unique node in the subtree of x such that t(y) is not also a node in the subtree of x. (See Figure 2.)

Note that both the domain and the range of each of the above discrete functions consist of the set of nodes and thus are independent of the number of arcs. Since $|N|$ is the maximum number of nodes that could be in the tree, a one dimensional array of size $|N|$, called a *node length array*, is allocated to each function during computer implementation. The procedures for updating the values of the functions when the tree is reconfigured are discussed in [1, 9, 13, 17]. Current network codes use only a subset of these functions. For example, the code that we developed for the U.S. Treasury to solve problems with 50,000 nodes and 62 million arcs on a UNIVAC 1108 uses only the predecessor, node potential, and thread functions because of central memory limitations. The most efficient code, ARC-II [1], whose times are reported in the previous section, uses the predecessor, thread, last node, node potential, and cardinality functions. The previously fastest code PNET-I [10] uses the predecessor, thread, node potential, and depth functions. The selection of which functions to use in developing a code largely depends on the developers trade-off utility between central processor times and available memory.

The original problem data (compactly stored) and the basis matrix (stored via linked lists) are the only data elements kept by network codes. No basis inverse is stored. In LP systems, inverses generally require considerable amounts of storage and involve numerous (error-producing) arithmetic operations. In network systems, the specialized labeling rules (that are designed to exploit the linked list storage structures) operate on the basis graph in a manner that obviates the use of a basis inverse.

3.1 Fundamental Steps of Special Purpose Primal Simplex Codes for Networks

The fundamental pivot, or basic exchange, step of the simplex method will now be briefly reviewed in the graphical setting. Assume that a primal feasible starting basis (possibly containing artificial arcs) has been determined and is represented as a rooted tree. To evaluate the nonbasic arcs to determine whether any of them "price out" profitably, and therefore are candidates to enter the basis, it is necessary to determine node potentials (values for the dual variables) d(i) which satisfy complementary slackness, i.e., which yield $- d(i) + d(j) = c_{ij}$ for each basic arc. Because of redundancy in the defining equation of pure network problems, one node potential may be specified arbitrarily. The root node is customarily selected for this purpose and assigned a potential of zero, whereupon the potentials of the other nodes are immediately determined in a cascading fashion by moving down the tree via the thread function and identifying the value for each node from its predecessor using the equation $- d(i) + d(j) = c_{ij}$. This highly efficient labeling procedure for traversing the tree to initialize and update these node potential values is fully described in [1, 3, 17].

A feasible basic solution is optimal if the updated cost coefficient π_{ij} $(= - d(i) + d(j) = c_{ij})$ is nonpositive for each of the nonbasic arcs with flow equal to the lower bound (without loss of generality we will henceforth assume that all lower bounds are 0) and nonnegative for each of the nonbasic arcs with flow equal to the upper bound U_{ij}. If the solution is not optimal, then a nonbasic arc which violates the nonnegativity (or nonpositivity) requirement for π_{ij} is selected to enter the basis. If the flow on the selected arc is zero (or U_{ij}), then the simplex method attempts to increase (or decrease) this flow.

The arc to leave the basis is determined by: (1) finding the unique path in the basis tree, called the *basis equivalent path*, which connects the two nodes of the entering arc, and (2) isolating a blocking arc in this path whose flow goes to zero or its upper bound ahead of (or at least as soon as) any others as a result of increasing or decreasing the flow on the entering arc.

An increase (decrease) in the flow of the incoming arc causes a corresponding increase (decrease) in the flow of all basis equivalent path arcs traversed in the forward direction and a corresponding decrease (increase) of all basis equivalent path arcs traversed in the reverse direction. Thus, if a forward direction arc already has a 0 flow or a reverse direction arc already has a U_{ij} flow, then such an arc qualifies as a blocking arc and the incoming arc cannot be assigned a non-zero flow change. By using the predecessor and cardinality function in combination, one can quickly and efficiently simultaneously find the basis equivalent path and a blocking arc. To illustrate, suppose arc (8,3) in Figure 2 is the entering arc. Its basis equivalent path can be found by first examining the cardinality function of the nodes associated with the entering arc; namely, $c(8) = 3$ and $c\ (3) = 7$. Since $c(8) < c(3)$, find the predecessor of node 8. Compare $c(3)$ with $c(p(8)) = c(4) = 6$ and simultaneously determine the maximum flow change possible on the *basis link* between nodes 8 and 4. Continue this process of traversing the predecessor path of the current node with the smaller cardinality value until the "two" current nodes have the same cardinality value. If these "two" nodes are the same node, stop. The basis equivalent path has been traversed and a blocking arc found. Otherwise, continue from either of the two current nodes until the stopping criterion is satisfied.

It is important to note that the above procedure can be substantially improved computationally by modifying the algorithm to also stop when a zero flow change arc is encountered. Since computational testing has shown that 90% of the pivots made in solving network problems are degenerate, this augmented rule yields improved solution times [2, 8].

Once the entering and leaving arcs are known, the basis exchange is completed simply by updating the flow values on the basis equivalent path, determining new node potentials for the new basis tree, and updating the other functions used to represent the tree. Only a subset of the node potentials change during a pivot and these can be updated rather than being determined from scratch. (In a capacitated problem, the entering and leaving arcs can be the same arc. In this case, the updating is very simple.)

To update the node potentials, assume that the nonbasic arc (p,q) is to enter into the basis and the basic arc (r,s) is to leave the basis. If arc (r,s) is deleted from the basis (before adding arc (p,q)), two subtrees, K and $\overline{K}$, are formed, each containing one of the two nodes of the incoming arc (p,q). Let K denote the subtree which does not contain the root node of the full basis. The node potentials for the new basis may be obtained [1] by updating only those potentials of the nodes in K or $\overline{K}$, as follows. If p is in K, add $\delta = -d(p) + d(q) - c_{pq}$ to the potentials of each node in K. Otherwise, q is in K and $-\delta$ is used in the above operations. (Note that $\delta > 0$ if arc (p,q) is nonbasic with zero flow and $\delta < 0$ if arc (p,q) is nonbasic with U_{pq} flow.)

The above procedure can be efficiently performed by using the thread function to simultaneously locate K and update the node potentials. Further, since the node potentials in either subtree K or $\overline{K}$ can be updated, the above procedure can be substantially enhanced by using the cardinality to select the smaller tree and updating its node potentials [1].

The procedures for updating the functions to represent the tree directly depend on which of the functions are being used. [1] describes these procedures when the predecessor, thread, cardinality, and last node functions are used. [13] describes these procedures when the predecessor and thread are used. The discussion in [13] can be easily modified when the depth function is used in conjunction with the predecessor and thread functions.

If the latter two combinations of functions are used, the updating procedures basically involve rerooting subtree K at its associated node of the entering arc and attaching this rerooted subtree to $\overline{K}$ via the entering arc. As shown in [1], when the predecessor, thread, cardinality, and last node functions are used, either subtree can be efficiently rerooted and attached to the other subtree. This has several important computational advantages when integrating the operations of finding the basis equivalent path, the leaving variable, updating the node potentials, and updating the functions used to represent the tree.

3.2 Network Basis Traversal and Relations to Inverse Updating

The network labeling and list procedures provide a means for traversing (and modifying) relevant portions of the basis tree. The basis tree is given a "top-to-bottom" orientation by the predecessor and thread functions. Accordingly, there are two major types of traversal required to execute the basis exchange steps, called "upward traversal" and "downward traversal."

The first, upward traversal, is associated with operations normally requiring pre-multiplication by the inverse, such as determining the representation of a variable to enter the basis. This operation is performed by traversing the unique paths from selected nodes up to their "junction" point. Simultaneously, the equivalent triangular system of basis equations is solved, in effect, by back substitution. This approach has two advantages. First, original problem data are used to compute the representation; thus, roundoff error is minimized. Second, operations involving elements of the basis representation with weights of zero, or the execution of checks to identify such elements, are eliminated since the upward traversal of the basis graph isolates the elements that receive non-zero weights.

Downward traversal is analogous to post-multiplication by the inverse. This operation is used to calculate updated dual variable values (node potentials) associated with the problem nodes. At each iteration, new dual values must be computed for the nodes in a particular subtree associated with the arc leaving the basis. (The set of nodes whose potentials must be updated for GN problems has a slightly more complex characterization.) These nodes can all be encountered, and their new node potentials determined, by downward traversal using the thread function. For a pure network, it is only necessary to add a constant to the potential of each node in the subtree. In the case of a GN problem, each new value is computed by a simple operation which is equivalent to solving an equation in just one variable. An additional computational advantage of this procedure is that only the dual variables whose values change are examined at each iteration. Thus this operation again strictly eliminates checking or performing arithmetic operations on zero elements.

3.3 Integrated Operations

In the better network codes, the updating of the dual variables is integrated with the updating of the basis graph. The complete basis exchange step involves finding the representation of the arc to enter the basis, determining the arc to leave the basis, updating the node potentials, and restructuring (and partly reorienting) the basis graph by removing the arc leaving the basis and inserting the arc entering the basis. This last step, which is equivalent to updating the basis inverse, is accomplished simply by changing a few pointers in the list structures: no arithmetic operations are involved and, consequently, no round-off error introduced. The integration of this basis update with the dual update, by which both processes are carried out simultaneously, is made possible by the specialized labeling procedures [1].

These advantageous features of network systems have motivated researchers to develop extensions for solving LP problems with embedded networks. The NETSG code is an instance of one such extension. The next section briefly describes the fundamental ideas underlying such extensions and indicates their potential value.

4. EXPLOITING EMBEDDED NETWORK STRUCTURE

The procedures for exploiting embedded network structure are based upon partitioning the coefficient matrix of an LP problem into network and non-network components. Such a partitioning scheme arranges the rows so that each column has at most two non-zero entries in the first m rows out of a total of m + q rows. By reference to this partitioning, a basis compactification procedure is employed that induces an identical row partitioning on the basis matrix

B so that B can be expressed in the form

$$B = \begin{bmatrix} G_m & E_q \\ A_m & A_q \end{bmatrix}.$$

The submatrix G_m is an m x m basis for the underlying network (or networks) composing the first m row of the LP problem. The basis inverse B^{-1} may be stated relative to this same partitioning as:

$$B^{-1} = \begin{bmatrix} (G_m^{-1} + G_m^{-1}E_qQ^{-1}A_mG_m^{-1}) & (-G_m^{-1}E_qQ^{-1}) \\ (-Q^{-1}A_mG_m^{-1}) & Q^{-1} \end{bmatrix}$$

where $Q = A_q - A_mG_m^{-1}E_q$.

The motivation for this partitioning is to factor out the matrix G_m. By its connection with the network, G_m may be stored as a graph and any operations involving G_m^{-1} may be performed by the special labeling and basis traversal techniques discussed in the preceding section. This means that G_m^{-1} need not be explicitly generated and the full basis inverse B^{-1} can be determined simply by referring to the partitioned components of the basis B and the q x q matrix Q^{-1}. Thus, Q^{-1} may be viewed as the *working basis inverse* for such problems. Since the row (and column) dimension of Q^{-1} is equal to the number of non-network constraints of the LP problem, the algorithmic steps of the simplex method can be executed much more efficiently and with considerably less demand on computer memory by merging the network updating of G_m^{-1} with the standard updating of Q^{-1}. The benefits previously discussed for dealing with network basis updating thereby carry over to the partitioned LP problem with only the residual portion of the basis associated with Q^{-1} being subject to the slower customary process with its greater attendant susceptibility to numerical inaccuracy. In addition, since the ordering of the rows to achieve the partition need be done only once (and often will automatically be accomplished by the initial formulation), the augmentation of commercial LP systems with such a "special order network" *(SON)* feature would constitute a noteworthy and beneficial enhancement.

The development of the SON feature will, in our opinion, produce the next major computational advance in large-scale linear programming. In fact, it should have a more profound effect than the development of generalized upper bounding (GUB). This belief is based on the following:

1) GUB is a specialization of these procedures. This can be seen simply by observing that the GUB constraints can be scaled and summed to form another constraint which, when appended to the GUB constraints, yields a network.

2) SON extends GUB in several important ways. For example, it eliminates the non-overlapping variable requirement of GUB in an efficient manner. Other attempts to accomplish this by "generalized GUB" procedures do not share the computational power of network techniques.

3) Computational experience with the SON feature in prototype applications involving both a simple form of the problem, solved completely by SON, and a fully general form of the problem, solved partly by SON, already demonstrate that substantial computational savings are possible. In particular, the NETSG code, previously reported to be 25 times faster than APEX-III (and with substantially better memory requirements), is a direct application of SON to the

situation of a single linear constraint where the network portion is a generalized network with arbitrary multipliers.

At the opposite end of the spectrum, we have implemented a "first pass" execution of the SON feature for a major automobile manufacturer engaged in solving more general LP problems with large imbedded network components. The firm was solving the LP problems of this application on an IBM 370/145 using MPSX, incurring computer run times in excess of 30 minutes per problem. We superimposed a network system on the MPSX system in a manner designed to operate on the network portion of the problem to produce an advanced starting basis for MPSX. This "first pass" application by itself reduced the total solution time from over 30 minutes to 10 minutes.

There is another important application of the SON feature that extends beyond its use in solving LP problems. This application has to do with solving pure and mixed integer programming (IP) problems. Current research has shown that the best solution approach is to use formulations which keep the number of integer variables as small as possible and which yield strong LP relaxations, rather than minimizing the number of constraints. Fortunately the IP problem manipulation schemes for obtaining stronger LP relaxations often induce network structure. To illustrate this statement, consider the constraint

$$x_1 + x_2 + x_3 - 3x_4 \leq 0$$

where x_1, x_2, x_3, x_4 are 0-1 variables. It is well known that replacing this constraint by the constraints:

$$s_1 + x_1 = x_4$$
$$s_2 + x_2 = x_4$$
$$s_3 + x_3 = x_4$$

yields a stronger LP relaxation. Note however that by performing elementary row operations this latter set of constraints is equivalent to:

$$\begin{array}{llllll} 0 = & x_1 - x_2 & & + s_1 - s_2 & & \\ 0 = & & x_2 - x_3 & & + s_2 - s_3 & \\ 0 = & & x_3 & & & s_3 - x_4 \end{array}$$

Subtracting these constraints produces $0 = - x_1 - s_1 + x_4$. Appending this equation to the others yields a network.

By combining the IP problem manipulation approach with the SON feature the working basis inverse of the LP problem would be reduced by the number of constraints that would otherwise be added to accommodate relationships of the form indicated. Thus the addition of the SON feature to commercial LP systems would allow practitioners to use stronger IP/LP formulations.

The above example is only one of many important uses of the SON feature in integer programming. Almost every practical IP application--fixed charge, location allocation, project selection, capacity expansion, and set-up scheduling problems have substantial network substructures. In sum, it seems clear that the SON feature would constitute an important computational advance for large-scale LP and IP.

The concepts we have sketched in this paper demonstrate how the incorporation of the SON feature into optimization software can become a reality, following in the path of the remarkably efficient network computer codes now in existence.

REFERENCES

1. R. Barr, F. Glover, and D. Klingman, "Enhancements of Spanning Tree Labeling Procedures for Network Optimization," Research Report CCS 262, Center for Cybernetic Studies, University of Texas at Austin, 1976.

2. R. Barr, F. Glover, and D. Klingman, "The Alternating Basis Algorithm for Assignment Problems," Research Report CCS 263, Center for Cybernetic Studies, University of Texas at Austin, 1977.

3. G. Bradley, G. Brown, and G. Graves, "Design and Implementation of Large Scale Primal Transshipment Algorithms," Technical Report NPS55BZBW76091, Naval Postgraduate School, Monterey, California, 1976.

4. A. Charnes and W. Cooper, *Management Models and Industrial Applications of Linear Programming*, Vols. I and II, Wiley, New York, 1961.

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

6. J. Gilsinn and C. Witzgall, "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees," NBS Technical Note 772, U.S. Department of Commerce, 1973.

7. F. Glover, J. Hultz, and D. Klingman, "Improved Computer-Based Planning Techniques," Research Report CCS 283, Center for Cybernetic Studies, University of Texas at Austin, 1977.

8. F. Glover, J. Hultz, D. Klingman, and J. Stutz, "A New Computer-Based Planning Tool," Research Report CCS 289, Center for Cybernetic Studies, University of Texas at Austin, 1976.

9. F. Glover, D. Karney, and D. Klingman, "The Augmented Predecessor Index Method for Locating Stepping Stone Paths and Assigning Dual Prices in Distribution Problems," *Transportation Science*, 6, 2 (1972), 171-179.

10. F. Glover, D. Karney, and D. Klingman, "Implementation and Computational Study on Start Procedures and Basis Change Criteria for a Primal Network Code," *Networks*, 4, 3 (1974), 191-212.

11. F. Glover and D. Klingman, "Network Application in Industry and Government," Research Report CCS 247, Center for Cybernetic Studies, University of Texas at Austin, 1975.

12. F. Glover, D. Klingman, and C. McMillan, "The NETFORM Concept," Research Report CCS 281, Center for Cybernetic Studies, University of Texas at Austin, 1977.

13. F. Glover, D. Klingman, and J. Stutz, "The Augmented Threaded Index Method for Network Optimization," *INFOR*, 12, 3 (1974), 293-298.

14. F. Glover and J. Mulvey, "Equivalence of the 0-1 Integer Programming Problem to Discrete Generalized and Pure Networks," MSRS 75-19, University of Colorado, Boulder, Colorado, 1975.

15. R. Helgason, J. Kennington, and H. Lall, "Primal Simplex Network Codes: State-of-the-Art Implementation Technology," Technical Report IEOR 76014, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas, Texas, 1976.

16. J. Hultz and D. Klingman, "Solving Singularly Constrained Generalized Network Problems," Research Report CCS 256, Center for Cybernetic Studies, University of Texas at Austin, 1976.

17. V. Srinivasan and G. Thompson, "Accelerated Algorithms for Labeling and Relabeling of Trees with Applications for Distribution Problems," *JACM*, 19, 4 (1972), 712-726.

18. H. Taha, *Operations Research*, The Macmillan Company, New York, 1971.

19. H. Wagner, *Principles of Operations Research*, Prentice-Hall, Englewood Cliffs, New Jersey, 1969.

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Center for Cybernetic Studies
The University of Texas at Austin

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

(6) 3. REPORT TITLE
Network Versus Linear Programming Algorithms and Implementations.

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
(5) N00014-75-C-0616, N00014-75-C-0569

(10) 5. AUTHOR(S) (First name, middle initial, last name)
Fred Glover
John Hultz
Darwin Klingman

(9) Research rept.

(11) 6. REPORT DATE
September 1977 (12) 31p.

7a. TOTAL NO. OF PAGES
26

7b. NO. OF REFS
19

8a. CONTRACT OR GRANT NO.
N00014-75-C-0616;0569

b. PROJECT NO.
NR047-021

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
Center for Cybernetic Studies
Research Report CCS 306

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
(14) CCS-306

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Office of Naval Research (Code 434)
Washington, DC

13. ABSTRACT

Over the years modelers and practitioners have become so adept at designing large scale linear programming problems that in some cases the complexity is staggering. ~~However,~~ such problems place a tremendous burden on current standard linear programming systems ~~and computer resources.~~ This has led to a demand for alternative design and solution procedures.

Noting that the vast majority of linear programming problems contain at least some embedded network structure, researchers in the network area have made many advances in the past few years. ~~In~~ fact, these advances have led to the ~~important new area of network computer~~ implementation technology.

This paper provides insights as to why network techniques are having such a large impact on modelers and why network solution procedures are being selected over standard linear programming systems for use in many real world applications. As dramatic proof of the savings offered by network solution procedures, various network computer codes are compared with the standard linear programming system APEX-III. Results indicate that the network techniques are 200 times faster on highly structured problems and as much as 25 times faster on more complex embedded network problems.

406 197

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6811

Unclassified
Security Classification

A-31408

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Networks						
Linear Programming						
Basis Inverse						
Embedded Networks						
Graphs						
Integer Programming						

DD FORM 1 NOV 65 1473 (BACK)
S/N 0102-014-6800

Unclassified
Security Classification

A-31409