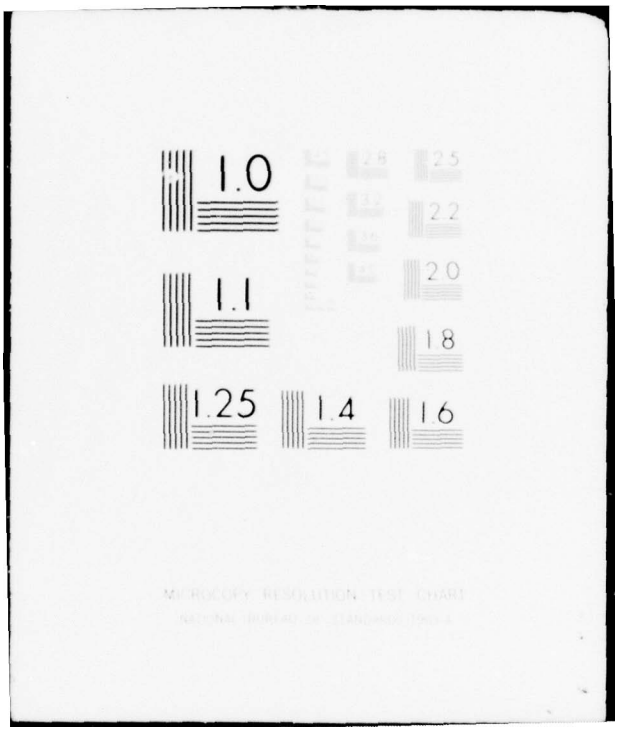
AD-A044 116 TEXAS UNIV AT AUSTIN CENTER FOR CYBERNETIC STUDIES F/G 9/2
A NEW COMPUTER-BASED PLANNING TOOL.(U)
DEC 76 F GLOVER, J HULTZ, D KLINGMAN, J STUTZ N00014-75-C-0616
UNCLASSIFIED CCS-289 NL

1 OF 1
AD A044116

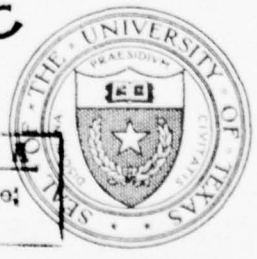
END
DATE FILMED
10-77
DDC


1.0
1.1
1.25
1.4
1.6
2.5
2.2
2.0
1.8

12

CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712



(12)

(9) Research Report CCS-289 (14)

(6) A NEW COMPUTER-BASED PLANNING TOOL

by

(10) F. Glover*
J. Hultz**
D. Klingman***
J. Stutz****

(11) December 1976 (12) 53 p.

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

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

***Professor of Operations Research and Computer Sciences, BEB 608, University of Texas, Austin, TX 78712.

****Associate Professor of Operations Research and Computer Sciences, BEB 613, University of Texas, Austin, TX 78712.

(15) This research was partly supported by ONR Contract N00014-76-C-0383 with Decision Analysis and Research Institute and by Project NR047-021, ONR Contracts N00014-75-C-0616 and N00014-75-C-0569 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, TX 78712
(512) 471-1821

406199

ABSTRACT

The purpose of this paper is to document this recent emergence of *generalized networks* as a fundamental computer-based planning tool and to demonstrate the power of the associated modeling and solution technologies when used in *concert* to solve real-world applications. To accomplish this, the paper is divided into two parts. Part I focuses on how generalized networks have been and may be used to model a diverse collection of significant practical problems. First we discuss (in nontechnical terms) the model structure of a generalized network (GN) and provide a brief historical survey of applications which have been modeled as GN problems. Next we present somewhat newer modeling techniques which draw heavily on generalized networks as a major component. The pictorial aspect of these techniques has proven to be extremely valuable in both communicating and refining nonlinear and combinatorial relationships.

Part II discusses the design and analysis of large-scale generalized network computer solution techniques. The central ideas that have brought GN problems into their own as a fundamental planning tool, by providing an effective methodology for computer implementations are clearly outlined. Part II further contains an in-depth computational study of solution strategies for GN problems within the framework of specializations of the primal simplex method. The study identifies an efficient solution procedure that derives from an integrated system of start, pivot, and degeneracy rules. The resulting method is shown on large problems to be *at least 50 times faster* than the sophisticated state-of-the-art LP system, APEX-III. Finally, the study demonstrates that the memory requirements of the method, as well as its solution times, are sufficiently small to warrant its use as a *computer-based planning tool* not only in a batch processing environment, but also in an interactive environment.

Introduction

Management Science has progressed rapidly since World War II, creating a virtual explosion of new knowledge about ways to solve optimization problems in industry and government. The single most important factor motivating this explosion of new knowledge has been the parallel growth of the computer industry. The computer has given management scientists the capability to record and manipulate extremely large amounts of data in an efficient manner. Without this capability, many of the tools of management science would be mere theoretical niceties.

The advent of the computer has given rise to the development of *computer-based* planning models. The techniques for building, solving, refining, and analyzing such models have undergone a steady evolutionary development as computer hardware has changed. As a result of this evolution, linear programming (LP) has emerged as one of the most widely used tools of *large-scale computer-based planning*. This is largely due to the fact that LP models portray many practical applications and current commercial LP computer packages have been able to solve problems sufficiently large to meet the requirements of real-world settings.

General linear programming techniques have not proven to be totally satisfactory, however. The conceptual design, data collection, and computational capabilities of such models have often fallen short of what is required for a truly useful *decision support tool*. As a result, management scientists have intensively studied the structures of LP models and have determined *fundamental building blocks* which appear in many applications. Network models have long been recognized as being key building blocks for many problem formulations. Studies of the structure of LP models from practical applications have underscored this conclusion. Because of this, mathematical programmers have devised ingenious solution algorithms for particular types of network problems. The computer age has spurred the development of computer codes embodying these algorithms which

have proven in *practice* to be dramatically more effective than general LP codes for solving these problems [1, 11, 19, 20, 28, 32, 38, 42].

As a consequence, various subclasses of network models, such as shortest path, maximum flow, minimum spanning tree, assignment, transportation, and transshipment models, have come to be considered as rightfully belonging to the class of *fundamental computer-based planning tools*. That is to say, a model is customarily regarded as a *fundamental computer-based planning tool* if it is capable of accurately depicting a variety of significant applications either directly, or as an integral component. Additionally, if the underlying model is a special class of an LP model, then large-scale algorithms and computer codes must exist which are at least an order of magnitude faster than state-of-the-art commercial LP packages for this model type.

Previously, generalized network (GN) models [12, 14, 30] have not been considered to be fundamental computer-based planning tools. Bradley [10] in 1975 stated that GN problems "in the near future . . . could come to be regarded as a fundamental model." The primary reason this important problem class has not previously achieved this distinction has been the lack of efficient large-scale computer codes. At an early stage in the evolution of the computer age, researchers developed ingenious special purpose solution algorithms for these problems. Further, in the 50's the early pioneers of management science identified important problem classes which could be modeled as GN problems. Unfortunately, these model identification efforts did not continue with any rigor in the 60's because no effective special purpose computer codes appeared. Thus, such models could only be solved with general purpose LP codes.

Recently, however, the situation has been changing and the somewhat overdue computer codes for generalized networks are at last appearing. As a consequence,

modelers have begun to devote attention to determining if an LP model is a GN problem and, more importantly, to devising formulations in which generalized networks play the role of critical components.

The purpose of this paper is to document this recent emergence of *generalized networks* as a fundamental computer-based planning tool and to demonstrate the power of the associated modeling and solution technologies when used in *concert* to solve real-world applications. To accomplish this, the paper is divided into two parts. Part I focuses on how generalized networks have been and may be used to model a diverse collection of significant practical problems. First we discuss (in nontechnical terms) the model structure of a generalized network and provide a brief historical survey of applications which have been modeled as GN problems. Next we present somewhat newer modeling techniques which draw heavily on generalized networks as a major component. This collection of modeling techniques is called the NETFORM (network formulation) concept or approach. The pictorial aspect of this approach has proven to be extremely valuable in both communicating and refining nonlinear and combinatorial relationships. Additionally, the NETFORM concept often yields a formulation that enables the problem to be solved as a sequence of GN problems with dramatic gains in efficiency over alternative approaches. To provide an understanding of this approach and the role of generalized networks within it, two real-world applications are described which have profited by its use. Thus, Part I illustrates the manner in which generalized networks are emerging as **fundamental building blocks** for modeling, communicating and solving a multitude of problems.

Part II discusses the design and analysis of large-scale generalized network computer solution techniques. The central ideas that have at last brought GN problems into their own as a fundamental planning tool, by providing an effective

methodology for computer implementations, are clearly outlined. Part II further contains an in-depth computational study of solution strategies for GN problems within the framework of specializations of the primal simplex method. The study identifies an efficient solution procedure that derives from an integrated system of start, pivot, and degeneracy rules. The resulting method is shown on large problems to be *at least 50 times faster* than the sophisticated state-of-the-art LP system, APEX-III. (Thus, the method can solve a problem every week for a year and consume the same amount of computer time required to solve the problem only once with the LP system.) Finally, the study demonstrates that the memory requirements of the method, as well as its solution times, are sufficiently small to warrant its use as a *computer-based planning tool* not only in a batch processing environment, but also in an interactive environment.

PART I - GENERALIZED NETWORK MODEL

1.0 PROBLEM DEFINITION

The generalized network problem represents a large class of LP problems. This class includes any LP problem whose coefficient matrix, ignoring simple upper bound constraints, contains at most two non-zero entries in each column. A large portion of the literature on LP problems has been devoted to the special cases of the GN problem in which the non-zero elements of a column consist of a 1 and a -1 (either initially or by linear transformation). This condition identifies the problem as a pure network, whose instances include shortest path, maximum flow, assignment, transportation, and transshipment problems. The GN problem, by allowing other non-zero doubletons (and singletons) in a column, is actually the broadest classification of linear network related problems. Practical settings in which such GN problems arise include resource allocation, produc-

tion, distribution, scheduling, capital budgeting, and many other problem types which will be elaborated subsequently.

The most effective procedures for modeling and communicating pure network problems are based on viewing these problems as directed graphs. A generalized network can also be represented as a directed graph as follows.

Under the assumed existence of a finite optimum, it is possible to transform the coefficient matrix (by scaling or by complementing a variable relative to its upper bound), so that if a column has two non-zero entries, at least one of these is -1. In this way, a directed arc is "formed" from the node associated with the -1 to the node associated with the other non-zero entry. (If both entries are -1, the arc may be directed either way.) Columns with single non-zero entries give rise to arcs incident on only one node.

There is an important distinction between arcs in pure network problems and arcs in GN problems. An arc of a generalized network has a *multiplier* associated with it. This multiplier is the non-zero coefficient associated with the node at the head of the arc (i.e., to which the arc is directed). In pure networks, this multiplier is always +1.

Consider the following GN problem:

$$\text{Minimize} \quad 1x_{12} + 5x_{13} + 3x_{23} + 1x_{24} - 4x_{32} - 9x_{34}$$

$$\begin{aligned}
\text{subject to:} \quad & -1x_{12} - 1x_{13} && && && = -5 \\
& 2x_{12} && - 1x_{23} - 1x_{24} + 1/3x_{32} && && = 0 \\
& && 1/2x_{23} + 1x_{23} && - 1x_{32} - 1x_{34} && = 0 \\
& && && -1/5x_{24} && + 3x_{34} = 10
\end{aligned}$$

$$0 \leq x_{12} \leq 3,\ 0 \leq x_{13} \leq 4,\ 0 \leq x_{23} \leq 6,$$

$$0 \leq x_{24} \leq 5,\ 0 \leq x_{32} \leq 3,\ 0 \leq x_{34} \leq 7$$

The network associated with this problem is shown in Figure 1. As with pure network problems, each row of the coefficient matrix is associated with a node and each column with an arc. (In other words, a node corresponds to a problem equation and an arc corresponds to a problem variable.) The arc is directed from the node associated with the -1 entry toward the node associated with the other non-zero entry. Likewise, each arc has a cost, lower bound, and upper bound which are shown in Figure 1 within the square and parentheses, respectively. The non-zero multiplier associated with an arc is shown in Figure 1 within a triangle. The constant terms (right hand sides) of the problem equations identify supply and demand requirements attached to the corresponding nodes. A negative constant term identifies a supply (which by convention equals the absolute value of this term), a positive constant term identifies a demand, and a 0 constant term identifies a "conservation condition" in which the amount of flow entering the node must be exactly matched by the amount of flow leaving the node.

As flow passes across an arc in a generalized network problem, it is acted upon by the non-zero multiplier. Thus, the amount starting out on an arc will not necessarily be the amount arriving at the opposite end. In particular, the multiplier indicates that the flow entering the arc is multiplied by the value of the multiplier as that flow leaves the arc. For example, if 2 units start on the arc from node 1 to node 2 in Figure 1, 4 units will arrive at node 2 since the multiplier is 2. Likewise, 10 units starting on the arc from node 2 to node 4 will result in 2 units arriving at node 4 since the multiplier in this case is 1/5. It should be noted that *the arc's cost, lower bound, and upper bound apply only to the units of flow entering the arc.*

Another important feature of GN problems is that total supply may not be the same as total demand. In pure network problems, total supply always equals total demand. However, the effect of multipliers is such that total supply and total demand may, in fact, be entirely different. This can result in odd structural consequences, such as absorbing and generating cycles. (See [3, 29, 30].)

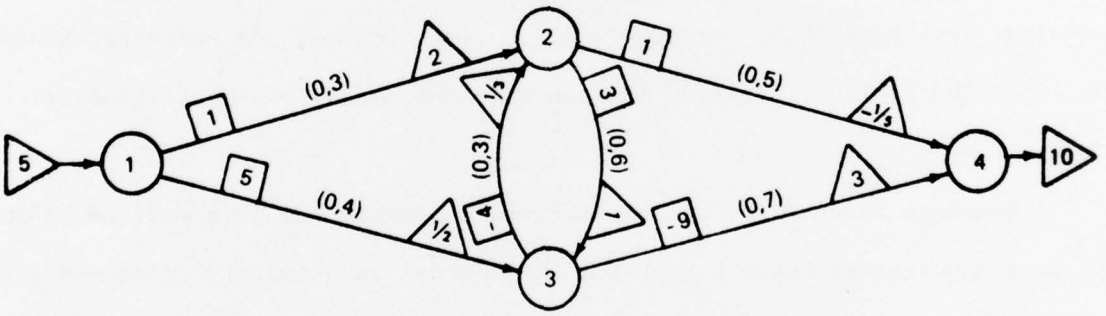


Figure 1

Generalized Network

2.0 APPLICATIONS OF GENERALIZED NETWORKS

Generalized networks can be used to model numerous problems for which there is no pure network equivalent. There are essentially two ways in which the multipliers on the arcs of generalized networks may be interpreted. First, they can be viewed as simply modifying the amount of flow of particular goods. In this way, generalized networks can represent such situations as evaporation, seepage, deterioration, breeding, interest rates, sewage treatment, purification processes with varying efficiencies, machine efficiencies and structural strength design. However, it is also possible to interpret the multiplication process as a transformation from one type of good to another. With this interpretation, it is possible to model such processes as manufacturing, production, fuel to energy conversions, blending, crew scheduling, manpower to job requirements, and currency exchanges. The following applications lend insight into the possible uses of generalized networks.

A complete water distribution system with losses has been modeled by Bhaumik [8] as a generalized network problem. This model is primarily concerned with the movement of water through canals to various reservoirs. However, the model must also consider the retention of water over several time periods. The multipliers in this case represent the loss effect due to both evaporation and seepage.

Turner and Gilliam [17] have proposed a file reduction model which has the form of a generalized transportation model with a single extra constraint. This model is designed to facilitate the reduction of extremely large microdata files to smaller, statistically representative files. The objective, in this case, is to minimize the amount of information lost in the reduction process. The arcs represent paths from the original records to the reduced records. A non-zero flow on an arc implies that the originating record is to be represented by the terminal

record. The multipliers on the arcs are used to insure that the reduced file is truly representative of all of the original records.

Kim [35] has utilized generalized networks to represent copper refining processes. The electrolytic refining procedure, in this case, is modeled by a large d-c electrical network. The arcs are current paths with the multipliers representing the appropriate resistances. In this way, Kim analyzes the effect of short circuits in the refining process.

Charnes and Cooper [12] identify applications of generalized networks for both plastic-limit analysis and warehouse funds-flow models. In plastic-limit analysis, the network is generated by forming the equations for horizontal and vertical equilibrium and by employing a coupling technique. The warehouse funds-flow model is actually a multi-time period model. The arcs are used to represent sales, production, and the inventory holding of both products and cash. The multipliers are introduced to facilitate the conversions between cash and products.

A cash management problem has been modeled as a generalized network by Crum [13]. This model for the multi-national firm incorporates transfer pricing, receivables and payables, collections, dividend payments, interest payments, royalties, and management fees. The arcs represent possible cash flow patterns and the multipliers represent costs, savings, liquidity changes, and exchange rates. Other applications include machine loading problems [12, 14, 44], blending problems [12, 44], the caterer problem [14, 44], and scheduling problems such as production and distribution problems, crew scheduling, aircraft scheduling, and manpower training [12, 14, 44].

3.0 INTEGER GENERALIZED NETWORKS

The uses of arc multipliers just discussed do not by any means exhaust their range of application. In fact, upon adding the requirement of discreteness, which

forces the flows on particular arcs to occur in integer quantities, the GN problem is capable of modeling an unexpected diversity of problems. For example, introducing discreteness into the GN model produces a framework for problems such as scheduling variable length television commercials into time slots, assigning jobs to computers in computer networks, scheduling payments on accounts where contractual agreements specify "lump sum" payments, and designing communication networks with capacity constraints. While these are "direct" applications, the use of special modeling principles, sometimes called NETFORM principles, enable even somewhat more complex applications to be modeled and solved as integer GN problems. In fact, the NETFORM principles and techniques make it possible to model any 0-1 LP problem as an integer GN problem [23, 27]. These procedures extend quite naturally to accommodate mixed integer 0-1 LP problems where the continuous part of the problem is a transportation, transshipment or generalized network problem itself. Reference [43] illustrates this extension and shows how contemporary financial capital allocation problems can be modelled as integer GN problems. Many other important real-world applications have such a "mixed" structure, including a variety of energy models, plant location models, and physical distribution models.

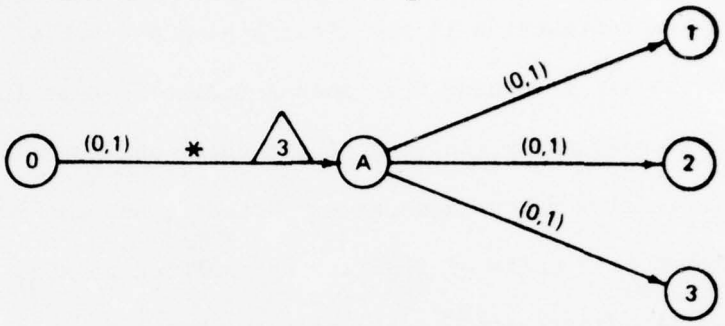
The NETFORM representation is mathematically equivalent to any of the algebraic representations that can be arrived at by customary mathematical programming formulation techniques. However, by the creation of the network-related structures, which may or may not be implicit in any customary formulation, the NETFORM representation permits the application of specialized solution methods, tailored to employ algorithmic advances discussed in Part II for exploiting the graphical relationships of network components. The remainder of this section briefly describes the basic principles of the NETFORM approach and discusses two applications which have profited from its use.

Figure 2 illustrates one of the useful modeling devices of the NETFORM approach that finds application in a variety of settings. The costs, bounds, and multipliers are represented in the same fashion as earlier. In addition, the *asterisk* on the arc from node 0 to node A indicates that its flow must be an *integer* amount. Consequently, in view of the upper and lower bounds on this arc, the only acceptable flow values are exactly 0 and 1, and the multiplier thus ensures that either 0 or 3 units of flow are transmitted to node A. Further, the only possible way to distribute 3 units of flow into node A is to send exactly one unit to each of the nodes 1, 2, and 3 since each of the three arcs leaving A has an upper bound of 1. Thus, in sum, the following effect has been achieved: *when the flow on the arc from node 0 to node A is 0, the flow on each of the three arcs out of node A is 0; when the flow on the arc from node 0 to A is 1, the flow on each of the three arcs out of node A is 1.*

It should be noted that multipliers may further be attached to the arcs leaving node A, so that their flows may be further transformed. Thus, for example, the flow on the arc from node 0 to node A can represent an investment decision (invest if flow = 1, do not invest if flow = 0), and the flows on arcs out of A can represent components of the investment (e.g., particular stocks in a portfolio, tracts of land in a real estate venture, items of equipment in a manufacturing operation, etc.). Multipliers on the letter arcs would then express the number of items of each of these investment components that are obtained by selecting the main investment. (For example, a particular equipment investment may be composed of six machines of type 1, eight machines of type 2, and so forth.) The combination of arc multipliers and the 0-1 integer restriction gives rise to what is called an *integer generalized network* or a *0-1 generalized network*. This NETFORM modeling tool has a variety of important uses, as demonstrated more concretely by the following two real-world applications.

Figure 2

Generalized Network with Integer Flow Restrictions



Air Force Course Scheduling

The Air Force requires Undergraduate Flight Training (UFT) graduates to take advanced flight training before their first operational assignment. UFT graduates must also take from one to four survival training courses. Each individual has a different background and therefore may require a different mix of these courses. Furthermore, both the flight and the survival training courses are offered only at certain times, are subject to enrollment limits, and generally have prerequisites. A set of feasible course schedules is identified for each UFT graduate, and given a "cost rating" by a computer program called the UFT Pipieline Scheduling Model. Feasibility and cost considerations depend on additional factors such as attendance requirements at Combat Crew Training courses, various modes of transportation, the number of dead days in the pipeline, the opportunity for the UFT graduates to take leave as desired, etc.

The objective is to select a particular course schedule for each UFT graduate so that the complete set of schedules selected will satisfy all class enrollment limits and yield the smallest total cost. To solve this problem, the personnel manager in the Training Pipeline Management Division *manually* assigns each graduate to one of his feasible schedules, trying to assure that all enrollment limits will be satisfied. Clearly, this is a difficult and time-consuming task to do by hand; further, the total cost of these manual assignments may be far from optimal.

In search of a better approach, the Air Force developed an integer programming formulation for this problem. However, the IP formulation turned out to be almost totally resistant to solution. Consequently, we reformulated this integer programming problem as a 0-1 GN problem shown in Figure 3.

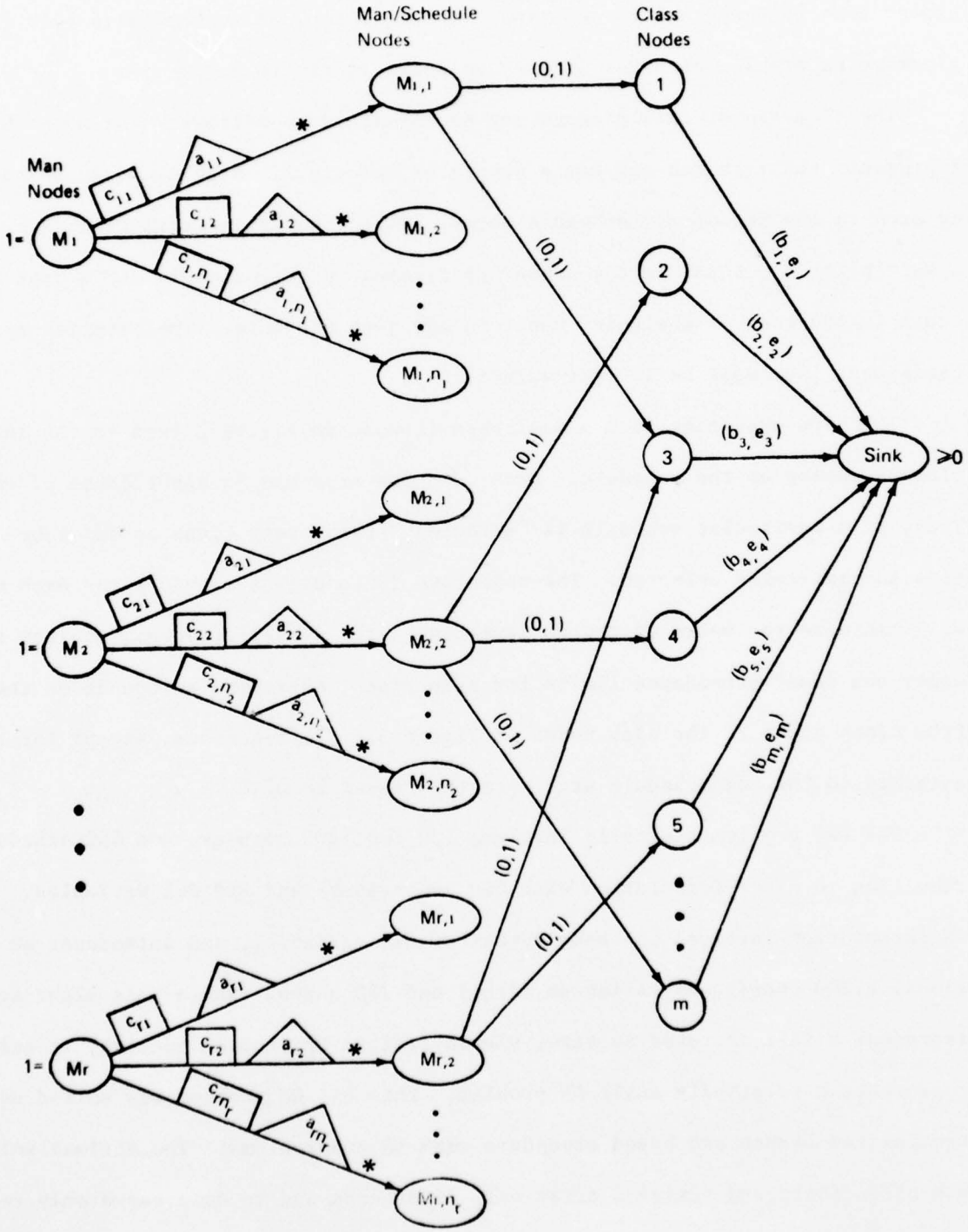
The elements of this diagram may be explained as follows. The node Mi represents the i-th man and has a supply of exactly 1. Each man node is connected by arcs to its set of man/schedule nodes. These connecting man/schedule arcs have a multiplier a_{ij} equal to the number of classes in the schedule and a cost c_{ij} equal to the cost of assigning man i to his j-th schedule. The asterisk again indicates that flow must be integer-valued.

The arcs emanating from a man/schedule node in Figure 3 lead to the individual classes making up the schedule. Each of these arcs has an upper bound of one. Thus, if a particular schedule is "selected," then every class in the schedule is also automatically selected. The objective is to pick a schedule for each man that will minimize the value of the assignments on the overall program, subject to the upper and lower attendance limits for each class, expressed as bounds on the arcs from class nodes to the sink nodes of .igure 3. All arc costs, except for those attached to the man/schedule arcs, are thus equal to 0.

The UFT problem typically involves 120 men, 200 classes, and 460 schedules, resulting in a 0-1 formulation with 520 constraints and 460 0-1 variables. The 0-1 GN formulation involves the same number of 0-1 variables, and introduces an additional 2,200 continuous variables (arcs) and 780 nodes. While this might seem to represent a fair increase in size, viewed from an LP problem context, it actually represents a relatively small GN problem. This 0-1 GN problem was solved using a specialized branch and bound procedure with GN subproblems. The optimal solution was often found and verified after only 30 seconds and in some cases only required

Figure 3

UFT NETFORM Formulation



a total solution time of 10 seconds on a CDC 6600. Thus, the problem was transformed from one that had been extremely difficult to solve as an integer program to one that was solved easily as a NETFORM.

Refueling Nuclear Reactors

A mixed integer programming problem for determining the minimum cost refueling schedule for nuclear reactors, modeled by Kazmersky [34], has similarly benefited by the use of the NETFORM concept. Although the mixed integer programming formulation bore no apparent connection to networks, we discovered a way to express the problem by a convenient NETFORM representation after interacting closely with Dr. Kazmersky. This representation was not only equivalent to the original formulation but also succeeded in reducing its size. We will forego the details of the transformation of the original problem to a 0-1 GN problem because the steps are somewhat intricate and the original formulation by itself consumes more than twenty pages of [34]. However, we were able to exploit the 0-1 GN formulation by developing a branch and bound solution procedure which employed the GN optimization procedures discussed in Part II of this paper. Using this computer system, four versions of this problem were solved using data supplied by the TVA. The first three versions required half an hour to two hours to solve on an IBM 370/168 using MPSX. By contrast, these versions were easily solved in 10 to 20 minutes using the 0-1 GN formulation and the specialized solution approach. The fourth version, which involved 173 constraints, 126 zero-one variables, and 511 continuous variables, was by far the most difficult. Again, using MPSX on an IBM 370/168, the original mixed integer formulation was run for eight hours and then taken off the machine to avoid further computer run costs. The best (minimum cost) solution obtained had an objective function value of $136,173,440. With a 30 minute time limit imposed on the 0-1 GN solution effort, a solution was obtained that was more than $10,000,000

cheaper, with an objective function value of $125,174,727. Consequently, this application shows that problems too complex to be solved optimally (within practical time limits) by standard approaches may be helped substantially by the NETFORM approach.

4.0 IMPLICATIONS FOR THE USE OF GN MODELS

The foregoing development provides two powerful incentives for adopting a GN formulation where appropriate. One is the ability to conceptualize a GN formulation graphically rather than algebraically. This pictorial aspect has highly beneficial effects for teaching people how to formulate their problems and for communicating mathematical models to nonscientific users. The other major incentive is the ability to solve GN problems--and by extension, a variety of problems with GN components--with a remarkable degree of efficiency. Part II establishes the underlying support for this second incentive. Namely, it presents an in-depth computational analysis of algorithmic rules and computer implementation procedures which are capable of solving large-scale GN problems well above an order of magnitude faster than state of the art LP codes.

PART II - DESIGN AND ANALYSIS OF LARGE-SCALE GENERALIZED NETWORK COMPUTER PROCEDURES

1.0 OVERVIEW

This part presents an extensive computer analysis of algorithmic rules for GN problems. Computational studies of pure network solution procedures have done much to advance the state-of-the-art. Excellent testing has been performed on transportation [18, 20, 28, 36, 39, 44] and transshipment [4, 5, 11, 19, 26, 33, 37, 42] computer codes. These studies have provided critical insights into the best methods for solving such problems as well as providing benchmark data for

future solution procedures.

However, there have been no in-depth studies to date concerning the much broader class of GN problems. This is not to say that computer codes do not exist for solving such problems. Code development has been reported by Eisemann [15], Maurras [40], Glover, Klingman, and Stutz [25], Bhaumik and Jensen [9], Langley [38], and Balachandran [2], among others. Most of these papers report findings for only certain classes of GN problems and all of them are limited in the scope of the computational analysis. Thus, an important body of empirical research is lacking in the network literature.

The code NETG reported by Glover, Klingman, and Stutz [25] was selected to form the basis for the computational testing of this study. This code is an implementation of the highly efficient extended augmented predecessor index (EAPI) procedure [18, 24], and embodies many of the latest advances in solution methodology for generalized network problems. In addition, the code is written entirely in FORTRAN, has been thoroughly debugged and tested during more than three years of developmental work, and is organized to allow parameter manipulations and subroutine insertions to be carried out conveniently. These features are extremely important to an empirical study of strategic implementation possibilities.

In any computer implementation, there are numerous steps that can be performed in alternative ways. Experience from previous studies of pure network problems has shown that the determination of an effective set of decision rules to handle such alternatives can have an enormous impact on the efficiency of the implemented solution method. Consequently, a primary objective of this study is to investigate decision rules for the GN problem and establish their relative merits. The rules determined to be best have been integrated to produce a code which has been tested against the highly efficient linear programming system, APEX-III. The results of

this comparison indicate that the special purpose GN code is at least 50 times faster than APEX-III on large GN problems.

2.0 PROBLEM STATEMENT

We may formalize the illustrations of Part I by defining a *generalized network* problem as follows:

$$\text{Minimize } c^T x \tag{1}$$

subject to:

$$Gx = b \tag{2}$$

$$0 \leq x \leq u, \tag{3}$$

where each column of the coefficient matrix G contains at most two non-zero entries. It will be assumed that u is finite (e.g., using "regularization" [12] if necessary). The problem as stated may be conceptualized as a graph containing vertices and non-directed edges. However, we will further assume that if a column of G has two non-zero entries, then at least one of these is negative and by appropriate scaling has a value of -1. This allows the problem to be interpreted in a directed graph (digraph) setting, as noted in Part I.

Each column of G is associated with a directed edge (*arc*) of the digraph and each row is associated with a vertex (*node*). An arc runs from the tail node, associated with the -1 coefficient, to the head node, associated with the other non-zero entry. In the case of a single non-zero entry in the column of G, the associated arc is called a *self-loop* and is incident on a single node. The *flow* on the arc is defined to be the value assigned to the corresponding component of x (i.e., to the variable whose column of G gives rise to the arc). Thus, by this association between arcs and variables, each arc has an associated cost per unit flow (the corresponding component of c) and an upper bound on the flow (the corresponding component of u). The coefficient in the column of G associated with the head of the arc is called the arc multiplier.

The right-hand side value (component of b) associated with a particular node determines whether flow will be inserted or removed from the network at that node. If the right-hand side value is negative, flow is inserted and the node is called a source node. Likewise, if the right-hand side value is positive, the node is called a sink node and flow is removed at that point. If a particular node has both arcs heading into it as well as out of it, then it is called a transshipment node. All other nodes will be either pure sources or pure sinks.

Since the computer code NETG, on which this study is based, employs a specialization of the primal simplex method, a brief discussion of this specialization is in order. We begin by examining the basis structure for this specialization.

It may be assumed without loss of generality that G has rank m, where m is the number of rows in G. Otherwise, the problem is equivalent to a set of disjoint minimum cost flow networks (see [21]). Any basis for the problem, then, will be composed of m linearly independent column vectors selected from G. Graphically, a working basis of the simplex method is a set of m arcs incident on the m nodes of the problem. It has been shown [16, 24, 38, 40] that the graph of a basis will be composed of a set of disjoint quasi-trees. Each quasi-tree is a simple tree to which a single arc is added. The additional arc forms exactly one cycle, which is a unique series of arcs leading from a node back to that node. By convention, to allow this characterization to apply to the case in which the additional arc is a self-loop, a self-loop is regarded as a cycle.

The EAPI method, on which NETG is based, is specifically designed to store the bases of generalized network problems in the graphical quasi-tree form. All of the primal simplex operations are carried out by working with the basis graphs which are stored using linked list procedures. The inherent advantages of this method will be fully described subsequently.

3.0 DATA STRUCTURES IN THE EAPI METHOD

A generalized network problem is simply a type of LP problem and can thus be solved using any standard LP solution technique. Improvements in inversion and reinversion processes, data compactification, and pivot selection strategies have provided dramatic increases in the efficiency of primal simplex computer codes in recent years. In many cases, special structure such as embodied in the generalized network problem is detected by these codes. This information is then used to reduce storage requirements and to simplify operations. However, none of the current LP systems is capable of fully exploiting the structure of generalized network problems.

One of the conspicuously exploitable features of generalized network problems is the sparsity of the coefficient matrix, and current LP codes are of course designed to take advantage of sparsity to store data economically. If the problem has been transformed to digraph form, however, storage may be further reduced. By the use of simple ordered lists to capture the digraph structure, NETG is designed to store only the cost coefficient, the non-zero multiplier, and the upper bound for each column of the coefficient matrix. In this way, problem data can often be resident in fast main memory even for extremely large problems.

As noted in section 2, bases for generalized network problems have a very special structure. With possible reordering of the rows and columns, the basis matrix forms a block diagonal matrix. Each of the blocks is either triangular or near-triangular and can be represented as a quasi-tree. Johnson [31, 32] originally proposed a linked list procedure for storing simple trees and suggested its use for the more complex quasi-trees. Effective labeling procedures for restructuring (updating) quasi-trees by reference to such lists are the core of the EAPI method developed by Glover, Klingman, and Stutz [24].

The original problem data (compactly stored) and the basis matrix (stored via linked lists) are the only data elements that need to be kept by a specialized code such as NETG. The main factor here is that a basis inverse need not be maintained. Inverses generally require considerable amounts of storage and involve numerous (error-producing) arithmetic operations to utilize and maintain them. Instead, the specialized labeling rules of the EAPI method operate on the basis graph in a manner that obviates the use of a basis inverse.

The EAPI method [24] orients each quasi-tree so that the cycle is conceptually located at the top, with attached trees hanging downward. This orientation is called a rooted cycle. The linked lists and labeling procedures provide the means of traversing the tree in both upward and downward directions.

Without detailing minutely the rules and processes of these procedures, it is useful to sketch their main functions in overview. In particular, the two types of quasi-tree traversal accommodated by the procedures are used to carry out basis exchange steps. The first, upward traversal, is associated with operations normally requiring pre-multiplication by the inverse, such as determining the representation of an entering vector (arc). This operation is performed by traversing the unique paths from selected vertices up to their associated cycle(s). Simultaneously, the equivalent triangular system of equations is solved, in effect, by back substitution. A directed trace of the cycle(s) using the cycle factor(s) [22, 24] completes the operation. The process of determining the representation of a vector requires traversing at most two quasi-trees and only generates the non-zero entries of this representation.

Downward quasi-tree traversal is analogous to post-multiplication by the inverse. This operation is used to calculate updated dual variable values. In a network interpretation, there is a dual variable associated with each of the nodes in

the problem. These are often referred to in the literature as node potentials. At each iteration, new dual values associated with a subset of the nodes in a single quasi-tree must be determined. A single value associated with a node on the cycle may be determined using the cycle factor [22, 24]. The resulting system of equations is triangular, and may be solved by traversing downward through the quasi-tree, again employing back substitution. This process only involves those dual values that change during a basis exchange.

Therefore, the specialized procedures reduce both data storage requirements and arithmetic operations required to carry out basis exchange steps. However, there are several alternative methods for implementing the graph processing steps that underly these procedures. The primary purpose of this study is to determine which of these implementation methods are the most efficient.

4.0 EXPERIMENTAL DESIGN

The computer code NETG is coded entirely in standard FORTRAN IV. The use of machine dependent operations is totally avoided in order to ease the transition to various computers. However, this fact also lends credence to the results obtained from different computers, since none has a particular coding advantage. The program was initially coded, debugged, and tested, using the RUN compiler on a CDC 6600 computer with a maximum main memory allocation of 130,000 words. The complete capacitated algorithm occupies 8N + 4A + 8500 words of central memory, where N is the number of nodes and A is the number of arcs in the specific problem being solved.

Since most of the testing performed is of a comparative nature, it was desirable to obtain a set of problems that met certain specifications and that could be made available on a repeated basis. For this reason, a generalized network problem generator was developed. This code is a logical extension of the NETGEN

[37] problem generator for pure network problems. NETGEN uses a random number generator to create problems that conform to certain structural characteristics supplied by the user. The user can control the number of nodes, sources, sinks, transshipment nodes, and arcs as well as the cost range, the total supply, and the capacity restrictions. The new code, NETGENG, has the added feature that the user can specify a range of values from which the arc multiplier values are chosen.

Using NETGENG, the problem set in Table I was generated for the purpose of computational testing. The information shown is the input specification for NETGENG. It should be possible to recreate this data set exactly on virtually any medium to large scale computer. The stated problems were specifically chosen so that the effects of problem structure on solution time could be noted. The problems vary in size from 200 nodes and 1500 arcs up to 1000 nodes and 7000 arcs. The 200, 300, 400, and 1000 node problems consist of different types of capacitated and uncapacitated generalized transportation and transshipment problems.

With the problem set complete, the process of selecting decision rules began. The types of decision rules tested include start procedures, pivot selection techniques, methods for exploiting degeneracy, tolerance levels, Big-M values, and pivot tie-breaking rules. It was economically infeasible to perform tests on all possible combinations of decision rules or even to test each rule on all of the generated problems. For this reason, it was imperative to design the tests to yield useful and valid results while minimizing computational time. An outline of each test is given in the next section.

Table I

Problem Specifications

Problem	Nodes	Sources	Sinks	Transshipment Sources	Transshipment Sinks	Arcs	Cost Range	Multiplier Range	Total Supply	Percent High Cost	Percent Capacitated	Bound Range	Random Number Seed
1	200	100	100	0	0	1500	1-100	.5 -1.5	100000	0	0	-	13502460
2	200	100	100	0	0	2000	1-100	.5 -1.5	100000	0	100	1000-2000	13502460
3	200	5	195	0	0	4000	1-100	.5 -1.5	100000	0	100	1000-2000	13502460
4	200	15	50	10	20	6000	1-100	.25- .95	100000	0	100	1000-2000	13502460
5	300	150	150	0	0	1500	1-100	.5 -1.5	100000	0	0	-	13502460
6	300	5	295	0	0	2000	1-100	.5 -1.5	100000	0	0	-	13502460
7	300	135	165	135	130	4000	1-100	.5 -1.5	100000	0	0	-	13502460
8	300	10	40	5	15	6000	1-100	.5 -1.5	100000	0	0	-	13502460
9	300	2	50	1	20	7000	1-100	.5 -1.5	100000	0	0	-	13502460
10	400	200	200	0	0	2000	1-100	.5 -1.5	200	0	0	-	13502460
11	400	3	397	0	0	4000	1-100	.2 -1.4	100000	0	0	-	13502460
12	400	20	100	5	50	5000	1-100	.3 -1.7	100000	0	0	-	13502460
13	400	25	70	10	20	6000	1-100	.5 -1.5	100000	0	100	1000-2000	13502460
14	400	30	50	0	0	7000	1-100	.5 -1.5	100000	0	100	1000-2000	13502460
15	1000	5	995	0	0	4000	1-100	.2 -6.0	200000	0	100	4000-6000	13502460
16	1000	20	100	5	20	6000	1-100	.4 -1.4	200000	0	100	4000-6000	13502460
17	1000	20	50	20	50	6500	1-100	.5 -1.5	200000	0	0	-	13502460
18	1000	30	400	10	30	7000	1-100	.7 -1.2	200000	0	0	-	13502460

5.0 Computational Results

The testing of alternative decision rules was performed on a CDC 6600 computer located at the University of Texas at Austin. In each of the comparative tests, an attempt was made to execute the codes involved during comparable time periods. However, it should be noted that minor differences between two solution times should be statistically ignored. For this reason, many tests were run on large problems so that timing variations become less significant. Each code version makes use of a real-time clock routine supplied by CDC. This routine can be employed using a FORTRAN subroutine call and is generally accurate to two decimal places.

Start Procedures

The first phase of testing involves a comparison of three different start procedures. All of the starts tested are based on techniques that have proved effective for pure network problems. The first of these is the artificial start procedure. It attaches an artificial arc (self-loop) to every node in the problem. These artificial arcs are then assigned extremely large (Big-M) cost coefficients. The artificial method is the fastest start procedure to execute, but it does not yield an advanced initial basis.

The second method tested is the sequential source minimum (SSM) procedure. This method sequentially examines each node in the problem. If the node has associated supply, flow is assigned to the arc having the least cost that leads to a node with positive demand, or, if none exists, to a node with zero demand. The flow is set equal to the minimum of the supply, the upper bound on the arc, or the demand (if non-zero). If the flow on an arc is set equal to the supply or the demand, the associated node is eliminated from further consideration. The number of passes made through the nodes is at the discretion of the user. If the process is termin-

ated before supply and demand have been exhausted, then artificial arcs are appended. For the purposes of testing, the number of passes was set to 1, 2, 3, 5, and exhaustive.

The last start method tested is the exhaustive node supply. This method is similar to the sequential source minimum in the way it assigns flow to arcs. However, the procedure continues to assign flow out of a particular node until the supply at that node is exhausted or until no further arcs exist. At that point, the next node with supply is considered. Upon completion, remaining supply and demand are met by appending artificial arcs.

Each of the start methods described above was tested using two distinct pivot selection criteria. These are the node first negative and the node most negative criteria. Both methods are based on examining the non-basic arcs leading out of a given node. The node first negative method selects the first encountered pivot eligible arc for the basis exchange. The node most negative method, on the other hand, selects the best pivot eligible arc (in terms of the magnitude of the updated cost coefficient) from the arcs out of the node. All other code parameters were held constant in all of the tests. The results of testing on the first 14 test problems can be found in Tables II and III. Regardless of pivot criteria, the exhaustive pass SSM procedure is predominately the best start method in terms of resultant total solution time. It provides a reasonable trade-off between the time spent selecting an initial basis and the time recovered from a reduced number of pivots. As compared to the artificial start procedure, there are cases in which the exhaustive pass SSM method reduces total pivots by as much as 61% and total solution time by as much as 55%.

Table II

Comparison of Start Procedures*

(Row First Negative Pivot Criteria)

	Artificial			1 Pass SSM			2 Pass SSM			3 Pass SSM			5 Pass SSM			Exhaustive Pass SSM			Exhaustive Node Supply		
Problem	Total	Start	Pivots	Total	Start	Pivots	Total	Start	Pivots	Total	Start	Pivots	Total	Start	Pivots	Total	Start	Pivots	Total	Start	Pivots
1	19.98	.02	2839	19.81	.05	2531	15.62	.08	1758	15.25	.08	1673	12.78	.10	1646	12.78	.09	1646	15.80	.10	1934
2	25.96	.02	3623	19.64	.06	2584	16.72	.10	2204	15.23	.10	1947	15.03	.11	1751	11.59	.12	1417	15.58	.11	1694
3	10.25	.02	1509	9.32	.03	1371	10.22	.05	1526	9.51	.07	1436	10.54	.11	1515	5.94	.28	706	5.63	.23	642
4	26.67	.02	4800	30.54	.03	4870	27.51	.05	4425	24.24	.07	3421	25.34	.10	3917	22.60	.12	3508	24.36	.10	3627
5	30.24	.03	3699	27.65	.08	3057	26.24	.08	2773	21.72	.11	2326	25.15	.11	2704	24.04	.11	2612	25.36	.11	2998
6	18.55	.03	2259	18.97	.06	2212	17.92	.08	2117	19.96	.11	2373	19.26	.16	2238	11.34	1.66	1104	11.62	1.51	1166
7	47.99	.03	5511	37.86	.09	4032	32.11	.10	3543	28.68	.13	3094	26.17	.15	2733	27.98	.16	2856	26.86	.16	2845
8	27.30	.03	4649	24.79	.03	4273	22.37	.05	3913	24.11	.06	4276	19.78	.06	3472	15.86	.14	2752	17.74	.05	3096
9	25.75	.01	4491	25.90	.03	4472	20.75	.03	3691	25.38	.03	4106	21.61	.03	3821	16.29	.13	2935	23.64	.03	4099
10	46.24	.05	5708	35.69	.08	3364	31.34	.13	3007	32.24	.13	2757	31.66	.14	2789	31.97	.13	2789	32.82	.13	2902
11	15.34	.03	1656	15.42	.10	1654	16.90	.04	1800	17.26	.08	1818	17.18	.13	1814	17.45	1.73	1660	17.99	1.44	1725
12	36.06	.03	4921	33.40	.04	4609	31.52	.04	4534	29.13	.06	4056	30.21	.08	4118	23.52	.19	3068	30.83	.06	4130
13	98.84	.03	11920	89.81	.09	11277	83.51	.06	10640	82.79	.08	10067	56.75	.13	7247	52.92	.28	5877	78.17	.10	9591
14	98.93	.03	12870	109.91	.06	13062	91.15	.08	11827	74.25	.10	9545	71.36	.16	8975	59.49	.30	7081	71.14	.12	9199

* Times listed are in CPU seconds.

Table III

Comparison of Start Procedures*

(Row West Negative Pivot Criterion)

Problem	Artificial			1 Pass SSM			2 Pass SSM			3 Pass SSM			5 Pass SSM			99 Pass SSM			Exhaustive Row Supply		
	Total	Start	Pivots	Total	Start	Pivots	Total	Start	Pivots	Total	Start	Pivots	Total	Start	Pivots	Total	Start	Pivots	Total	Start	Pivots
1	9.65	.02	1258	7.77	.07	966	7.32	.09	876	7.44	.08	881	6.60	.12	753	6.65	.11	753	6.46	.08	759
2	12.45	.02	1492	10.48	.06	1229	9.35	.08	1119	8.69	.10	862	6.93	.11	764	7.43	.13	806	7.35	.11	818
3	7.56	.02	729	8.11	.03	761	8.32	.05	768	7.41	.08	693	7.94	.10	744	4.30	.28	356	4.27	.07	369
4	19.98	.02	2541	20.25	.02	2364	20.41	.05	2402	21.61	.07	2543	22.43	.08	2569	23.01	.13	2691	12.66	.15	1701
5	15.44	.03	1780	14.05	.08	1501	14.66	.09	1523	11.89	.10	1212	10.70	.11	1110	12.91	.12	1523	14.52	.11	1474
6	9.58	.02	756	9.75	.07	753	9.77	.08	756	9.39	.11	724	9.35	.16	717	6.25	1.66	326	4.22	1.51	281
7	21.29	.03	2335	18.72	.08	1838	16.05	.11	1588	15.83	.13	1576	17.36	.16	1608	17.57	.16	1667	18.03	.15	1673
8	19.61	.02	2680	15.54	.03	2640	16.46	.03	2449	13.49	.05	1976	12.72	.07	1953	11.98	.13	1701	15.48	.05	2339
9	16.33	.03	2483	18.19	.02	2716	18.21	.03	2575	17.98	.05	2622	18.94	.03	2811	15.65	.11	2267	16.45	.03	2422
10	15.63	.03	2343	21.43	.10	1815	16.55	.12	1474	20.47	.13	1611	17.20	.15	1462	17.09	.15	1462	20.26	.13	1661
11	9.18	.03	549	8.70	.05	573	8.76	.08	575	9.22	.10	598	8.86	.13	576	11.54	1.20	565	11.18	1.45	563
12	22.59	.03	2885	21.13	.05	2759	21.67	.06	2762	17.47	.07	2173	17.84	.08	2222	14.71	.18	1866	19.95	.07	2519
13	63.31	.02	7300	68.28	.08	7288	58.70	.07	6463	53.91	.12	5889	47.26	.12	5160	41.56	.28	4564	50.18	.10	5601
14	62.97	.03	7014	56.83	.05	6508	57.90	.08	6469	50.77	.09	5801	42.76	.15	4820	36.95	.29	4041	47.53	.11	5424

* Times listed are in CPU seconds.

Pivot Selection Criteria

Tables II and III can also be used to perform an initial analysis of pivot selection criteria. It should be noted that the total solution times in Table III strictly dominate those in Table II. The obvious indication is that it is worthwhile to spend time searching for the "best" pivot eligible arc. Selecting the "best" out of a single node, as the most negative method does, can reduce total solution time by as much as 48%. For this reason, additional testing was conducted to try to find the best pivot selection criteria.

An obvious choice for a pivot selection strategy is the candidate list method. An S-R candidate list procedure employs an array of length R. This list contains the pointers to pivot eligible arcs selected by using the node most negative procedure R successive times. After each of S pivots, the best arc that is still pivot eligible in the list is selected to enter the basis. If there are no eligible arcs on the list or if the list has been used S times, the list is refilled by calling the node most negative procedure R more times.

A number of variations of this method have been computationally tested. The first, G1, employs a 1-1 candidate list. This is equivalent to the node most negative procedure and is included as a comparative device. G2 initially uses an 8-16 list. However, if the list cannot be filled, the size of the list is set to the number found, and S is set equal to 1/2R. G3 is similar to G2 except that if the list cannot be filled, it is reduced in size to a 1-1 list. The fourth version searches through a maximum of 100 nodes to fill an 8-16 list. If the list cannot be filled, R is set to the number found and S = 1/2R. G5 is the same as G4 except that 200 nodes are searched before the list size is altered.

These five codes were tested using the 1000 node problems (15-18). A 2-pass sequential source minimum start was used and all other parameters were held constant.

The results are shown in Table IV. It is evident that G2 and G3 clearly dominate the other codes, with G2 having a slight edge over G3. Note that by employing a candidate list, total solution time was reduced by as much as 40% over a simple node negative criteria. This test indicates that candidate lists should be used, but it does not show the best initial list size. This determination will be made subsequently.

Flow Update Procedures

The initial version of the computer code, G2, often performs wasted operations in the process of updating the flows on basic arcs. First, it has no check routines for identifying a degenerate pivot during the calculation of a representation. Thus, unnecessary representation components are found and flows are modified by a zero amount. G6 was designed to eliminate this problem. It identifies a degenerate pivot and totally skips the flow update procedures whenever possible. Another version, G7, was created to allow for storing the representation vector. Both G2 and G6 recalculate the representation during flow updates. G7 maintains an extra array to store this information.

A comparison of G2, G6, and G7 on the 1000 node problems is given in Table V. All three versions used a 2-pass sequential source minimum start, an 8-16 candidate list, and all other parameters were held constant. It is obviously advantageous to check for degenerate pivots as G6 does. Although the total number of pivots is not necessarily reduced, the total solution time can be reduced by as much as 25%. Note that the number of pivots differs due to the fact that tolerances for zero values are used. Thus, G6 selects the first "zero" arc to leave the basis while G2 picks the "minimum zero" arc to leave the basis. However, the tests on G7 indicate that it is not advantageous to go one step further and keep a representation array. Apparently the overhead of maintaining this extra information is not offset by a reduction in other calculations.

Table IV

Pivot Selection*

	G1		G2		G3		G4		G5	
Problem	Total Time	Total Pivots	Total Time	Total Pivots	Total Time	Total Pivots	Total Time	Total Pivots	Total Time	Total Pivots
15	96.75	3088	80.15	2374	70.37	2288	72.74	2286	72.65	2286
16	75.60	7514	45.35	4501	45.66	4504	55.66	5455	45.62	4513
17	63.12	7840	42.14	4950	42.51	4950	42.49	5039	42.26	4969
18	98.97	7654	64.73	4738	65.28	4771	91.86	7254	92.07	7254

*Times listed are in CPU seconds.

Table V

Flow Update Procedures*

	G2				G6				G7			
Problem	Total Time	Start Time	Total Pivots	Degenerate Pivots	Total Time	Start Time	Total Pivots	Degenerate Pivots	Total Time	Start Time	Total Pivots	Degenerate Pivots
15	80.15	.24	2374	0	80.11	.25	2374	0	80.36	.22	2374	0
16	45.35	.11	4501	3447	42.70	.10	5086	3904	44.53	.10	4997	3865
17	42.14	.11	4950	4426	31.33	.10	4824	4314	30.68	.10	4373	3885
18	64.73	.10	4738	2154	59.56	.10	4705	2070	65.49	.11	4802	2100

*Times listed are in CPU seconds.

Candidate List Parameters

An important consideration that was not resolved by the pivot criteria section is the determination of the proper candidate list parameters. This involves setting both the size of the list and the number of times the list is to be used before refilling it. Candidate list sizes of 1-1, 5-10, 8-16, and 15-25 have all been tested on the 1000 node problems using the G6 code. A five pass SSM start was employed, while parameters other than list size were held constant. A list size of 5-10 can be seen in Table VI to dominate all other strategies. It is interesting to note that a larger list size does not guarantee a reduction in the total number of pivots.

Pivot Tolerance

Tolerance levels are used in computer programming to avoid the problems caused by round-off error. These tolerances give ranges about values within which equality is assumed. Such tolerances must be used in the testing of updated cost coefficients. Negative coefficients indicate pivot eligible arcs; however, very small negative numbers may actually be equivalent to zero.

In order to find the best tolerance level, values of 0.000001, 0.01, 0.5, and 1.0 were tested. These results can be found in Table VII. The G6 code was used, employing a 5-10 candidate list and an exhaustive pass SSM start procedure. It is extremely interesting to note the varying effects that these tolerance levels have upon solution times. Yet all of these codes arrived at solutions with the same optimum objective function value. The value of the tolerance dramatically changes the choices of pivot eligible arcs. Extremely small values force the code to perform a series of disadvantageous pivots. On the other hand, large tolerances allow *the code to initially overlook* advantageous pivot selections. The best strategy is to select a moderate tolerance value of 0.01.

Table VI

Candidate List Size*

Problem	1-1		5-10		8-16		15-25	
	Total Time	Total Pivots	Total Time	Total Pivots	Total Time	Total Pivots	Total Time	Total Pivots
15	72.43	2316	79.87	2328	79.92	2328	80.01	2328
16	53.83	6172	35.64	3999	40.06	4260	41.99	4423
17	43.86	6423	30.59	4278	36.48	4850	37.24	5046
18	96.57	7364	55.74	4309	56.42	4098	55.96	4318

*Code G6 used.

Times listed are in CPU seconds.

Table VII

Pivot Tolerance*

	0.000001			0.01			0.5			1.0		
Problem	Total Time	Total Pivots	Degenerate Pivots	Total Time	Total Pivots	Degenerate Pivots	Total Time	Total Pivots	Degenerate Pivots	Total Time	Total Pivots	Degenerate Pivots
15	59.98	1479	0	56.79	1437	0	59.38	1472	0	56.57	1417	0
16	39.27	4095	3124	35.08	3699	2813	37.62	4027	3087	41.09	4281	3266
17	33.16	4296	4040	29.63	3977	3561	33.47	4265	3942	36.46	4667	4214
18	51.19	3607	1786	51.10	3571	1749	55.39	3799	1840	51.28	3537	1776

* Times listed are in CPU seconds.

Big-M Value

The final parameter value to be tested is the Big-M value. None of the codes developed implements a Phase I-Phase II procedure. Therefore, an exact value of Big-M must be selected. All of the test problems have cost ranges of 1-100. Experience with pure network computer codes indicates that Big-M should be set as small as possible (still insuring feasibility). The Big-M values were tested using the G6 code, with an exhaustive pass SSM start, a 5-10 candidate list, and a pivot tolerance of 0.000001. The test results are shown in Table VIII. Values of 10000, 2000, 1000, 500, 250, 150, and 100 were all tested. If 100 is eliminated because of resultant infeasibilities, a value of 150 clearly dominates all others in terms of total solution time. Note that in problem 16, the total solution time was reduced by over 42% simply by changing the Big-M value from 10000 to 150.

Breaking Pivot Ties

The last decision rule to be tested was the one for resolving ties in the test for a minimum ratio. The previously described codes have simply selected the most convenient (often the first encountered) minimum ratio. A new code version, G8, has been developed to test a heuristic rule for breaking pivot ties. The rule specifies that the ratio with the largest denominator be selected from among the set of tied ratios. Note that the resultant code cannot take advantage of degeneracy, and, therefore, is a modification of G1 and was tested against G1.

The results of testing G8 on all of the generated problems are shown in Table IX. In the majority of cases, G8 reduced the total number of pivots performed. However, this reduction was not enough to offset the increased number of operations involved with the pivot selection.

Table VIII

Big-M Values*

Problem	10,000			2,000			1,000			500			250			150			100		
	Total Time	Total Pivots	Degenerate Pivots	Total Time	Total Pivots	Degenerate Pivots	Total Time	Total Pivots	Degenerate Pivots	Total Time	Total Pivots	Degenerate Pivots	Total Time	Total Pivots	Degenerate Pivots	Total Time	Total Pivots	Degenerate Pivots	Total Time	Total Pivots	Degenerate Pivots
15	59.98	1479	0	54.76	1391	0	53.69	1369	0	53.90	1364	0	52.74	1332	0	51.05	1304	0	49.01	1270	0
16	39.27	4095	3124	39.24	3991	3030	39.75	4172	3243	35.37	3582	2784	30.96	3414	2705	22.77	2764	2244	16.04[a]	2109	1730
17	55.16	4296	4040	30.36	4148	3744	29.90	4000	3632	30.48	4002	3588	27.25	3752	3392	29.60	3968	3576	27.65	3729	3413
18	51.19	3607	1786	54.20	3968	1888	47.72	3344	1638	45.95	3330	1678	46.99	3410	1688	41.99	3112	1512	38.07[b]	2865	1429

* Times listed are in CPU seconds.

[a] Infeasible; 5 basic artificials.

[b] Infeasible; 5 basic artificials.

Table IX

Breaking Pivot Ties*

Problem	G1		G8	
	Total Time	Total Pivots	Total Time	Total Pivots
1	6.65	753	6.87	753
2	7.43	806	8.53	890
3	4.70	356	3.43	225
4	23.01	2691	17.32	2044
5	12.91	1323	13.15	1279
6	6.25	326	5.57	326
7	17.57	1667	16.78	1521
8	11.98	1701	13.83	2107
9	15.65	2267	14.59	2219
10	17.09	1461	17.09	1481
11	11.54	565	12.04	564
12	14.71	1866	16.82	2149
13	41.56	4364	38.78	3999
14	36.95	4041	31.31	3578
15	59.40	1658	59.46	1658
16	77.80	7515	66.20	6868
17	61.68	6924	51.71	6872
18	86.66	5955	80.66	6051

* Times listed are in CPU seconds.

2.6 Code Comparisons

It is difficult to fully assess the efficiency of a method unless that method is placed in comparison with other widely accepted methods. For this reason, NETG, along with the previously determined decision rules, was compared with a number of well-known linear programming computer codes. These codes have been divided into two categories: 1) specialized network codes, and 2) standard linear programming computer packages.

The first comparison was against several specialized programs for solving pure network problems. Pure network problems are specializations of generalized networks in which all of the arc multipliers are 1. The pure network codes are all equipped to take advantage of the simplifications that arise from the unit multipliers. NETG, on the other hand, contains the cumbersome mechanisms for handling non-unit multipliers and quasi-tree structures. This comparison, therefore, indicates the relative "cost" of these additional mechanisms.

The specialized codes used in the comparison were ARC-II [5], PNET-I [19], I/O PNET-I [33], Bennington [7], Boeing, General Motors, SHARE [41], and SUPERK [4]. The Boeing and General Motors codes were developed by the respective corporations. All of these programs are coded in standard FORTRAN-IV. The tests were performed using the RUN compiler on a CDC 6600 computer. Each of these programs, with the exception of I/O PNET-I, maintains all instructions and data in central memory. I/O PNET-I uses problem data from external disk storage and is, therefore, capable of solving much larger problems than the other codes.

PNET-I, from which I/O PNET-I was formed, is one of the fastest known special purpose primal simplex codes. It employs a list structuring procedure which is more advanced than the one used by NETG (see [26]). ARC-II, which is faster than PNET-I, uses sophisticated primal labeling techniques [5]. SUPERK, Boeing, General Motors, and SHARE are all variations of the out-of-kilter algorithm.

The problem set for the comparison consists of 35 pure network problems generated by NETGEN [37]. The input specifications for these problems can be found in [37]. Problems 1-5 are 100 x 100 transportation problems; problems 6-10 are 150 x 150 transportation problems; problems 11-15 are 200 x 200 assignment problems; and problems 16-35 are 400-1500 node capacitated and uncapacitated transshipment problems.

The results of the tests are shown in Table X. It can be seen that NETG is approximately three times as slow as ARC-II which is by far the fastest of the group. This result gives some indication of the additional calculations required by a generalized network code over a pure network code. An interesting result is that the NETG times are roughly comparable with the SUPERK times. This is important because SUPERK is one of the fastest out-of-kilter implementations.

The second phase of comparison involved NETG and a state of the art linear programming code called APEX-III. APEX-III is maintained by CDC and is operational on all CDC 6000 series and CYBER-70 series computers. The purpose of this test was to indicate the advantages that specialized procedures have over standard approaches. However, it should be noted that APEX-III employs highly sophisticated techniques.

The two codes were tested using seven problems generated by NETGENG. NETGENG is capable of randomly generating generalized network problems of virtually any size. The input specifications for the selected problems can be found in Table XI. They range in size from a 50 x 50 generalized transportation problem to a 1000 node generalized transshipment problem.

To perform the comparison between NETG and APEX-III, a CDC CYBER-74 computer was used and NETG was compiled using the CDC FTN compiler. The results can be found in Table XII. The basis of comparison for these tests was a quantity called an SBU. Each procedure incurs SBU's based on the amount of CPU seconds used, I/O

Table X

Solution Times (in seconds)

Problems	NETG	ARC-II	I/O PNET-I	PNET-I	BENN	SUPERK	GM	SHARE	Boeing
1	2.11	.78	1.75	1.17	20.25	5.68	46.25	17.76	30.25
2	2.61	.89	1.96	1.35	24.36	6.47	63.30	21.34	21.59
3	3.44	1.01	2.13	1.74	34.56	6.87	105.72	26.16	31.47
4	3.52	.95	2.35	1.47	31.45	6.57	70.74	25.13	36.47
5	3.81	1.25	2.92	1.73	52.10	6.77	90.10	30.97	47.73
6	5.67	2.11	4.50	3.06	61.00	11.05	92.32	46.40	46.64
7	7.17	2.23	5.20	3.57	DNR	12.86	157.31	65.92	113.12
8	8.87	2.99	7.18	4.20	DNR	13.69	160.71	81.00	175.10
9	8.46	2.99	6.76	4.35	DNR	13.40	158.01	81.21	186.99
10	8.72	4.02	7.37	5.57	DNR	14.13	197.82	84.24	184.75
11	7.31	1.92	3.71	3.12	17.44	6.44	35.67	19.93	30.39
12	8.32	2.36	7.75	4.48	20.31	6.47	28.43	21.17	22.08
13	7.67	3.13	8.58	4.91	24.92	7.25	31.39	25.81	20.02
14	9.55	2.96	8.46	5.56	27.40	6.95	18.62	24.95	23.11
15	10.97	3.12	8.60	5.91	DNR	7.56	23.48	27.05	21.08
16	3.71	1.38	2.60	2.15	11.77	5.27	60.27	21.51	15.05
17	7.49	1.87	3.65	2.90	20.10	8.36	96.66	32/40	64.64
18	4.44	1.26	2.65	1.70	11.31	5.13	61.54	20.06	18.31
19	5.98	1.72	3.61	2.40	20.62	8.49	DNR	31.75	61.07
20	5.12	1.28	2.47	2.47	10.38	4.69	DNR	18.11	25.72
21	7.23	1.83	4.19	2.46	20.35	7.96	DNR	32.60	61.39
22	4.61	1.26	2.60	2.01	9.97	4.60	DNR	17.91	24.84
23	7.12	1.67	4.08	2.74	19.81	7.91	DNR	32.66	67.96
24	6.12	1.52	3.60	2.91	11.71	5.59	DNR	25.27	21.57
25	6.90	1.83	5.49	3.96	18.27	8.37	DNR	33.19	48.40
26	4.99	1.08	2.66	4.15	11.38	5.51	DNR	25.05	19.34
27	6.88	1.62	4.03	4.31	16.37	7.50	DNR	30.45	41.98
28	15.11	4.40	10.34	5.67	DNR	13.91	DNR	53.87	83.98
29	18.72	4.87	12.51	6.55	DNR	14.51	DNR	52.55	117.83
30	17.77	4.88	15.32	8.10	DNR	16.00	DNR	61.33	152.21
31	18.26	5.68	13.45	8.48	DNR	17.05	DNR	61.33	135.73
32	29.77	7.42	20.33	13.59	DNR	22.88	DNR	78.63	553.93
33	26.64	7.82	20.17	17.65	DNR	25.89	DNR	101.92	210.14
34	32.24	8.21	19.42	14.86	DNR	25.42	DNR	92.25	248.16
35	37.47	8.81	26.16	17.13	DNR	29.96	DNR	DNR	DNR

NA - Code and data would not fit in 104,000 words of memory.

DNR - Did not run.

Table XI

Problem Specifications

Problem	Nodes	Sources	Sinks	Arcs	Multiplier Range		Cost Range		Upper Bound Range		Transshipment		Total Supply	Percent High Cost	Random Number Seed
					Min	Max	Min	Max	Min	Max	Sources	Sinks			
1	100	50	50	1000	.5-1.5		1-100		--		0	0	100000	0	13502460
2	100	5	50	1000	.5-1.5		1-100		--		2	10	100000	0	13502460
3	100	5	50	1000	.5-1.5		1-100		100-1000		2	10	100000	0	13502460
4	250	125	125	4000	.5-1.5		1-100		--		0	0	100000	0	13502460
5	250	15	75	4000	.5-1.5		1-100		--		5	15	100000	0	13502460
6	500	50	200	5000	.3-1.7		1-100		--		10	50	100000	0	13502460
7	1000	200	500	6000	.2-1.4		1-100		--		25	75	100000	0	13502460

Table XII

NETG vs. APEX-III

Problem	NETG		APEX-III	
	SBU's[a]	Cost[b]	SBU's	Cost
1	7.51	$1.35	62.65	$ 11.28
2	7.29	$1.31	80.93	$ 14.57
3	9.70	$1.75	94.72	$ 17.05
4	16.65	$3.00	453.02	$ 81.54
5	14.74	$2.65	742.61	$133.67
6	22.55	$4.06	1044.34	$187.98[c]
7	50.22	$9.04	1633.64	$294.06[d]

[a]CYBER-74 System Billing Unit.

[b]Computed at $0.18 per SBU.

[c]Stopped after 10,000 iterations.
Objective Function Value = 25,337,282.
Optimal Objective Function Value = 3,354,927.

[d]Stopped after 10,000 iterations.
Objective Function Value = 1,340,958,349.
Optimal Objective Function Value = 3,964,490.

operations performed, and central memory used. In this way, SBU's may be used to compute a total cost for the job. Cost figures have been included, based on the lowest CDC price per SBU, $0.18.

The results are quite remarkable, especially when the dollar charges are compared. NETG is in some cases more than 50 times more efficient than APEX-III. Problems 6 and 7 had to be prematurely terminated on APEX-III (after 10,000 iterations) due to the exhorbitant processing costs involved.

REFERENCES

1. H. Aashtiani and T. Magnanti, "Implementing Primal-Dual Network Flow Algorithms," Working Paper OR 055-76, Massachusetts Institute of Technology, 1976.

2. V. Balachandran, "An Integer Generalized Transportation Model for Optimal Job Assignment in Computer Networks," *Operations Research*, 24, 4 (1976), 742-759.

3. E. Balas and P. Ivanescu (Hammer), "On the Generalized Transportation Problem," *Management Science*, 1 (1964), 188-202.

4. R. Barr, F. Glover, and D. Klingman, "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," *Mathematical Programming*, 7, 1 (1974), 60-87.

5. 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.

6. 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.

7. G. Bennington, "An Efficient Minimal Cost Flow Algorithm," O. R. Report 75, North Carolina State University, Raleigh, North Carolina, 1972.

8. G. Bhaumik, *Optimum Operating Policies of a Water Distribution System with Losses*, Unpublished Dissertation, University of Texas at Austin, August, 1973.

9. G. Bhaumik and P. Jenson, "A Computationally Efficient Algorithm for the Network with Gains Problem," Working Paper, Department of Mechanical Engineering, University of Texas at Austin, 1974.

10. G. Bradley, "Survey of Deterministic Networks," *AIIE Transactions*, 7, 3 (1975), 222-234.

11. 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.

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

13. R. Crum, "Cash Management in the Multinational Firm: A Constrained Generalized Network Approach," Working Paper, The University of Florida, Gainesville, Florida, 1976.

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

15. K. Eisemann, "The Generalized Stepping Stone Method for the Machine Loading Model," *Management Science*, 11, 1 (1964), 154-177.

16.

17. G. Gilliam and J. Turner, "A Profile Analysis Network Model to Reduce the Size of Microdata Files," Working Paper, Office of Tax Analysis, Office of the Secretary of the Treasury, Washington, D.C., 1974.

18. 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.

19. 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.

20. F. Glover, D. Karney, D. Klingman, and A. Napier, "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," *Management Science*, 20, 5 (1974), 793-819.

21. F. Glover and D. Klingman, "On the Equivalence of Some Generalized Network Problems to Pure Network Problems," *Mathematical Programming*, 4, 3 (1973), 351-361.

22. F. Glover and D. Klingman, "A Note on Computational Simplifications in Solving Generalized Transportation Problems," *Transportation Science*, 7, 4 (1973), 351-361.

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

24. F. Glover, D. Klingman, and J. Stutz, "Extensions of the Augmented Predecessor Index Method to Generalized Network Problems," *Transportation Science*, 7, 4 (1973), 377-384.

25. F. Glover, D. Klingman, and J. Stutz, "Implementation and Computational Study of a Generalized Network Code," Presented at the 44th National ORSA Conference, San Diego, California, 1973.

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

27. 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.

28. B. Harris, "A Code for the Transportation Problem of Linear Programming," *JACM*, 23, 1 (1976), 155-157.

29. J. Hultz, *Algorithms and Applications for Generalized Networks*, Unpublished Dissertation, University of Texas at Austin, 1976.

30. W. Jewell, "Optimal Flow Through Networks with Gains," *Operations Research*, 10, 4 (1962), 476-499.

31. E. Johnson, "Programming in Networks and Graphs," ORC Report 65-1, University of California at Berkeley, 1965.

32. E. Johnson, "Networks and Basic Solutions," *Operations Research*, 14, 4 (1966), 619-623.

33. D. Karney and D. Klingman, "Implementation and Computational Study on an In-Core Out-of-Core Primal Network Code," *Operations Research*, 24, (1976).

34. P. Kazemersky, *A Computer Code for Refueling and Energy Scheduling Containing an Evaluator of Nuclear Decisions for Operation*, Unpublished Dissertation, Ohio State University, 1974.

35. Y. Kim, "An Optimal Computational Approach to the Analysis of a Generalized Network of Copper Refining Process," Presented at the Joint ORSA/TIMS/AIIE Conference, Atlantic City, New Jersey, 1972.

36. D. Klingman, A. Napier, and G. Ross, "A Computational Study of the Effects of Problem Dimensions on Solution Times for Transportation Problems," *JACM*, 22, 3 (1975), 413-424.

37. 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, 5 (1974), 814-821.

38. R. Langley, *Continuous and Integer Generalized Flow Problems*, Unpublished Dissertation, Georgia Institute of Technology, 1973.

39. R. Langley, J. Kennington, and C. Shetty, "Computational Devices for the Capacitated Transportation Problem," *Naval Research Logistics Quarterly*, 21, 4 (1974), 637-647.

40. J. Maurras, "Optimization of the Flow Through Networks with Gains," *Mathematical Programming*, 3, (1972), 135-144.

41. "Out-of-Kilter Network Routine," SHARE Distribution 3536, SHARE Distribution Agency, Hawthorne, New York, 1967.

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

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

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
A New Computer-Based Planning Tool

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)
J. Hultz J. Stutz
F. Glover
D. Klingman

6. REPORT DATE
December 1976

7a. TOTAL NO. OF PAGES
48

7b. NO. OF REFS
44

8a. CONTRACT OR GRANT NO.
N00014-75-C-0616;0569 and N00014-76-C-0383

b. PROJECT NO.
NR047-021

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
Center for Cybernetic Studies
Research Report CCS 289

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

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, D.C.

13. ABSTRACT

The purpose of this paper is to document this recent emergence of generalized networks as a fundamental computer-based planning tool and to demonstrate the power of the associated modeling and solution technologies when used in concert to solve real-world applications. To accomplish this, the paper is divided into two parts. Part I focuses on how generalized networks have been and may be used to model a diverse collection of significant practical problems. First we discuss (in nontechnical terms) the model structure of a generalized network (GN) and provide a brief historical survey of applications which have been modeled as GN problems. Next we present somewhat newer modeling techniques which draw heavily on generalized networks as a major component. The pictorial aspect of these techniques has proven to be extremely valuable in both communicating and refining nonlinear and combinatorial relationships.

Part II discusses the design and analysis of large-scale generalized network computer solution techniques. The central ideas that have brought GN problems into their own as a fundamental planning tool, by providing an effective methodology for computer implementations are clearly outlined. Part II further contains an in-depth computational study of solution strategies for GN problems within the framework of specializations of the primal simplex method. The study identifies an efficient solution procedure that derives from an integrated system of start, pivot, and degeneracy rules. The resulting method is shown on large problems

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Unclassified

Security Classification

A-31408

43. L. Tavis, R. Crum, and D. Klingman, "Implementation of Large-Scale Financial Planning Models: Solution Efficient Transformations," Research Report CCS 267, Center for Cybernetic Studies, University of Texas at Austin, 1976.

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

13. Abstract (Continued)

to be at least 50 times faster than the sophisticated state-of-the-art LP system, APEX-III. Finally, the study demonstrates that the memory requirements of the method, as well as its solution times, are sufficiently small to warrant its use as a computer-based planning tool not only in a batch processing environment, but also in an interactive environment.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Linear Programming						
Networks						
Generalized Networks						
Modeling						
Transshipment						
Graphs						
Computer Based Planning