AN ABSTRACT OF THE THESIS OF

Maryam Bolouri for the degree of Doctor of Philosophy in Statistics presented on July 27, 1989.

Title: Network Models with Generalized Upper Bound Side Constraints

Redacted for privacy

Abstract approved: [illegible]
Jeffrey L. Arthur

The objective of this thesis is to develop and computationally test a new algorithm for the class of network models with generalized upper bound (GUB) side constraints. Various algorithms have been developed to solve the network with arbitrary side constraints problem; however, no algorithm that exploits the special structure of the GUB side constraints previously existed. The proposed algorithm solves the network with GUB side constraints problem using two sequences of problems. One sequence yields a lower bound on the optimal value for the problem by using a Lagrangean relaxation based on relaxing copies of some subset of the original variables. This is achieved by first solving a pure network subproblem and then solving a set of single constraint bounded variable linear programs. Because only the cost coefficients change from one pure network subproblem to another, the optimal solution for one subproblem is at least feasible, if not optimal, for the next pure network subproblem. The second sequence yields an upper bound on the optimal value by using a decomposition of the problem based on changes in the capacity vector. Solving for the decomposed problem corresponds to solving for pure network subproblems that have undergone changes in lower and/or upper bounds. Recently developed reoptimization techniques are incorporated in the algorithm to find an initial (artificial) feasible solution to the pure network subproblem.

A program is developed for solving the network with GUB side constraints problem by using the relaxation and decomposition techniques. The algorithm has been tested on problems with up to 200 nodes, 2000 arcs and 100 GUB constraints. Computational experience indicates that the upper bound procedure seems to perform well; however, the lower bound procedure has a fairly slow convergence rate. It also indicates that the lower bound step size, the initial lower bound value, and the lower and upper bound iteration strategies have a significant effect on the convergence rate of the lower bound algorithm.

Network Models with Generalized Upper Bound Side Constraints

by

Maryam Bolouri

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed July 27, 1989

Commencement June 1990

APPROVED:

Redacted for privacy

Associate Professor of Statistics in charge of major

Redacted for privacy

Chairman of Department of Statistics

Redacted for privacy

Dean of Graduate School

Date thesis is presented July 27, 1989

Typed by Maryam Bolouri

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Network Models with Generalized Upper Bound Side Constraints

CHAPTER 1

INTRODUCTION

Network flow models arise in a wide variety of applications such as production-distribution systems, communications systems, and pipe network systems. All the activities, other than slacks and surpluses, can be presented as arcs connecting pairs of nodes in a network. The values of the variables associated with these activities can be interpreted as flows in the network and the constraints as flow balance equations. The arcs may represent pipes in a water distribution network, telephone lines in a communication network, etc.; and the nodes may be interpreted as locations or terminals connected by the arcs.

Unfortunately, many real world models do not possess pure network structure. Often additional linear constraints are essential to model crucial policy restrictions. These additional constraints are generally referred to as side constraints. There are numerous applications of the network with side constraints model arising in practical settings. Applications utilizing several types of side constraints models are briefly described below.

1. Glover, Glover, Lorenzo and McMillan (1982) developed a model for Frontier Airlines whose goal was setting prices more adaptively and changing them more rapidly, i.e., to determine the number of passengers at each fare class on each flight segment that will optimize revenue for any given set of prices, flight segment capacities and passenger carrying demand. The system is designed to accommodate a network of 600 flights and 30000 passenger itineraries (PI) with up to 5 fare classes per PI. The number of side constraints ranges from 1800 to 2400.

2. Klingman, Mote and Phillips (1988) developed an optimization–based logistics model for W. R. Grace. W. R. Grace company is one of the nation's largest suppliers of phosphate–based chemical products such as fertilizer. The mathematical model contains 12 monthly time periods and is used primarily for annual planning purposes. The system is designed to provide management with important variable cost and material balance information so as to minimize the sum over all periods of shipping costs, production costs, and

inventory costs. It typically has 3408 nodes, 21504 arcs and 288 side constraints.

3. Ali, Helgason and Klingman (1987) developed an integrated man–machine decision support system for the U.S. Air Force whose goal is to select the least cost set of cargo–routes which satisfy the point–to–point demands for cargo movement among the 60 Air Force bases. The system is designed to aid the Air Force Logistics Command in making annual design changes in route structure for a large routing and distribution system. It is also designed to accomodate a network of 60 bases with up to 313 flights. The number of side constraints ranges from 234 to 310.

In a network without additional constraints there are of course still the conservation equations; in other words total flow into a node must equal the sum of total flow out of that node and the node requirement. The conservation equations are handled graphically by network solution procedures. Bounds specified for individual arcs or variables are handled implicitly in a manner analogous to the bounded–variable simplex method for linear programs.

1.1 Formal Definition

The general problem to be considered may be defined as a mathematical program with the following form:

$$\begin{array}{rl} \text{Minimize} & cx + dy \\ \text{Subject to} & Ax \qquad = r \\ & Ex + Py = b \\ & l(x) \leq x \leq u(x) \\ & l(y) \leq y \leq u(y) \end{array}$$

In this formulation, A is an (mXn) matrix, E is a (pXn) matrix and P is a (pXq) matrix. The r is a (mX1) vector; x, l(x) and u(x) are (nX1) vectors; b is a (pX1) vector; y, l(y) and u(y) are (qX1) vectors; c and d are respectively (1Xn) and (1Xq) vectors.

A major portion of the LP literature has been devoted to the following problems:

(a) Standard LP Problems (m=n=0 and P is an arbitrary matrix), that is,

$$\begin{array}{rl} \text{Minimize} & dy \\ \text{Subject to} & Py = b \\ & l(y) \leq y \leq u(y) \end{array}$$

Problems having inequality constraints may be placed in this form via the addition of slack or surplus variables. The rules for accomplishing this transformation can be found in any text on linear programming. [Lasdon (1970) and Murty (1976)]

(b) LP/GUB Problems (q=0, A is an arbitrary matrix and E contains at most one nonzero entry per column), that is,

$$\begin{aligned} \text{Minimize} \quad & cx \\ \text{Subject to} \quad & Ax = r \\ & Ex = b \\ & l(x) \leq x \leq u(x) \end{aligned}$$

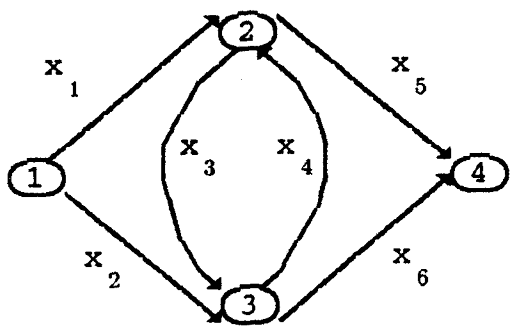
This problem arises from an LP in standard form in which the constraints fall into two sets. The first set of m constraints (i.e. Ax = r) is of an arbitrary nature. The last set of p constraints (i.e. Ex = b) is termed the generalized upper bound (GUB) constraints, i.e., they satisfy the property that every variable in the model appears in at most one constraint in this set of constraints. [Murty (1983, pp. 359-368)]

(c) Pure Network Problems (p=q=0 and A is a node-arc incidence matrix), that is,

$$\begin{aligned} \text{Minimize} \quad & cx \\ \text{Subject to} \quad & Ax = r \\ & l(x) \leq x \leq u(x) \end{aligned}$$

The matrix A is defined to be a node-arc incidence matrix, that is, a matrix where each column has exactly two nonzero entries, one being a +1 and the other a -1. The rows of the node-arc incidence matrix correspond to the nodes of the network and the columns to the arcs. The convention used here is that if arc k is directed from node i to node j then row i will contain a -1 and row j a +1 in column k of A. In such cases nodes i and j are referred to respectively as the from-node and the to-node for arc k. For example, Figure 1.1 presents a simple network and its associated node-arc incidence matrix. The vector r defines the requirements at the various nodes. For our convention, supply nodes have a negative value in r, demand nodes a positive value in r and transshipment nodes a zero r value. In addition, r is such that the sum of its components is zero, that is, total supply equals total demand. The vector x of decision variables corresponds to the flows across the arcs. The l and u vectors represent the lower and upper bounds respectively that are placed on x with $-\infty < l < u < \infty$. Many special cases of pure network problems have been studied. Some of these and their specialization are:

1. Uncapacitated Transshipment Problem. This problem is a specialization of the pure network in which the arcs have infinite capacity ($u_j = \infty$ for all



$$A = \begin{array}{c|cccccc|} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 1 & -1 & -1 & & & & \\ 2 & 1 & & -1 & 1 & -1 & \\ 3 & & 1 & 1 & -1 & & -1 \\ 4 & & & & & 1 & 1 \end{array}$$

Figure 1.1 : A Network and its Associated Node-Arc Incidence Matrix

j).

2. Capacitated Transportation Problem. This is a special case of the pure network problem in which the nodes can be partitioned into two sets, one consisting solely of supply points and the other only of demand points, such that all arcs originate from supply nodes and terminate in demand nodes. Figure 1.2 illustrates a typical structure for a capacitated transportation problem.
3. Transportation Problem. This problem is a special case of the capacitated transportation problem in which the arcs have infinite capacity.
4. Assignment Problem. This is a special case of the transportation problem in which the number of supply nodes is equal to the number of demand nodes and all demands and supplies are unity (i.e., $|r_i| = 1$ for all i).
5. Shortest Path Problem. Given a network whose arc cost is given the physical interpretation of arc length, the shortest path problem is to find that sequence of arcs connecting node s to node t such that the sum of the arc costs on the path is minimized, where s and t may be any given node pair. This problem can be viewed as a special

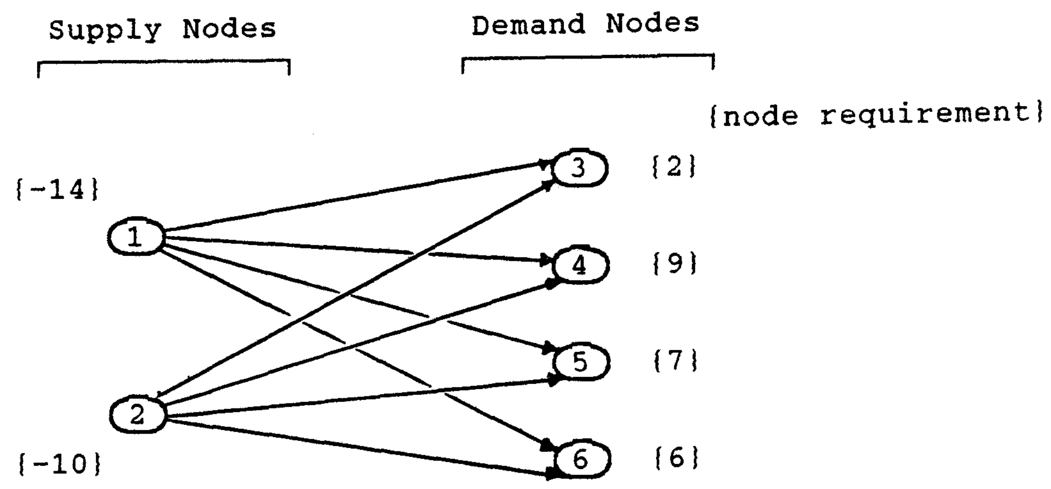


Figure 1.2 : Transportation Problem

case of the uncapacitated transshipment problem in which $r_s = -1$, $r_t = 1$, and all other requirements are zero. Suppose we wish to find the shortest path from node 1 to node 4 in the network of Figure 1.1. The network corresponding to this problem is given in Figure 1.3. The optimal flow pattern in Figure 1.3 implies that the shortest path consists of arcs (1,2), (2,3) and (3,4).

6. Maximal Flow Problem. For this problem arc capacities ($l_j \leq x_j \leq u_j$ for all j) are the only relevant parameters. For any given node pair s and t, the problem is to find the maximal continuous flow from node s to node t. Suppose we wish to determine the maximal continuous flow from node 1 to node 4 in the network of Figure 1.1. The revised network is illustrated in Figure 1.4. Note that the value of the maximum flow is equal to 5.

(d) Multicommodity Network Problem (q=0, A is a block–diagonal matrix with each block a node–arc incidence matrix from one commodity and E is a special structured matrix), i.e.,

[cost,flow]
{node requirement}

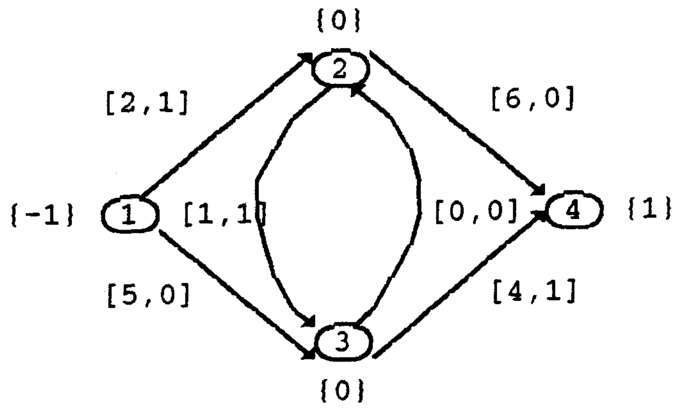


Figure 1.3 : Shortest Path Problem

[flow,upper bound,lower bound]

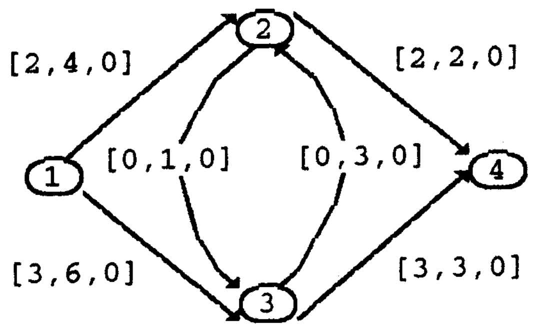


Figure 1.4 : Maximal Flow Problem

$$\begin{aligned} \text{Minimize} \quad & cx \\ \text{Subject to} \quad & Ax = r \\ & Ex = b \\ & l(x) \leq x \leq u(x) \end{aligned}$$

Problems of this type arise when a network of nodes and arcs is shared by several different items (commodities). When k commodities are present, E is of the special structure $[\, D^1 \mid D^2 \mid \ldots \mid D^k]$ where D^i for $i=1,\ldots,k$ are diagonal matrices. In many cases $D^i = I$ for $i=1,\ldots,k$. [Kennington and Helgason (1980, pp. 124-165)]

(e) Generalized Network Problems ($p=q=0$ and A contains at most two nonzero entries per column), i.e.,

$$\begin{aligned} \text{Minimize} \quad & cx \\ \text{Subject to} \quad & Ax = r \\ & l(x) \leq x \leq u(x) \end{aligned}$$

This is a special case of LP in which each column of the constraint matrix has at most two nonzero entries. In many practical applications the two nonzero entries in each column are of opposite sign. One may associate a graph with any generalized network problem. This graph consists of undirected arcs, in contrast to network graphs (see Figure 1.1).
[Kennington and Helgason (1980, pp. 91-123)]

(f) Network with Side Constraints Problem (A is a node-arc incidence matrix, and E and P are arbitrary matrices), i.e.

$$\begin{aligned} \text{Minimize} \quad & cx + dy \\ \text{Subject to} \quad & Ax = r \\ & Ex + Py = b \\ & l(x) \leq x \leq u(x) \\ & l(y) \leq y \leq u(y) \end{aligned}$$

This problem arises when in addition to the constraints on the flow in a network other linear constraints are present. [Kennington and Helgason (1980, pp. 166-182)]

(g) Network with GUB Constraints Problem (q=0, A is a node-arc incidence matrix and E contains at most one nonzero entry per column, i.e.

$$\begin{aligned} \text{Minimize} \quad & cx + dy \\ \text{Subject to} \quad & Ax = r \\ & Ex \leq b \\ & l(x) \leq x \leq u(x) \end{aligned}$$

(Note that by adding the vector of slacks y and the matrix P=I, this is of the form of the general category of problems considered here.) This is a special case of the network with side constraints problem in which the side constraints are generalized upper bounding (GUB) constraints. This is the problem of interest and an algorithm for solving this problem will be presented in Chapter 3.

1.2 Outline of the Remaining Chapters

The organization of the dissertation will be to first present a summary of the major results for relevant topics in optimization theory; this will be done in Chapter 2 along with a review of the literature. The basic methodology used to solve the network model with generalized upper bound constraints is explained in detail in Chapter 3. Chapter 4 presents an overview of software considerations. Chapter 5 presents our computational results along with a discussion of relevant findings. Finally, Chapter 6 provides a summary of the findings and some suggestions for further study.

CHAPTER 2
FOUNDATIONS

2.1 Background Results

The results in this section will draw from the work of Ali, Allen, Barr and Kennington (1986), Fisher (1981), Held, Wolfe and Crowder (1974), Kennington and Helgason (1980), Shapiro (1979), Wagner (1975) and Bolouri and Arthur (1989). The purpose of this section is to present a summary of the relevant results for (i) solving pure network flow problems via the simplex method on a graph, (ii) projecting an infeasible point onto a feasible region, (iii) the subgradient method for nondifferentiable optimization, (iv) the Lagrangean dual problem and (v) the single constraint, bounded variable linear programming (SCBVLP) problem. All of these ideas will be utilized in describing the solution of network models with generalized upper bound side constraints.

2.1.1 Pure Network Problems

The minimum cost network flow or pure network problem is a special structured linear program of the form:

$$\text{(NP)} \quad \begin{array}{ll} \text{minimize} & c\bar{x} \\ \text{subject to} & \\ & A\bar{x} = \bar{r} \\ & \bar{l} \leq \bar{x} \leq \bar{u} \end{array}$$

where A is a node–arc incidence matrix with m nodes and n arcs. Arc (i,j) is directed from node i to node j, and its flow, unit cost, lower bound, and upper bound are given, respectively, as $\bar{x}_{ij}$, c_{ij}, $\bar{l}_{ij}$ and $\bar{u}_{ij}$. The constant $\bar{r}_k$ represents the requirement at node k. The objective is to determine a set of flows which meet the node requirements and bound restrictions at a minimum total cost.

In practice, (NP) is transformed to yield a slightly simplified form with zero lower bounds. By defining $\bar{x} = x + \bar{l}$, the problem becomes:

$$\begin{array}{ll} \text{minimize} & cx + \alpha \\ \text{subject to} & \\ & Ax = r \\ & 0 \leq x \leq u \end{array}$$

where $\alpha = c\bar{l}$, $r = \bar{r} - A\bar{l}$, and $u = \bar{u} - \bar{l}$. It is this form of the problem that is typically implemented in computer codes, with the lower bounds maintained separately for reconstruction of problem (NP) upon solution of the transformed problem.

Since the system **Ax** = **r** has rank m-1 (see Kennington and Helgason (1980, p. 56)), an additional arc a, called the *root arc*, is added to problem (NP) to get :

$$(\mathcal{NP}) \quad \text{minimize} \quad \mathbf{cx} \qquad (1)$$

$$\text{subject to} \quad \mathbf{Ax} + a\mathbf{e}^{\ell} = \mathbf{r} \qquad (2)$$

$$0 \leq \mathbf{x} \leq \mathbf{u} \qquad (3)$$

$$0 \leq a \leq 0 \qquad (4)$$

where $\mathbf{e}^{\ell}$ is a vector with 1 in the ℓth position and zeros elsewhere, $1 \leq \ell \leq m$. ℓ is usually referred to as the *root node*. Then the constraint matrix $[\ \mathbf{A} \mid \mathbf{e}^{\ell}\ ]$ has full row rank. It has been shown (Kennington and Helgason (1980, p. 57)) that the only bases for $[\ \mathbf{A} \mid \mathbf{e}^{\ell}\ ]$ are $\mathbf{e}^{\ell}$ along with a set of linear independent columns from **A**.

2.1.1.1 Operations with the Network Basis

This section will introduce some notions from graph theory that will be used in the characterization of a basis for the problem $(\mathcal{NP})$. A *network* is a (directed) graph $\mathcal{G} = [\ \mathcal{N}, \mathcal{A}]$ where $\mathcal{N}$ is a finite set of nodes and $\mathcal{A}$ is a set of directed arcs joining pairs of nodes of $\mathcal{N}$. A *path* in $\mathcal{G}$ is an alternating sequence of distinct nodes and arcs such that each arc is incident to the two nodes immediately preceding it and following it. A path links its first element to its last element. A *cycle* in $\mathcal{T}$ is a path in $\mathcal{T}$ whose two endpoints are not distinct. A cycle links its first element to itself. If every pair of nodes in $\mathcal{T}$ is joined by a path then $\mathcal{T}$ is said to be *connected*. A graph which has no cycles is said to be *acyclic*. A *subgraph* of a graph $\mathcal{T}$ is a graph composed of a subset of the nodes and arcs of $\mathcal{T}$. A *spanning subgraph* of $\mathcal{T}$ is one which contains all the nodes in $\mathcal{T}$.

An important type of a graph is a tree. A *tree* is a connected acyclic graph. The following proposition gives other characterizations of a tree and will be stated without proof. (A proof can be found in Kennington and Helgason (1980, pp. 203-206).)

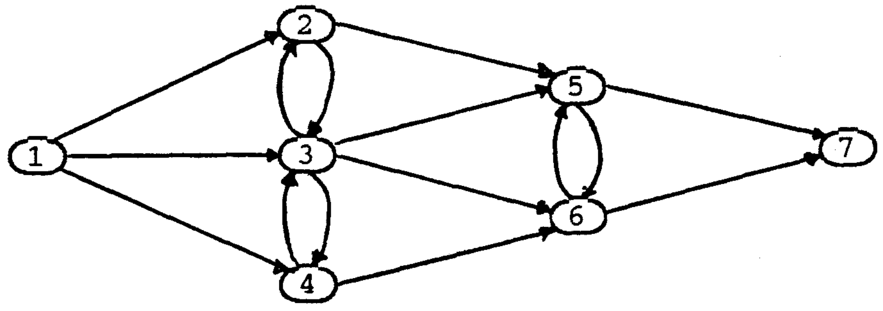
<u>Proposition</u>: The following statements are equivalent:

1. $\mathcal{T}$ is a tree.
2. Every distinct pair of points of $\mathcal{T}$ are joined by a unique path.
3. $\mathcal{T}$ is connected and the number of nodes is one more than the number of arcs.
4. $\mathcal{T}$ is acyclic and the number of nodes is one more than the number of arcs.

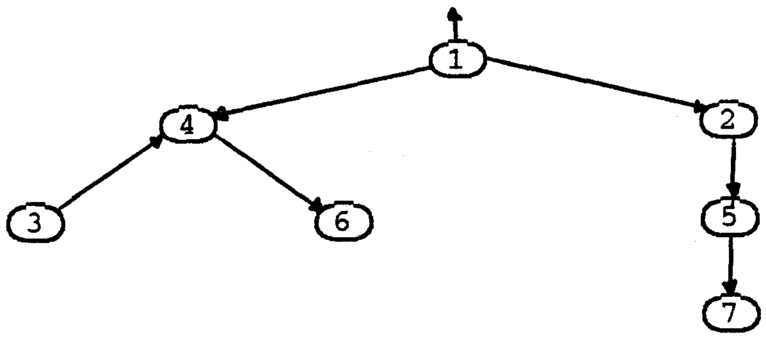
A tree that is a spanning subgraph of a graph $\mathcal{T}$ is called a *spanning tree* for $\mathcal{T}$.

Recall that a network basis has one additional column e^{ℓ} which is represented on a graph by a link leaving the root node ℓ and having no to-node. Furthermore, the corresponding graph is called a *rooted graph*, and a spanning subgraph of a rooted graph that is a tree is termed a *rooted spanning tree* for the rooted graph. It is well known that the set of all arcs that form a basis for a network, together with the set of all nodes in the network, form a rooted spanning tree (Kennington and Helgason (1980, pp. 58-59)). Figure 2.1(a) shows a network and Figure 2.1(b) a corresponding rooted spanning tree (basis tree). The representation of the network bases as rooted trees is one of the key reasons network optimization codes are so efficient. The remainder of this section will be devoted to a few illustrations that serve to point out how the computational savings occur.

Consider the rooted tree illustrated in Figure 2.1(b) with the corresponding basis (for clarification, node numbers are supplied next to the basis matrix):



a.) Example Network



b.) Rooted Spanning Tree Corresponding to a Basis for the Network

Figure 2.1 : Example Network and its Basis Tree

$$B = \begin{array}{c} 7 \\ 5 \\ 6 \\ 3 \\ 4 \\ 2 \\ 1 \end{array} \left[\begin{array}{ccccccc} 1 & & & & & & \\ -1 & 1 & & & & & \\ & & 1 & & & & \\ & & & -1 & & & \\ & & -1 & 1 & 1 & & \\ & -1 & & & & 1 & \\ & & & & -1 & -1 & -1 \end{array} \right] \tag{5}$$

Note that B has been triangularized by row and column interchanges. (An algorithm for triangularizing a network basis can be found in Kennington and Helgason (1980, p. 60)).

A representation of B is required for two types of calculations; premultiplication of B by a row vector and postmultiplication of B by a column vector. These calculations can be represented symbolically as

$$\boldsymbol{\pi} \mathbf{B} = \mathbf{c} \tag{6}$$

which is solved for $\boldsymbol{\pi}$, and

$$\mathbf{B}\mathbf{y} = \mathbf{d} \tag{7}$$

which is solved for $\mathbf{y}$, where $\boldsymbol{\pi}$, $\mathbf{c}$, $\mathbf{y}$ and $\mathbf{d}$ are appropriately dimensioned row and column vectors. The special structure of the matrix B greatly reduces the computational effort.

Consider solving (6) for $\boldsymbol{\pi}$ with the matrix B from (5) when

$$\mathbf{c} = [5,0,2,0,1,1,0] \tag{8}$$

This gives the following set of equations:

$$\begin{aligned}
\pi_7 - \pi_5 \qquad\qquad\qquad\qquad\qquad &= 5 \\
\pi_5 \qquad\qquad\qquad - \pi_2 \qquad &= 0 \\
\pi_6 \qquad - \pi_4 \qquad\qquad\quad &= 2 \\
- \pi_3 + \pi_4 \qquad\qquad\quad &= 1 \\
\pi_4 \qquad - \pi_1 &= 1 \\
\pi_2 - \pi_1 &= 1 \\
- \pi_1 &= 0
\end{aligned} \tag{9}$$

To solve for π, the value of the last component is obtained first and the values of the remaining components are iteratively obtained by backward substitution. The solution of (9) is

$$\pi = [6,1,3,1,1,1,0] \tag{10}$$

The above procedure may be used for solving any triangular system of equations but the calculation could be simplified by using a basis tree representation of B.

Recall that for the revised simplex method any equation for the reduced costs can be expressed as :

$$c^N - c^B B^{-1} A_N \tag{11}$$

where B is a basis, c^B is the cost associated with the basic columns, c^N is the cost associated with the nonbasic columns A_N and B^{-1} is the basis inverse. Again, one may make use of the triangularity of B and solve for

$\pi = c^B B^{-1}$, called the dual solution, or node potentials. The system of equations to be solved is

$$\pi = c^B B^{-1} \tag{12}$$

or

$$\pi B = c^B \tag{13}$$

where π is the row vector representing the duals or node potentials. Without loss of generality, assume that the rth column of B, B_r, corresponds to the arc (i,j). Then B_r has a value of -1 in the row corresponding to node i and a value of 1 in the row corresponding to node j, and zeros elsewhere. Then the rth equation of (13) is

$$\pi B_r = c_{ij}$$

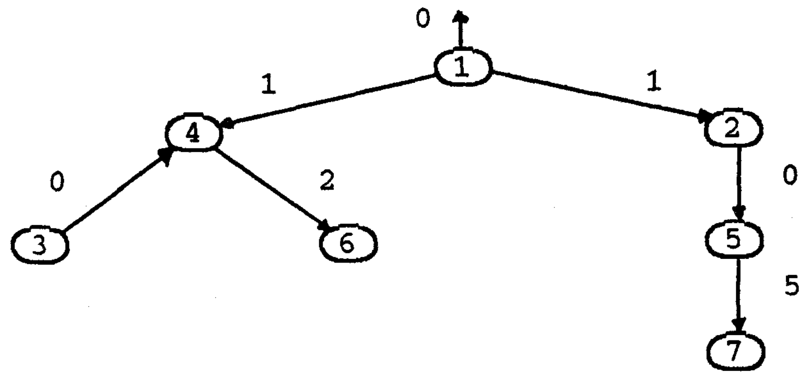
or equivalently,

$$-\pi_i + \pi_j = c_{ij} \tag{14}$$

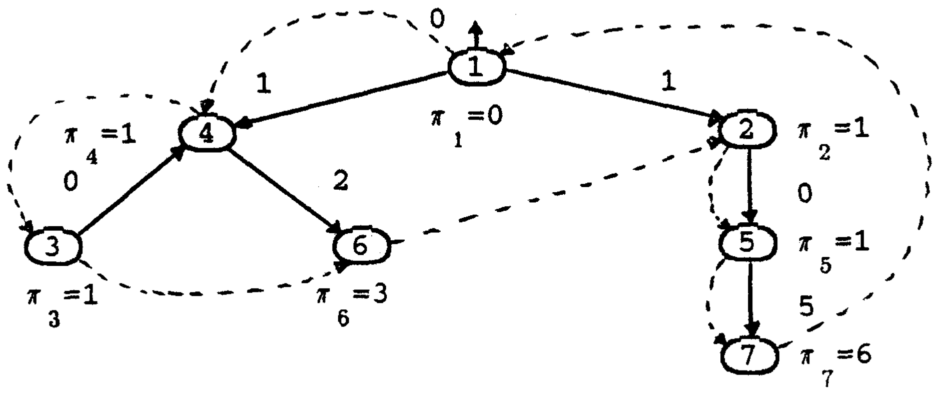
where c_{ij} is the cost associated with the arc (i,j). In general, for any basis B, and basis tree $\mathcal{T}_B$ with root node ℓ, (13) can be reduced to

$$\begin{aligned} \pi_\ell &= 0 \\ -\pi_i + \pi_j &= c_{ij} \qquad \text{for } (i,j) \in \mathcal{T}_B \end{aligned} \tag{15}$$

where the π values are associated with the nodes of the basis tree and the c_{ij} values are associated with the arcs. Figure 2.2(a) shows a basis tree with the set of cost coefficients from (8) on the arcs. Because the set of constraints for the network has rank one less than the



a.) Basis Tree with the Arc Costs Assigned



b.) Basis Tree with Node Potentials Determined

Figure 2.2 : Example Basis Tree

number of nodes, we can arbitrarily assign a value to any of the nodes. The convention is to assign a value of zero to the potential of the root node.

Consider node 4 first. In order to determine the potential for node 4, π_4, the following equation has to be solved

$$\pi_4 = c_{14} + \pi_1 \tag{16}$$

The value of c_{14} is equal to 1. The π_4 is equal to c_{14} or 1 since the value of π_1 is zero. To calculate the remaining node potentials the same operations are performed; each making use of potentials calculated in the previous step. In order to make sure the calculations are performed in the right order, a labelling scheme referred to as the *thread* function is used. The thread, written as t(x) where x is a node number, may be thought of as a thread which passes through each node exactly once in a top to bottom, left to right order starting from the root node. The thread function is shown in Figure 2.2(b) as a dashed line and specifies the order in which the nodes in the basis tree are visited, e.g. t(6)=2 and t(7)=1. It is now a simple matter to solve (13). Figure 2.2(b) shows the complete solution. An algorithm for solving (13) can be found in Kennington and Helgason (1980, p. 63).

Now consider solving (7) for y with a **d** vector of

$$\mathbf{d} = \begin{array}{c} 7 \\ 5 \\ 6 \\ 3 \\ 4 \\ 2 \\ 1 \end{array} \begin{bmatrix} 0 \\ -2 \\ 4 \\ -3 \\ 0 \\ 6 \\ 0 \end{bmatrix} \tag{17}$$

where again the corresponding node numbers are included. The equations to be solved are:

$$\begin{array}{rrrrrrrcr}
y_1 & & & & & & & = & 0 \\
-y_1 & +y_2 & & & & & & = & -2 \\
 & & y_3 & & & & & = & 4 \\
 & & & -y_4 & & & & = & -3 \\
 & & -y_3 & +y_4 & +y_5 & & & = & 0 \\
 & -y_2 & & & & +y_6 & & = & 6 \\
 & & & & -y_5 & -y_6 & -y_7 & = & 0
\end{array} \tag{18}$$

To solve this system of equations we use forward substitution starting with $y_1=0$. The solution to (18) is

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \\ 4 \\ 3 \\ 1 \\ 4 \\ -5 \end{bmatrix} \tag{19}$$

In order to use the basis tree to solve (18) we assign the d_i values to each node i in the basis tree and use them to determine values associated with the arcs. The chain specified by the dashed line in Figure 2.3 indicates the order in which the nodes are visited in determining the arc values. Since the order is opposite that of the thread function, this chain will be referred to as the *reverse thread* function, written as r(x) when x is a node number; for example, r(1)=7 and r(4)=1 in Figure 2.3. In order to implement the procedure it is also necessary to maintain a function that specifies for any given node its immediate predecessor. This function will be referred to as the *predecessor* function and written as p(x) when x is a node number where p(x)=0 if node x is the root node; e.g. p(3)=4.

To determine the arc values using the tree we begin at the node r(1), or node 7. If an arc is directed into a particular node it will receive the value on the node, if it is directed out of a node it will receive the negative of the value on the node. Then the value on the node

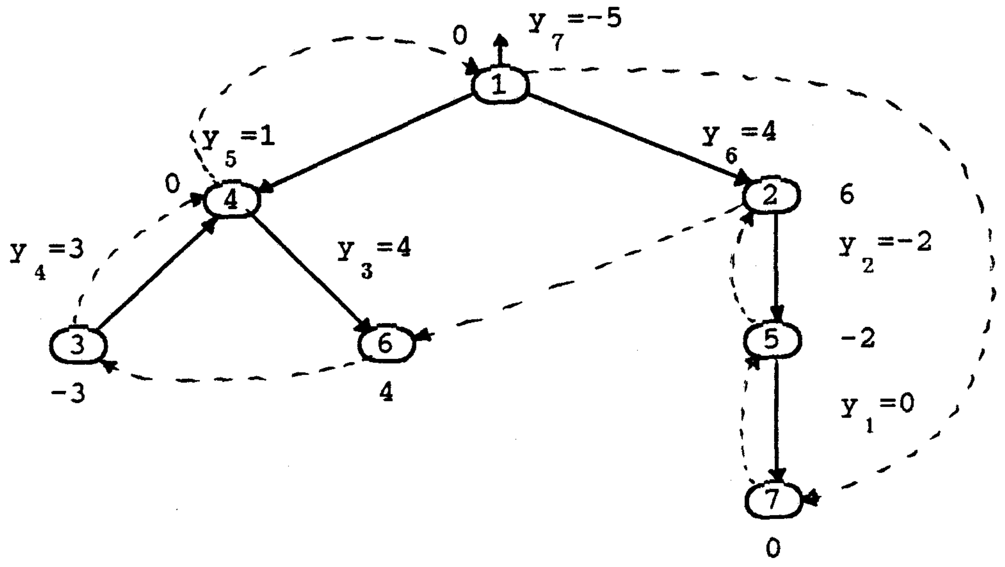


Figure 2.3 : Example Basis Tree

p(7), or node 5, is revised to be the old value on the node p(7) plus the value on the node 7. The same procedure is followed for r(7), the reverse thread of 7, until the predecessor of the current node is zero. For example, at node 6 the arc value, y_3, is equal to 4. Then the revised value on the predecessor of node 6, p(6)=4, would be 4 + 0 = 4. Then at node r(6)=3 the arc value, y_4, is equal to 3. The revised value on node p(3)=4 would be the old revised value on node 4 plus the value on node 3, that is, the value of the sum 4 + (-3) = 1. This value is used instead of 0 in determining the value to be assigned to the arc (1,4). Figure 2.3 shows the finished tree. Note that the operations performed are extremely simple, resulting in a very efficient method for solving systems of the form of (7). A general algorithm necessary to solve (7) is presented in Kennington and Helgason (1980, p. 171).

For completeness, the steps of the specialized primal simplex method on a graph are summarized as follows:

Step 0 : Determine Node Potentials.

Assume that an initial feasible basis (possibly containing artificial arcs) has been determined and stored as a rooted tree. Flows on the arcs have been determined as discussed above. The node potentials π_k for each node k are determined (on a graph) using the technique explained previously.

Step 1 : Identify the Outgoing and Incoming Arcs.

The basis exchange step of the simplex method selects an incoming arc and outgoing arc from the nonbasic and basic arcs respectively. The incoming arc is that particular nonbasic arc which is profitable to enter the basis. That is, it is a nonbasic arc that has zero flow and a negative reduced cost or has saturating (upper bound) flow and a positive reduced cost. If no such arc exists, the problem is solved. The outgoing arc, the arc to leave the basis, is an arc in the basis equivalent path (i.e. the unique path in the basis tree which connects the two nodes of the incoming arc) whose flow goes to zero or its upper bound sooner than any others as a result of a flow change in the incoming arc. The basis equivalent path can be determined by tracing the predecessors of the two nodes to their initial point of intersection.

Step 2 : Execute the Basis Exchange.

The outgoing and incoming arcs swap their basic and nonbasic statuses to become nonbasic and basic, respectively; the basis tree functions, basis flows and node potentials are then updated and the method returns to step 1 with a new feasible basis.

Consider Figure 2.4 and assume that link [4,12] has been selected to enter the basis. The basis equivalent path for link [4,12] is the set of links in the predecessor path of the basis tree from node 4 to node 2 (i.e. links [4,3] and [3,2]) and from node 12 to node 2 (i.e. links [12,11], [11,8], [8,6] and [6,2]). Node 2 is referred to as the *intersection node*.

Various data structures have been developed to facilitate implementation of the algorithm. All the data structures use the predecessor and thread functions plus various combinations of other functions. Each label or function used in the data structures requires a node–length array. Let T be the basis tree and T(x) be the subtree of T that is rooted at node x (hence the subtree that includes x and all its successors under the predecessor ordering). The following functions are widely used in the data structures.

p(x) = the predecessor of node x where p(x)=0 if x is a root node.

t(x) = the thread of x.

r(x) = the reverse thread of x.

c(x) = the number of nodes in T(x) (called the cardinality of x).

f(x) = the "last node" of the nodes in T(x) (hence the last node in the thread in T(x)).

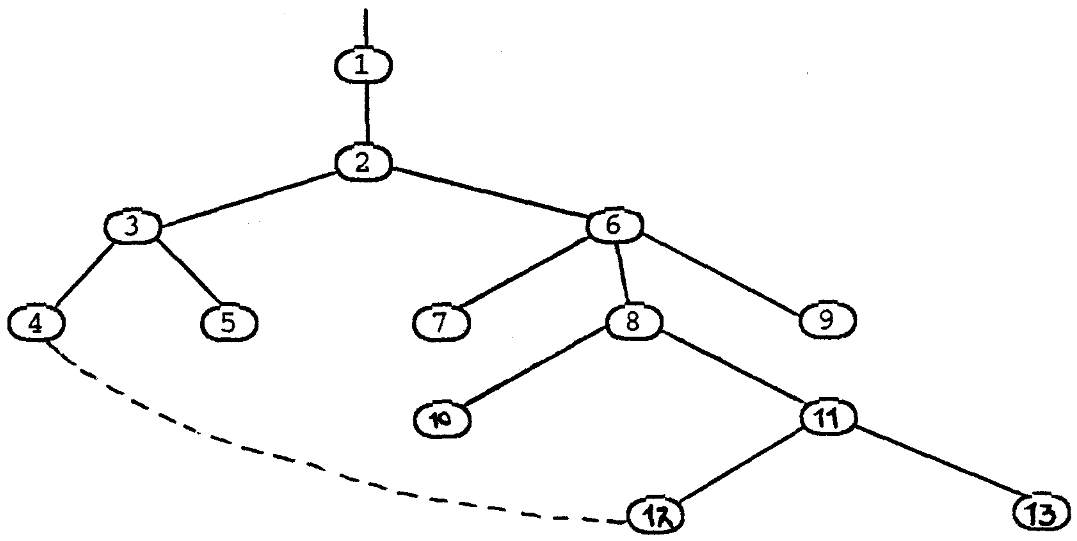


Figure 2.4 : Sample Rooted Spanning Tree

d(x) = the length (i.e. number of arcs) of the path linking any node x to the root node, where d(x)=0 if x is a root node (called the distance of x).

Table 2.1 illustrates the node functions for the spanning tree of Figure 2.4.

Table 2.1 : Node Functions for the Rooted Spanning Tree

$k \in \mathcal{N}$	p(k)	t(k)	r(k)	c(k)	f(k)	d(k)
1	0	2	9	13	9	0
2	1	3	1	12	9	1
3	2	4	2	3	5	2
4	3	5	3	1	4	3
5	3	6	4	1	5	3
6	2	7	5	8	9	2
7	6	8	6	1	7	3
8	6	10	7	5	13	3
9	6	1	13	1	9	3
10	8	11	8	1	10	4
11	8	12	10	3	13	4
12	11	13	11	1	12	5
13	11	9	12	1	13	5

2.1.1.2 Reoptimization Procedures for Pure Network Algorithms

This section will introduce procedures that use the spanning tree properties of network simplex bases and the elegant data structures to quickly reoptimize networks which have undergone costs and/or bounds changes. In each instance, the methods keep as much of the existing solution as possible. (Ali, Allen , Barr and Kennington (1986)).

Consider problem (*NP*) again:

$$\begin{array}{ll} \text{minimize} & cx \\ \text{subject to} & \\ & Ax + ae^{\ell} = r \\ & 0 \leq x \leq u \\ & 0 \leq a \leq 0 \end{array}$$

The set of arcs, $\mathcal{A}$, can be partitioned into three subsets:

$$B = \left\{ (i,j) : \text{arc } (i,j) \text{ is basic} \right\}$$
$$N^0 = \left\{ (i,j) : \text{arc } (i,j) \text{ is nonbasic with zero flow} \right\}$$
$$N^u = \left\{ (i,j) : \text{arc } (i,j) \text{ is nonbasic with flow equal to its upper bound} \right\}$$

With this partitioning, problem (*NP*) can be restated as follows:

$$\min \sum_{(i,j)\in \mathcal{A}} c_{ij}\, x_{ij} \tag{20}$$

s.t.

$$\sum_{(i,j)\in B} x_{ij} - \sum_{(j,k)\in B} x_{jk} + ae^{j} = rr_{j} \quad j \in \mathcal{N} \tag{21}$$

$$0 \le x_{ij} \le u_{ij} \qquad \text{for } (i,j) \in B \tag{22}$$

$$x_{ij} = 0 \qquad \text{for } (i,j) \in N^{0} \tag{23}$$

$$x_{ij} = u_{ij} \qquad \text{for } (i,j) \in N^{u} \tag{24}$$

$$0 \le a \le 0 \tag{25}$$

$$\mathcal{A} = B \cup N^{0} \cup N^{u} \tag{26}$$

where rr_j is called the *reduced requirement* at node j, and

$$rr_{j} = r_{j} - \left(\sum_{(i,j)\in N^{u}} x_{ij} - \sum_{(j,k)\in N^{u}} x_{jk} \right) \tag{27}$$

It is clear that once the nonbasic sets N^0 and N^u are assigned, equations (21) uniquely determine the basic arc flows. In addition, the flows give a basic feasible solution if constraints (22) are satisfied.

The node labels that are useful in the reoptimization procedures are p(k), π_k, rr(k), t(k), r(k) and FLOW(k) for a node number k where rr(k) is the reduced requirement at node k and FLOW(k) is the flow on either (k,p(k)) ∈ B or

$(p(k),k) \in B$, whichever exists, with $FLOW(\ell) = 0$. The reduced requirements $rr(i)$, $i \in \mathcal{N}$, need not be maintained explicitly, since they can be reconstructed from a given set of nonbasic and basic flows. The following procedure reconstructs the reduced requirements. Upon completion, each $rr(i)$, $i \in \mathcal{N}$, corresponds to demand if positive and supply if negative.

```
PROCEDURE X2D
Step 1 : Set $i \leftarrow 1$ and set $rr(i) \leftarrow FLOW(i)$.
Step 2 : Increment. Set $i \leftarrow i + 1$.
         If $i > m$, go to step 3; otherwise, go to step 1.
Step 3 : Let $j = t(\ell)$ and $rr(\ell) = 0$.
Step 4 : Set $i \leftarrow 1$.
Step 5 : If $(j,p(j)) \in B$, then let
         $rr(p(j)) = rr(p(j)) + rr(j)$ and $rr(j) = -rr(j)$;
         otherwise, let $rr(p(j)) = rr(p(j)) - rr(j)$.
Step 6 : Increment. Set $i \leftarrow i + 1$.
         If $i > m - 1$, terminate; otherwise, let $j = t(j)$
         and go to step 5.
```

For a given basis B, changes in node requirements are incorporated via changes in the flow on basic arcs. Notice that changes to any bounds for nonbasic arcs can also be made on this set of reduced requirements, since each nonbasic arc in N^u (N^0) with an altered upper (lower) bound remains nonbasic at its new upper (lower) bound, thus forcing a change in the basic flows in order to preserve the conservation of flow. The following procedure redistributes this modified set of values for each rr(i), $i \in \mathcal{N}$, to the basic variables.

```
PROCEDURE D2X
Step 1 : Set i ← 1.  Let FLOW(i) = rr(i).
Step 2 : Increment.  Set i ← i + 1.
         If i > m, go to step 3; otherwise, go to step 1.
Step 3 : Let j = r(ℓ). Set k ← 1.
Step 4 : Let FLOW(p(j)) = FLOW(p(j)) + FLOW(j).
         If (j,p(j)) ∈ B, let FLOW(j) = -FLOW(j).
         Let j = r(j).
Step 5 : Increment.  Set k ← k + 1.
         If k > m - 1, terminate; otherwise, go to step 4.
```

Upon completion, the basis will have a set of flows satisfying (21) but not necessarily (22). Any basic variable whose flow violates one of its bounds must be handled using the appropriate cases given below.

Assume we have an optimal extreme point solution to problem ($\mathcal{NP}$), and we are interested in making one or more of the following changes to the original problem (NP) and finding the optimal solution to the revised problem.

CHANGING A UNIT COST

Changes in unit costs require relatively simple treatment for reoptimization since the primal feasibility of the basis is unaffected. Suppose that the unit cost on arc (i,j) is changed from c_{ij} to $\hat{c}_{ij}$ for a net change of $\Delta c = \hat{c}_{ij} - c_{ij}$. Let $\hat{\pi}_i$ be the new dual variables, determined as follows.

Case 1 : Arc (i,j) is basic.

a. If $i = p(j)$, set $\Delta c = -\Delta c$.

b. Set $\hat{\pi}_k = \pi_k + \Delta c$ for $k \in T(i)$

$\hat{\pi}_k = \pi_k - \Delta c$ for $k \in T - T(i)$

Case 2 : Arc (i,j) is nonbasic.

No changes to the dual variables are required.

Notice that if several arcs' costs are modified, it would be advantageous to replace c_{ij} with $\hat{c}_{ij}$ and solve for the dual variables using the procedure discussed in the previous section with the optimal extreme point solution as a starting basic feasible solution for the modified problem.

CHANGING AN UPPER BOUND

Suppose that the upper bound on arc (i,j), in problem (NP), is changed from $\bar{u}_{ij}$ to $\bar{u}^*_{ij} \geq T_{ij}$ for a net change of $\Delta u = \bar{u}^*_{ij} - \bar{u}_{ij} = \bar{u}^*_{ij} - (u_{ij} + T_{ij})$. Let $\hat{u}_{ij} = \bar{u}^*_{ij} - T_{ij}$, then $\Delta u = \hat{u}_{ij} - u_{ij}$. That is, we have an optimal solution $\tilde{x}$ for problem

$$\begin{aligned} &\text{minimize} \quad cx \\ &\text{subject to} \\ &\qquad Ax + ae^{\ell} = r \\ &\qquad 0 \leq x \leq u \\ &\qquad 0 \leq a \leq 0 \end{aligned}$$

and we would like to find an initial basic feasible solution for the problem

$$\begin{aligned} &\text{minimize} \quad cx \\ &\text{subject to} \\ &\qquad Ax + ae^{\ell} = r \\ &\qquad 0 \leq x \leq \hat{u} \\ &\qquad 0 \leq a \leq 0 \end{aligned}$$

Case 1 : Arc (i,j) is basic.

a. $\tilde{x}_{ij} \leq \hat{u}_{ij}$.

No change is required and the current solution is both feasible and optimal.

b. $\tilde{x}_{ij} > \hat{u}_{ij}$.

Make arc (i,j) nonbasic at the new upper bound $\hat{u}_{ij}$ and replace it in the basis tree with the artificial arc (i,j) having a flow of $\tilde{x}_{ij} - \hat{u}_{ij}$.

Case 2 : Arc (i,j) is nonbasic.

a. $\tilde{x}_{ij} = 0$.

No changes are required and the current solution is both feasible and optimal.

b. $\tilde{x}_{ij} = u_{ij}$.

Set $\tilde{x}_{ij}$ to its new upper bound $\hat{u}_{ij}$. Apply procedure X2D to construct a vector of reduced requirements, rr(i), from basic flows, FLOW(i). Set rr(i) = rr(i) + Δu and rr(j) = rr(j) - Δu. Apply procedure D2X to construct a set of basic flows, FLOW(i), from a set of reduced requirements, rr(i). Adjust any (basic) flows exceeding their bounds as follows:

For all arcs (p,q) $\in$ B, if $\tilde{x}_{pq} > u_{pq}$, go to case 1b; if $\tilde{x}_{pq} < 0$, set $\hat{x}_{pq} = x_{pq}$, make arc (p,q) nonbasic at its lower bound (zero), and replace it in the basis tree with an artificial reverse arc (q,p), having a flow of $- \hat{x}_{pq}$.

Note that if several arcs' bounds are modified, it would be advantageous to check nonbasic arcs first, and therefore procedures X2D and D2X need only be applied once. In this case the basic arcs are only checked once for feasibility.

CHANGING A LOWER BOUND

Suppose that the lower bound on arc (i,j), in problem (NP), is changed from $\bar{l}_{ij}$ to $\hat{l}_{ij}$ for a net change of $\Delta l = \hat{l}_{ij} - \bar{l}_{ij}$. That is, the new problem to solve is

$$\begin{array}{ll} \text{minimize} & c\bar{x} \\ \text{subject to} & \\ & A\bar{x} = \bar{r} \\ & \hat{l} \le \bar{x} \le \bar{u} \end{array}$$

which is equivalent to

$$\begin{array}{ll} \text{minimize} & c\hat{x} \\ \text{subject to} & \\ & A\hat{x} + ae^{\ell} = \hat{r} \\ & 0 \le \hat{x} \le \hat{u} \\ & 0 \le a \le 0 \end{array}$$

where $\bar{x} = \hat{x} + \hat{l}$, $\hat{r} = \bar{r} - A\hat{l}$ and $\hat{u} = \bar{u} - \hat{l}$.

But an optimal extreme solution is available for the problem

$$\begin{array}{ll} \text{minimize} & cx \\ \text{subject to} & \\ & Ax + ae^{\ell} = r \\ & 0 \leq x \leq u \\ & 0 \leq a \leq 0 \end{array}$$

denoted $\{\tilde{x}_{ij}\}$ where $x = \bar{x} - \bar{l}$, $r = \bar{r} - A\bar{l}$ and $u = \bar{u} - \bar{l}$. Furthermore,

$$\hat{x} = \bar{x} - \bar{l} - \Delta l = x - \Delta l,$$
$$\hat{r} = \bar{r} - A\bar{l} - A\Delta l = r - A\Delta l,$$

and $$\hat{u} = \bar{u} - \bar{l} - \Delta l = u - \Delta l.$$

Case 1 : Arc (i,j) is basic.

a. $\Delta l \leq \tilde{x}_{ij} \leq u_{ij}$.

Setting $\hat{x}_{ij} = \tilde{x}_{ij} - \Delta l$ yields a solution that is both feasible and optimal.

b. $\tilde{x}_{ij} < \Delta l$.

Make arc (i,j) nonbasic at its lower bound (zero), and replace it in the basis tree with an artificial reverse arc (j,i), having a flow of $\Delta l - \tilde{x}_{ij}$.

Case 2 : Arc (i,j) is nonbasic.

a. $\tilde{x}_{ij} = u_{ij}$.

No basic flow change is required, the current solution is both feasible and optimal, and $\hat{u}_{ij} = u_{ij} - \Delta l$.

b. $\tilde{x}_{ij} = 0$.

Set $\hat{x}_{ij}$ to be 0.

Apply procedure X2D to construct a vector of reduced requirements, rr(i), from basic flows, FLOW(i). Set rr(i) = rr(i) + Δu and rr(j) = rr(j) - Δu. Apply procedure D2X to construct a set of basic flows, FLOW(i), from a set of reduced requirements, rr(i). Adjust any (basic) flows exceeding their bounds as follows:

For all arcs $(p,q) \in B$, if $\tilde{x}_{pq} > u_{pq}$, adjust the flows according to case 1b of CHANGING AN UPPER BOUND ; if $\tilde{x}_{pq} < 0$, set $\hat{x}_{pq} = \tilde{x}_{pq}$, make arc (p,q) nonbasic at its lower bound (zero), and replace it in the basis tree with an artificial reverse arc (q,p), having a flow of $- \hat{x}_{pq}$.

Notice that if several lower and/or upper bounds on arcs are modified case 1 and case 2 of the upper and lower bound change need only be applied once.

To illustrate the above procedures consider the following example:

$$\min\ 10x_1 + 11x_2 \qquad + 12x_5 + 12x_6 + 10x_7 + \qquad + 6x_9 + 14x_{10}$$

subject to

$$\begin{aligned}
-x_1 - x_2 - x_3 &= -10\\
- x_4 - x_5 - x_6 &= -14\\
x_1 + x_4 - x_7 + x_8 &= 8\\
x_2 + x_5 + x_7 - x_8 - x_9 + x_{10} &= 8\\
x_3 + x_6 + x_9 - x_{10} &= 8
\end{aligned}$$

$0 \le x_1 \le 4$, $0 \le x_2 \le 10$, $0 \le x_3 \le 4$, $0 \le x_4 \le 4$,
$0 \le x_5 \le 14$, $0 \le x_6 \le 4$, $0 \le x_7 \le 10$, $0 \le x_8 \le 14$,
$0 \le x_9 \le 14$, $0 \le x_{10} \le 10$.

An optimal solution to the problem is
$x=(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10})=(4,2,4,4,6,4,0,0,0,0)$.
Figure 2.5 shows the network and its corresponding optimal basis tree. Note that arcs x_1, x_3 and x_4 are nonbasic arcs at their upper bounds. Now suppose that the upper bounds on arcs x_2, x_3, x_4 and x_6 are changed to $\hat{u}_2 = 7$, $\hat{u}_3 = 3$, $\hat{u}_4 = 6$ and $\hat{u}_6 = 3$. In order to find a feasible starting point, let $x_3 = 3$ (its new upper bound) and $x_4 = 6$ (its new upper bound) and find the new basic flows. Applying procedure X2D gives

i	1	2	3	4	5
rr(i)	-2	-10	0	8	4

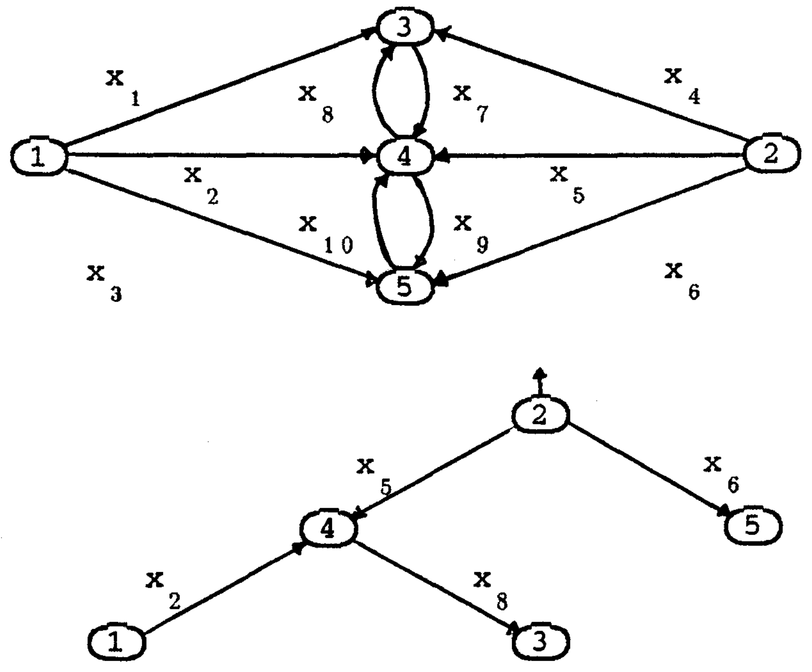


Figure 2.5 : A Network and its Optimal Rooted Spanning Tree

Arc (1,5) has a net change of -1, so rr(1)=-3 and rr(5)=5. Arc (2,3) has a net change of 2, thus rr(2)=-8 and rr(3)=-2. These result in a new vector of reduced requirements

i	1	2	3	4	5
rr(i)	-3	-8	-2	8	5

Applying procedure D2X yields a new basis vector

i	1	2	3	4	5
x(i)	3	0	-2	3	5

That is,

$x=(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10})=(4,3,3,6,3,5,0,-2,0,0)$. Now all the nonbasic arcs are within their bounds. But x_6 and x_8 violate their upper bound and lower bound, respectively. Therefore, x_6 is set to 3 and made nonbasic, and it is replaced by the artificial $x_{11}=2$ in the basis (x_{11} is the artificial arc (2,5)). x_8 is set to 0 and made nonbasic and it is replaced by the artificial arc $x_{12}=2$ in the basis (x_{12} is the artificial arc (3,4)). Hence the starting solution for the modified problem is

$x = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})$
$= (4,3,3,6,3,3,0,0,0,0,2,2)$,

where arcs x_2, x_5, x_{11} and x_{12} are basic, and arcs x_1, x_3, x_4 and x_6 are nonbasic at their upper bounds.

2.1.2 Projection Operators

Some algorithms (e.g. the subgradient algorithm) make use of a procedure called a *projection operator*. Let Γ be any compact, convex and nonempty set. The projection of a point $\bar{x} \notin \Gamma$ onto Γ, denoted by $P[\bar{x}]$, is defined to be any point $x \in \Gamma$ that is nearest $\bar{x}$ with respect to the Euclidean norm. Held, Wolfe and Crowder (1974) have suggested an extremely simple means for obtaining projection operators P that project an infeasible point $\bar{x}$ onto a feasible region $\left\{ y = (y_1, \dots, y_n) : \sum_j e_j y_j = b, 0 \le y_j \le u_j, e_j > 0, j=1,\dots,n \right\}$. Mathematically we wish to solve the following problem:

$$\min \left\{ \sum_j (x_j - \bar{x}_j)^2 : \sum_j e_j x_j = b,\ 0 \le x_j \le u_j,\ e_j > 0,\ 1 \le j \le n \right\} \tag{28}$$

Kennington and Helgason (1980, p. 228) present a simple efficient algorithm for solving (28). They show that to solve (28) one need only find the appropriate λ such that

$f(\lambda) = b$ where

$$f(\lambda) = \sum_j e_j x_j(\lambda) = \sum_j \max \left\{ \min \left(e_j \bar{x}_j - e_j^2 \lambda \;,\; e_j u_j \right) , \; 0 \right\}. \tag{29}$$

Note that $x_j(\lambda)$ may be expressed as:

$$x_j(\lambda) = \begin{cases} u_j & \lambda \le \dfrac{\bar{x}_j - u_j}{e_j} \\ \bar{x}_j - e_j \lambda & \dfrac{\bar{x}_j - u_j}{e_j} < \lambda \le \dfrac{\bar{x}_j}{e_j} \\ 0 & \lambda > \dfrac{\bar{x}_j}{e_j} \end{cases}$$

Clearly each $x_j(\lambda)$ is piecewise linear and monotonically nonincreasing. Hence $f(\lambda)$ is also piecewise linear and monotonically nonincreasing since the positive sum of such functions preserves this property.

To illustrate a typical f suppose that $\bar{x} = (\bar{x}_1, \bar{x}_2)' = (5,6)'$ and we wish to find a point $x = (x_1, x_2)'$ such that

$$\min \left\{ \|x - \bar{x}\|^2 \; : \; 5x_1 + 7x_2 = 30, \; 0 \le x_1 \le 4, \; 0 \le x_2 \le 5 \right\}.$$

Then

$$x_1(\lambda) = \begin{cases} 4 & \lambda \le \frac{1}{5} \\ 5 - 5\lambda & \frac{1}{5} < \lambda \le 1 \\ 0 & \lambda > 1 \end{cases}$$

$$x_2(\lambda) = \begin{cases} 5 & \lambda \le \frac{1}{7} \\ 6 - 7\lambda & \frac{1}{7} < \lambda \le \frac{6}{7} \\ 0 & \lambda > \frac{6}{7} \end{cases}$$

and

$$f(\lambda) = 5x_1(\lambda) + 7x_2(\lambda) = \begin{cases} 55 & \lambda \le \frac{1}{7} \\ 62 - 49\lambda & \frac{1}{7} < \lambda \le \frac{1}{5} \\ 67 - 74\lambda & \frac{1}{5} < \lambda \le \frac{6}{7} \\ 25 - 25\lambda & \frac{6}{7} < \lambda \le 1 \\ 0 & \lambda > 1 \end{cases}$$

We solve for λ such that $f(\lambda) = 30$. It is clear from Figure 2.6 that $\lambda = 0.5$. Thus

$$P\left[\begin{pmatrix} 7 \\ 3 \end{pmatrix} \right] = \begin{pmatrix} 2.5 \\ 2.5 \end{pmatrix}.$$

Kennington and Helgason (1980, pp. 231–232) also present an algorithm for obtaining λ such that $f(\lambda) = b$. The procedure consists of a binary search to bracket λ between two breakpoints, followed by a linear interpolation. The algorithm is summarized below:

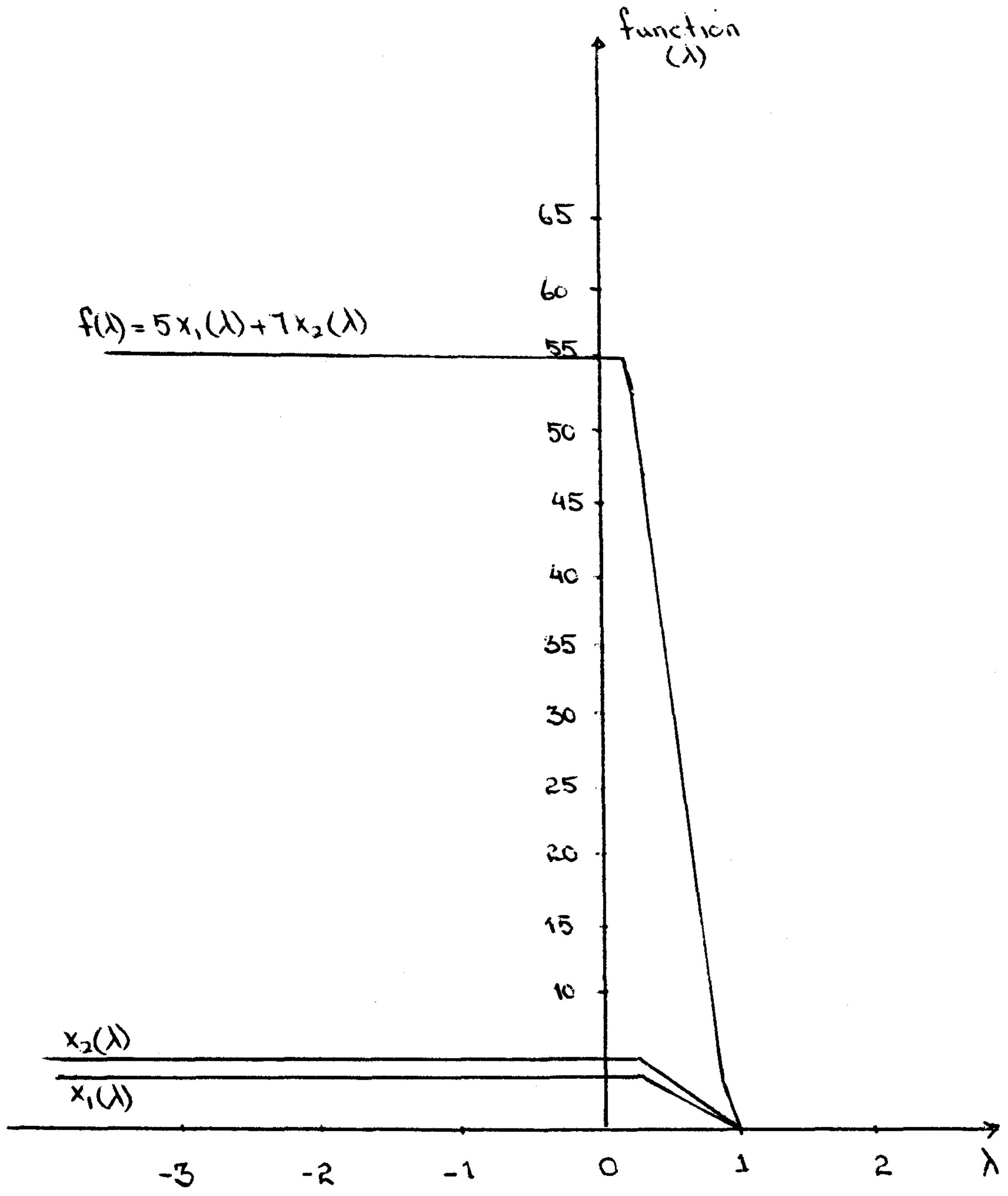


Figure 2.6 : Illustration of $f(\lambda)$, $x_1(\lambda)$ and $x_2(\lambda)$

ALG 2.1 ALGORITHM FOR λ

Step 0 : Initialization.

Let $a_1 \le a_2 \le \dots \le a_{2n}$ denote the ordered 2n breakpoints $\frac{\bar{x}_j - u_j}{e_j}$ and $\frac{\bar{x}_j}{e_j}$ for j=1,...,n.

If $b > \sum_j e_j u_j$ or $b < 0$ terminate with no feasible solution; otherwise set $l=1$, $r=2n$, $L=\sum_j e_j u_j$ and $R=0$.

Step 1 : Test for Bracketing.

If $r-l=1$, go to step 4; otherwise set $m=[\frac{l+r}{2}]$, the greatest integer $\le \frac{l+r}{2}$.

Step 2 : Compute New Value.

$$B = \sum_j \max \left\{ \min \left(e_j \bar{x}_j - e_j^2 a_m , e_j u_j \right) , 0 \right\}.$$

Step 3 : Update.

If $B = b$, terminate with $\lambda = a_m$. If $B > b$, set $l=m$, $L=B$, and go to step 1. If $B < b$, set $r=m$, $R=B$, and go to step 1.

Step 4 : Interpolate.

Terminate with $\lambda = a_l + \frac{(a_r - a_l)(b - L)}{R - L}.$

To demonstrate the above algorithm consider the previous example again.

Step 0 : The breakpoints are $\frac{\bar{x}_1 - u_1}{e_1} = \frac{1}{5}$, $\frac{\bar{x}_2 - u_2}{e_2} = \frac{1}{7}$, $\frac{\bar{x}_1}{e_1} = 1$, $\frac{\bar{x}_2}{e_2} = \frac{6}{7}$. Thus the ordered breakpoints

are: $a_1 = \frac{1}{7} < a_2 = \frac{1}{5} < a_3 = \frac{6}{7} < a_4 = 1.$

Set $l = 1$, $r = 4$, $L = 55$ and $R = 0$.

Step 1 : Since $r - l > 1$, set $m = [\frac{5}{2}] = 2$. Then

Step 2 : $B = \max\left\{ \min\left(25 - 5\ ,\ 20\right),\ 0 \right\} +$

$\max\left\{ \min\left(42 - \frac{49}{5}\ ,\ 35\right),\ 0 \right\} = 20 + \frac{161}{5}$

$= \frac{261}{5}.$

Step 3 : Since $B = \frac{261}{5} > b = 30$, set $l = 2$ and $L = \frac{261}{5}$.

Step 1 : Since $r - l = 2 > 1$, set $m = [\frac{6}{2}] = 3$.

Step 2 : $B = \max\left\{ \min\left(25 - \frac{150}{7}\ ,\ 20\right),\ 0 \right\} +$

$\max\left\{ \min\left(42 - 42\ ,\ 35\right),\ 0 \right\} = \frac{25}{7} + 0$

$= \frac{25}{7}.$

Step 3 : Since $B = \frac{25}{7} < b = 30$, set $r=3$, $R = \frac{25}{7}$.

Step 1 : Now $r - l = 1$, thus

Step 4 : $\lambda = \frac{1}{2}$; hence $x_1 = 2.5$ and $x_2 = 2.5$.

Figure 2.7 illustrates the problem. It is clear that the closest point to $\bar{x} = (5,6)'$ that is in the feasible region is $x = (2.5,2.5)$. The algorithm attempts to find this point by way of the line -|-|-|-|- shown in the figure. At the first breakpoint, a_1, $f(a_1)$ is equal to 55 which is greater than 30. The algorithm then moves to the second breakpoint, a_2. At a_2, $f(a_2)$ is equal to $\frac{261}{5}$ which is still greater than 30. At a_3, $f(a_3)$ is equal to $\frac{25}{7}$ which is less than 30. Hence $f(\lambda) = 30$ for some λ such that $a_2 < \lambda < a_3$. Since we know that $f(\lambda)$ is linear between the breakpoints, a line interpolation would give us a value of λ for which $f(\lambda) = 30$.

2.1.3 <u>Subgradient Optimization</u>

Consider the nonlinear program given by

$$\begin{array}{ll} \min & g(y) \\ \text{s.t.} & y \in G \end{array}$$

a_i	$(x_1(a_i), x_2(a_i))'$	$f(a_i)$
$\frac{1}{7}$	$(4,5)$	55
$\frac{1}{5}$	$(4,4.6)$	52.2
$\frac{6}{7}$	$(\frac{5}{7},0)$	$\frac{25}{7}$
1	$(0,0)$	0

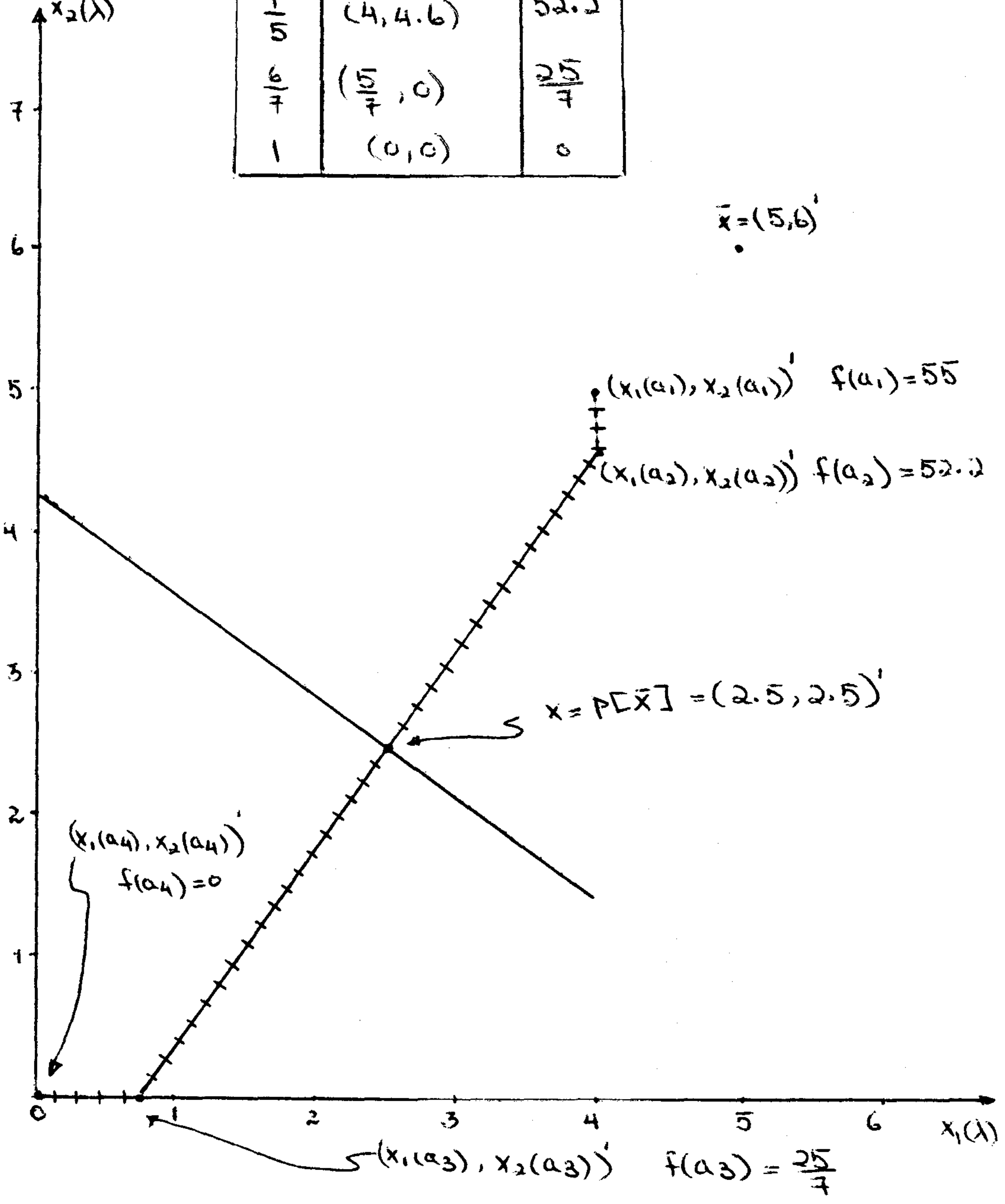


Figure 2.7 : Illustration of the Problem and ALG 2.1

where g is a convex, real-valued function defined over the compact, convex, and nonempty set G. A vector η is called a *subgradient* of g(y) at $\bar{y}$ if

$$g(y) - g(\bar{y}) \geq \eta\,(y - \bar{y}) \qquad \text{for all } y \in G.$$

If g(y) is differentiable at $\bar{y}$, the only subgradient at $\bar{y}$ is the gradient. For points at which g(y) is not differentiable, the subgradient is any linear support of g at $\bar{y}$. For any $\bar{y} \in G$, denote the set of all subgradients of g at $\bar{y}$ by $\partial g(\bar{y})$.

The subgradient optimization algorithm may be viewed as a generalization of the steepest descent (ascent) method for convex (concave) problems in which any subgradient is substituted for the gradient at a point where the gradient does not exist. Although subgradients do not necessarily provide improving directions, the convergence of the subgradient algorithm is assured under fairly minor conditions on the step sizes (see Held, Wolfe and Crowder (1974) and Allen, Helgason, Kennington and Shetty (1987)).

Using the projection operator, the subgradient algorithm in its most general form follows:

ALG 2.2 THE SUBGRADIENT ALGORITHM

```
Step 0 : Initialization.
    Let $y_0 \in G$, select a set of step sizes $s_0, s_1, s_2, \ldots,$
    and set $i \leftarrow 0$.
Step 1 : Find Subgradient.
    Obtain some $\eta_i \in \partial g(y_i)$.  If $\eta_i = 0$, terminate with $y_i$
    optimal.
Step 2 : Move to New Point.
    Set $y_{i+1} \leftarrow P[y_i - s_i \eta_i]$, set $i \leftarrow i + 1$, and return to
    step 1.
```

The termination criteria in step 1 may not hold at any member of G and is thus computationally inefficient. Hence, some other stopping rule must be devised. In practice this is often a limit on the number of iterations.

Various proposals have been offered for the selection of the step sizes. Four general schemes which have been suggested are:

(i) $s_i = \lambda_i$

(ii) $s_i = \dfrac{\lambda_i}{\|\eta_i\|}$

(iii) $$s_i = \frac{\lambda_i}{\|\eta_i\|^2}$$

(iv) $$s_i = \frac{\lambda_i \, [g(y_i) - \tilde{g}\,]}{\|\eta_i\|^2}$$

where $\tilde{g}$ is an estimate of g^*, the optimal objective value.

Convergence of the subgradient algorithm is established by means of conditions on the constants λ_i. If the conditions

$$\lambda_i > 0 \text{ for all } i, \ \lim_{i \to \infty} \lambda_i = 0, \text{ and } \sum_{i=0}^{\infty} \lambda_i = \infty$$

are satisfied then convergence of a subsequence of iterates is guaranteed when the step sizes defined by schemes (i) – (iii) are used (Kennington and Helgason (1980, pp. 222–227)). For step sizes defined by scheme (iv), under the assumption that

$$0 < \lambda_i \le \beta < 2, \ \sum_{i=0}^{\infty} \lambda_i = \infty \text{ and } \tilde{g} \le g^*,$$

for any given $\delta > 0$, there exists an iterate M (see Allen, Helgason, Kennington and Shetty (1987)) such that

$$g(y_M) \le g^* + \left(\frac{\beta}{2 - \beta}\right)(g^* - \tilde{g}) + \delta \ .$$

In this case, one eventually obtains an objective value whose error (at worst) is arbitrarily close to $\frac{\beta}{2-\beta}$ times the error present in the estimate $\bar{g}$ of g^*.

2.1.4 Lagrangean Dual Problem

Consider the arbitrary optimization problem (called the primal problem)

$$\begin{aligned} z = \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g(\mathbf{x}) \leq 0 \\ & \mathbf{x} \in X \subseteq \mathbb{R}^n \end{aligned} \tag{30}$$

where f, $g(\mathbf{x}) = [g_1(x),\ldots,g_m(x)]$ are real-valued functions and X is a nonempty set. If (30) does not have a feasible solution, we take $z = +\infty$. The constraints $g(\mathbf{x}) \leq 0$ are the so-called "complicating" constraints, i.e., the problem would be much simpler to solve without them.

One way to try to avoid the complicating constraints $g(\mathbf{x}) \leq 0$ is to price them out by placing the term $ug(\mathbf{x})$ in the objective function where **u** (called the Lagrangean multipliers) is a nonnegative m-vector of associated dual variables. We define the *Lagrangean function*

$$L(u) = \min_{x \in X} \left\{ f(x) + ug(x) \right\} \tag{31}$$

It is also convenient to define:

$$L(x,u) = f(x) + ug(x) \tag{32}$$

Let $\bar{x}$ denote a specific primal solution computed from (31) for a specific $\bar{u} \geq 0$; that is, $L(\bar{u}) = L(\bar{x},\bar{u})$. There is no guarantee that $\bar{x}$ is optimal or even feasible for (30), but the rationale for selecting $\bar{u}$ is embodied in the following conditions.

Definition : A pair $(\bar{x},\bar{u})$ with $\bar{x} \in X$, $\bar{u} \geq 0$ satisfies the Global Optimality Condition (GOC) for the primal problem (30) if

1. $f(\bar{x}) + \bar{u}g(\bar{x}) = \min_{x \in X} \left\{ f(x) + \bar{u}g(x) \right\}$
2. $\bar{u}g(\bar{x}) = 0$
3. $g(\bar{x}) \leq 0$

The Lagrangean dual problem to the primal problem (30) is obtained by finding the greatest lower bound to z; namely

$$d = \sup_{u \geq 0} L(u) \tag{33}$$

Clearly $d \leq z$ and without further assumptions on the primal problem (30), it is possible that $d < z$ (called a *duality gap*).

The following theorem gives sufficient conditions for $\bar{x}$ to be optimal in the primal problem (30). (A proof can be found in Shapiro (1979, p. 144)).

<u>Theorem</u> : If $(\bar{x},\bar{u})$ satisfies the GOC, then $\bar{x}$ is optimal in the primal problem (30).

The Lagrangean function L defined in (31) has all the nice properties, such as continuity and concavity, except once-differentiability. The function is nondifferentiable at any $\bar{u}$ where $L(\bar{u})$ has multiple optima. Although it is differentiable almost everywhere, it generally is nondifferentiable at an optimal point. It is apparent that $L(u)$ is subdifferentiable everywhere, i.e., the Lagrangean function $L(u)$ has subgradients [For a proof see Bazaraa and Shetty (1979, p. 190)]. It can easily be shown that the vector $g(\bar{x})$ is a subgradient of the Lagrangean function L at any u for which $\bar{x}$ solves (31). Any other subgradient is a convex combination of these subgradients. Because the subgradient method is easy to program and has worked well on many practical problems, it has become the most popular method for solving the

Lagrangean dual problem (33). For a complete discussion of various methods for solving the Lagrangean dual problem (33) see Bazaraa and Shetty (1979), Fisher (1981), Geoffrion (1974), Lasdon (1970) and their references.

2.1.5 A Singly-Constrained Bounded Variable Linear Program

The problem considered here is of the following form:

(SCBVLP)

$$\text{minimize} \quad z(y) = \sum_{j=1}^{n} w_j y_j \tag{34}$$

subject to

$$\sum_{j=1}^{n} e_j y_j \leq b \tag{35}$$

$$0 \leq y_j \leq u_j \tag{36}$$

where w_j and e_j $(j = 1, \ldots, n)$ and b are arbitrary real numbers, and $0 < u_j < \infty$ for all j. Specific instances of related problems have appeared in the literature, to wit:

Example (a). If it is assumed that $b \geq 0$, and for each j, $w_j < 0$, $e_j > 0$, and $u_j = \infty$, then an optimal solution to the resulting problem is given in Shapiro (1979, pp. 116–117). By ordering the variables so that

$$\frac{w_1}{e_1} \leq \frac{w_2}{e_2} \leq \ldots \leq \frac{w_n}{e_n}$$

then the (greedy) optimal solution is $y_1 = b/e_1$ and $y_j = 0$ for $j = 2, \ldots, n$.

Example (b). As in (a), except that $u_j < \infty$ for all j. Then the problem is equivalent to a continuous relaxation of an integer knapsack problem. In this case (see Wagner (1975, p. 494)) an optimal solution is found by ordering the variables as in example (a) and then setting:

$$y_j = \begin{cases} u_j & \text{for } j = 1,2,\ldots,k-1 \\ \dfrac{b - \sum_{i=1}^{k-1} a_i u_i}{a_k} & \text{for } j = k \\ 0 & \text{for } j=k+1,\ldots,n, \end{cases}$$

where (k - 1) is the largest integer such that $(b - \sum_{i=1}^{k-1} a_i u_i) \geq 0$. For more details see Chvatal (1980), Ingargiola and Korsh (1977), Murty (1976, p. 446), Wagner (1975, p. 494) and their references.

Our purpose is to extend the algorithm in (b) to the more general problem SCBVLP. To this end, some basic properties of the problem and its optimal solution must first be developed.

Assume the index set $J = \{1, 2, \ldots, n\}$ has been partitioned into three subsets

$$J^+ = \{ j \in J : e_j > 0 \}$$
$$J^- = \{ j \in J : e_j < 0 \}$$
$$J^0 = \{ j \in J : e_j = 0 \}$$

and define $b^* = b - \sum_{j \in J^-} e_j u_j$.

<u>Proposition</u> <u>1.</u> Problem SCBVLP is feasible if and only if $b^* \geq 0$.

<u>Proof:</u> ($\Rightarrow$) If SCBVLP is feasible, then there exists $\hat{y}_j$, $j \in J$ such that $0 \leq \hat{y}_j \leq u_j$ and $\sum_{j \in J} e_j \hat{y}_j \leq b$.

i.e. $$\sum_{j \in J^+} e_j \hat{y}_j + \sum_{j \in J^-} e_j \hat{y}_j \leq b$$

or $$\sum_{j \in J^+} e_j \hat{y}_j \leq b - \sum_{j \in J^-} e_j \hat{y}_j .$$

But $j \in J^- \Rightarrow e_j \hat{y}_j \geq e_j u_j$ and $j \in J^+ \Rightarrow e_j \hat{y}_j \geq 0$, therefore

$$0 \leq \sum_{j \in J^+} e_j \hat{y}_j \leq b - \sum_{j \in J^-} e_j \hat{y}_j \leq b - \sum_{j \in J^-} e_j u_j = b^*.$$

($\Leftarrow$) If $b^* = b - \sum_{j \in J^-} e_j u_j \geq 0$, then define $\hat{y}$ by

$$\hat{y}_j = \begin{cases} 0 & \text{if } j \in J^0 \cup J^+ \\ u_j & \text{if } j \in J^- \end{cases}$$

Obviously $\hat{y}$ satisfies (3), and

$$\sum_{j \in J} e_j \hat{y}_j = \sum_{j \in J^-} e_j \hat{y}_j = \sum_{j \in J^-} e_j u_j \leq b.$$

Thus $\hat{y}$ is feasible to SCBVLP. □

We now assume SCBVLP is feasible. Then an optimum exists, since the problem is bounded by (36). The following results characterize optimal solutions to SCBVLP.

Proposition 2. In any optimal solution y^* to SCBVLP, if $j \in J^0$ then

$$y_j^* = \begin{cases} u_j & \text{if } w_j < 0 \\ 0 & \text{if } w_j > 0 \\ \text{any value in } [0, u_j] & \text{if } w_j = 0 \end{cases}$$

Proof: Obvious. □

Proposition 3. In any optimal solution y^* to SCBVLP, if $j \in J^-$ and $w_j < 0$, then $y_j^* = u_j$.

Proof: Let y^* be an optimal solution, and suppose there exists $k \in J^-$ such that $w_k < 0$ but $y_k^* < u_k$.

Let $\hat{y}$ be defined by

$$\hat{y}_j = \begin{cases} u_j & \text{if } j = k \\ y_j^* & \text{otherwise} \end{cases}$$

Then $0 \leq \hat{y}_j \leq u_j$ and

$$\sum_{j \in J} e_j \hat{y}_j = \sum_{\substack{j \in J \\ j \neq k}} e_j \hat{y}_j + e_k \hat{y}_k = \sum_{\substack{j \in J \\ j \neq k}} e_j y_j^* + e_k u_k$$

But $y_k^* < u_k$ and $k \in J^-$ imply $e_k u_k < e_k y_k^*$. Thus

$$\sum_{j \in J} e_j \hat{y}_j < \sum_{\substack{j \in J \\ j \neq k}} e_j y_j^* + e_k y_k^* = \sum_{j \in J} e_j y_j^* \leq b.$$

Therefore $\hat{y}$ is feasible to SCBVLP. Furthermore,

$$z(\hat{y}) = \sum_{j \in J} w_j \hat{y}_j = \sum_{\substack{j \in J \\ j \neq k}} w_j \hat{y}_j + w_k \hat{y}_k = \sum_{\substack{j \in J \\ j \neq k}} w_j y_j^* + w_k u_k$$

and since $w_k < 0$ and $y_k^* < u_k$, this yields

$$z(\hat{y}) < \sum_{\substack{j \in J \\ j \neq k}} w_j y_j^* + w_k y_k^* = z(y^*)$$

which contradicts the optimality of y^*. □

Corollary 1. If $j \in J^-$ and $w_j = 0$, then $\hat{y}_j = u_j$ in an optimal solution $\hat{y}$ to SCBVLP.

Proof: Follows that of Proposition 3, except that $z(\hat{y}) = z(y^*)$ in the last line. □

Proposition 4. In any optimal solution y^* to SCBVLP, if $j \in J^+$ and $w_j > 0$, then $y_j^* = 0$.

Proof: Let y^* be an optimal solution, and suppose there exists $k \in J^+$ such that $w_k > 0$ but $y_k^* > 0$.

Let $\hat{y}$ be defined by

$$\hat{y}_j = \begin{cases} 0 & \text{if } j = k \\ y_j^* & \text{otherwise} \end{cases}$$

Then $0 \le \hat{y}_j \le u_j$ and

$$\sum_{j \in J} e_j \hat{y}_j = \sum_{\substack{j \in J \\ j \ne k}} e_j y_j^* + e_k \hat{y}_k < \sum_{\substack{j \in J \\ j \ne k}} e_j y_j^* + e_k y_k^* \le b$$

where the strict inequality follows from $e_k > 0$, $y_k^* > 0$ and $\hat{y}_k = 0$. So $\hat{y}$ is feasible to SCBVLP. In addition, since $w_k > 0$ we get

$$z(\hat{y}) = \sum_{\substack{j \in J \\ j \ne k}} w_j \hat{y}_j + w_k \hat{y}_k = \sum_{\substack{j \in J \\ j \ne k}} w_j y_j^* + w_k \cdot 0$$

$$< \sum_{\substack{j \in J \\ j \ne k}} w_j y_j^* + w_k y_k^* = z(y^*)$$

contradicting the optimality of y^*. □

<u>Corollary 2.</u> If $j \in J^+$ and $w_j = 0$, then $\hat{y}_j = 0$ in an optimal solution $\hat{y}$ to SCBVLP.

<u>Proof:</u> Follows the construction of $\hat{y}$ in the proof of Proposition 4, except we conclude $z(\hat{y}) = z(y^*)$. □

With these results, problem SCBVLP can be reduced to a problem involving only two sets of variables, namely those in the *restricted* sets

$$J_r^- = \{ j \in J^- : w_j > 0 \}$$
$$J_r^+ = \{ j \in J^+ : w_j < 0 \}$$

This reduced problem takes on the form:

$$\text{minimize} \quad \tilde{z}(y) = \sum_{j \in J_r^-} w_j y_j + \sum_{j \in J_r^+} w_j y_j + \tilde{k} \tag{37}$$

subject to

$$\sum_{j \in J_r^-} e_j y_j + \sum_{j \in J_r^+} e_j y_j \leq \tilde{b} \tag{38}$$

$$0 \leq y_j \leq u_j \qquad j \in J_r^- \cup J_r^+ \tag{39}$$

where $\tilde{k} = \sum_{\substack{j \in J^0 \\ w_j < 0}} w_j u_j + \sum_{\substack{j \in J^- \\ w_j \leq 0}} w_j u_j$ is a constant and $\tilde{b} = b - \sum_{\substack{j \in J^- \\ w_j \leq 0}} e_j u_j$.

Denote $|J_r^-| = k_1$ and $|J_r^+| = k_2$. Obviously, if $k_1 = k_2 = 0$, then the solution to SCBVLP is completely determined by the above results, and no problem (37)–(39) exists. Furthermore, if $k_1 = 0$, then problem (37)–(39) is in the form of example (b), and the optimal solution to (37)–(39) is determined via the algorithm stated therein, thus completing the solution to problem SCBVLP.

Finally, suppose $k_1 > 0$. Consider the transformation on (37)–(39) defined by

$$\bar{y}_j = \begin{cases} y_j & \text{for } j \in J_r^+ \\ u_j - y_j & \text{for } j \in J_r^- \end{cases}$$

Direct substitution and algebraic manipulation results in the following problem equivalent to (37)–(39):

$$\text{minimize} \quad \bar{z}(y) = \sum_{j \in J_r^-} \bar{w}_j \bar{y}_j + \sum_{j \in J_r^+} \bar{w}_j \bar{y}_j + \bar{k} \tag{40}$$

subject to

$$\sum_{j \in J_r^-} \bar{e}_j \bar{y}_j + \sum_{j \in J_r^+} \bar{e}_j \bar{y}_j \leq \bar{b} \tag{41}$$

$$0 \leq \bar{y}_j \leq u_j \qquad j \in J_r^- \cup J_r^+ \tag{42}$$

where

$$\bar{w}_j = \begin{cases} w_j & \text{for } j \in J_r^+ \\ -w_j & \text{for } j \in J_r^- \end{cases}$$

$$\bar{e}_j = \begin{cases} e_j & \text{for } j \in J_r^+ \\ -e_j & \text{for } j \in J_r^- \end{cases}$$

$$\bar{k} = \tilde{k} + \sum_{j \in J_r^-} w_j u_j$$

and

$$\bar{b} = \tilde{b} - \sum_{j \in J_r^-} e_j u_j = \tilde{b} - \sum_{\substack{j \in J^- \\ w_j > 0}} e_j u_j$$

$$= \left(b - \sum_{\substack{j \in J^- \\ w_j \le 0}} e_j u_j \right) - \sum_{\substack{j \in J^- \\ w_j > 0}} e_j u_j$$

$$= b - \sum_{j \in J^-} e_j u_j = b^*$$

Problem (40)–(42) is now in the form of example (b). Thus, its solution – and the completion of the solution to problem SCBVLP – can be determined.

The greedy algorithm given by Wagner (1975, p. 494) has been modified to solve SCBVLP directly, using the results obtained previously. The algorithm is summarized below:

ALG 2.3 ALGORITHM FOR SCBVLP PROBLEM

Step 0 : Input n, b and for each $j = 1, \ldots, n$, e_j, w_j and u_j.

Step 1 : Determine J^-, J^0, J^+.

Compute

$$b^* = b - \sum_{j \in J^-} e_j u_j .$$

If $b^* < 0$, terminate; the problem is infeasible. Otherwise, proceed to step 2.

Step 2 : For those $j \in J^0$, set

$$y_j^* = \begin{cases} 0 & \text{if } w_j \geq 0 \\ u_j & \text{if } w_j < 0 \end{cases}$$

For those $j \in J^+$ with $w_j \geq 0$, set $y_j^* = 0$.
For those $j \in J^-$ with $w_j \leq 0$, set $y_j^* = u_j$.

Step 3 : Determine J_r^- and J_r^+. If $J_r = J_r^- \cup J_r^+ = \phi$, terminate; an optimal solution has been determined in steps 1 and 2. Otherwise, let $|J_r| = k$, and order the elements of J_r so that

$$\frac{w_{(1)}}{e_{(1)}} \leq \frac{w_{(2)}}{e_{(2)}} \leq \ldots \leq \frac{w_{(k)}}{e_{(k)}} < 0$$

Let $j = 1$.

Step 4 : Set

$$y^*_{(j)} = \begin{cases} \text{minimum } \{ \dfrac{b^*}{e_{(j)}}, u_{(j)} \} & \text{if } e_{(j)} > 0 \\ u_{(j)} - \text{minimum } \{ \dfrac{b^*}{-e_{(j)}}, u_{(j)} \} & \text{if } e_{(j)} < 0 \end{cases}$$

Adjust b^* according to

$$b^* = \begin{cases} b^* - e_{(j)} y^*_{(j)} & \text{if } e_{(j)} > 0 \\ b^* + e_{(j)} (u_{(j)} - y^*_{(j)}) & \text{if } e_{(j)} < 0 \end{cases}$$

Step 5 : Let $j = j + 1$. If $j \leq k$, go to step 4. Otherwise, terminate with an optimal solution.

The algorithm has been implemented in a FORTRAN code called SCBVLP and a complete listing is found in Bolouri and Arthur (1989).

To illustrate algorithm ALG 2.3 consider the following example.

minimize $6y_1 - 3y_2 - y_3 + y_4 - 6y_5 + 7y_6$

subject to

$$- y_1 + y_2 + 5y_3 - 100y_4 - 11y_7 \leq 18$$

$$0 \leq y_1 \leq 10 , \ 0 \leq y_2 \leq 16 , \ 0 \leq y_3 \leq 14 ,$$

$$0 \leq y_4 \leq 12, \ 0 \leq y_5 \leq 5 , \ 0 \leq y_6 \leq 10,$$

$$0 \leq y_7 \leq 2.$$

Step 1 : $J^{+} = \{ 2, 3 \}$, $J^{-} = \{ 1, 4, 7 \}$ and

$J^{0} = \{ 5, 6 \}$.

$b^{*} = 18 - (-1)(10) - (-100)(12) - (-11)(2)$

$= 1250$.

$b^{*} > 0$, thus the problem is feasible.

Step 2 : Since $w_5=-6 < 0$ and $w_6=7 > 0$, then

$y^{*}_5=5$ and $y^{*}_6=0$.

$7 \in J^{-}$ and $w_7=0$ implies that $y^{*}_7=2$.

Step 3 : $J^{-}_{r} = \left\{ 1, 4 \right\}$, $J^{+}_{r} = \left\{ 2, 3 \right\}$ and

$J_r = \left\{ 1, 2, 3, 4 \right\}$. Then k=4 and

$$\frac{w_1}{e_1} = -6 < \frac{w_2}{e_2} = -3 < \frac{w_3}{e_3} = -\frac{1}{5} < \frac{w_4}{e_4} = -\frac{1}{100} < 0$$

j=1.

Step 4 : $e_1=-1 < 0$, thus $y^{*}_1 = 10 - \min(10, \frac{1250}{1}) = 0$, and

$b^{*} = 1250 + (-1)(10) = 1240$.

Step 5 : $j=2 < k=4$.

Step 4 : $e_2=1 > 0$, thus $y^{*}_2 = \min(16, \frac{1240}{1}) = 16$,

and $b^{*} = 1240 - (1)(16) = 1224$.

Step 5 : Since $j=3 < k=4$, then

Step 4 : $e_3=5 > 0$, thus $y_3^*=14$ and $b^*= 1154$.

Step 5 : Since $j=4=k$, then

Step 4 : $e_4=-100 < 0$, thus $y_4^* = .46$ and
$b^*= 0$.

Step 5 : Since $j=5 > k=4$, terminate with
$y = (y_1^*, y_2^*, y_3^*, y_4^*, y_5^*, y_6^*, y_7^*) =$
$= (0, 16, 14, 0.46, 5, 0, 2)$ an optimal solution
to the SCBVLP problem.

2.2 Literature Review

Just where the study of networks may be said to have originated is a debatable question. Hitchcock (1941) was the first to formulate and solve for a certain minimum cost transportation problem. Koopmans' (1947) work on the same category of problems was the first to interpret properties of optimal and nonoptimal solutions with respect to the linear graph associated with a network of routes. Because of this and the work done by Hitchcock, the classical case is often referred to as the Hitchcock–Koopmans Transportation Problem.

A few years later Dantzig (1951) showed how his general algorithm for solving linear programs, the simplex method, could be simplified and made more effective for the special case of the transportation models. Dantzig (1963) further examined the results and showed that a basis can be represented as a rooted spanning tree.

The first data structure suggested for implementation is presented in Johnson (1966). He proposed a labelling procedure that only requires three labels at each node. This procedure could be used to carry out the steps of the simplex algorithm completely in terms of the graph. Glover, Karney and Klingman (1972) elaborated on Johnson's procedure by providing a method for characterizing successive basis trees with minimal relabelling. This procedure, which is called the augmented predecessor index (API) method, also provided the most efficient way of finding the representation of the arc coming into the basis, pricing out the basic arcs and updating basis labels. Glover et al. have shown that the major updating calculations of a basis exchange step can be restricted to just one of the two subtrees created by dropping the outgoing arc. Srinivasan and Thompson (1972) have proposed an alternate procedure for doing this. Their procedure requires sorting the nodes of the subtree by their distances from the root and then a full subtree update of both the distance and the cardinality function at each basis exchange step is performed. Glover, Karney, Klingman and Napier's (1974) use of the API method resulted in an efficient special purpose primal simplex transportation code. Srinivasan and Thompson (1973) succeeded in reducing the solution times of their accelerated primal transportation code by half by incorporating the API into their algorithm. Glover, Klingman and Stutz (1974) developed a new list structure, called the augmented threaded index (ATI) method, for recording and updating the basis tree for adjacent extreme point ("simplex type") network algorithms. The ATI method is computationally more efficient and requires less computer memory to implement than all alternate list structures. This method uses only two pointers per node to search a spanning tree both upward and downward, while previously proposed structures required at least three pointers per node. Glover, Karney and Klingman (1974) have shown that the ATI method improves the efficiency of their transshipment code by 10% while requiring less computer memory. More recently, Barr, Glover and Klingman (1979) have developed a new type of relabelling scheme, called the extended threaded index (XTI) method, that can be used to implement the previous list structures (and particularly the ATI method) with greater efficiency. Computational results show that the XTI procedure is approximately twice as fast as the ATI procedure (the previously fastest procedure in the literature) for implementing the basis exchange operations. However, memory requirements were quite close for all of the procedures.

The pricing strategy, the selection of that particular nonbasic arc for which flow change will be allowed, is an important tactic within network programs. For most pricing strategies the arc file is managed in what is called a candidate list. The candidate list serves as a depository for a whole set or a defined subset of nonbasic arcs. Arcs are exchanged (pivoted) between basic status and this candidate list. The searching procedure for an arc in the candidate list is managed as a wrap-around stack. That is, if in a previous search the last element scanned resides in position j, the next search would start in position j+1. Whenever the end of the array is reached, scanning continues at position j=1. The simplex algorithm terminates whenever a candidate list cannot be formed because all arcs are ineligible for pivoting. Pricing strategies range from selection of the first candidate arc found (usually referred to as the "first encountered improvement rule"), to selection of a nonbasic arc in the candidate array having the most improved reduced cost (usually referred to as the "most improvement rule"). Glover, Karney, Klingman and Napier

(1974) and Srinivasan and Thompson (1973) have advocated the "most improvement rule". Other pivoting rules are extensively analyzed by Srinivasan and Thompson (1973) for dense transportation problems and by Glover, Karney, Klingman and Napier (1974) for relatively small sparse transportation problems. Also see Bradley, Brown and Graves (1977), and Mulvey (1978).

An alternative method, known as the out-of-kilter algorithm, was first developed by Fulkerson (1961). Unlike the primal simplex on a graph algorithm, the out-of-kilter algorithm is not a specialization of a more general method. This algorithm defines certain "kilter" conditions which, taken together, constitute primal and dual feasibility criteria for arcs in a network. The method brings each non-conforming ("out-of-kilter") arc "into kilter" by adjusting its flow (the primal variable) or changing its node potentials (the dual variables). A different point of view of this algorithm is given by Barr, Glover and Klingman (1974). They have reformulated the algorithm so that it employs a new labelling scheme, and a special classification scheme for determining the "kilter status" of each arc. Barr, Glover and Klingman (1974) and Glover, Karney and Klingman (1974) have shown with computational tests that basic primal approaches are considerably more efficient than the out-of-kilter algorithms.

The results for generalized networks may be traced to Dantzig's (1963) linear programming book. A basis for a generalized network problem has a graph—theoretic structure. Each component of the graph associated with a basis is either a rooted tree or a one—tree (a tree with an additional arc forming one cycle). As in the pure network problems, the simplex operations can be carried out on the graph, thereby eliminating the need for matrix operations. For a complete description of the algorithm and discussion of data structures for implementation see Brown and McBride (1984), Brown, McBride and Wood (1985), Glover and Klingman (1973), Glover, Klingman and Stutz (1974) and Kennington and Helgason (1980).

In what follows it is assumed that "side constraints" are necessary to model key policy restrictions. While the addition of non—network constraints greatly improves the realism and effectiveness of the models, it also increases significantly the difficulty of their solution. Various algorithms have been developed to solve the network with side constraints problem. The most popular of these is the primal partitioning modification of the simplex method. The advantage of this technique is that many of the simplex operations involving the basis inverse can be designed to exploit the embedded network structure. Since a portion of the basis may be stored as a rooted spanning tree as in pure network codes, the operations needed to perform the computation of the dual variables used in pricing and the computation of the updated column used in the ratio test may be done much more efficiently than in standard linear programming packages. The development of this algorithm can be traced to Bennett (1966), Charnes and Cooper (1961), Chen and Saigal (1977), Hartman and Lasdon (1972), and Kaul (1965). A complete description of the algorithm and an effective implementation are presented in Barr, Farhangian and Kennington (1986). Glover and Klingman (1981) describe a highly efficient algorithm that modifies and implements the steps of the primal simplex algorithm for the completely general case of embedded pure network problems. The efficiency is the direct result of exploiting the pure network portion of the coefficient matrix and the network–LP interface by special labelling and updating procedures.

Further specializations of this method have been developed for restricted classes of networks, such as multicommodity network flow problems (see Ali, Barnett, Farhangian and Kennington (1984)) or transportation problems (see Klingman and Russell (1975)). The papers of Hartman and Lasdon (1972), Graves and McBride (1976) and Kennington (1977) deal with primal partitioning techniques for multicommodity network flow problems. Kennington (1977) also discusses implementation techniques for solving multicommodity transportation problems that make use of the primal partitioning simplex technique. See also Kennington and Helgason (1980).

Another common method that has been successfully used to solve various classes of network with side constraints models has been the Lagrangean relaxation or subgradient optimization method. In this method, the side constraints are placed into the objective function where a penalty is assessed if they are not satisfied. Thus, the linear programming subproblems which must be solved are pure network problems with different objective function coefficients for each subproblem due to modifications of the Lagrange multipliers. Because only the cost coefficients change from subproblem to subproblem, the optimal solution for one subproblem is at least feasible, if not optimal, for the next subproblem. Shepardson and Marsten (1980) have used this approach to solve the two duty period scheduling problem which they reformulated as a one duty period problem with side constraints. Ford and Fulkerson (1958), Held, Wolfe and Crowder (1974), Kennington and Helgason (1980), Kennington and Shalaby (1977), Tomlin (1966) and Weigel and Cremeans (1972) have proposed variations of this method for solving multicommodity network flow problems. Aggarwal (1985) and Price and Gravel (1984) have successfully adapted this approach for solving the constrained assignment problem. For more complete details of this technique, see Kennington and Shalaby (1977) and Kennington and Helgason (1980).

Often the side constraints, which by definition are not of a network form, may have some other special structure. For example, in the multicommodity network flow problem, where there are limits on the total amounts of commodities that can flow through shared arcs, the side constraints that are formed on each such arc are generalized upper bounding (GUB) constraints. Ali, Barnett, Farhangian, Kennington, McCarl, Patty, Shetty and Wong (1984) solved this problem by using a primal partitioning algorithm. The basis inverse is maintained as a set of rooted spanning trees (one for each commodity) and a working basis inverse in product form. This working basis inverse has dimension equal to the number of binding GUB constraints. The initial basis is obtained using a multicommodity variation of the routine used in NETFLO (see Kennington and Helgason (1980, page 244)).

Another network with side constraints problem which falls into this category of problems is the equal flow problem. In this problem, the side constraints correspond to pairs of arcs which are restricted to have the same amount of flow. Ali, Kennington and Shetty (1985) addressed this problem by solving two sequences of pure network problems to generate an upper and lower bound. When the gap between the bounds becomes acceptably small, this method terminates with a feasible solution which can be guaranteed to be within a pre-determined percentage of the optimal.

The network with GUB constraints problem may be solved as a linear program with GUB constraints using any of the existing standard methods. Dantzig and Van Slyke (1967) were the first to propose a well-known Generalized Upper Bounding Technique (GUBT) which is a specialization of the revised simplex method. GUBT solves such an LP by subdividing the coefficient matrix into non-GUB and GUB components. By reference to this subdivision, the technique maintains the inverse of a working basis whose order equals the number of non-GUB constraints, instead of maintaining the inverse of the basis of order equal to the total number of constraints as required by the usual revised simplex method. All the quantities needed for carrying out the simplex method on this LP are derived using the inverse of the working basis and the original data in the problem, and after each pivot step only the inverse of the working basis is updated. When the number of GUB constraints is large, GUBT results in substantial savings in the memory space requirements and the total computational effort over the implementation of the conventional revised simplex method applied to the same LP. For more complete details of this technique see Lasdon (1970) and Murty (1976). Other methods for solving LPs with GUB constraints were proposed by various researchers independently but are quite similar in nature; these include Bennett (1966), Graves and McBride (1976), Hartman and Lasdon (1970), Kaul (1965), and Sakarovitch and Saigal (1967).

However, in order to reduce both solution time as well as computer storage requirements, it is often advantageous to use any existing network structure when solving these problems. A new algorithm for solving networks with GUB side constraints using Lagrangean relaxation and decomposition approaches, along with a discussion of software considerations, is presented in the following chapters.

CHAPTER 3
SOLUTION METHODOLOGY

In this chapter a new algorithm for solving the network with generalized upper bound side constraints problem will be discussed. The solution technique exploits the special structure of the side constraints and maintains as much of the characteristics of the pure network problem as possible. The solution technique consists of solving two sequences of problems. One sequence yields tighter lower bounds on the optimal value by using a Lagrangean relaxation based on relaxing "copies" of some subset of the original variables. The second sequence yields a tighter upper bound on the optimal value for the problem by using a decomposition of the problem based on changes in the capacity vector and maintains a feasible solution at all times. Both the lower and upper bounding algorithms have been developed in the context of the general subgradient algorithm presented in Section 2.1.3.

Mathematically, the network flow problem with generalized upper bound (GUB) constraints is expressed as:

(*NPG*)

$$z(\mathbf{x}) = \text{minimize} \quad c^0\mathbf{x}^0 + c^1\mathbf{x}^1 + c^2\mathbf{x}^2 + \ldots + c^p\mathbf{x}^p$$

subject to

$$\begin{array}{lr} \mathbf{A}^0\mathbf{x}^0 + \mathbf{A}^1\mathbf{x}^1 + \mathbf{A}^2\mathbf{x}^2 + \ldots + \mathbf{A}^p\mathbf{x}^p & = r \\ \mathbf{E}^1\mathbf{x}^1 \qquad\qquad\qquad\qquad & \leq b_1 \\ \mathbf{E}^2\mathbf{x}^2 \qquad\qquad\qquad & \leq b_2 \\ & \vdots \\ \mathbf{E}^p\mathbf{x}^p & \leq b_p \\ 0 \leq \mathbf{x}^i \leq \mathbf{u}^i \qquad \text{for } i = 0,\ldots,p & \end{array}$$

where $\mathbf{A}^j$ is an (mxn_j) matrix for j=0,...,p, $\mathbf{E}^k$ is a $(1\mathrm{xn}_k)$ vector for k=1,...,p, $\mathbf{x}^i$ and $\mathbf{u}^i < \infty$, for i=0,...,p, are the decision variables and capacity vectors, respectively. The matrix $\mathbf{A} = [\mathbf{A}^0|\mathbf{A}^1|\mathbf{A}^2|\ldots|\mathbf{A}^p]$ is a node-arc incidence matrix. For each k, $1 \leq k \leq p$, the vector $\mathbf{E}^k$ is of the form $[e^k_1, e^k_2, \ldots, e^k_{n_k}]$ where $e^k_j \neq 0$ for $j=1,\ldots,n_k$. All the variables in the vector $\mathbf{x}^k$ belong to the kth GUB constraint, for $1 \leq k \leq p$. Thus, the variables in the vector $\mathbf{x}^0$ do not appear in any GUB constraint. Problem (*NPG*) can also be expressed as :

$$z(\mathbf{x}) = \text{minimize} \sum_{i=0}^{p} c^i \mathbf{x}^i$$

subject to

$$\sum_{i=0}^{p} \mathbf{A}^i \mathbf{x}^i = \mathbf{r}$$

$$\sum_{j=1}^{n_k} e_j^k \; x_j^k \le b_k \qquad \text{for } k=1,\dots,p$$

$$0 \le \mathbf{x}^i \le \mathbf{u}^i \qquad \text{for } i=0,\dots,p$$

where e_j^k ($j=1,\dots,n_k$ and $k=1,\dots,p$) is any nonzero real number.

3.1 The Lower Bound for Problem *NPG*

A lower bound on the objective function of the problem (*NPG*) can be obtained by using the Lagrangean dual of the problem. The natural Lagrangean relaxation for the problem (*NPG*) is obtained by dualizing the GUB constraints, resulting in :

$$(\mathit{LR}) \qquad Lr(\mathbf{v}) = \min \sum_{i=0}^{p} c^i \mathbf{x}^i + \sum_{k=1}^{p} v_k (\mathbf{E}^k \mathbf{x}^k - b_k)$$

$$\text{s.t.} \sum_{i=0}^{p} \mathbf{A}^i \mathbf{x}^i = \mathbf{r}$$

$$0 \le \mathbf{x}^i \le \mathbf{u}^i \qquad \text{for } i=0,\dots,p$$

where $v_k \ge 0$ is the Lagrange multiplier associated with the kth GUB constraint and $\mathbf{v} = (v_1, v_2, \dots, v_p)$. The Lagrangean dual of the problem (*LR*) is

(D) $$\max_{v \geq 0} \; Lr(v)$$

The Lagrangean relaxation used in the lower bound technique is obtained by copying certain variables in *(NPG)*, using ideas presented in Glover and Klingman (1988). This scheme is interesting in that the Lagrangean subproblems keep all the original constraints. This technique is referred to as the *Lagrangean decomposition* and its dual is called the *decomposed Lagrangean dual.* The enlarged equivalent representation of problem *(NPG)* is:

(NPGC)
$$\begin{array}{lll}
\min & \sum_{i=0}^{p} c^i x^i & \\
\text{s.t.} & \sum_{i=0}^{p} A^i x^i = r & \\
& 0 \leq x^i \leq u^i & \text{for } i=0,\ldots,p \\
& E^k y^k \leq b_k & \text{for } k=1,\ldots,p \\
& y^k = x^k & \text{for } k=1,\ldots,p \\
& 0 \leq y^k \leq u^k & \text{for } k=1,\ldots,p
\end{array}$$

The Lagrangean decomposition for *(NPGC)* is obtained by dualizing the "copy" constraints $y^k=x^k$ for $k=1,\ldots,p$, which will yield a decomposable problem.

$$(LD) \qquad L(\mathbf{w}) = \min \sum_{i=0}^{p} c^i \mathbf{x}^i + \sum_{k=1}^{p} \mathbf{w}^k(\mathbf{y}^k - \mathbf{x}^k)$$

$$\text{s.t.} \quad \sum_{i=0}^{p} A^i \mathbf{x}^i = \mathbf{r}$$

$$0 \le \mathbf{x}^i \le u^i \qquad \text{for } i=0,\dots,p$$

$$E^k \mathbf{y}^k \le b_k \qquad \text{for } k=1,\dots,p$$

$$0 \le \mathbf{y}^k \le u^k \qquad \text{for } k=1,\dots,p$$

where $\mathbf{w}^k$ is the $(n_k x1)$ vector of Lagrange multipliers associated with the kth "copy" constraint, $\mathbf{w}^k=(w^k_1,\dots,w^k_{n_k})$ and $\mathbf{w}=(\mathbf{w}^1,\dots,\mathbf{w}^p)$. Note that since $\mathbf{x}$ and $\mathbf{y}$ are bounded vectors, the feasible region to *(LD)* is finite. Thus $L(\mathbf{w})$ is finite for all finite $\mathbf{w}$. For convenience, we assume that problem *(NPG)* is feasible. The decomposed Lagrangean dual of the problem *(LD)* is

$$(DD) \qquad \max_{\mathbf{w}} \; L(\mathbf{w})$$

The following proposition provides justification for considering problem *(DD)*.

<u>Proposition 1</u> : $\max_{\mathbf{w}} \; L(\mathbf{w}) = z(\mathbf{x})$.

Proof :

$$\max_{w} L(w) = \max_{w} \min \sum_{i=0}^{p} c^i x^i + \sum_{k=1}^{p} w^k (y^k - x^k)$$

$$\begin{array}{lll} \text{s.t.} & \sum_{i=0}^{p} A^i x^i = r & \\ & 0 \le x^i \le u^i & \text{for } i=0,\ldots,p \\ & E^k y^k \le b_k & \text{for } k=1,\ldots,p \\ & 0 \le y^k \le u^k & \text{for } k=1,\ldots,p \end{array}$$

$$= \max_{w} \min \ c^0 x^0 + \sum_{k=1}^{p} (c^k - w^k) x^k + \sum_{k=1}^{p} w^k y^k$$

$$\begin{array}{lll} \text{s.t.} & \sum_{i=0}^{p} A^i x^i = r & (\pi^1) \\ & x^i \le u^i & (\sigma_i^1) \\ & x^i \ge 0 & \\ & E^k y^k \le b_k & (\pi_k^2) \\ & y^k \le u^k & (\sigma_k^2) \\ & y^k \ge 0 & \end{array}$$

where π^1, σ_i^1 (for i=0,...,p), π_k^2 and σ_k^2 (for k=1,...,p) are the dual variables associated with each type of constraint. By LP duality,

$$\max_{w} L(w) =$$

$$\max_{w} \max \quad \pi^1 r + \sum_{i=0}^{p} \sigma_i^1 u^i + \sum_{k=1}^{p} \pi_k^2 b_k + \sum_{k=1}^{p} \sigma_k^2 u^k$$

s.t.

$$\pi^1 A^0 + \sigma_0^1 \le c^0$$

$$\pi^1 A^k + \sigma_k^1 \le c^k - w^k \qquad \text{for } k=1,\dots,p$$

$$\pi_k^2 E^k + \sigma_k^2 \le w^k \qquad \text{for } k=1,\dots,p$$

π^1 is unrestricted in sign

$\sigma_i^1 \le 0$ for $i=0,\dots,p$

$\pi_k^2 \le 0$ for $k=1,\dots,p$

$\sigma_k^2 \le 0$ for $k=1,\dots,p$

$$= \max \quad \pi^1 r + \sum_{i=0}^{p} \sigma_i^1 u^i + \sum_{k=1}^{p} \pi_k^2 b_k + \sum_{k=1}^{p} \sigma_k^2 u^k$$

s.t.

$$\pi^1 A^0 + \sigma_0^1 \le c^0$$

$$\pi^1 A^k + \sigma_k^1 + w^k \le c^k \qquad \text{for } k=1,\dots,p$$

$$\pi_k^2 E^k + \sigma_k^2 - w^k \le 0 \qquad \text{for } k=1,\dots,p$$

π^1 and w are unrestricted in sign

$\sigma_i^1 \le 0$ for $i=0,\dots,p$

$\pi_k^2 \le 0$ for $k=1,\dots,p$

$\sigma_k^2 \le 0$ for $k=1,\dots,p$

By LP duality,

$$
\begin{aligned}
\max_{\mathbf{w}} L(\mathbf{w}) \\
= \quad & \min \sum_{i=0}^{p} c^i \mathbf{x}^i \\
& \text{s.t.} \sum_{i=0}^{p} \mathbf{A}^i \mathbf{x}^i = \mathbf{r} \\
& \qquad 0 \leq \mathbf{x}^i \leq \mathbf{u}^i && \text{for } i=0,\ldots,p \\
& \qquad \mathbf{E}^k \mathbf{y}^k \leq b_k && \text{for } k=1,\ldots,p \\
& \qquad \mathbf{y}^k = \mathbf{x}^k && \text{for } k=1,\ldots,p \\
& \qquad 0 \leq \mathbf{y}^k \leq \mathbf{u}^k && \text{for } k=1,\ldots,p \\
= \quad & z(\mathbf{x}).
\end{aligned}
$$

Therefore $\max_{\mathbf{w}} L(\mathbf{w})$ is equal to $z(\mathbf{x})$. □

Proposition 2 below, referred to as the weak duality theorem, shows that the objective value of any feasible solution to the decomposed Lagrangean dual (*DD*) yields a lower bound on the objective value of any feasible solution to problem (*NPG*).

<u>Proposition 2</u> : Let $\bar{\mathbf{x}}$ be a feasible solution to problem (*NPG*), i.e., $\mathbf{A}\bar{\mathbf{x}} = \mathbf{r}$, $0 \leq \bar{\mathbf{x}} \leq \mathbf{u}$ and $\mathbf{E}^k \bar{\mathbf{x}}^k \leq b_k$ for $k=1,\ldots,p$. Let $(\bar{\mathbf{x}},\bar{\mathbf{y}})$ be a feasible solution to problem (*NPGC*), i.e., $\sum_{i=0}^{p} \mathbf{A}^i \bar{\mathbf{x}}^i = \mathbf{r}$, $0 \leq \bar{\mathbf{x}}^i \leq \mathbf{u}^i$ for $i=0,\ldots,p$, $\mathbf{E}^k \bar{\mathbf{y}}^k \leq b_k$, $\bar{\mathbf{y}}^k = \bar{\mathbf{x}}^k$ and $0 \leq \bar{\mathbf{y}}^k \leq \mathbf{u}^k$ for $k=1,\ldots,p$. Let $\bar{\mathbf{v}} = (\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_p)$ be a feasible solution to (*D*), i.e. $\bar{\mathbf{v}} \geq 0$. Let $\bar{\mathbf{w}} = (\bar{\mathbf{w}}^1, \bar{\mathbf{w}}^2, \ldots, \bar{\mathbf{w}}^p)$

be such that $\overline{w}^k = -\overline{v}_k E^k$ for $1 \leq k \leq p$. Then

$$Lr(\overline{v}) \leq L(\overline{w}) \leq \sum_{i=0}^{p} c^i \overline{x}^i .$$

Proof :

$$\sum_{i=0}^{p} c^i \overline{x}^i = \sum_{i=0}^{p} c^i \overline{x}^i + \sum_{k=1}^{p} \overline{w}^k (\overline{y}^k - \overline{x}^k)$$

since $\overline{y}^k = \overline{x}^k \Rightarrow \overline{w}^k(\overline{y}^k - \overline{x}^k) = 0$

$$\geq \min_{x,y} \left\{ \sum_{i=0}^{p} c^i x^i + \sum_{k=1}^{p} \overline{w}^k (y^k - x^k) : \sum_{i=0}^{p} A^i x^i = r, \right.$$
$$0 \leq x^i \leq u^i \text{ for } i=0,\dots,p,\ E^k y^k \leq b_k,$$
$$\left. 0 \leq y^k \leq u^k \text{ for } k=1,\dots,p \right\}$$

$$= L(\overline{w})$$

$$= \min_{x} \left\{ c^0 x^0 + \sum_{k=1}^{p} (c^k - \overline{w}^k) x^k : \sum_{i=0}^{p} A^i x^i = r, \right.$$
$$\left. 0 \leq x^i \leq u^i \text{ for } i=0,\dots,p \right\} + \min_{y} \left\{ \sum_{k=1}^{p} \overline{w}^k y^k : \right.$$
$$\left. E^k y^k \leq b_k,\ 0 \leq y^k \leq u^k \text{ for } k=1,\dots,p \right\}$$

$$= \min_{x} \left\{ c^0 x^0 + \sum_{k=1}^{p} (c^k + \overline{v}_k E^k) x^k : \sum_{i=0}^{p} A^i x^i = r, \right.$$
$$\left. 0 \leq x^i \leq u^i \text{ for } i=0,\dots,p \right\} + \min_{y} \left\{ \sum_{k=1}^{p} -\overline{v}_k E^k y^k \right.$$
$$\left. : E^k y^k \leq b_k,\ 0 \leq y^k \leq u^k \text{ for } k=1,\dots,p \right\}$$

$$= \min_{\mathbf{x}} \left\{ c^0\mathbf{x}^0 + \sum_{k=1}^{p} (c^k+\bar{v}_k E^k)\mathbf{x}^k : \sum_{i=0}^{p} A^i\mathbf{x}^i = r, \right.$$

$$\left. 0 \leq \mathbf{x}^i \leq u^i \text{ for } i=0,\ldots,p \right\} - \sum_{k=1}^{p} \bar{v}_k b_k + \sum_{k=1}^{p} \bar{v}_k b_k +$$

$$\min_{\mathbf{y}} \left\{ \sum_{k=1}^{p} -\bar{v}_k E^k\mathbf{y}^k : E^k\mathbf{y}^k \leq b_k,\ 0 \leq \mathbf{y}^k \leq u^k \text{ for} \right.$$

$$\left. k=1,\ldots,p \right\}$$

$$= \min_{\mathbf{x}} \left\{ \sum_{i=0}^{p} c^i\mathbf{x}^i + \sum_{k=1}^{p} \bar{v}_k (E^k\mathbf{x}^k - b_k) : \sum_{i=0}^{p} A^i\mathbf{x}^i = r, \right.$$

$$\left. 0 \leq \mathbf{x}^i \leq u^i \text{ for } i=0,\ldots,p \right\} + \sum_{k=1}^{p} \bar{v}_k b_k +$$

$$\min_{\mathbf{y}} \left\{ \sum_{k=1}^{p} -\bar{v}_k E^k\mathbf{y}^k : E^k\mathbf{y}^k \leq b_k,\ 0 \leq \mathbf{y}^k \leq u^k \text{ for} \right.$$

$$\left. k=1,\ldots,p \right\}$$

$$= Lr(\bar{v}) + \min_{\mathbf{y}} \left\{ \sum_{k=1}^{p} \bar{v}_k (b_k - E^k\mathbf{y}^k) : E^k\mathbf{y}^k \leq b_k, \right.$$

$$\left. 0 \leq \mathbf{y}^k \leq u^k \text{ for } k=1,\ldots,p \right\}$$

$\geq Lr(\bar{v})$ since the second term in the above summation is nonnegative. □

Propositions 1 and 2 establish that the optimal objective values of (*NPG*) and the decomposed Lagrangean dual are equal and that any feasible solution to the Lagrangean decomposition (*LD*) provides a lower bound on the optimal objective value for (*NPG*). In order to facilitate the solution of the Lagrangean decomposition, we need to examine the properties of the function L(w). The following result describes L(w).

<u>Proposition 3</u> : L(w) is concave.

<u>Proof</u> :

Consider w_1, w_2 and $\lambda \in [0,1]$. Let $(\bar{x},\bar{y})$ be such that

$$\sum_{i=0}^{p} c^i \bar{x}^i + \sum_{k=1}^{p} w^k (\bar{y}^k - \bar{x}^k)$$

$$= \min_{x,y} \left\{ \sum_{i=0}^{p} c^i x^i + \sum_{k=1}^{p} w^k (y^k - x^k) : \sum_{i=0}^{p} A^i x^i = r, \right.$$
$$0 \le x^i \le u^i \text{ for } i=0,\dots,p,\ E^k y^k \le b_k,$$
$$\left. 0 \le y^k \le u^k \text{ for } k=1,\dots,p \right\}$$

where $w = \lambda w_1 + (1-\lambda) w_2$

Then

$$L(w) = L(\lambda w_1 + (1-\lambda) w_2)$$
$$= \sum_{i=0}^{p} c^i \bar{x}^i + \sum_{k=1}^{p} [\lambda w_1^k + (1-\lambda) w_2^k]\ (\bar{y}^k - \bar{x}^k)$$

By definition of L,

$$L(w_1) \le \sum_{i=0}^{p} c^i \bar{x}^i + \sum_{k=1}^{p} w_1^k\ (\bar{y}^k - \bar{x}^k)$$

$$L(w_2) \le \sum_{i=0}^{p} c^i \bar{x}^i + \sum_{k=1}^{p} w_2^k\ (\bar{y}^k - \bar{x}^k)$$

Multiplying the first inequality by λ and the second by $(1-\lambda)$, we have

$$\lambda L(w_1) + (1-\lambda)L(w_2) \leq \sum_{i=0}^{p} c^i \bar{x}^i + \sum_{k=1}^{p} [\lambda w_1^k + (1-\lambda) w_2^k] (\bar{y}^k - \bar{x}^k)$$

$$= L(w)$$

Hence L(w) is concave. □

Since L(w) is piecewise linear concave (Held, Wolfe and Crowder (1974)), one must have a way for determining a subgradient of L(w) at a given **w**. Consider the following result.

<u>Proposition 4</u> : Let $(\bar{x},\bar{y})$ be an optimal solution to (*LD*) at $\bar{w}$. Then $\left(\bar{y}^1 - \bar{x}^1, \ldots, \bar{y}^p - \bar{x}^p \right)^t$ is a subgradient of L at $\bar{w}$.

<u>Proof</u> :

Let (x,y) be an optimal solution to (*LD*) at w. Then

$$L(w) = \sum_{i=0}^{p} c^i x^i + \sum_{k=1}^{p} w^k (y^k - x^k) \leq \sum_{i=0}^{p} c^i \bar{x}^i + \sum_{k=1}^{p} w^k (\bar{y}^k - \bar{x}^k)$$

and

$$L(\bar{w}) = \sum_{i=0}^{p} c^i \bar{x}^i + \sum_{k=1}^{p} \bar{w}^k (\bar{y}^k - \bar{x}^k) .$$

Thus

$$L(w) - L(\bar{w}) \leq \sum_{k=1}^{p} (w^k - \bar{w}^k)(\bar{y}^k - \bar{x}^k) .$$

Hence $\left\langle \bar{y}^1 - \bar{x}^1, \ldots, \bar{y}^p - \bar{x}^p \right\rangle^t$ is a subgradient of L at $\bar{w}$. □

The following result will aid us in finding an optimal solution to problem (*NPG*).

<u>Proposition 5</u> : (Optimality Conditions)

Let $(\bar{x},\bar{y})$ be an optimal solution to (*LD*) at **w**, i.e., $L(w) = \sum_{i=0}^{p} c^i\bar{x}^i + \sum_{k=1}^{p} w^k(\bar{y}^k-\bar{x}^k)$. If

$\left\langle \bar{x}^1, \ldots, \bar{x}^p \right\rangle^t = \bar{y} = \left\langle \bar{y}^1, \ldots, \bar{y}^p \right\rangle^t$, then $\bar{x}$ is an optimal solution to problem (*NPG*).

<u>Proof</u> :

If $\left\langle \bar{x}^1, \ldots, \bar{x}^p \right\rangle^t = \left\langle \bar{y}^1, \ldots, \bar{y}^p \right\rangle^t$, then

(i) $\sum_{i=0}^{p} c^i\bar{x}^i + \sum_{k=1}^{p} w^k(\bar{y}^k-\bar{x}^k) =$

$$\min_{x,y} \left\{ \sum_{i=0}^{p} c^i x^i + \sum_{k=1}^{p} w^k(y^k-x^k) : \sum_{i=0}^{p} A^i x^i = r, \right.$$
$$0 \le x^i \le u^i \text{ for } i=0,\ldots,p,\ E^k y^k \le b_k,$$
$$\left. 0 \le y^k \le u^k \text{ for } k=1,\ldots,p \right\}$$

(ii) $(w^1,\ldots,w^p)\left\langle \bar{y}^1 - \bar{x}^1, \ldots, \bar{y}^p - \bar{x}^p \right\rangle^t = 0$ since $\bar{x}^k = \bar{y}^k$ for $k=1,\ldots,p$

(iii) $\bar{x}^k = \bar{y}^k$ for $k=1,\ldots,p$ (given)

that is, $(\bar{x},\bar{y},w)$ satisfy the Global Optimality Conditions (see Section 2.1.4). Hence $(\bar{\mathbf{x}},\bar{\mathbf{y}})$ is optimal to *(NPGC)*. Hence $\bar{\mathbf{x}}$ is optimal to *(NPG)*. □

Consider the Lagrangean decomposition function again. For any given value of the vector $\mathbf{w}$,

$$L(\mathbf{w}) = \min_{\mathbf{x}} \left\{ \mathbf{c}^0\mathbf{x}^0 + \sum_{k=1}^{p} (\mathbf{c}^k-\mathbf{w}^k)\mathbf{x}^k : \sum_{i=0}^{p} \mathbf{A}^i\mathbf{x}^i = \mathbf{r},\ 0 \le \mathbf{x}^i \le \mathbf{u}^i \text{ for } i=0,\dots,p \right\} + \min_{\mathbf{y}} \left\{ \sum_{k=1}^{p} \mathbf{w}^k\mathbf{y}^k : \mathbf{E}^k\mathbf{y}^k \le b_k,\ 0 \le \mathbf{y}^k \le \mathbf{u}^k \text{ for } k=1,\dots,p \right\}$$

$$= \min_{\mathbf{x}} \left\{ \mathbf{c}^0\mathbf{x}^0 + \sum_{k=1}^{p} (\mathbf{c}^k-\mathbf{w}^k)\mathbf{x}^k : \sum_{i=0}^{p} \mathbf{A}^i\mathbf{x}^i = \mathbf{r},\ 0 \le \mathbf{x}^i \le \mathbf{u}^i \text{ for } i=0,\dots,p \right\}$$
$$+ \sum_{k=1}^{p} \min_{\mathbf{y}} \left\{ \sum_{j=1}^{n_k} w^k_j y^k_j : \sum_{j=1}^{n_k} e^k_j y^k_j \le b_k,\ 0 \le \mathbf{y}^k \le \mathbf{u}^k \right\}$$

where the first minimization is a pure network problem and the second consists of p singly-constrained bounded variable LP (SCBVLP) problems. Algorithms for solving these were briefly presented in Section 2.1.5. Hence, for a given $\mathbf{w}$, an optimal solution to problem *(LD)* is found by first solving the pure network problem and then solving the single-constraint bounded variable LPs. The following summarizes the algorithm for solving the Lagrangean decomposition function $L(\mathbf{w})$, for any given value of the vector w.

ALG 3.1 ALGORITHM FOR L(w)

Step 1 : Solve the pure network problem

$$\min_{\mathbf{x}} \left\{ c^0\mathbf{x}^0 + \sum_{k=1}^{p}(c^k-\mathbf{w}^k)\mathbf{x}^k : \sum_{i=0}^{p} A^i\mathbf{x}^i = r, \right.$$
$$\left. 0 \le \mathbf{x}^i \le u^i \text{ for } i=0,\dots,p \right\}.$$

Let $\bar{\mathbf{x}}$ denote its optimal solution.

Step 2 : set $k \leftarrow 1$

Step 3 : Let $\bar{b}_k = b_k - \sum_{j\in J^k_-} e^k_j u^k_j$ where $J^k_- = \{ j : e^k_j < 0 \}$

for each k.

Set $i \leftarrow 1$.

Step 4 : Order $\frac{w^k_i}{e^k_i}$ for $i=1,\dots,n_k$ from smallest to

largest. Denote the ordered values by

$$\frac{w^k_{(1)}}{e^k_{(1)}} \le \frac{w^k_{(2)}}{e^k_{(2)}} \le \dots \le \frac{w^k_{(n_k)}}{e^k_{(n_k)}} .$$

Let $\bar{y}^k_{(i)}$ and $u^k_{(i)}$ correspond to the ith ordered

value $\frac{w^k_{(i)}}{e^k_{(i)}}$.

Step 5 : If $\frac{w^k_{(i)}}{e^k_{(i)}} < 0$, then

$$\bar{y}^k_{(i)} = \begin{cases} \min\left(\frac{\bar{b}_k}{e^k_{(i)}}, u^k_{(i)}\right) & \text{if } e^k_{(i)} > 0 \\ u^k_{(i)} - \min\left(\frac{\bar{b}_k}{|e^k_{(i)}|}, u^k_{(i)}\right) & \text{if } e^k_{(i)} < 0 \end{cases}$$

Set

$$\bar{b}_k = \begin{cases} \bar{b}_k - e^k_{(i)} \bar{y}^k_{(i)} & \text{if } e^k_{(i)} > 0 \\ \bar{b}_k + e^k_{(i)} (u^k_{(i)} - \bar{y}^k_{(i)}) & \text{if } e^k_{(i)} < 0 \end{cases}$$

and go to step 6.

If $\frac{w^k_{(i)}}{e^k_{(i)}} \geq 0$, then

$$\bar{y}^k_{(i)} = \begin{cases} u^k_{(i)} & \text{if } e^k_{(i)} < 0 \\ 0 & \text{if } e^k_{(i)} > 0 \end{cases}$$

Step 6 : set $i \leftarrow i + 1$.

If $i > n_k$, the kth SCBVLP problem is done, go to step 7; otherwise, go to step 5.

Step 7 : Set $k \leftarrow k + 1$.

If $k > p$, terminate with $\bar{y}$ an optimal solution to the one constraint bounded variable LPs, and an optimal objective value for the Lagrangean decomposition function is

$$L(w) = c^0\bar{x}^0 + \sum_{k=1}^{p} (c^k - w^k)\bar{x}^k + \sum_{k=1}^{p} w^k\bar{y}^k \ ;$$

otherwise, go to step 3.

The lower bounding algorithm modifies the generalized subgradient algorithm to maximize L(w). For a given w, L(w) is solved to optimality by ALG 3.1. If the optimality conditions (Proposition 5) are satisfied the method is terminated; otherwise, a step is taken in the direction of a subgradient of L at w, a new w is determined and the process is repeated. Since there is a possibility of never reaching the optimal solution (cyclic solutions), the method terminates upon reaching an arbitrary iteration limit. The major difficulty involves finding the appropriate step size. This problem is handled by adjusting the step size based on the behavior of the function, as will be discussed in detail in Chapter 5. The algorithm makes use of a scalar, UBND, representing an upper bound for the problem. Since the solution procedure progressively recalculates the lower bound for the network with GUB constraints problem, the lower bound algorithm uses the best value for the UBND that was obtained in the previous upper bound iteration. The following summarizes the lower bound algorithm.

ALG 3.2 ALGORITHM FOR THE LOWER BOUND

Step 1 : Initialization.

Initialize UBND, step size d and tolerance ϵ.

Let w=0.

Step 2 : Find Subgradient.

Let $\bar{x}$ solve

$$L_1(w) = \min_{x} \left\{ c^0x^0 + \sum_{k=1}^{p} (c^k - w^k)x^k \ : \ \sum_{i=0}^{p} A^i x^i = r, \quad 0 \leq x^i \leq u^i \text{ for } i=0,\dots,p \right\}.$$

Let $\bar{y}$ solve

$$L_2(w) = \sum_{k=1}^{p} \min_{y} \left\{ \sum_{j=1}^{n_k} w^k_j y^k_j \ : \ \sum_{j=1}^{n_k} e^k_j y^k_j \leq b_k, \quad 0 \leq y^k \leq u^k \right\}.$$

If $\bar{x} = \bar{y}$, terminate with $\bar{x}$ an optimal solution to the problem (*NPG*).

Let LBND = $L_1(w) + L_2(w)$.

If (UBND - LBND) $\leq \epsilon\,|\text{UBND}|$, terminate with $\bar{x}$ near optimal; otherwise, let $\eta = \left\langle \bar{y}^1 - \bar{x}^1, \ \dots, \ \bar{y}^p - \bar{x}^p \right\rangle^t$.

Step 3 : Move to New Point.

$w = w + d\eta$.

Adjust the step size d.

Step 4 : Repeat the Process.

Go to step 2.

By taking note of the structure of $L_1(w)$ one can see that once an optimal solution is found for a given w, then this optimal solution remains feasible to the pure network problem for all other values of w.

3.2 <u>The Upper Bound for Problem *NPG*</u>

An upper bound on the objective function of problem (*NPG*) can be obtained by using a decomposition approach. After artificial variables have been added, (*NPG*) becomes

$$
(NPGA) \qquad \begin{array}{lll}
\min & \sum_{i=0}^{p} c^i x^i + Ms & \\
\text{s.t.} & \sum_{i=0}^{p} A^i x^i + Is = r & \\
 & \sum_{j=1}^{n_k} e^k_j \, x^k_j \le b_k & \text{for } k=1,\ldots,p \\
 & 0 \le x^i \le u^i & \text{for } i=0,\ldots,p \\
 & s \ge 0 &
\end{array}
$$

where M is a vector of large positive numbers, and s is a vector of artificial variables. It will be shown that an alternate formulation of *(NPGA)* can be obtained by decomposing problem *(NPGA)* is given by,

(NPGD)
$$\begin{array}{lll} \min & g(y) & \\ \text{s.t.} & \displaystyle\sum_{j=1}^{n_k} e_j^k\, y_j^k = b_k & \text{for } k=1,\dots,p \\ & 0 \le y^k \le u^k & \text{for } k=1,\dots,p \end{array}$$

where for any vector $\mathbf{y} = \left(\mathbf{y}^1, \dots, \mathbf{y}^p \right)^t$,

$$\begin{array}{llll} g(\mathbf{y}) = \min & \displaystyle\sum_{i=0}^{p} \mathbf{c}^i \mathbf{x}^i + \mathbf{Ms} & & \\ \quad \text{s.t.} & & & \\ & \displaystyle\sum_{i=0}^{p} \mathbf{A}^i \mathbf{x}^i + \mathbf{Is} = \mathbf{r} & & (\pi) \\ & \mathbf{x}^0 \le \mathbf{u}^0 & & (\sigma^0) \\ & \mathbf{x}^0 \ge 0 & & (\mu^0) \\ & \mathbf{x}_j^k \le \mathbf{y}_j^k & \text{for } j \in J_+^k & (\sigma_j^k) \\ & \mathbf{x}_j^k \le \mathbf{u}_j^k & \text{for } j \in J_-^k & (\mu_j^k) \\ & \mathbf{x}_j^k \ge \mathbf{y}_j^k & \text{for } j \in J_-^k & (\sigma_j^k) \\ & \mathbf{x}_j^k \ge 0 & \text{for } j \in J_+^k & (\mu_j^k) \\ & \mathbf{s} \ge 0 & & \end{array} \qquad (32)$$

where for each k, $1 \le k \le p$, $J_+^k = \{ j : e_j^k > 0 \}$ and $J_-^k = \{ j : e_j^k < 0 \}$. Furthermore, let π, σ^0, μ^0, σ_j^k and μ_j^k be the dual variables associated with each type of constraint in (32). By duality theory, g(y) =

$$\max \pi r + \sigma^0 u^0 + \sigma^1 y^1 + \dots + \sigma^p y^p + \sum_{k=1}^{p} \sum_{j \in J_-^k} \mu_j^k u_j^k$$

s.t.

$$\pi A^0 + \sigma^0 + \mu^0 = c^0$$

$$\pi A^k + \sigma^k + \mu^k = c^k \qquad \text{for } k=1,\dots,p$$

$$\pi \le M , \; \sigma^0 \le 0 , \; \mu^0 \ge 0$$

$$\sigma_j^k \le 0 \quad \text{for } k=1,\dots,p,\ j \in J_+^k$$

$$\sigma_j^k \ge 0 \quad \text{for } k=1,\dots,p,\ j \in J_-^k$$

$$\mu_j^k \le 0 \quad \text{for } k=1,\dots,p,\ j \in J_-^k$$

$$\mu_j^k \ge 0 \quad \text{for } k=1,\dots,p,\ j \in J_+^k$$

The following proposition justifies the decomposition of problem (*NPG*) into problem (*NPGD*).

Proposition 6 : The decomposed problem (*NPGD*) is equivalent to problem (*NPG*).

Proof :

For each k, $1 \le k \le p$, let

$S_1^k = \{ x^k : E^k x^k \le b_k,\ 0 \le x^k \le u^k \} \in \mathbb{R}^{n_k}$, and

$S_2^k = \{ (x^k, y^k) : E^k y^k = b_k,\ 0 \le y^k \le u^k ,\ 0 \le x_j^k \le y_j^k$ for $j \in J_+^k,\ y_j^k \le x_j^k \le u_j^k$ for $j \in J_-^k \} \in \mathbb{R}^{2n_k}$.

Let $x^k \in S_1^k$, i.e., $0 \le x^k \le u^k$ and $\sum_{j=1}^{n_k} e_j^k x_j^k \le b_k$.

case 1 : If $\sum_{j \in J_-^k} e_j^k x_j^k = 0$, then let $y_j^k = x_j^k$ for $j \in J_-^k$.

Since $\sum_{j \in J_+^k} e_j^k u_j^k > b_k$ (otherwise the kth single constraint bounded variable LP is satisfied for any x^k with $0 \le x^k \le u^k$ and hence can be removed from the problem), by increasing some or all of the x_j^k to their upper bound we can find a vector y^k such that,

$$\sum_{j=1}^{n_k} e_j^k y_j^k = b_k \ , \ 0 \le y^k \le u^k \ , \ 0 \le x_j^k \le y_j^k \text{ for } j \in J_+^k,$$

and $y_j^k = x_j^k$ for $j \in J_-^k$.

case 2 : If $\sum_{j \in J_-^k} e_j^k x_j^k \ne 0$, then by decreasing some or all of the x_j^k (for $j \in J_-^k$) to zero and by increasing some or all of the x_j^k (for $j \in J_+^k$) to their upper bounds [notice that this is always possible since the worst scenario would be the case when x_j^k (for $j \in J_-^k$) is decreased until $\sum_{j \in J_-^k} e_j^k x_j^k = 0$ and then we have case 1], we can find a vector y^k such that

$$\sum_{j=1}^{n_k} e_j^k y_j^k = b_k \ , \ 0 \le y^k \le u^k \ , \ 0 \le x_j^k \le y_j^k \text{ for } j \in J_+^k,$$

and $y_j^k \le x_j^k \le u_j^k$ for $j \in J_-^k$.

Hence $x^k \in S_1^k$ implies that $(x^k, y^k) \in S_2^k$.

Now suppose that $(\mathbf{x}^k,\mathbf{y}^k) \in S_2^k$, i.e., $\sum_{j=1}^{n_k} e_j^k y_j^k = b_k$, $0 \leq \mathbf{y}^k \leq \mathbf{u}^k$, $0 \leq x_j^k \leq y_j^k$ for $j \in J_+^k$, and $y_j^k \leq x_j^k \leq u_j^k$ for $j \in J_-^k$, then

for $j \in J_+^k$, $0 \leq x_j^k \leq y_j^k \Rightarrow 0 \leq e_j^k x_j^k \leq e_j^k y_j^k$

$$\Rightarrow \sum_{j \in J_+^k} e_j^k x_j^k \leq \sum_{j \in J_+^k} e_j^k y_j^k;$$

for $j \in J_-^k$, $y_j^k \leq x_j^k \leq u_j^k \Rightarrow e_j^k u_j^k \leq e_j^k x_j^k \leq e_j^k y_j^k$

$$\Rightarrow \sum_{j \in J_-^k} e_j^k x_j^k \leq \sum_{j \in J_-^k} e_j^k y_j^k.$$

Thus, $\sum_{j=1}^{n_k} e_j^k x_j^k \leq \sum_{j=1}^{n_k} e_j^k y_j^k = b_k$ and $0 \leq \mathbf{x}^k \leq \mathbf{u}^k$.

Hence, $(\mathbf{x}^k,\mathbf{y}^k) \in S_2^k$ implies that $\mathbf{x}^k \in S_1^k$.

So solving

$$\begin{aligned}
\min \quad & \sum_{i=0}^{p} c^i \mathbf{x}^i + \mathbf{Ms} \\
\text{s.t.} \quad & \\
& \sum_{i=0}^{p} \mathbf{A}^i \mathbf{x}^i + \mathbf{Is} = \mathbf{r} \\
& \mathbf{E}^k \mathbf{x}^k \leq b_k \\
& 0 \leq \mathbf{x}^0 \leq \mathbf{u}^0 \\
& 0 \leq \mathbf{x}^k \leq \mathbf{u}^k \qquad \text{for } k=1,\ldots,p \\
& \mathbf{s} \geq 0
\end{aligned}$$

is equivalent to solving

$$\begin{array}{lll}
\min & \sum_{i=0}^{p} \mathbf{c}^i \mathbf{x}^i + \mathbf{Ms} & \\
\text{s.t.} & & \\
& \sum_{i=0}^{p} \mathbf{A}^i \mathbf{x}^i + \mathbf{Is} = \mathbf{r} & \\
& 0 \le \mathbf{x}^0 \le \mathbf{u}^0 & \\
& \mathbf{E}^k \mathbf{y}^k = \mathrm{b}_k & \text{for } k=1,\dots,p \\
& 0 \le \mathbf{y}^k \le \mathbf{u}^k & \text{for } k=1,\dots,p \\
& 0 \le \mathrm{x}_j^k \le \mathrm{y}_j^k & \text{for } k=1,\dots,p,\ j \in J_+^k \\
& \mathrm{y}_j^k \le \mathrm{x}_j^k \le \mathrm{u}_j^k & \text{for } k=1,\dots,p,\ j \in J_-^k \\
& \mathbf{s} \ge 0 &
\end{array}$$

which in turn is equivalent to solving

$$\begin{array}{lll}
\min_{\mathbf{y}} \ \min_{\mathbf{x}} & \sum_{i=0}^{p} \mathbf{c}^i \mathbf{x}^i + \mathbf{Ms} & \\
\text{s.t.} & & \\
& \sum_{i=0}^{p} \mathbf{A}^i \mathbf{x}^i + \mathbf{Is} = \mathbf{r} & \\
& 0 \le \mathbf{x}^0 \le \mathbf{u}^0 & \\
& 0 \le \mathrm{x}_j^k \le \mathrm{y}_j^k & \text{for } k=1,\dots,p,\ j \in J_+^k \\
& \mathrm{y}_j^k \le \mathrm{x}_j^k \le \mathrm{u}_j^k & \text{for } k=1,\dots,p,\ j \in J_-^k \\
& \mathbf{E}^k \mathbf{y}^k = \mathrm{b}_k & \text{for } k=1,\dots,p \\
& 0 \le \mathbf{y}^k \le \mathbf{u}^k & \text{for } k=1,\dots,p \\
& \mathbf{s} \ge 0 &
\end{array}$$

this is just problem (*NPGD*)

$$\begin{array}{lll} \min & g(y) & \\ \text{s.t.} & \sum_{j=1}^{n_k} e_j^k \, y_j^k = b_k & \text{for } k=1,\dots,p \\ & 0 \le y^k \le u^k & \text{for } k=1,\dots,p \end{array}$$

where g(y) is defined by (32).

Hence, problem (*NPGD*) is equivalent to problem (*NPG*). □

Proposition 6 establishes that the decomposition assures the satisfaction of the GUB constraints. We now prove that the objective function g(y) of the problem (*NPGD*) is a convex function.

<u>Proposition 7</u> : The function g(y) is convex over

$$\Gamma = \left\{ y : \sum_{j=1}^{n_k} e_j^k \, y_j^k = b_k, \ 0 \le y^k \le u^k \text{ for } k=1,\dots,p \right\}.$$

<u>Proof</u>:

Let $\bar{y}$ and $\hat{y}$ be chosen so that $\bar{y} \in \Gamma$ and $\hat{y} \in \Gamma$. Choose $\alpha \in [0,1]$. Let

$$S = \Big\{ (\pi, \sigma^0, \sigma^1, \dots, \sigma^p, \mu^0, \mu^1, \dots, \mu^p) : \pi \le M, \ \sigma^0 \le 0,$$
$$\mu^0 \ge 0 \text{ and for } k=1,\dots,p \ \sigma_j^k \le 0 \text{ for } j \in J_+^k, \ \sigma_j^k \ge 0 \text{ for}$$
$$j \in J_-^k, \ \mu_j^k \le 0 \text{ for } j \in J_-^k, \ \mu_j^k \ge 0 \text{ for } j \in J_+^k \Big\}.$$

Then

$$g(\alpha\bar{y} + (1-\alpha)\hat{y}) = g(\alpha\bar{y}^1 + (1-\alpha)\hat{y}^1 , \ldots , \alpha\bar{y}^p + (1-\alpha)\hat{y}^p)$$

$$= \max\Big\{\pi r + \sigma^0u^0 + \sum_{k=1}^{p}\sigma^k[\alpha\bar{y}^k + (1-\alpha)\hat{y}^k] + \sum_{i=1}^{p}\mu^iu^i :$$
$$\pi A^i + \sigma^i + \mu^i \leq c^i \text{ for } i=0,\ldots,p,$$
$$(\pi,\sigma^0,\sigma^1,\ldots,\sigma^p,\mu^0,\mu^1,\ldots,\mu^p) \in S\Big\}$$

$$= \max\Big\{ \alpha[\pi r + \sigma^0u^0 + \sum_{k=1}^{p}\sigma^k\bar{y}^k + \sum_{i=1}^{p}\mu^iu^i] + (1-\alpha)[\pi r +$$
$$\sigma^0u^0 + \sum_{k=1}^{p}\sigma^k\hat{y}^k + \sum_{i=1}^{p}\mu^iu^i]: \pi A^i + \sigma^i + \mu^i \leq c^i \text{ for}$$
$$i=0,\ldots,p, \ (\pi,\sigma^0,\sigma^1,\ldots,\sigma^p,\mu^0,\mu^1,\ldots,\mu^p) \in S\Big\}$$

$$\leq \alpha\max\Big\{\pi r + \sigma^0u^0 + \sum_{k=1}^{p}\sigma^k\bar{y}^k + \sum_{i=1}^{p}\mu^iu^i]:$$
$$\pi A^i + \sigma^i + \mu^i \leq c^i \text{ for } i=0,\ldots,p,$$
$$(\pi,\sigma^0,\sigma^1,\ldots,\sigma^p,\mu^0,\mu^1,\ldots,\mu^p) \in S\Big\}$$
$$+ (1-\alpha)\max\Big\{\pi r + \sigma^0u^0 + \sum_{k=1}^{p}\sigma^k\hat{y}^k + \sum_{i=1}^{p}\mu^iu^i]:$$
$$\pi A^i + \sigma^i + \mu^i \leq c^i \text{ for } i=0,\ldots,p,$$
$$(\pi,\sigma^0,\sigma^1,\ldots,\sigma^p,\mu^0,\mu^1,\ldots,\mu^p) \in S\Big\}$$

$$= \alpha g(\bar{y}) + (1-\alpha)g(\hat{y}).$$

Hence $g(y)$ is convex over Γ. □

The above proposition implies that problem (*NPGD*) is a convex program with linear constraints. Since g is also piecewise linear (Held, Wolfe and Crowder (1974)), problem (*NPGD*) may be solved using a specialization of the subgradient optimization algorithm. The algorithm begins with an initial feasible solution, i.e., $y \in \Gamma = \left\{ y : \sum_{j=1}^{n_k} e_j^k y_j^k = b_k,\ 0 \leq y^k \leq u^k \text{ for } k=1,\dots,p \right\}$. The procedure involves minimizing the directional derivative of g subject to feasibility restrictions, which uses a subgradient to determine a direction in which y can be moved or shows that y is optimal. If y is not optimal, a step is taken in the direction, a new feasible point is determined, and the process is repeated.

To apply the subgradient algorithm to (*NPGD*) one must have a way for determining a subgradient of g(y) at a given point. Consider the following proposition.

<u>Proposition 8</u> : Let $\bar{y} \in \Gamma$, let $(\bar{\pi}, \bar{\sigma}^0, \bar{\sigma}^1, \dots, \bar{\sigma}^p, \bar{\mu}^0, \bar{\mu}^1, \dots, \bar{\mu}^p)$ be the optimal dual solution to $g(\bar{y})$. Then $(\bar{\sigma}^1, \dots, \bar{\sigma}^p)$ is a subgradient of g at $\bar{y}$.

<u>Proof</u>:

Let y be any vector in Γ, and let $(\pi, \sigma^0, \sigma^1, \dots, \sigma^p, \mu^0, \mu^1, \dots, \mu^p)$ denote the optimal dual solution to g at y. Then

$$g(y)-g(\bar{y}) = \left(\pi r + \sigma^0 u^0 + \sum_{k=1}^{p} \sigma^k y^k + \sum_{i=1}^{p} \mu^i u^i \right) - \left(\bar{\pi} r + \bar{\sigma}^0 u^0 + \sum_{k=1}^{p} \bar{\sigma}^k \bar{y}^k + \sum_{i=1}^{p} \bar{\mu}^i u^i \right)$$

$$\geq \left\langle \bar{\pi} r + \bar{\sigma}^0 u^0 + \sum_{k=1}^{p} \bar{\sigma}^k y^k + \sum_{i=1}^{p} \bar{\mu}^i u^i \right\rangle - \left\langle \bar{\pi} r + \bar{\sigma}^0 u^0 + \sum_{k=1}^{p} \bar{\sigma}^k \bar{y}^k + \sum_{i=1}^{p} \bar{\mu}^i u^i \right\rangle$$

$$= \sum_{k=1}^{p} \bar{\sigma}^k (y^k - \bar{y}^k)$$

$$= (\bar{\sigma}^1, \ldots, \bar{\sigma}^p)\ (y - \bar{y}).$$

Hence, $(\bar{\sigma}^1, \ldots, \bar{\sigma}^p)$ is a subgradient of g at $\bar{y}$. □

Therefore the subgradient is available from the solution of the network problem. Consider solving the pure network problem

$$\begin{aligned}
\min \quad & \sum_{i=0}^{p} c^i x^i + Ms \\
\text{s.t.} \quad & \\
& \sum_{i=0}^{p} A^i x^i + Is = r \\
& 0 \leq x^0 \leq u^0 \\
& 0 \leq x^k_j \leq y^k_j \qquad \text{for } k=1,\ldots,p,\ j \in J^k_+ \\
& y^k_j \leq x^k_j \leq u^k_j \qquad \text{for } k=1,\ldots,p,\ j \in J^k_- \\
& s \geq 0
\end{aligned}$$

If $s \neq 0$ in the optimal solution, then the problem has no feasible solution. Otherwise denote the optimal solution by $\bar{x}$. Let $(\pi, \sigma^0, \sigma^1, \ldots, \sigma^p, \mu^0, \mu^1, \ldots, \mu^p)$ solve

$\max\left\{\pi r + \sigma^0 u^0 + \sum_{k=1}^{p} \sigma^k y^k + \sum_{i=1}^{p} \mu^i u^i : \pi A^i + \sigma^i + \mu^i \leq c^i\right.$ for $i=0,\ldots,p$, $\pi \leq M$, $\sigma^0 \leq 0$, $\mu^0 \geq 0$ and for $k=1,\ldots,p$ $\sigma_j^k \leq 0$ for $j \in J_+^k$, $\sigma_j^k \geq 0$ for $j \in J_-^k$, $\mu_j^k \leq 0$ for $\left. j \in J_-^k , \mu_j^k \geq 0 \text{ for } j \in J_+^k\right\}$.

Note that for each k, $1 \leq k \leq p$, $\sigma^k + \mu^k = c^k - \pi A^k$ and $\sigma_j^k \leq 0$ for $j \in J_+^k$, $\sigma_j^k \geq 0$ for $j \in J_-^k$, $\mu_j^k \leq 0$ for $j \in J_-^k$, and $\mu_j^k \geq 0$ for $j \in J_+^k$. Thus if the jth column of A^k, denoted by A_j^k, corresponds to x_j^k and its associated arc is incident from node f(j) to node t(j), we have that

for $j \in J_+^k$

$$\sigma_j^k \leq \min \{ c_j^k - \pi A_j^k , 0 \} = \min \left\{c_j^k + \pi_{f(j)} - \pi_{t(j)} , 0 \right\}$$

and for $j \in J_-^k$

$$\sigma_j^k \geq \max \{ c_j^k - \pi A_j^k , 0 \} = \max \left\{c_j^k + \pi_{f(j)} - \pi_{t(j)} , 0 \right\}$$

But,

for all j, $\bar{x}_j^k$ basic $\Rightarrow c_j^k + \pi_{f(j)} - \pi_{t(j)} = 0 \Rightarrow \sigma_j^k \leq 0$;

for $j \in J_+^k$, $\bar{x}_j^k = 0$ and nonbasic $\Rightarrow c_j^k + \pi_{f(j)} - \pi_{t(j)} \geq 0$ $\Rightarrow \sigma_j^k \leq 0$;

$\bar{x}_j^k = y_j^k$ and nonbasic $\Rightarrow c_j^k + \pi_{f(j)} - \pi_{t(j)} \leq 0 \Rightarrow \sigma_j^k \leq c_j^k + \pi_{f(j)} - \pi_{t(j)}$;

for $j \in J_-^k$, $\bar{x}_j^k = y_j^k$ and nonbasic $\Rightarrow c_j^k + \pi_{f(j)} - \pi_{t(j)} \geq 0 \Rightarrow \sigma_j^k \geq c_j^k + \pi_{f(j)} - \pi_{t(j)}$;

$\bar{x}_j^k = u_j^k$ and nonbasic $\Rightarrow c_j^k + \pi_{f(j)} - \pi_{t(j)} \leq 0 \Rightarrow \sigma_j^k \geq 0$.

Hence, for all j, $1 \leq j \leq n_k$, one solution for σ_j^k is

$$\sigma_j^k = \begin{cases} c_j^k + \pi_{f(j)} - \pi_{t(j)} & \text{if } \bar{x}_j^k = y_j^k \text{ and nonbasic} \\ 0 & \text{otherwise} \end{cases}$$

Thus, in using the subgradient optimization algorithm for the problem (*NPGD*) at each point **y**, the subgradient can be calculated directly using the above development.

It is possible that moving a step along the subgradient yields a point which does not belong to the set $\Gamma = \left\{ \mathbf{y} : \sum_{j=1}^{n_k} e_j^k y_j^k = b_k,\ 0 \leq \mathbf{y}^k \leq \mathbf{u}^k \text{ for } k=1,\ldots,p \right\}$. This point is projected onto the set Γ by means of a projection operator. What remains to complete the procedure is to find an efficient way for projecting an infeasible point onto Γ.

Given a point $\bar{\mathbf{y}} \notin \Gamma$, we must solve the following problem:

$$\min\left\{\|y - \bar{y}\| : \sum_{j=1}^{n_k} e_j^k y_j^k = b_k,\ 0 \le y^k \le u^k \text{ for } k=1,\ldots,p\right\}$$

$$= \min\left\{\sum_{k=1}^{p}\sum_{j=1}^{n_k}(y_j^k - \bar{y}_j^k)^2 : \sum_{j=1}^{n_k} e_j^k y_j^k = b_k,\ 0 \le y^k \le u^k \text{ for } k=1,\ldots,p\right\}.$$

But this program decomposes on k, therefore we must solve

$$\min\left\{\sum_{j=1}^{n_k}(y_j^k - \bar{y}_j^k)^2 : \sum_{j=1}^{n_k} e_j^k y_j^k = b_k,\ 0 \le y^k \le u^k\right\} \qquad (33)$$

for each GUB constraint k=1,...,p.

For each k, if $\sum_{j=1}^{n_k} e_j^k \bar{y}_j^k = b_k$ and $0 \le \bar{y}^k \le u^k$ then $\bar{y}_j^k$ is within the subset of feasible region Γ_k where $\Gamma_k = \left\{ y^k : \sum_{j=1}^{n_k} e_j^k y_j^k = b_k,\ 0 \le y^k \le u^k \right\}$ and $\Gamma = \mathop{X}_{k=1}^{p} \Gamma_k$

(X denotes the k cartesian products); thus let $y_j^k = \bar{y}_j^k$ for $j=1,\ldots,n_k$. Otherwise, (33) takes the form

$$\min \quad \frac{1}{2} y^k D^k y^k - 2\, \bar{y}^k y^k \tag{34}$$

$$\sum_{j=1}^{n_k} e_j^k\, y_j^k - b_k = 0 \qquad (\lambda^k) \tag{35}$$

$$y_j^k - u_j^k \leq 0 \qquad (\omega_j^k) \tag{36}$$

$$- y_j^k \leq 0 \qquad (\nu_j^k) \tag{37}$$

where $D^k=2I$, and λ^k, ω_j^k and ν_j^k are the Kuhn–Tucker multipliers associated with the three types of constraints. The Kuhn–Tucker conditions for (34)–(37) are:

$$2y_j^k - 2\bar{y}_j^k + \omega_j^k - \nu_j^k + e_j^k\, \lambda^k = 0 \quad \text{for all } j, \tag{38}$$

$$\omega_j^k\, (y_j^k - u_j^k) = 0 \quad \text{for all } j, \tag{39}$$

$$\nu_j^k\, y_j^k = 0 \quad \text{for all } j, \tag{40}$$

$$\nu_j^k,\ \omega_j^k \geq 0 \quad \text{for all } j \tag{41}$$

plus (35), (36) and (37).

Consider the following solution as a function of λ^k:

$$\left.\begin{aligned}
y_j^k(\lambda^k) &= \begin{cases} \dfrac{1}{e_j^k}\max\left\{\min\left(e_j^k\bar{y}_j^k-(e_j^k)^2\dfrac{\lambda^k}{2}, e_j^k u_j^k\right), 0\right\} & \text{if } e_j^k>0 \\ \dfrac{1}{e_j^k}\min\left\{\max\left(e_j^k\bar{y}_j^k-(e_j^k)^2\dfrac{\lambda^k}{2}, e_j^k u_j^k\right), 0\right\} & \text{if } e_j^k<0 \end{cases} \\
\omega_j^k(\lambda^k) &= \begin{cases} e_j^k \max\left\{\dfrac{2\bar{y}_j^k}{e_j^k} - \lambda^k - \dfrac{2u_j^k}{e_j^k}, 0\right\} & \text{if } e_j^k>0 \\ e_j^k \min\left\{\dfrac{2\bar{y}_j^k}{e_j^k} - \lambda^k - \dfrac{2u_j^k}{e_j^k}, 0\right\} & \text{if } e_j^k<0 \end{cases} \\
\nu_j^k(\lambda^k) &= \begin{cases} e_j^k \max\left\{-\dfrac{2\bar{y}_j^k}{e_j^k} + \lambda^k, 0\right\} & \text{if } e_j^k>0 \\ e_j^k \min\left\{-\dfrac{2\bar{y}_j^k}{e_j^k} + \lambda^k, 0\right\} & \text{if } e_j^k<0 \end{cases}
\end{aligned}\right\} (42)$$

For any selection of λ^k, this solution clearly satisfies (36), (37) and (41). The following propositions show that (42) will also satisfy (38), (39) and (40).

<u>Proposition 9</u> : The solution given by (42) satisfies (39).

<u>Proof</u>:

case 1. $\dfrac{2\bar{y}_j^k}{e_j^k} - \lambda^k - \dfrac{2u_j^k}{e_j^k} < 0$

a. $e_j^k > 0 \Rightarrow \omega_j^k = 0 \Rightarrow \omega_j^k(y_j^k - u_j^k) = 0.$

b. $e_j^k < 0 \Rightarrow \bar{y}_j^k - e_j^k \frac{\lambda^k}{2} > u_j^k \Rightarrow e_j^k \bar{y}_j^k - (e_j^k)^2 \frac{\lambda^k}{2} < e_j^k u_j^k < 0$

$\Rightarrow y_j^k = u_j^k \Rightarrow \omega_j^k(y_j^k - u_j^k) = 0.$

case 2. $\dfrac{2\bar{y}_j^k}{e_j^k} - \lambda^k - \dfrac{2u_j^k}{e_j^k} \geq 0$

a. $e_j^k > 0 \Rightarrow e_j^k \bar{y}_j^k - (e_j^k)^2 \frac{\lambda^k}{2} \geq e_j^k u_j^k > 0 \Rightarrow y_j^k = u_j^k$

$\Rightarrow \omega_j^k(y_j^k - u_j^k) = 0.$

b. $e_j^k < 0 \Rightarrow \omega_j^k = 0 \Rightarrow \omega_j^k(y_j^k - u_j^k) = 0.$

Hence, the solution given by (42) satisfies (39). □

Proposition 10 : The solution given by (42) satisfies (40).

Proof :

case 1. $- \dfrac{2\bar{y}_j^k}{e_j^k} + \lambda^k < 0$

a. $e_j^k > 0 \Rightarrow \nu_j^k = 0 \Rightarrow \nu_j^k y_j^k = 0.$

b. $e^k_j < 0 \Rightarrow e^k_j\bar{y}^k_j-(e^k_j)^2\frac{\lambda^k}{2} > 0 \Rightarrow y^k_j = 0 \Rightarrow \nu^k_j y^k_j = 0.$

case 2. $-\frac{2\bar{y}^k_i}{e^k_j} + \lambda^k \geq 0$

a. $e^k_j > 0 \Rightarrow e^k_j\bar{y}^k_j-(e^k_j)^2\frac{\lambda^k}{2} \leq 0 \Rightarrow y^k_j = 0 \Rightarrow \nu^k_j y^k_j = 0.$

b. $e^k_j < 0 \Rightarrow \nu^k_j = 0 \Rightarrow \nu^k_j y^k_j = 0.$

Hence, the solution given by (42) satisfies (40). □

<u>Proposition 11</u> : The solution given by (42) satisfies (38).

<u>Proof</u> :

case 1. $e^k_j > 0$

a. $e^k_j\bar{y}^k_j-(e^k_j)^2\frac{\lambda^k}{2} \geq e^k_j u^k_j > 0$

$\Rightarrow y^k_j = u^k_j, \frac{2\bar{y}^k_i}{e^k_j} -\lambda^k- \frac{2u^k_i}{e^k_j} \geq 0,$

and $\frac{2\bar{y}^k_i}{e^k_j} -\lambda^k > 0.$

$$\frac{2\bar{y}^k_j}{e^k_j} - \lambda^k - \frac{2u^k_j}{e^k_j} \geq 0 \Rightarrow \omega^k_j = 2\bar{y}^k_j - e^k_j\lambda^k - 2u^k_j.$$

$$\frac{2\bar{y}^k_j}{e^k_j} - \lambda^k > 0 \Rightarrow \nu^k_j = 0.$$

Thus

$$2y^k_j - 2\bar{y}^k_j + \omega^k_j - \nu^k_j + e^k_j\lambda^k$$

$$= 2u^k_j - 2\bar{y}^k_j + (2\bar{y}^k_j - e^k_j\lambda^k - 2u^k_j) - 0 + e^k_j\lambda^k = 0.$$

b. $0 < e^k_j\bar{y}^k_j - (e^k_j)^2\frac{\lambda^k}{2} < e^k_j u^k_j$

$$\Rightarrow y^k_j = \bar{y}^k_j - e^k_j\frac{\lambda^k}{2},\quad \frac{2\bar{y}^k_j}{e^k_j} - \lambda^k - \frac{2u^k_j}{e^k_j} < 0,$$

and $\dfrac{2\bar{y}^k_j}{e^k_j} - \lambda^k > 0.$

$$\frac{2\bar{y}^k_j}{e^k_j} - \lambda^k - \frac{2u^k_j}{e^k_j} < 0 \Rightarrow \omega^k_j = 0.$$

$$\frac{2\bar{y}^k_j}{e^k_j} - \lambda^k > 0 \Rightarrow \nu^k_j = 0.$$

Thus

$$2y_j^k - 2\bar{y}_j^k + \omega_j^k - \nu_j^k + e_j^k \lambda^k$$

$$= 2(\bar{y}_j^k - e_j^k \frac{\lambda^k}{2}) - 2\bar{y}_j^k + 0 - 0 + e_j^k \lambda^k = 0.$$

c. $e_j^k \bar{y}_j^k - (e_j^k)^2 \frac{\lambda^k}{2} \leq 0$

$$\Rightarrow y_j^k = 0 \text{ and } \frac{2\bar{y}_i^k}{e_j^k} - \lambda^k \leq 0.$$

$$\frac{2\bar{y}_i^k}{e_j^k} - \lambda^k \leq 0 \Rightarrow \frac{2\bar{y}_i^k}{e_j^k} - \lambda^k - \frac{2u_i^k}{e_j^k} \leq 0 \text{ and}$$

$$\nu_j^k = -2\bar{y}_j^k + e_j^k \lambda^k.$$

$$\frac{2\bar{y}_i^k}{e_j^k} - \lambda^k - \frac{2u_i^k}{e_j^k} \leq 0 \Rightarrow \omega_j^k = 0.$$

Thus

$$2y_j^k - 2\bar{y}_j^k + \omega_j^k - \nu_j^k + e_j^k \lambda^k$$

$$= 0 - 2\bar{y}_j^k + 0 - (-2\bar{y}_j^k + e_j^k \lambda^k) + e_j^k \lambda^k = 0.$$

case 2. $e_j^k < 0$

a. $e_j^k u_j^k < 0 \leq e_j^k \bar{y}_j^k - (e_j^k)^2 \frac{\lambda^k}{2}$

$$\Rightarrow y_j^k = 0, \ \frac{2\bar{y}_j^k}{e_j^k} - \lambda^k - \frac{2u_j^k}{e_j^k} > 0, \text{ and } \frac{2\bar{y}_j^k}{e_j^k} - \lambda^k \geq 0.$$

$$\frac{2\bar{y}_j^k}{e_j^k} - \lambda^k - \frac{2u_j^k}{e_j^k} > 0 \Rightarrow \omega_j^k = 0.$$

$$\frac{2\bar{y}_j^k}{e_j^k} - \lambda^k \geq 0 \Rightarrow \nu_j^k = -2\bar{y}_j^k + e_j^k \lambda^k.$$

Thus

$$2y_j^k - 2\bar{y}_j^k + \omega_j^k - \nu_j^k + e_j^k \lambda^k$$

$$= 0 - 2\bar{y}_j^k + 0 - (-2\bar{y}_j^k + e_j^k \lambda^k) + e_j^k \lambda^k = 0.$$

b. $e_j^k u_j^k \leq e_j^k \bar{y}_j^k - (e_j^k)^2 \frac{\lambda^k}{2} < 0$

$$\Rightarrow y_j^k = \bar{y}_j^k - e_j^k \frac{\lambda^k}{2}, \ \frac{2\bar{y}_j^k}{e_j^k} - \lambda^k - \frac{2u_j^k}{e_j^k} \geq 0,$$

$$\text{and } \frac{2\bar{y}_j^k}{e_j^k} - \lambda^k < 0.$$

$$\frac{2\bar{y}^k_j}{e^k_j} - \lambda^k - \frac{2u^k_j}{e^k_j} \geq 0 \Rightarrow \omega^k_j = 0.$$

$$\frac{2\bar{y}^k_j}{e^k_j} - \lambda^k < 0 \Rightarrow \nu^k_j = 0.$$

Thus

$$2y^k_j - 2\bar{y}^k_j + \omega^k_j - \nu^k_j + e^k_j\lambda^k$$

$$= 2(\bar{y}^k_j - e^k_j\frac{\lambda^k}{2}) - 2\bar{y}^k_j + 0 - 0 + e^k_j\lambda^k = 0.$$

c. $e^k_j\bar{y}^k_j - (e^k_j)^2\frac{\lambda^k}{2} < e^k_j u^k_j < 0$

$$\Rightarrow y^k_j = u^k_j,\ \frac{2\bar{y}^k_j}{e^k_j} - \lambda^k - \frac{2u^k_j}{e^k_j} < 0 \text{ and } \frac{2\bar{y}^k_j}{e^k_j} - \lambda^k < 0.$$

$$\frac{2\bar{y}^k_j}{e^k_j} - \lambda^k - \frac{2u^k_j}{e^k_j} < 0 \Rightarrow \omega^k_j = 2\bar{y}^k_j - e^k_j\lambda^k - 2u^k_j.$$

$$\frac{2\bar{y}^k_j}{e^k_j} - \lambda^k < 0 \Rightarrow \nu^k_j = 0.$$

Thus

$$2y^k_j - 2\bar{y}^k_j + \omega^k_j - \nu^k_j + e^k_j\ \lambda^k$$

$$= 2u_j^k - 2\bar{y}_j^k + (2\bar{y}_j^k - e_j^k\lambda^k - 2u_j^k) - 0 + e_j^k\lambda^k = 0.$$

Hence, the solution given by (42) solves (38). □

Hence to solve (34)–(37) one need only find the appropriate λ^k such that (35) is satisfied. Let

$$f(\lambda^k) = \sum_{j \in J_+^k} \max\left\{\min\left(e_j^k\bar{y}_j^k - (e_j^k)^2\frac{\lambda^k}{2},\ e_j^k u_j^k\right),\ 0\right\}$$
$$+ \sum_{j \in J_-^k} \min\left\{\max\left(e_j^k\bar{y}_j^k - (e_j^k)^2\frac{\lambda^k}{2},\ e_j^k u_j^k\right),\ 0\right\}$$

where $J_+^k = \left\{j : e_j^k > 0\right\}$ and $J_-^k = \left\{j : e_j^k < 0\right\}$ for each k, $1 \leq k \leq p$. Then we must find λ^k such that $f(\lambda^k) = b_k$. So $y_j^k(\lambda^k)$ may be expressed as follows:

for $e_j^k > 0$,

$$y_j^k(\lambda^k) = \begin{cases} u_j^k & \lambda^k \leq \dfrac{2(\bar{y}_j^k - u_j^k)}{e_j^k} \\ \bar{y}_j^k - e_j^k\dfrac{\lambda^k}{2} & \dfrac{2(\bar{y}_j^k - u_j^k)}{e_j^k} < \lambda^k \leq \dfrac{2\bar{y}_j^k}{e_j^k} \\ 0 & \lambda^k > \dfrac{2\bar{y}_j^k}{e_j^k} \end{cases}$$

for $e_j^k < 0$,

$$y_j^k(\lambda^k) = \begin{cases} 0 & \lambda^k \le \dfrac{2\bar{y}_j^k}{e_j^k} \\ \bar{y}_j^k - e_j^k \dfrac{\lambda^k}{2} & \dfrac{2\bar{y}_j^k}{e_j^k} < \lambda^k \le \dfrac{2(\bar{y}_j^k - u_j^k)}{e_j^k} \\ u_j^k & \lambda^k > \dfrac{2(\bar{y}_j^k - u_j^k)}{e_j^k} \end{cases}$$

Clearly each $y_j^k(\lambda^k)$ for $j \in J_+^k$ is piecewise linear and monotonically nonincreasing and each $y_j^k(\lambda^k)$ for $j \in J_-^k$ is piecewise linear and monotonically nondecreasing. Thus each $f(\lambda^k)$ is piecewise linear and monotonically nonincreasing. In this instance, a revised version of ALG 2.1 of Section 2.1.2 is used for obtaining λ^k for the case where the coefficients e_j^k could take any value not equal to zero. The revised algorithm follows.

<u>ALG 3.3 ALGORITHM FOR λ^k</u>

Step 1 : Set $k \leftarrow 1$.

Step 2 : Initialization.

Let $a_1^k \le a_2^k \le \ldots \le a_{2n_k}^k$ denote the ordered $2n_k$ breakpoints $\dfrac{2(\bar{y}_j^k - u_j^k)}{e_j^k}$ and $\dfrac{2\bar{y}_j^k}{e_j^k}$ for $j=1,\ldots,n_k$. If

$b_k < \sum_{j \in J_-^k} e_j^k u_j^k$ where $J_+^k = \left\{ j : e_j^k > 0 \right\}$ and

$J_-^k = \left\{ j : e_j^k < 0 \right\}$, terminate with no feasible solution;

otherwise set $l^k = 1$, $r^k = 2n_k$, $L^k = \sum_{j \in J_+^k} e_j^k u_j^k$ and

$R^k = \sum_{j \in J_-^k} e_j^k u_j^k$.

Step 3 : Test for Bracketing.

If $r^k - l^k = 1$, go to step 6; otherwise set $m = [\frac{l^k + r^k}{2}]$,

the greatest integer $\leq \frac{l^k + r^k}{2}$.

Step 4 : Compute New Value.

$$B^k = \sum_{j \in J_+^k} \max\left\{ \min\left(e_j^k \bar{y}_j^k - (e_j^k)^2 \frac{a_m^k}{2} \;,\; e_j^k u_j^k \right),\; 0 \right\}$$

$$+ \sum_{j \in J_-^k} \min\left\{ \max\left(e_j^k \bar{y}_j^k - (e_j^k)^2 \frac{a_m^k}{2} \;,\; e_j^k u_j^k \right),\; 0 \right\}.$$

Step 5 : Update.

If $B^k = b_k$, terminate with $\lambda^k = a_m^k$. If $B^k > b_k$, set $l^k = m$, $L^k = B^k$, and go to step 3. If $B^k < b_k$, set $r^k = m$, $R^k = B^k$, and go to step 3.

Step 6 : Interpolate.

Terminate with $$\lambda^k = a_{\ell^k}^k + \frac{(a_{r^k}^k - a_{\ell^k}^k)(b_k - L^k)}{R^k - L^k}.$$

Step 7 : Increment.

Set $k \leftarrow k + 1$.

If $k > p$, terminate with a λ^k for each of SCBVLP;

otherwise, go to step 2.

The subgradient optimization algorithm for problem (*NPGD*) makes use of a lower bound, LBND, on the optimal objective function in the termination criterion each time the procedure is invoked. The following summarizes the upper bound algorithm for problem (*NPG*).

ALG 3.4 ALGORITHM FOR UPPER BOUND

Step 1 : Initialization.

Initialize LBND, step size t and ϵ.

Step 2 : Find Subgradient and Step Size.

Let $\bar{x}$ and $\bar{\pi}$ be the vectors of optimal primal and dual variables for

$$\min \left\{ \sum_{i=0}^{p} c^i x^i + Ms : \sum_{i=0}^{p} A^i x^i + Is = r,\ s \geq 0,\ 0 \leq x^0 \leq u^0, \text{and for } k=1,\dots,p \quad 0 \leq x_j^k \leq y_j^k \text{ for } j \in J_+^k \text{ and } y_j^k \leq x_j^k \leq u_j^k \quad \text{for } j \in J_-^k \right\}$$

Let UBND = $c\bar{x}$.

If (UBND-LBND) $\leq \epsilon |\text{UBND}|$, terminate with $\bar{x}$ near optimal. Otherwise, let, for k=1,...,p,

$$\sigma_j^k = \begin{cases} c_j^k + \bar{\pi}_{f(j)} - \bar{\pi}_{t(j)} & \text{if } \bar{x}_j^k = y_j^k \text{ and nonbasic} \\ 0 & \text{otherwise} \end{cases}$$

$$\eta = \left(\sigma_1^1, \dots, \sigma_{n_1}^1, \dots, \sigma_1^p, \dots, \sigma_{n_p}^p \right)^t .$$

Step 3 : Move to New Point.

$y \rightarrow P[y - t\eta]$.

Adjust the step size t.

Step 4 : Repeat the Process.

Go to step 2.

3.3 An Algorithm for Problem *NPG*

The pure network problem with generalized upper bound constraints can be implemented using decomposition (upper bound algorithm) without the lower bound procedure. It is also possible to obtain a lower bound on the optimal value of the problem ($\mathcal{NPG}$) by implementing the lower bound algorithm independently. By merging the two algorithms, a procedure which adjusts the lower and upper bounds progressively can be used.

The algorithm for the network with GUB constraints problem begins with **w** (the vector of Lagrange multipliers of the lower bound procedure) equal to 0, this involves ignoring the GUB constraints and solving the network portion. Let $\bar{\mathbf{x}}$ denote the optimal solution to the network problem. If the optimal solution to the pure network problem satisfies the GUB constraints, the solution is also optimal for the problem ($\mathcal{NPG}$). Otherwise, a step is taken in the direction of a subgradient of L at **w** = 0 and a new **w** is determined. The optimal solution to the initial p SCBVLP problems is obtained using ALG 3.1 for the case when w=0. The initial solution in the upper bound procedure is the solution $\bar{\mathbf{x}}$ to the last pure network problem solved in the lower bound procedure. The most recent pure network solution in the previous lower bound procedure is used in reoptimizing the network in the lower bound algorithm each time it is invoked.

Each time the lower bound procedure is called, a maximum of LITER iterations are performed. Each time the upper bound procedure is called, a maximum of UITER iterations are performed.

ALG 3.5 ALGORITHM FOR THE PROBLEM *NPG*

Step 0 : Initialization.

Initialize UITER, LITER, step size d and ϵ.

Let $w = 0$.

Let $\bar{x}$ solve

$$\min \left\{ \sum_{i=0}^{p} c^i x^i + Ms : \sum_{i=0}^{p} A^i x^i + Is = r,\ s \geq 0, \\ 0 \leq x^i \leq u^i, \text{and for } i=0,\ldots,p \right\}$$

If $\bar{x}$ satisfies the GUB constraints, terminate with $\bar{x}$ optimal to the original problem.

Let $\bar{y}$ be an initial solution to p SCBVLPs. That is, for each k, $1 \leq k \leq p$,

$$\bar{y}_j^k = \begin{cases} 0 & \text{if } e_j^k > 0 \\ u_j^k & \text{if } e_j^k < 0 \end{cases}$$

Let $\eta_L = \left(\bar{y}^1 - \bar{x}^1, \ldots, \bar{y}^p - \bar{x}^p \right)^t$.

If $\eta_L = 0$, terminate with $\bar{x}$ optimal.

Let $w = d\ \eta_L$.

Adjust the step size d.

Let UBND $= \infty$ and LBND $= \sum_{i=0}^{p} c^i \bar{x}^i$.

Step 1 : Set Iteration Count.

Set $L \leftarrow 0$ and $U \leftarrow 0$.

```
Step 2 : Compute Lower Bounds.
    a. Call ALG 3.2 (steps 2 and 3).
    b. Set $L \leftarrow L + 1$.
       If $L < LITER$, go to step 2a.

Step 3 : Compute Upper Bounds.
    a. Call ALG 3.4 (steps 2 and 3).
    b. Set $U \leftarrow U + 1$.
       If $U < UITER$, go to step 3a.
       Otherwise, go to step 1.
```

3.4 Finding an Initial Basic Feasible Solution

The initialization step of ALG 3.5 assumes that a basic feasible solution with which to initiate the algorithm can be found. The purpose of this section is to describe a strategy for obtaining such a solution. For this method a combination of original arcs and artificial arcs are allowed to carry flow.

Assume the problem has been transformed so that all lower bounds are zero. A heuristic procedure is used to obtain the initial basic feasible solution; the main idea of this procedure is to quickly find paths through the network that will transport a large amount of items from the supply nodes to the demand nodes.

The heuristic first starts with the initialization phase. Let the root node, ROOT, be equal to any one of the supply nodes. For each demand node $q \in \mathcal{N}$, add an artificial arc (ROOT,q) with $c_{ROOT,q} = u_{ROOT,q} = +\infty$. These arcs are all made part of the spanning tree, and each is assigned an initial flow equal to the unsatisfied demand at the node that is connected to the root, that is, $x_{ROOT,q} = r_q$. The flow $x_{ROOT,q}$ will be decreased if a set of arcs is found that allows for the achievement of r_q from one or more supply nodes.

A list $\mathcal{SL}$ is then formed consisting of supply nodes, ordered by magnitude of node number in the original problem, with the node having the smallest number appearing first in the list. For each node $p \in \mathcal{SL}$, define a quantity US_p, which is called the undistributed supply. Initially, $US_p = -r_p$ for $p \in \mathcal{SL}$. Also for any node $p \in \mathcal{SL}$, let $T_p = \{(p,q): (p,q) \in \mathcal{A}\}$; T_p is simply the set of all arcs whose "from" node is p. This completes the initialization phase of the heuristic.

The main portion of the heuristic attempts to build forward chains (directed paths) beginning at each supply node and terminating at some node already in the tree. Each chain consists initially of a single node and may be extended by the addition of new nodes and connecting arcs.

The node most recently added to the chain will be referred to as the highest node on the chain. Chains are extended only at the highest node. Eventually each chain is connected to the spanning tree either by an artificial arc from the highest node in the chain to the ROOT or by an arc $(p,q) \in \mathcal{A}$, where p is the highest node in the chain and q is a demand node.

The procedure consists of two phases. In phase 1, part of a spanning tree is formed so as to transport the undistributed supply to demands via chains. For each supply node p with undistributed supply $US_p > 0$, we append an arc $(p,q) \in T_p$ with flow US_p, cost of $COST_{p,q}$ and bound of $u_{p,q}$ if :

(i) q is a demand node, flow on arc (ROOT,q) is positive, $US_p < u_{p,q}$ and $US_p \leq x_{ROOT,q}$.

(ii) q is either a transshipment or a supply node which is not in any chain and $US_p < u_{p,q}$.

In case (i) the chain is completed. In case (ii) q becomes the highest node in the chain. If none of the above cases hold, arc (p,q) may become nonbasic at its upper bound if :

(iii) q is a demand node, flow on arc (ROOT,q) is positive, $u_{p,q} \leq US_p$ and $u_{p,q} \leq x_{ROOT,q}$.

(iv) q is either a transshipment or a supply node which is not in any chain and $u_{p,q} \leq US_p$.

If no q $\in T_p$ satisfies the last four cases p is connected to the root node via an artificial arc having flow of US_p; and p is then removed from $\mathcal{SL}$. The process is repeated until $\mathcal{SL} = \phi$. At this point phase 1 is completed. In phase 2, either artificial arcs or real arcs are added with a flow of zero in order to connect the isolated transshipment nodes to the tree so as to complete the spanning tree.

<u>ALG 3.6 : Finding An Initial (Artificial) Feasible Basis</u>

```
(PHASE 1)
Step 0 : Begin.
Step 1 : Select First Node in Supply List ($\mathcal{SL}$).
   a.    If $\mathcal{SL} = \phi$ , go to Step 9 (Check for termination)
         Otherwise, let p be the first node in $\mathcal{SL}$.
   b.    If p = ROOT, remove p from $\mathcal{SL}$ and go to Step 0.
Step 2 : Check for the Undistributed Supply.
         If  $US_p$=0, go to Step 8 (Connect p to ROOT).
Step 3 : Select a Node that is a To Node of p.
         If $T_p = \phi$ , go to Step 8 (Connect p to ROOT).
         Otherwise, let q be the first node in $T_p$.
         If $r_q > 0$, go to Step 4.
         Otherwise, go to Step 5.
```

Step 4 : A Demand Node.

a. (Demand is satisfied via real arcs)

If $x_{ROOT,q} = 0$, remove q from T_p and go to Step 3.

b. (Demand not completely satisfied via real arcs)

Go to Step 6.

Step 5 : A Supply or Transshipment Node.

a. If q is part of a chain, remove q from T_p and go to Step 3.

b. Otherwise, go to Step 7.

Step 6 : A Demand Node May Receive Supply

a. (It is an arc that may be set to upper bound)

If $u_{p,q} \leq \min[US_p, x_{ROOT,q}]$, let $x_{p,q}=u_{p,q}$, $US_p=US_p - u_{p,q}$, $x_{ROOT,q}=x_{ROOT,q}-u_{p,q}$, remove q from T_p and go to Step 2.

b. (It is an arc that may become basic).

If $US_p < u_{p,q}$ & $US_p \leq x_{ROOT,q}$, let $x_{p,q}=US_p$, $x_{ROOT,q}= x_{ROOT,q}- US_p$, $US_p = 0$, connect the chain with p as the highest node in the chain to the tree via (p,q), remove p from $\mathcal{SL}$ and go to Step 1.

c. (It is an arc that cannot be set to upper bound or made basic).

Remove q from T_p and go to Step 3.

Step 7 : A Supply or a Transshipment Node May Transfer the Supply.

a. (It is an arc that may be set to upper bound).

If $u_{p,q} \leq US_p$, let $x_{p,q}=u_{p,q}$, $US_p=US_p-u_{p,q}$, place q in the last position of $\mathcal{SL}$ with $US_q = US_q + u_{p,q}$, remove q from T_p, and go to Step 2.

b. (It is an arc that may become basic).

If $US_p < u_{p,q}$, let $x_{p,q}=US_p$, remove p from $\mathcal{SL}$, extend the chain by connecting p to q via (p,q) so that q becomes the highest node in the chain, place q in the last position of $\mathcal{SL}$ with $US_q = US_q + US_p$, $US_p = 0$, and go to Step 1.

Step 8 : Connect p to Tree With an Artificial Arc.

Remove p from $\mathcal{SL}$. Create an artificial arc, (p,ROOT) with $c_{p,ROOT} = u_{p,ROOT} = +\infty$, $x_{p,ROOT} = US_p$. Connect the chain with p as its highest node to the utmost right hand side of the tree via (p,ROOT) and go to Step 1.

(PHASE 2)

Step 9 : Initialize Node Counter.

Let p = NUMSUP + NUMDEM + 1.

Step 10: Test for Connectedness of Tree

If all the transshipment nodes are connected to the tree, Terminate.

Step 11: Connect to Tree With an Artificial Arc.

a. Create an artificial arc with $c_{p,ROOT} = u_{p,ROOT} = +\infty$, $x_{p,ROOT} = 0$. Connect (p,ROOT) to the utmost right hand corner of tree.

b. (Increment node counter)

$p = p + 1$.

c. If $p >$ number of nodes, Terminate.

d. Go to Step 10.

CHAPTER 4
SOFTWARE DESCRIPTION

This chapter describes the use of **NETGUB** for the solution of the network with GUB constraints problem. The primary purpose is to provide documentation of the subroutines which compose the optimization software.

4.1 Data Structures

NETGUB makes use of 14 arc–length arrays to store arc information. The arcs are rearranged internally with all arcs incident from node 1, followed by all arcs incident from node 2, ..., followed by all arcs incident from node NODES. Table 4.1 gives the use of each of the fourteen arrays and the name of the subroutines that make use of these arrays. **NETGUB** makes use of 9 node–length arrays to store node information. Table 4.2 gives the use of each of the nine arrays and the name of the routines that make use of them. Three GUB–length arrays are used to represent the GUB information. Table 4.3 gives the use of each of these arrays and the name of the routines that make use of them.

Table 4.1 : Arc–Length Arrays

Arc length arrays	Description	Subroutines that use these
ARCNAM	Name of arc	**INPUT,INIT,LBALG,UBALG PURNET,REOPT**
COST	Unit cost on arc	**INPUT,INIT,LBALG,UBALG PURNET,REOPT,START**
CTEMP	Temporary unit cost or net change in unit cost on arc	**INPUT,INIT,LBALG PURNET**
GCOEF	Coefficient of arc in the GUB constraint	**INPUT,INIT,LBALG,UBALG SCBVLP,REOPT,PROJOP UINIT**
GFLOW	Flow on arc	**INIT,BASSAV,BASRED LBALG,UBALG,PURNET SCBVLP,REOPT,START UINIT**
GUBADD	Arc address	**INPUT,INIT,LBALG,UBALG SCBVLP,PROJOP,UINIT**
LGMULT	Lagrange multiplier corresponding to an arc	**INIT,LBALG,SCBVLP**
LOWER	Lower bound on arc flow	**INPUT,INIT,LBALG,UBALG PURNET,REOPT**

Table 4.1 (continued)

Arc length arrays	Description	Subroutines that use these
STATUS	Status of an arc. 0, if arc is basic; 1, if arc is nonbasic at lower bound; 2, if arc is nonbasic at upper bound; 3, if arc is fixed.	INPUT,INIT,BASSAV BASRED,LBALG,UBALG PURNET,REOPT,START
TO	To-node of an arc	INPUT,INIT,LBALG,UBALG PURNET,REOPT,START
UPPER	Upper bound on arc flow	INPUT,INIT,LBALG,UBALG PURNET,SCBVLP,REOPT PROJOP,START,UINIT
UTEMP	Temporary upper bound or net change in upper bound on arc flow	UINIT,UBALG,REOPT PROJOP
YFLOW	Flow on arc that is in a GUB constraint	INIT,LBALG,SCBVLP
ZFLOW	Flow on arc that is in a GUB constraint	UINIT,UBALG,PROJOP REOPT

Table 4.2 : Node–Length Arrays

Node length arrays	Description	Subroutines that use these
BASIS	Arc joining node with its predecessor. +, if arc is (p(i),i); -, if arc is (i,p(i))	**INIT,BASSAV,BASRED LBALG,UBALG,PURNET REOPT,START**
CARD	Cardinality of a node in the basis tree	**INIT,BASSAV,BASRED LBALG,UBALG,PURNET REOPT,START**
FLOW	Flow on basic arc (i,p(i)) or (p(i),i)	**INPUT,INIT,BASSAV BASRED,LBALG,UBALG PURNET,REOPT,START**
FROM	The location of the first arc in the TO array with node as its from–node	**INPUT,INIT,LBALG UBALG,PURNET,REOPT START**
LNOD	Last node in the subtree of this node	**INIT,BASSAV,BASRED LBALG,UBALG,PURNET REOPT,START**
NVAL	Node requirement	**INPUT,INIT,START**
PI	Dual variable value	**INIT,LBALG,UBALG PURNET,REOPT**
PRED	Predecessor node	**INIT,BASSAV,BASRED LBALG,UBALG,PURNET REOPT,START**
THD	Thread node	**INIT,BASSAV,BASRED LBALG,UBALG,PURNET REOPT,START**

Table 4.3 : GUB-Length Arrays

GUB length arrays	Description	Subroutines that use these
GVAL	Right hand side of a GUB constraints	INPUT, INIT, LBALG UBLAG, SCBVLP, PROJOP UINIT
NUMVRG	Number of arcs in a GUB constraints	INPUT, INIT, LBALG UBALG, SCBVLP, UINIT
REDGUB	Determines if a GUB constraint can be eliminated. 1, if yes; 0, if no.	INPUT, INIT, LBALG UBALG, UINIT

Table 4.4 gives the use of each of the constant parameters. Before compiling **NETGUB**, one has to decide on the maximum size problem required to be solved. The dimensions of the parameters should be changed to accommodate the desired size problem. The dimensions are currently set to 2000 for arc–length arrays, 500 for node–length arrays and 1000 for GUB–length arrays. The parameters are currently set to 20 and 10, respectively, for maximum iterations for lower and upper bound procedures, and .01 for tolerance.

4.2 <u>Main Program and Subroutines</u>

This section concentrates on the description of the different subroutines called in the main program. Figure 4.1 shows the various subroutines that constitute the main program for solving the network with GUB constraints problem. Descriptions of each subroutine follow. In the following discussions i will designate an arbitrary node, while j and k will denote an arc and a GUB constraint, respectively.

A. Subroutine **INPUT**

There are four sets of information that are specified in the input; the nonzero requirements at nodes; the number of GUB constraints; the number of variables in each

Table 4.4 : Constant Parameters

Constant parameter	Type	Description
EPSILON	R*8	Tolerance on the difference between lower and upper bound on the objective function value at termination
LITER	I	Maximum iterations for the lower bound procedure
MAXARC	I	Maximum number of arcs
MAXGUB	I	Maximum number of GUB constraints
MAXNOD	I	Maximum number of nodes
UITER	I	Maximum iterations for the upper bound procedure

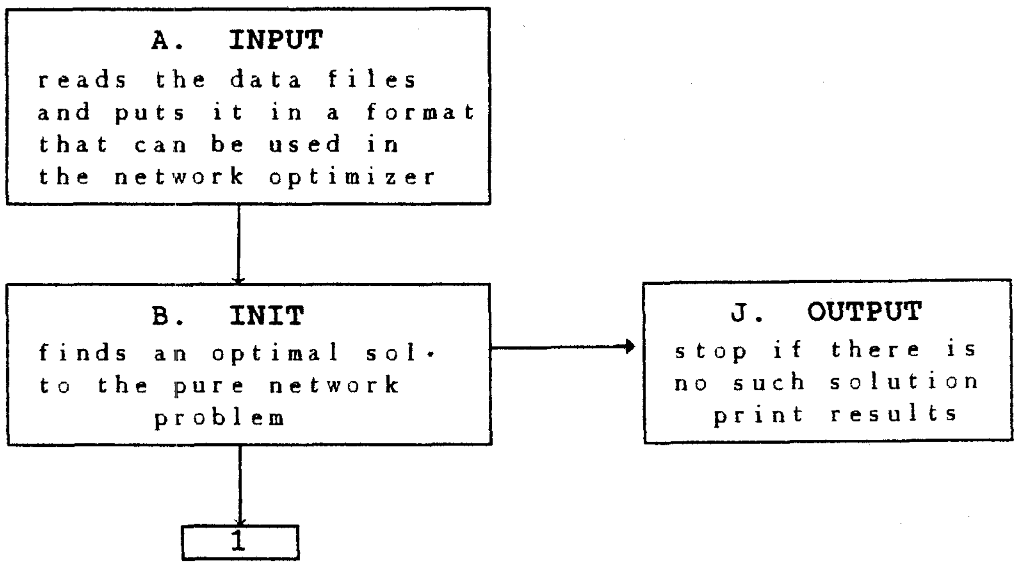

A. INPUT
reads the data files and puts it in a format that can be used in the network optimizer
B. INIT
finds an optimal sol. to the pure network problem
J. OUTPUT
stop if there is no such solution print results
1

Figure 4.1 : Flow Diagram of the Main Program for the Network with GUB Constraints Problem

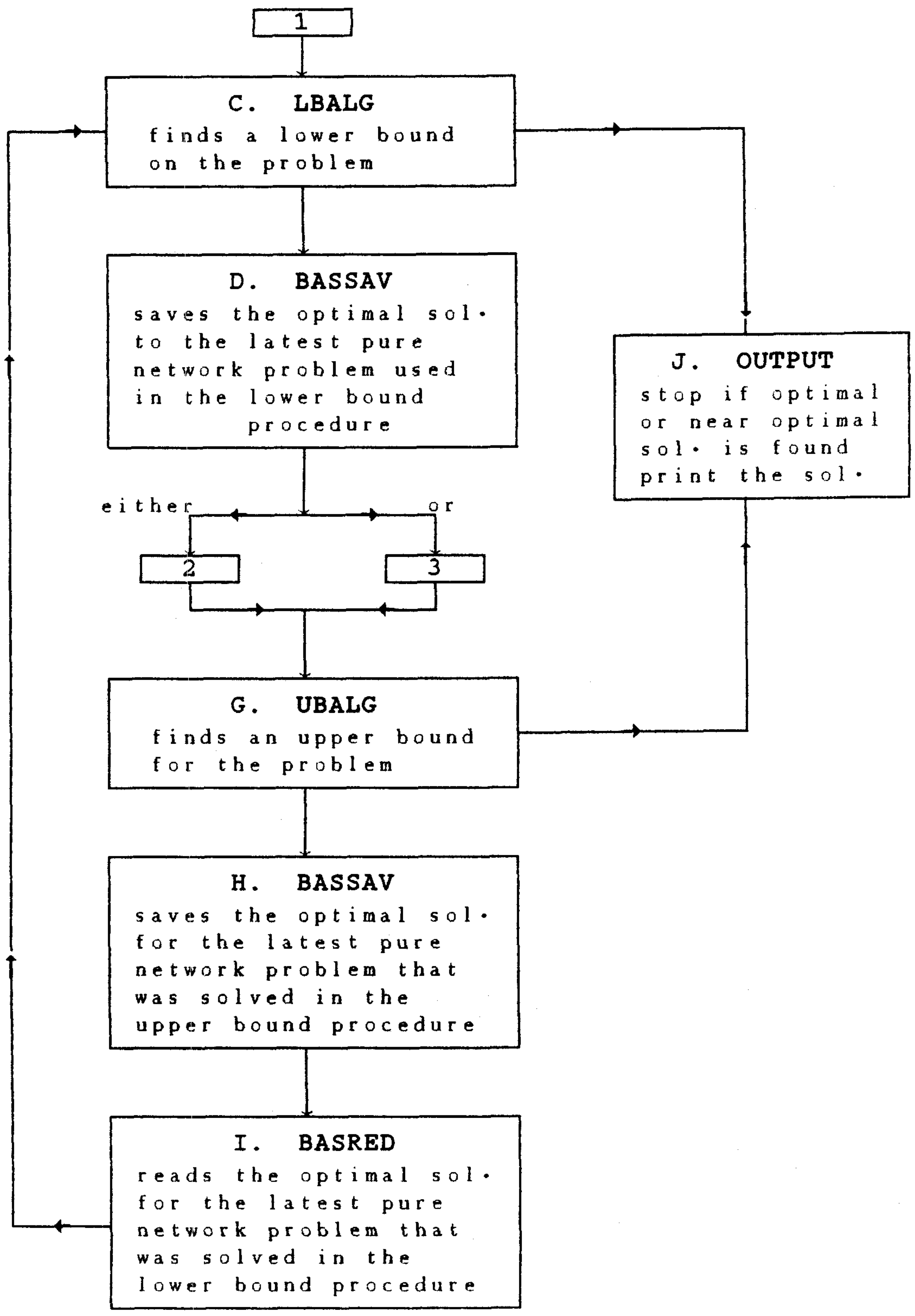

1
C. LBALG
finds a lower bound on the problem
D. BASSAV
saves the optimal sol. to the latest pure network problem used in the lower bound procedure
either
or
2
3
G. UBALG
finds an upper bound for the problem
H. BASSAV
saves the optimal sol. for the latest pure network problem that was solved in the upper bound procedure
I. BASRED
reads the optimal sol. for the latest pure network problem that was solved in the lower bound procedure
J. OUTPUT
stop if optimal or near optimal sol. is found print the sol.

Figure 4.1 (continued)

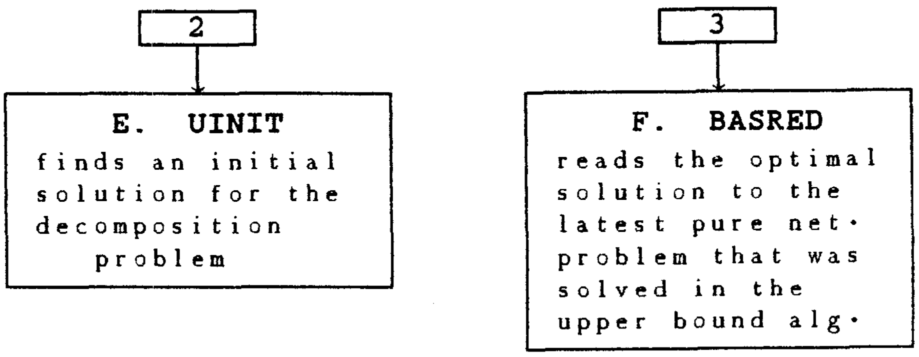

2
E. UINIT
finds an initial solution for the decomposition problem
3
F. BASRED
reads the optimal solution to the latest pure net· problem that was solved in the upper bound alg·

Figure 4.1 (continued)

GUB constraint and the RHS value; and for each arc, the arc name, from node, to node, cost, upper bound , lower bound and coefficient in GUB constraint (0, if not in a GUB constraint).

The input format used for all data and an example (file 8) is given in Appendix A. The input file is capable of storing successive problems in one file.

This routine sets up the data structures for the arc—length arrays **ARCNAM**(j), **FROM**(j), **TO**(j), **COST**(j), **UPPER**(j), **LOWER**(j) and **GCOEF**(j); **NVAL**(i); and all GUB—length arrays. The number of arcs, **ARCS**, number of nodes, **NODES**, and the total number of arcs in the GUB constraints, **GARCS**, are determined as the data is read in. This routine introduces a dummy node, **DUMMY**, when the total supply exceeds total demand; however, when total demand exceeds total supply, the problem is declared infeasible. Unlike some other pure network input files, the number of arcs out of each node need not be specified, since the routine calculates these, **NUMOUT**(i), as the arc information is read in.

This routine also converts the problem to one with zero lower bounds and at the same time adjusts for the objective function value, **TCOST**, of the network problem and also adjusts for the right hand side value of the GUB constraints in the following way:

```
Repeat for all j ∈ $\mathcal{A}$
   If LOWER(j) ≠ 0, then :
      UPPER(j) = UPPER(j) - LOWER(j).
      TCOST = TCOST + LOWER(j) * COST(j).
      If j appears in kth GUB constraint, then:
         GVAL(k) = GVAL(k) - LOWER(j) * GCOEF(j).
```

To account for the GUB constraint feasibility or for the constraints that can always be satisfied and hence can be eliminated, the following inner loop is used:

```
Repeat for all 1 ≤ k ≤ NGUBS
   TGVAL = SGVAL = 0.
   Repeat for all j ∈ $\mathcal{A}$ that are in the kth GUB
   constraint
      If GCOEF(j) > 0, then :
         TGVAL = TGVAL + UPPER(j) * GCOEF(j)
      If GCOEF(j) < 0, then :
         SGVAL = SGVAL - UPPER(j) * GCOEF(j).
      If TGVAL ≤ GVAL(k), then REDGUB(k) = 1.
      If GVAL(k) < -SGVAL, then the problem is
         declared infeasible.
```

The CTEMP array is used, in this routine only, to temporarily store SGVAL for each GUB constraint.

B. Subroutine **INIT**

This routine is set up to find an initial basic feasible solution for the lower bound procedure that was described in the initialization step (Step 0) of ALG 3.5 in Section 3.3. Figure 4.2 shows the various subroutines that are included in this subroutine.

This routine first attempts to find an initial feasible solution to the pure network portion of the problem by using the following subroutine.

a. Subroutine **START**

This routine makes use of an advanced start procedure discussed in Section 3.4 to produce an initial basic feasible solution and whenever necessary, introduces artificial variables. At the beginning of the routine the status of the data structure should be exactly as at the end of the input routine. Then ALG 3.6 is followed to find an initial (artificial) basic feasible solution. Also the data structure for the arc length arrays **GFLOW**(j) and **STATUS**(j); and for the node length arrays **BASIS**(i), **CARD**(i), **LNOD**(i), **PRED**(i), **THD**(i), **FLOW**(i) and **PI**(i) are set up.

Every time the flow associated with node i, **FLOW**(i), is determined, the objective function value, **TCOST**, is adjusted as follows

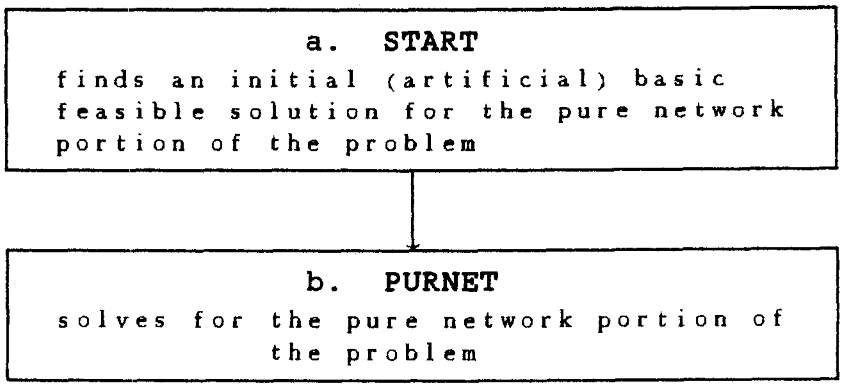

a. START
finds an initial (artificial) basic feasible solution for the pure network portion of the problem
b. PURNET
solves for the pure network portion of the problem

Figure 4.2 : Flow Diagram of the INIT Subroutine

TCOST = TCOST + FLOW(i) * COST(ABS(BASIS(i))).

Upon completion of routine **START**, the data structures for the network problem are all set up. The routine **INIT** then makes use of this advanced starting solution to optimize the pure network portion of the problem.

b. Subroutine **PURNET**

The routine **PURNET** performs primal simplex iterations until optimality criteria are satisfied. The algorithm procedures makes use of the Big–M method to produce a feasible solution (if the problem is feasible) and then the optimal solution. The cost of artificial variables (the value of M) used in the procedure is 100000. At the end of this routine the following arrays contain the information for the optimal solution:

GFLOW, **STATUS**, **PI**, **BASIS**, **CARD**, **LNOD**, **PRED**, and **FLOW**.
The optimal objective function value is contained in **TCOST**.

At the end of the **PURNET** routine, the array **GFLOW** corresponds to the optimal solution $\bar{x}$ in step 0 of ALG 3.5 in Section 3.3. At this point the routine **INIT** attempts to find an optimal solution for the initial SCBVLPs. The array **YFLOW** corresponds to an optimal solution $\bar{y}$ in step 0 of ALG 3.5 and is determined as follows:

```
Repeat for all j $\in \mathcal{A}$ that are in the kth GUB
    If REDGUB(k) = 0, then :
        If GCOEF(j) > 0, then YFLOW(j) = 0.
        Else YFLOW(j) = UPPER(j).
        LGMULT(n) = LBSTEP * (YFLOW(j) - GFLOW(j)).
    If REDGUB(k) = 1, then :
        YFLOW(j) = GFLOW(j).
        LGMULT(n) = 0.
```

The Lagrange multiplier array, **LGMULT**, is arranged with respect to the location of the arc in the input file. That is, **LGMULT**(n) is the lagrange multiplier corresponding to the nth arc in the input file with an arc address of **GUBADD**(n). The reason this array is stored in this manner will become clear later when one desires to order the components of the **LGMULT** array.

This routine also determines the array **CTEMP**. This array is used in routine **LBALG** to determine the new cost coefficients of the pure network problem in step 2 of ALG 3.2 in Section 3.1.

```
Repeat for all GUB k
   Repeat for $j \in \mathcal{A}$ that are in the kth GUB
      CTEMP(j) = The net change in unit cost of the
                 pure network problem.
               = 0                if REDGUB(k) = 1
                 - LGMULT(n)      if REDGUB(k) = 0.
```

At the end of routine **INIT**, a lower bound on the objective function value of the network with GUB constraints problem, **LBND**, exists. At this point an upper bound on the objective function value of the problem, **UBND**, is set arbitrarily to a large positive number.

C. Subroutine LBALG

This routine recomputes the value of **LBND**, the lower bound on the objective function value of problem (NPG) using steps 2 and 3 of ALG 3.2. Figure 4.3 shows the various subroutines that are called in this routine.

The first time this routine is called, the status of the arrays should be exactly the same as at the end of routine **INIT**. The optimal solution of the pure network problem of step 0 of ALG 3.5 is used as an advanced starting point for this network problem (step 2 of ALG 3.2). In the subsequent calls the status of the arrays **BASIS**, **CARD**, **FLOW**, **GFLOW**, **LNOD**, **PRED**, **STATUS** and **THD** are

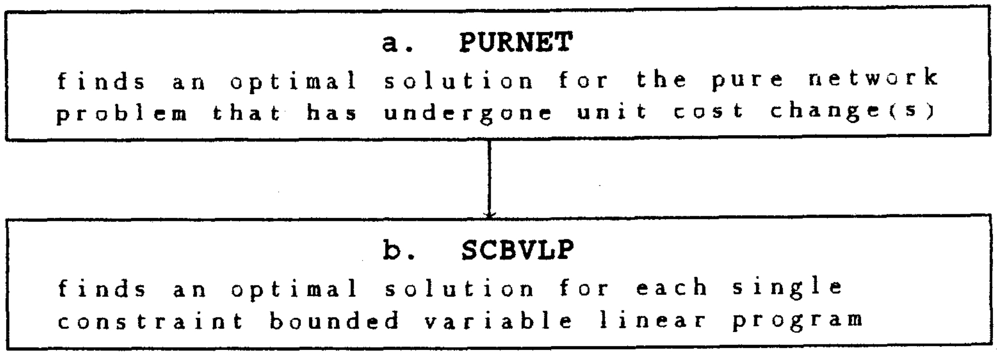

a. PURNET
finds an optimal solution for the pure network problem that has undergone unit cost change(s)
b. SCBVLP
finds an optimal solution for each single constraint bounded variable linear program

Figure 4.3 : Flow Diagram of the LBALG Subroutine

exactly as at the end of routine **BASRED**. In any case, the unit costs are changed so the **TCOST** is recomputed. In this instance, **TCOST** would be the corresponding objective function value for the present initial feasible solution. The changes are done in the following inner loop:

```
Repeat for all j $\in \mathcal{A}$
    If GCOEF(j) $\neq$ 0, then :
        CTEMP(j) = COST(j) - LGMULT(n).
    Else  CTEMP(j) = COST(j).
    TCOST = TCOST + CTEMP(j) * GFLOW(j)
where n is the location of arc j in the LGMULT array.
```

Thus, **CTEMP** is used here as the array of unit costs for the routine **PURNET** and it is this array that is passed on to it and not the array COST. Having the right unit costs, routine **PURNET** with arguments: **ARCNAM**, **BASIS**, **CARD**, **CTEMP**, **FLOW**, **FROM**, **FROMO**, **GFLOW**, **LNOD**, **LOWER**, **PI**, **PRED**, **STATUS**, **THD**, **TO**, and **UPPER** is called to solve for the network problem. At the end of routine **PURNET**, **GFLOW** contains the information for the optimal solution $\bar{x}$ in step 2 of ALG 3.2.

To solve for the SCBVLPs in step 2 of ALG 3.2, routine **SCBVLP** is constructed. This routine is called once for each GUB constraint with value of 0 in the corresponding component of **REDGUB**.

b. Subroutine **SCBVLP**

This routine solves for each GUB constraint (SCBVLP) separately. The arguments that are passed to this routine are: **GVAL**(k), **GFLOW**, **GUBADD**, **GCOEF**, **LGMULT**, **LOC1**, **LOC2**, **NUMVRG**(k), **UPPER**, and **YFLOW**, where k is the GUB constraint currently being solved for in this routine; initially LOC1 and LOC2 are, respectively, the locations of the first and the last arcs of the kth GUB constraint in the input file.

This routine makes use of ALG 2.3 of Section 2.1.5 to solve the SCBVLP. Array **YFLOW** would be constructed one component at a time to contain the information for the optimal solution $\bar{y}$ of step 2 of ALG 3.2.

Let n indicate the location of an arc anywhere between **LOC1** and **LOC2** inclusive for the kth GUB constraint. Let j be the arc corresponding to location n, i.e., j = **GUBADD**(n). **YFLOW** is constructed differently depending on whether

(i) **GCOEF**(j) $>$ 0 and **LGMULT**(n) $\geq$ 0 ;

(ii) **GCOEF**(j) $<$ 0 and **LGMULT**(n) $\leq$ 0;

(iii) either **GCOEF**(j) $>$ 0 and **LGMULT**(n) $<$ 0,

or **GCOEF**(j) $<$ 0 and **LGMULT**(n) $>$ 0.

This routine rearranges the arcs so that the arcs belonging to case (iii) are followed by the arcs that belong to either case (i) or case (ii). At this point **LOC2** is revised to be the location of the last arc that belongs to case (iii). Let **LASTRC** be the location of the last arc in this GUB constraint. Then, the arcs in **LOC1** to **LOC2** satisfy case (iii), and arcs in locations **LOC2** + 1 to **LASTRC** belong to either case(i) or case (ii). The arcs in locations **LOC1** to **LOC2** are then sorted according to step 4 of ALG 3.1 and the array **YFLOW** is constructed. The following loop shows how these components of **YFLOW** are computed. At the same time the adjusted right hand side value of the GUB constraint **ADJVAL** is computed. Initially **ADJVAL** stores the number $\bar{b}_k$ in step 3 of ALG 3.1.

```
Repeat for all n ∈ [ LOC1, LASTRC ]
   If GCOEF(j) > 0 and LGMULT(n) ≥ 0, then :
      YFLOW(j) = 0.
   If GCOEF(j) < 0, then :
      ADJVAL = ADJVAL - GCOEF(j) * UPPER(j).
      If LGMULT(n) ≤ 0, then :
         YFLOW(j) = UPPER(j)
         LBND = LBND + LGMULT(n) * YFLOW(j).
```

What is left is the construction of those components of **YFLOW** that follow case (iii). To do this, a routine

LOCSORT is used that finds the ordered values

$$\frac{\mathbf{LGMULT}(n)}{\mathbf{GCOEF}(j)} \text{ for all } n \in [\mathbf{LOC1},\mathbf{LOC2}] \qquad (1)$$

as was explained in step 4 of ALG 3.1. This routine is a slight modification of the routine **HPSORT** [Nijenhuis and Wilf (1978, p. 140)]. The arguments that are passed to this routine are : **NUM**, **LOC1**, **LOC2**, **LGMULT**, **GCOEF**, and **GUBADD**. At the end of this routine the arcs in locations **LOC1** to **LOC2** of the input file, and hence **LGMULT**, are rearranged so that arc j_1 in **LOC1** is the one with the smallest ratio (1), j_2 in **LOC1** + 1 is the second smallest ratio (1), ..., and j_{n_k} in **LOC2** is the one with the largest ratio (1). Having the ordered ratios, step 5 of ALG 3.1 is used to construct the remaining components of **YFLOW**. The following loop does this:

```
Repeat for all j in location $n \in$ [LOC1,LOC2]
   If GCOEF(j) > 0, then :

      $\text{YFLOW}(j) = \min \left\{ \text{UPPER}(j)\ ,\ \frac{\text{ADJVAL}}{\text{GCOEF}(j)} \right\}.$

      $\text{ADJVAL} = \text{ADJVAL} - \text{GCOEF}(j) * \text{YFLOW}(j).$
   Else
```

$$\text{YFLOW(j)} = \text{UPPER(j)} - \min\left\{\text{UPPER(j)}, \frac{\text{ADJVAL}}{-\text{GCOEF(j)}}\right\}$$

```
        ADJVAL = ADJVAL + GCOEF(j) (UPPER(j) -YFLOW(j))
     LBND = LBND + LGMULT(n) * YFLOW(j)
```

Hence, routine **SCBVLP** terminates with an optimal solution to the kth SCBVLP problem.

Returning to routine **LBALG**, if k is equal to NGUBS, then we are all done with the SCBVLPs; otherwise, the (k + 1)st SCBVLP is solved by returning to routine SCBVLP. After all of the SCBVLPs are done the routine recomputes the components of the array **LGMULT** and constructs the array **CTEMP** as the net change in unit costs of the problem. The following loop does this :

```
Repeat for all j∈𝒜
   CTEMP(j) = - LBSTEP ( YFLOW(j) - GFLOW(j) )
   LGMULT(n) = LGMULT(n) - CTEMP(j)
```

After computing the components of the arrays **LGMULT** and **CTEMP** the optimality criteria is checked. The optimality is reached either when **YFLOW**(j) is equal to **GFLOW**(j) for all j ∈ $\mathcal{A}$, or when (**UBND** - **LBND**) is less than or equal to **EPSILON** |**UBND**|. In this case we return to the main program with **FLGOPT** equal to 1.

D. Subroutine **BASSAV**

This routine saves the optimal solution of the pure network problem. The arguments that are passed to this routine are : **BASIS**, **CARD**, **FLOW**, **GFLOW**, **LNOD**, **PRED**, **STATUS** **THD** and either **LBOBAS** or **UBOBAS**, where **LBOBAS** and **UBOBAS** are logical units for scratch files. If this routine is called after the routine **LBALG**, then the status of the arrays are exactly as at the end of routine **LBALG** and **LBOBAS** is the scratch file that stores the values of these arrays to be recalled later on. If this routine is called after the routine **UBALG**, then the status of the arrays are exactly as at the end of routine **UBALG** and **UBOBAS** is the scratch file that stores these values.

At this instance, the optimal solution is stored to be used in subroutine **LBALG** on the next set of iterations.

On returning to the main program **NETGUB**, a lower bound on the objective function value of the program exists. If no feasible solution for the decomposition algorithm is known, then routine **UINIT** is the next routine that is called; otherwise, routine **BASRED** is called.

E. Subroutine UINIT

This routine is set up to find an initial basic feasible solution for problem (*NPGD*) using the last solution of the pure network problem in routine LBALG.

The ZFLOW array corresponds to a solution $\bar{z}$ for problem (*NPGD*). Initially, for each GUB constraint k the arrays ZFLOW and UTEMP are set as follows:

```
Repeat for j $\in \mathcal{A}$ that are in the kth GUB
   If REDGUB(k) = 0, then :
      ZFLOW(j) = GFLOW(j).
      If GCOEF(j) > 0, then UTEMP(j) = UPPER(j).
      Else  UTEMP(j) = 0.
      SUM = SUM + ZFLOW(j) * GCOEF(j).
   Else If GCOEF(j) > 0, then ZFLOW(j) = UPPER(j).
      Else ZFLOW(j) = 0.
      UTEMP(j) = 0.
```

If for any GUB constraint k, SUM is equal to GVAL(k), then components of ZFLOW that correspond to this GUB are all feasible; otherwise, routine PROJOP is used to find a feasible set of components. A description of routine PROJOP follows.

c. Subroutine PROJOP

This routine makes use of ALG 3.3 to find a feasible solution to problem (*NPGD*). This routine deals with one GUB constraint at a time. The breakpoints are stored in array BRKPNT, and the sum L and R at step 2 of ALG 3.3 are computed as these breakpoints are found, in the following manner.

```
Set T = R = 0.
Repeat for each GUB k
   Repeat for all j ∈ $\mathcal{A}$ that are in the kth GUB

      BRKPNT(i) = $\dfrac{2\ (ZFLOW(j) - UPPER(j))}{GCOEF(j)}$;

      BRKPNT(i+1) = $\dfrac{2\ ZFLOW(j)}{GCOEF(j)}$;

      If GCOEF(j) > 0, then :
         L = L + UPPER(j) * GCOEF(j).
      If GCOEF(j) < 0, then :
         R = R + UPPER(j) * GCOEF(j).
```

These breakpoints are ordered using a Heap sort. An implementation of the HPSORT routine is given in Nijenhuis and Wilf (1978, p. 140). The code used in NETGUB was also obtained from Nijenhuis and Wilf (1978, p. 140). The array needed for the **HPSORT** routine is **BRKPNT** together with the variable 2***NUMVRG**(k). At the end of the **HPSORT** routine the **BRKPNT** array is sorted into nondecreasing order. It is this array of breakpoints that is used in steps 4 through 6 of ALG 3.3 to compute λ^k and store the value in variable **LAMDA**. L1, L2, R1 and R2 are the variables that store the values of l^k, L^k, r^k, R^k in the algorithm, respectively. Having determined **LAMDA**, the corresponding components of **ZFLOW** are adjusted and at the same time the array **UTEMP** is determined as the following:

```
Repeat for all j $\in \mathcal{A}$ that are in this kth GUB
   If GCOEF(j) > 0, then :
      If LAMDA $\leq \frac{2(\text{ZFLOW}(j) - \text{UPPER}(j))}{\text{GCOEF}(j)}$, then :
         ZFLOW(j) = UPPER(j).
      Else if LAMDA $\leq$ 0, then :
         ZFLOW(j) = ZFLOW(j) - GCOEF(j) * $\frac{\text{LAMDA}}{2}$.
           Else ZFLOW(j) = 0.
      UTEMP(j) = ZFLOW(j) - UTEMP(j).
```

The array **UTEMP**(j) is used to store the difference between the previous feasible solution and the current feasible solution **ZFLOW** determined in the routine **PROJOP**. This array is used for a different purpose in the UBALG routine.

F. Subroutine BASRED

This routine reads the optimal solution of the pure network problem. The arguments that are passed to this routine are : BASIS, CARD, FLOW, GFLOW, LNOD, PRED, STATUS THD and either LBOBAS or UBOBAS, where LBOBAS and UBOBAS are logical units for scratch files. If this routine is called before the routine LBALG, then the arrays are reconstructed for the optimal solution of the pure network problem that was obtained in the last iteration of routine LBALG from the scratch file LBOBAS. If this routine is called before the routine UBALG, then the arrays are reconstructed for the optimal solution of the pure network problem that was obtained in the last iteration of routine UBALG from the scratch file UBOBAS.

At this point, the information is read in from the scratch file UBOBAS to be used in routine UBALG.

G. Subroutine UBALG

This routine recomputes the value of UBND, the upper bound on the objective function value of problem (*NPG*) using steps 2 and 3 of ALG 3.4. Figure 4.4 shows the various subroutines that are called in this routine.

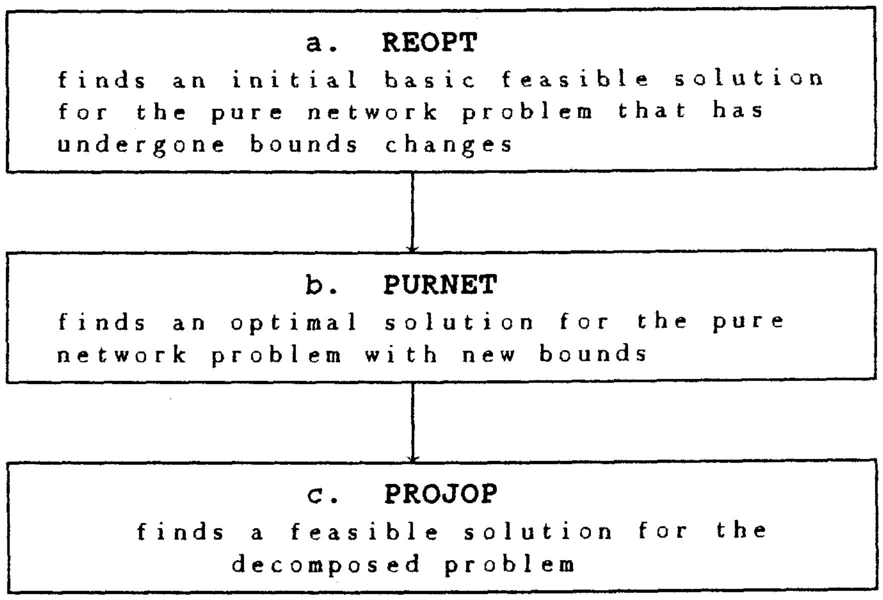

a. REOPT
finds an initial basic feasible solution for the pure network problem that has undergone bounds changes
b. PURNET
finds an optimal solution for the pure network problem with new bounds
c. PROJOP
finds a feasible solution for the decomposed problem

Figure 4.4 : Flow Diagram of the **UBALG** Subroutine

The first time this routine is called, the values of the arrays **BASIS**, **CARD**, **FLOW**, **GFLOW**, **LNOD**, **PRED**, **STATUS**, and **THD** are exactly as at the end of routine **LBALG**; but in the subsequent calls the values of these arrays are exactly as at the end of routine **BASRED**. The values in the remaining arrays are exactly as at the end of routine **LBALG**.

The network problem in step 2 of ALG 3.4 is converted to a problem with zero lower bounds and the objective function value is readjusted. That is, for arc j,

$$0 \leq \mathbf{GFLOW}(j) \leq \mathbf{UPPER}(j) \qquad \text{for } \mathbf{GCOEF}(j) = 0;$$

$$0 \leq \mathbf{GFLOW}(j) \leq \mathbf{ZFLOW}(j) \qquad \text{for } \mathbf{GCOEF}(j) > 0;$$

$$0 \leq \mathbf{GFLOW}(j) \leq \mathbf{UPPER}(j) - \mathbf{ZFLOW}(j) \qquad \text{for } \mathbf{GCOEF}(j) < 0.$$

But since the value of **ZFLOW**(j) might change from one iteration to the next, a reoptimization procedure for upper bounds changes that was discussed in Section 2.1.1.2 is used to find an advanced starting feasible solution for the network problem. At this point the vector **UTEMP** contains the net change in the upper bounds that was obtained in the last call of routine **PROJOP**. Here, the values of array **UTEMP** is exactly as at the end of routine **PROJOP**.

a. Subroutine **REOPT**

When there is a change in the bounds of the network problem in step 2 of ALG 3.4, this routine recomputes the values of the basic variables, i.e. the array **FLOW**; and hence **GFLOW**. The routine makes four arc–length passes to determine the new values of **FLOW** and hence **GFLOW**. In the first pass PROCEDURE X2D of Section 2.1.1.2 is used to compute the vector of reduced requirements. On the second pass the vector of reduced requirements is adjusted to account for the upper bound changes of nonbasic arcs only (if any exists). On the third pass this vector of reduced requirements is used to construct the flows on the basic arcs by using PROCEDURE D2X of Section 2.1.1.2. At this point, it is possible that the basic flows are either negative or larger than their upper bounds. Hence, the fourth pass adjusts for the basic flows that exceed their bounds as was discussed in case 2b of CHANGING AN UPPER BOUND in Section 2.1.1.2. It is at this pass that the **UTEMP** array is reconstructed to be the new upper bound on all the variables, that is, for all $j \in \mathcal{A}$

```
        UTEMP(j) = UPPER(j)               if GCOEF(j) = 0;
        UTEMP(j) = ZFLOW(j)               if GCOEF(j) > 0;
        UTEMP(j) = UPPER(j) - ZFLOW(j)    if GCOEF(j) < 0.
```

These values of the arrays **FLOW**, **GFLOW** and **UTEMP** are passed along to routine **UBALG**.

Back in routine **UBALG**, the routine **PURNET** with the vector of upper bounds **UTEMP** is called to solve for the pure network problem with **GFLOW** as an advanced starting feasible solution.

Having an optimal solution to the pure network problem, the routine stops with a near optimal solution if (**UBND** - **LBND**) is less than or equal to **EPSILON** $|\mathbf{UBND}|$; otherwise, the arrays of the subgradient **UBSUBG** and the feasible solution **ZFLOW** are constructed.

Three arc—length passes are made to determine **UBSUBG** and the new values of **ZFLOW**. In the first pass step 2 of ALG 3.4 is used to compute the subgradient. This is done as follows:

```
Repeat for all $j \in \mathcal{A}$ with $j = (f(j), t(j))$
   If STATUS(j) = 2 and GCOEF(j) > 0 or
      STATUS(j) = 1 and GCOEF(j) < 0, then :
      UBSUBG(j) = COST(j) + PI(f(j)) - PI(t(j)).
      NORM = NORM + $\|$ UBSUBG(j) $\|^2$.
```

where **NORM** is the variable that stores the norm of the subgradient. This would provide a tool for using any of the schemes (i) — (iv) that was discussed in Section 2.1.3. Presently **UBSTEP** is set to be

$$\frac{\text{UBND} - \text{LBND}}{2 * \text{NORM}} .$$

On the second pass the new vector of ZFLOW that corresponds to point $y - t\eta$ in step 3 of ALG 3.4 is computed.

```
Repeat for GUB k
   Repeat for all n ∈ [LOC1,LASTRC] with j=GUBADD(n)
      If REDGUB(k) = 0, then
         ZFLOW(j) = ZFLOW(j) - UBSTEP * UBSUBG(j)
      If REDGUB(k) = 1, then :
         UTEMP(j) = 0
         If GCOEF(j) > 0, then ZFLOW(j) = UPPER(j)
         Else ZFLOW(j) = 0.
```

On the third pass, the array ZFLOW is computed so that it is feasible for problem (*NPGD*). At the end of routine PROJOP, ZFLOW would be the next feasible solution to problem (*NPGD*).

H. Subroutine BASSAV

At this point, this routine stores the information in scratch file UBOBAS for use in the next set of iterations of routine UBALG.

I. Subroutine **BASRED**

At this point, this routine reads in the information from the scratch file **LBOBAS** for use in the next set of iterations of routine **LBALG**.

J. Subroutine **OUTPUT**

This routine produces the objective function values and the solution for both the lower and upper bound procedures. It also reports the total number of lower and upper bound iterations that were performed before reaching optimality or near optimality. This routine may be modified to produce an output report to specification for a given application.

CHAPTER 5
COMPUTATIONAL EXPERIMENTS

In this chapter a summary of some experimental results with the software described in Chapter 4 will be presented. The experiments were intended to give some general notion on the performance of the software and also to point out factors that have a substantial effect on this performance.

5.1 Test Problems

The algorithm has been tested on a set of 13 problems. All the problems were generated using the NETGEN (Klingman, Napier and Stutz (1974)) program to randomly generate the pure network portion of the problem. These pure network problems were solved using the NETFLO program (Kennington and Helgason (1980, p. 244)), a large–scale pure network problem solver. The idea was to use the pure network optimal solution in generating the GUB constraints portion so that the (network) solution remains feasible to the GUB constraints.

For each problem, the GUB constraints portion was generated as follows. A permutation of the arcs was determined. For each GUB constraint k, $1 \leq k \leq p$, the number of variables in that GUB constraint, n_k, was randomly generated. These numbers have a lower bound limit of 2 to ensure that each GUB constraint would consist of at least two variables. Then the first n_1 variables in the permutation were selected as the set of variables in the first GUB constraint, the next n_2 were selected as the set of variables in the second GUB constraint, ..., the next n_p were selected as the set of variables in the pth GUB constraint and the remaining variables were chosen to be the set of variables not in any GUB constraint. Having a set of variables for each GUB constraint, the coefficients of these variables were then generated to be within prespecified bounds (the lower bounds ranged from -5 to -2 and the upper bounds ranged from 2 to 5). These coefficients were further checked to ensure for their nonzero values. Any coefficient having a value of zero was then regenerated until a nonzero one was found. These coefficients together with the pure network solution provided a way for generating a right hand side value for each GUB constraint. In generating these values, the feasibility criteria (step 1 of ALG 2.3) of the GUB constraint was enforced. At this point the cost coefficients of the arcs with nonzero values in the pure network solution were increased by the amount c_{max}, defined to be the maximum of the cost coefficients in the original pure network problem, to ensure that the pure network solution was not optimal to the network with GUB constraints problem. Note that a problem generated in this manner is known to be feasible, with the pure network solution not optimal to the problem. Table 5.1 shows the main characteristics of these sample problems.

5.2 Performance Criteria

Experimental testing was carried out on a SUN 3/50 workstation which uses a MOTOROLA 68020 CPU with 68881 numeric co-processor, running at 16 MHZ. The computational results reported are the number of iterations performed for the lower and upper bound procedures before reaching a prespecified tolerance ϵ. The SUN FORTRAN function DTIME was considered for timing purposes; however due to the fact that solution time depends upon the number of users on the computer and also the available space on hard disk no times are reported. For example, problem 1 reached the 1% tolerance in 3.2 seconds and the 0.5% tolerance in 5.6 seconds on a day the computer was not overloaded, whereas on a day the computer was overloaded it took 32.4 seconds to reach the 0.5%

Table 5.1 : Sample Problems

Problem Number	Nodes	Arcs	Number of GUBs	Percentage of Arcs in GUBs
1	10	30	12	93.33%
2	50	250	75	99.20%
3	50	500	75	79.60%
4	50	500	75	99.60%
5	50	500	100	97.60%
6	100	500	100	99.80%
7	200	500	100	74.60%
8	50	1000	75	97.20%
9	50	1000	100	97.80%
10	50	1000	100	78.80%
11	200	1000	100	80.30%
12	50	2000	75	75.00%
13	200	2000	100	90.15%

tolerance. This indicated that the timings were not reliable. For the 1% tolerance the algorithm made 42 lower bound iterations and 20 upper bound iterations. For the 0.5% tolerance the number of lower and upper bound iterations were 100 and 45 respectively.

5.3 <u>Step Sizes</u>

As was mentioned in Section 2.1.3 step sizes play an important role in the convergence of the lower and upper bound algorithms. For this reason different step sizes were tried to find one suitable for these types of problems. As was also mentioned in Section 4.2, the step size used in the upper bound procedure is $(\text{UBND} - \text{LBND})/2\|\eta_u\|^2$, where LBND is the best lower bound value and η_u is the upper bound subgradient. This scheme seems to work fairly well. For example, in problem 2 UBND was within 0.5% of optimality in 10 upper bound iterations; in problem 4 it was within 0.7% of optimality in 275 upper bound iterations; and in problems 5 and 6 it was within 0.5% of optimality in 34 and 273 upper bound iterations, respectively. Hence the main interest was focused on the lower bound step sizes. A description of some of the lower bound step sizes that were tried together with their effect on convergence follows.

(a) <u>Fixed Step Size</u>. This would fall in the category of step sizes of scheme (i) in Section 2.1.3. The step sizes used were $d_i = \alpha d_{i-1}$, where d_0 is prespecified and α is a real number less than 1. This proved to be a very poor choice since for small step sizes the change in the cost coefficients and hence the change in the objective function value of the pure network subproblems becomes very small, thus causing the LBND to recycle (that is, the same LBND value was repeated). Secondly this normally occurs a long way from optimality since the adjustment of the step sizes is not based on the behavior of the lower bound function. Even if the initial solution is close to the optimal solution, the convergence depends on the choice of d_0 which in turn requires some prior knowledge of the optimal objective function value, which one obviously does not have.

(b) <u>Step Size Based on the Lower Bound Subgradient</u>. The step sizes are of the form $d_i = \lambda_i/\|\eta_1^i\|^2$, where η_1^i is the lower bound subgradient at the ith iteration and λ_i is a constant. The difficulty arises in the choice of λ_i. Ideally, if the solution to the current lower bound problem is close to feasibility, a small step size should be taken; whereas for solutions far away from feasibility a larger step size should be taken. But how large is large? It is clear that λ_i equal to some prespecified value λ_0, is a poor choice since d_i may then be large for small values of η_1^i, that is, as the solution gets closer to feasibility a larger step size is taken. A good choice of λ_i would seem to be one that reflects some measure of the violations of the GUBs. Hence $\lambda_i = \text{MAXVLN}_i$ was chosen, where MAXVLN_i is the maximum violation in a GUB constraint after the ith iteration.

An attempt was made to restrict the step sizes so that an improved lower bound value on the problem is guaranteed at each lower bound iteration, i.e. so the sequence of lower bounds was monotonically increasing. This was tried by limiting the step sizes to be the maximum of the current step size and the largest of the previous step sizes taken. This proved to have the same effect as the fixed step size approach. This is due to the fact that by limiting the step sizes, we force the lower bound function value to go in one direction only which might not be the "best" direction, that is, the direction might be a direction of descent rather than ascent.

Table 5.2 summarizes the effect of decreasing ϵ on the problems when using $d_i = (\text{MAXVLN}_i)/\|\eta_1^i\|^2$ as the lower bound step sizes. For each problem and for a given ϵ, the first number in the table is the total number of lower bound iterations and the second entry is the number of

Table 5.2 : Effect of Decreasing ϵ
(DNC → Did not converge after 2000 lower bound or 1000 upper bound iterations)

Problem Number	ϵ			
	5%	3%	1%	0.5%
1	(20)(8)	(30)(10)	(42)(20)	(100)(45)
2	(20)(1)	(23)(10)	(151)(70)	(600)(298)
3	(20)(1)	(20)(1)	(188)(90)	(624)(310)
4	(20)(1)	(60)(26)	(2000)(1000)	DNC
5	(42)(20)	(184)(90)	(1219)(600)	DNC*
6	(80)(32)	(171)(80)	(1183)(590)	DNC
7	(20)(1)	(33)(10)	(693)(340)	(1994)(990)
8	(31)(10)	(90)(40)	(1197)(590)	DNC
9	(122)(60)	(534)(260)	DNC**	DNC
10	(46)(20)	(80)(35)	(1220)(602)	DNC
11	(60)(22)	(300)(150)	DNC***	DNC
12	(20)(1)	(20)(2)	(116)(50)	(233)(110)
13	(20)(1)	(40)(12)	(1020)(510)	DNC

* The .55% tolerance was achieved after 2000 lower bound and 1000 upper bound iterations.

** The 2% tolerance was achieved after 2000 lower bound and 1000 upper bound iterations.

*** The 1.4% tolerance was achieved after 2000 lower bound and 1000 upper bound iterations.

upper bound iterations before reaching the tolerance ϵ. Recall that the program always performs 20 lower bound iterations before going to the upper bound algorithm (unless, of course, feasibility is reached in the lower bound algorithm). It is clear from these problems that the percentage of arcs in the GUB constraints has a major effect on the number of iterations performed and hence on the solution effort. For example, problems 3 and 4 are of the same size (50 nodes and 500 arcs). 79.60% and 99.60% of the arcs in problems 3 and 4, respectively, are in the GUB constraints. 188 lower bound and 90 upper bound iterations were needed before reaching the 1% tolerance in problem 3 whereas 2000 lower bound and 1000 upper bound iterations were needed in problem 4 to reach the same tolerance level. It is also clear that the convergence of the lower bound algorithm is quite slow. Consequently, further investigation is needed on the lower bound step sizes in order to improve on the rate of convergence of the lower bound algorithm.

5.4 Initial Lower Bound Solution

This section investigates the effect of the initial lower bound value for the problem on the convergence rate of the algorithm. Note that since **w=0** initially (step 1 of ALG 3.2), the initial lower bound value on the problem is equal to the optimal objective function value of the original pure network portion of the problem.

Table 5.3 demonstrates the percentage difference between the best UBND value at a given tolerance ϵ and the initial lower bound value. For instance, at the 3% tolerance, the difference between the initial lower bound value and the best UBND value is 1.80% for problem 3, 3.08% for problem 13, and 6.35% for problem 6. By comparing the number of lower bound iterations from Table 5.2 with the percentages in Table 5.3 (both at the 3% tolerance), it appears that when the initial lower bound is within 4% of the best upper bound, the algorithm almost always requires fewer than 100 lower bound iterations. This suggests that an improved estimate of w in calculating $L_1(w)$ in initialization could greatly enhance the performance of the lower bound algorithm. This is investigated further with the following example.

Consider problem 6 at the 3% tolerance level. From Table 5.2 we can see that this tolerance level was achieved after 171 lower bound and 80 upper bound iterations. Table 5.4 presents a summary of the best lower bound and upper bound values on the problem at some selected iteration counts. The initial lower bound value is 5.62% different from the optimal objective value of the problem. It is clear from Table 5.4 that the improvement

Table 5.3 : Distance Between the Initial Lower Bound Value and the Best Upper Bound.

Problem Number	ϵ			
	5%	3%	1%	0.5%
1	9.31%	9.00%	8.38%	8.10%
2	7.19%	7.14%	7.07%	6.99%
3	1.80%	1.80%	1.47%	1.22%
4	4.19%	3.42%	2.91%	
5	6.00%	5.69%	5.68%	5.68% *
6	6.73%	6.35%		
7	3.44%	3.13%	2.83%	2.83%
8	8.07%	7.63%	7.01%	
9	8.96%	8.37%	8.24% **	
10	13.49%	13.39%	12.67%	
11	5.19%	3.92%		
12	3.59%	3.53%	3.01%	2.95%
13	4.03%	3.08%	1.93%	

* At 0.55% tolerance level.
** At 2% tolerance level.

Table 5.4 : Performance of Problem 6 Before Reaching the 3% Tolerance Level

Total Number of Lower Bound Iterations	Total Number of Upper Bound Iterations	The Best LBND Value	The Best UBND Value
20	1	148777.13	166168.27
20	10	148777.13	160756.39
30	10	149137.08	160756.39
40	20	149534.85	159397.58
50	20	149918.59	159397.58
60	30	150284.27	158926.75
70	30	150597.03	158926.75
80	40	150941.40	158669.89
90	40	151318.02	158669.89
100	50	151645.62	158495.93
110	50	151919.13	158495.93
120	60	152210.06	158383.68
130	60	152496.88	158383.68
140	70	152737.78	158283.46
150	70	153020.51	158283.46
160	80	153198.63	158237.85
170	80	153438.83	158237.85
171	80	153491.91	158237.85

Optimal Value	= 157005.08
Initial Lower Bound Value	= 148184.00

of the lower bound values are quite small. For example, at the end of the 20th iteration, the difference between the best lower bound value and the optimal value is 5.24%, while the lower bound value has only had a 0.40% improvement (in 20 iterations). At the 60th iteration, this difference is 4.28%, with an improvement of 1.40% over the initial lower bound value. At the 170th iteration, the difference is 4.28%, with an improvement of 3.4% (over the initial lower bound value). On the contrary, the best UBND values are doing quite well. At the first iteration, the UBND value is within 5.51% of optimality; at the end of the 10th upper bound iteration, the best UBND has improved by 3.26%, giving a value that is within 2.33% of optimality. At the end of the 20th iteration, the best UBND value is within 1.5% of optimality; at the end of the 50th iteration, the best UBND value is within 0.94% of optimality, an improvement of over 4.62% from the first UBND value. At 3% tolerance, the best lower bound value is within 2.24% of optimality, whereas the best upper bound value is within 0.78%. This example indicates that if a better initial lower bound value were available, near optimality could have been reached for a smaller number of lower bound iterations and consequently the solution effort could have been reduced.

5.5 Lower and Upper Bound Iteration Strategies

Recall that the algorithm is developed so that the lower bound procedure is performed initially before the upper bound procedure; in addition, the code is written so that a set of 20 lower bound iterations are invoked before a set of 10 upper bound iterations. These values were arbitrarily selected; however it appears from these experiments that one may need to find a better initial strategy for the problem in order to improve on the performance of the algorithm. This is further illustrated by means of an example.

Consider problem 3 at the 3% tolerance level. This tolerance level is achieved after 20 lower bound and 1 upper bound iterations. Table 5.3 shows that the difference between the best UBND value and the initial lower bound value is 1.80%. This indicates that if one knew what the UBND value would have been prior to performing the lower bound procedure, only one lower bound iteration would have been needed to reach the 3% tolerance (and therefore also the 5% tolerance level). It might be advantageous to perform one of each of the lower and upper bound iterations and compute the difference between the two bounds before deciding on the number of lower and upper bound iterations.

CHAPTER 6
SUMMARY AND CONCLUSIONS

A relaxation and decomposition algorithm for the network problem with generalized upper bound side constraints was presented. The solution technique was developed to take advantage of the structure of the side constraints and simultaneously maintain as many of the characteristics of the pure network problems as possible. The sequence of relaxation problems, which yields lower bound values on the problem, has a fairly slow convergence rate to optimality. On the other hand, the sequence of decomposition problems which yields upper bound values on the problem seems to perform quite well. The solution technique seems best suited for a real—world situation in which one must quickly obtain near—optimal solutions.

6.1 Results of the Research

The purpose of this research was to develop and computationally test a new algorithm for the class of network with GUB side constraints problem. This class of problem is included in the class of network with arbitrary side constraints problem; however, no algorithm that exploits the special structure of the GUB side constraints had previously existed. The proposed algorithm solved the network with GUB side constraints problem using two sequences of problems. One sequence corresponded to computing improved lower bounds while the other corresponded to computing tighter upper bounds on the optimal value of the problem.

The lower bound procedure was developed to bound the optimal value from below by using a Lagrangean relaxation based on relaxing copies of some subset of the original variables. The Lagrangean relaxation, of course, did not enforce feasibility of the GUB constraints; hence a penalty was assessed when the solution to the pure network subproblem violated the GUB constraints. It was further established that a lower bound value could be found by first solving a pure network subproblem and then solving a set of single constraint bounded variable LPs. Because only the cost coefficients changed from one pure network subproblem to another (subproblem), the optimal solution for one subproblem was at least feasible, if not optimal, for the next pure network subproblem.

The upper bound procedure was developed to bound the optimal value from above by using a decomposition of the problem based on changes in the capacity vector. Solving for the decomposed problem corresponded to solving for pure network subproblems that had undergone bounds changes. The reoptimization procedures were used in finding a (artificial) feasible solution to a pure network subproblem by making use of the optimal solution to the previous pure network subproblem.

The Lagrangean relaxation and decomposition techniques have been widely used in mathematical programming, but no solution technique based on these was available for the network models with GUB side constraints; thus their performance for this class of problems had been unknown. The NETGUB program was developed for solving the network with GUB constraints problems by utilizing the relaxation and decomposition techniques.

The computational experiments indicated that further improvements on the mechanics of the lower bound procedure are needed to produce a more efficient algorithm. Further work is also required on the coding of the program itself to increase the efficiency of the algorithm.

6.2 Suggestions for Further Research

This section summarizes the suggestions that were raised in Chapter 5 for further improving the performance of the algorithm.

(a) It would be advantageous to find another way of initializing the problem in order to obtain a better initial lower bound value. The main difficulty here is in choosing an initial value for w. One would like to take advantage of the initial optimal solution to the pure network portion of the problem to choose a w so that the violations in the GUB constraints are reduced.

(b) It is clear from the experiments of Chapter 5 that further investigation on the lower bound step sizes is needed. The difficulty here is that these step sizes have to satisfy the conditions of Section 2.1.3; they also have to reflect the behavior of the lower bound function; and, in addition, they have to work for all the problems equally well. It is possible that one overall scheme may not do well for a given problem. One may need to have two schemes; one scheme when the lower bound values are far away from optimality (this may be determined by some measure of the tolerance) so that larger steps are taken, and another one when the lower bound value is close to optimality.

(c) It appears from the experiments that a better iteration strategy for the problem is needed. Is it advantageous, in the long run, to initially do one iteration of each of the lower and upper bound algorithms? The difficulty is in finding a strategy that works well for every given problem. Suppose that for a given problem one iteration of each of the algorithms is performed and the difference between the two bounds are calculated. If this difference is smaller than the prespecified tolerance level, the algorithm should clearly stop with a near optimal solution. If the difference is close but a little larger than the prespecified tolerance level, one strategy might be to perform one additional iteration of each of the algorithms. If the difference is large, it is not clear what the strategy should be since one has no idea which of the values needs to be improved the most. One possibility is to perform a set of lower bound iterations until small changes in the lower bound values are detected before going to the upper bound algorithm.

BIBLIOGRAPHY

Aggarwal, V. (1985), "A Lagrangean-Relaxation Method for the Constrained Assignment Problem", *Computers and Operations Research*, Vol 12, No. 1, pp. 97-106.

Ali, A. I., E. Allen, R. S. Barr and J. L. Kennington (1986), "Reoptimization Procedures for Bounded Primal Simplex Network Algorithms", *European Journal of Operational Research*, Vol. 23, pp. 256-263.

Ali, A. I., D. Barnett, K. Farhangian, J. Kennington, B. Mc Carl, B. Patty, B. Shetty and P. Wong (1984), "Multicommodity Network Problems: Applications and Computations", *IIE Transactions*, Vol. 16, No. 2, pp. 127-134.

Ali, A., R. Helgason and J. Kennington (1987), "An Air Force Logistics Decision Support System Using Multicommodity Network Models", *The Annals of the Society of Logistic Engineers*, Vol. 1, No. 2, pp. 93-105.

Ali, A. I., J. Kennington and B. Shetty (1985), "The Equal Flow Problem", Technical Report 85-OR-1, Department of Operations Research, Southern Methodist University, Dallas, Texas.

Allen, E., R. Helgason, J. Kennington, B. Shetty (1987), "A Generalization of Polyak's Convergence Result for Subgradient Optimization", *Mathematical Programming*, Vol. 37, pp. 309-317.

Barr, R. S., K. Farhangian and J. L. Kennington (1986), "Networks with Side Constraints: An LU Factorization Update", *The Annals of the Society of Logistics Engineers*, Vol. 1, NO. 1, pp. 66-85.

Barr, R. S., F. Glover and D. Klingman (1974), "An Improved Version of The Out-of-Kilter Method and a Comparative Study of Computer Codes", *Mathematical Programming*, Vol. 7, pp. 60-86.

Barr, R. S., F. Glover and D. Klingman (1979), "Enhancements to Spanning Tree Labelling Procedures for Network Optimization", *Infor*, Vol. 17, No. 1, pp. 16-34.

Bazaraa, M. S. and J. J. Jarvis (1977), *Linear Programming and Network Flows*, John Wiley and Sons, New York.

Bazaraa, M. S. and C. M. Shetty (1979), *Nonlinear Programming: Theory and Algorithms*, John Wiley and Sons, New York.

Bennett, J. M. (1966), "An Approach to Some Structured Linear Programming Problems", *Operations Research*, Vol. 14, pp. 636-645.

Bolouri, M. and J. L. Arthur (1989), "A Greedy Algorithm for the Single Constraint Linear Program with Bounded Variables", Technical Report No. 128, Department of Statistics, Oregon State University, Corvallis, Oregon.

Bradley, G. H., G. G. Brown and G. W. Graves (1977), "Design and Implementation of Large-Scale Primal Transshipment Algorithms", *Management Science*, Vol. 24, No. 1, pp. 1-34.

Brown, G. G. and R. D. McBride (1984), "Solving Generalized Networks", *Management Science*, Vol. 30, No. 12, pp. 1497-1523.

Brown, G. G. , R. D. McBride and R. K. Wood (1985), "Extracting Embedded Generalized Networks from Linear Programming Problems", *Mathematical Programming*, Vol. 32, No. 1, pp. 11-31.

Charnes, A. and W. Cooper (1961), *Management Models and Industrial Application of Linear Programming*, Vol. II, John Wiley, New York.

Chen, S. and R. Saigal (1977), "A Primal Algorithm for Solving a Capacitated Network Flow Problem with Additional Linear Constraints", *Networks*, Vol. 7, No. 1, pp. 59-79.

Chvatal, V. (1980), "Hard Knapsack Problems", *Operations Research*, Vol. 28, No. 6, pp. 1402-1411.

Dantzig, G. B. (1951), "Application of the Simplex Method to a Transportation Problem", in T. C. Koopmans, Ed., *Activity Analysis of Production and Allocation*, John Wiley and Sons, New York.

Dantzig, G. B. (1963), *Linear Programming and Extensions*, Princeton University Press, Princeton, N. J..

Dantzig, G. B. and R. M. Van Slyke (1967), "Generalized Upper Bounding Techniques", *Journal of Computer and System Sciences*, Vol. 1, No. 3, pp. 213-226.

Fisher, M. L. (1981), "The Lagrangean Relaxation Method for Solving Integer Programming Problems", *Management Science*, Vol. 27, No. 1, pp. 1-18.

Ford, L. R., Jr. and D. R. Fulkerson (1958), "A Suggested Computation for Maximal Multicommodity Network Flows", *Management Science*, Vol. 5, No. 1, pp. 97-101.

Fulkerson, D. R. (1961), "An Out—of—Kilter Method for Maximal—Cost Flow Problem", *Journal of the Society of Industrial and Applied Mathematics*, Vol. 9, No 1, pp. 18-27.

Geoffrion, A. M. (1974), "Lagrangian Relaxation and its Uses in Integer Programming", *Mathematical Programming Study*, Vol. 2, pp. 82-114.

Glover, F., R. Glover, J. Lorenzo and C. McMillan (1982), "The Passenger—Mix Problem in the Scheduled Airlines", *Interfaces*, Vol. 12, No. 3, pp. 73—80.

Glover, F., D. Karney and D. Klingman (1972), "The Augmented Predecessor Index Method for Locating Stepping Stone Paths and Assigning Dual Prices in Distribution Problems", *Transportation Science*, Vol. 6, No. 1, pp. 171—180.

Glover, F., D. Karney and D. Klingman (1974), "Implementation and Computational Comparisons of Primal, Dual and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems", *Networks*, Vol. 4, No. 3, pp. 191-212.

Glover, F., D. Karney, D. Klingman and A. Napier (1974), "A Computational Study on Start Procedures, Basis Change Criterion, and Solution Algorithms for Transportation Problems", *Management Science*, Vol. 20, No. 5, pp. 793-813.

Glover, F. and D. Klingman (1973), "On the Equivalence of Some Generalized Network Problems to Pure Network Problems", *Mathematical Programming*, Vol. 4, No. 3, pp. 369-378.

Glover, F. and D. Klingman (1981), "The Simplex SON Algorithm for LP/Embedded Network Problems", *Mathematical Programming Study 15*, pp. 148-176.

Glover, F. and D. Klingman (1988), "Layering Strategies for Creating Exploitable Structure in Linear and Integer Programs", *Mathematical Programming*, Vol. 40, pp. 165-181.

Glover, F., D. Klingman and G. T. Ross (1974), "Finding Equivalent Transportation Formulations for Constrained Transportation Problems", *Naval Research Logistics Quarterly*, Vol. 21, pp. 247—254.

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

Goffin, J. L. (1977), "On Convergence Rates of Subgradient Optimization Methods", *Mathematical Programming*, Vol. 13, pp. 329-347.

Graves, G. W. and R. D. McBride (1976), "The Factorization Approach to Large—Scale Linear Programming", *Mathematical Programming*, Vol. 10, No. 1, pp. 91-110.

Hartman, J. K. and L. S. Lasdon (1970), "A Generalized Upper Bounding Method for Doubly Coupled Linear Programs", *Naval Research Logistics Quarterly*, Vol. 17, No. 4, pp. 411-429.

Hartman, J. K. and L. S. Lasdon (1972), "A Generalized Upper Bounding Algorithm for Multicommodity Network Flow Problems", *Networks*, Vol. 1, pp. 333-354.

Held, M., P. Wolfe, H. Crowder (1974), "Validation of Subgradient Optimization", *Mathematical Programming*, Vol. 6, pp. 62-88.

Hitchcock, F. L. (1941), "The Distribution of a Product from Several Sources to Numerous Localities", *Journal of Mathematics and Physics*, Vol. 20, pp. 224-230.

Ingargiola, G. P. and J. F. Korsh (1977), "A General Algorithm for One—Dimensional Knapsack Problems", *Operations Research*, Vol. 25, No. 5, pp. 752-759.

Johnson, E. L. (1966), " Networks and Basic Solutions", *Operations Research*, Vol. 14, No. 4, pp. 619-623.

Kaul, R. N. (1965), "An Extension of Generalized Upper Bounded Techniques for Linear Programming", ORC Report 65-27, Department of Operations Research, University of California, Berkeley.

Kennington, J. L. (1977), "Solving Multicommodity Transportation Problems Using a Primal Partitioning Simplex Technique", *Naval Research Logistics Quarterly*, Vol. 24, No. 2, pp. 309-325.

Kennington, J. L. and R. V. Helgason (1980), *Algorithms for Network Programming*, Wiley, New York.

Kennington, J. L. and M. Shalaby (1977), "An Efficient Subgradient Procedure for Minimal Cost Multicommodity Flow Problems", *Management Science*, Vol. 23, No. 9, pp. 994-1004.

Klingman, D., J. Mote and N. V. Phillips (1988), "A Logistics Planning System at W. R. Grace", *Operations Research*, Vol. 36, No. 6, pp. 811-822.

Klingman, D.,A. Napier and J. Stutz (1974), "NETGEN : A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems", *Management Science*, Vol. 20, No. 5, pp. 814-821.

Klingman, D. and R. Russell (1975), "Solving Constrained Transportation Problems", *Operations Research*, Vol. 23, No. 1, pp. 91-106.

Koopmans, T. C. (1947), "Optimization Utilization of the Transportation System", *Proceeding of the International Statistical Conferences*, Washington, D.C..

Lasdon, L. S. (1970), *Optimization Theory for Large Systems*, Macmillan, New York.

Mulvey, J. M. (1978), "Pivot Strategies for Primal-simplex Network Codes", *Journal of the Association for Computing Machinery*, Vol. 25, No. 2, pp. 266-270.

Murty, K. G. (1976), *Linear and Combinatorial Programming*, Wiley, New York.

Murty, K. G. (1983), *Linear Programming*, Wiley, New York.

Nijenhuis, A. and H. S. Wilf (1978), *Combinatorial Algorithms : For Computers and Calculators*, Academic Press, Inc., Orlando, Florida.

Price, W. L. and M. Gravel (1984), "Solving Network Manpower Problems with Side Constraints", *European Journal of Operational Research*, Vol. 15, No. 2, pp. 196-202.

Sakaravitch, M. and R. Saigal (1967), "An Extension of Generalized Upper Bounding Techniques for Structured Linear Programs", *SIAM Journal of Applied Mathematics*, Vol. 15, No. 4, pp. 906-914.

Shapiro, J. M. (1979), *Mathematical Programming: Structures and Algorithms*, John Wiley and Sons, New York.

Shepardson, F. and R. E. Marsten (1980), "A Lagrangean Relaxation Algorithm for the Two Duty Period Scheduling Problem", *Management Science*, Vol. 26, No. 3, pp. 274-281.

Srinivasan, V. and G. L. Thompson (1972), "Accelerated Algorithms for Labelling and Relabelling of Trees, with Applications to Distribution Problems", *Journal of Association of Computing Machinery*, Vol. 19, No. 4, pp. 712.

Srinivasan, V. and G. L. Thompson (1973), "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm", *Journal of the Association for Computing Machinery*, Vol. 20, No. 2, pp. 194-213.

Tomlin, J. A. (1966), "Minimum—Cost Multicommodity Network Flows", *Operations Research*, Vol. 14, No. 1, pp. 45-51.

Wagner, H. M. (1975), *Principles of Operations Research*, 2nd Edition, Prentice—Hall, Inc., New Jersey.

Weigel, H. S. and J. E. Cremeans (1972), " The Multicommodity Network Flow Model Revised to Include Vehicle Per Time Period and Node Constraints", *Naval Research Logistics Quarterly*, Vol. 19, No. 1, pp. 77—89.

APPENDICES

APPENDIX A

Example Data File

NETGUB has the capability of solving successive problems in one run. The following is a description of the data required to specify each problem. In what follows, I8 indicates an integer field of 8 characters right justified, 2X indicates two blank spaces, A8 indicates a character field of 8 characters and F10.2 indicates a real field of 10 characters.

Card Group	Compo-sition	
1	One Card	NODSEC (node section title) (A8)
2	Card Set	Node Number, Node Requirement (non-zeros only) (A8) 2X (I10)
3	One Card	GUBSEC (GUB section title) (A8)
4	Card Set	Number of GUB constraints (I8) Number of var. in a GUB RHS value (I8) 2X (F10.2) (ordered by GUB constraint number)
5	One Card	ARCSEC (arc section title) (A8)
6	Card Set	Name, from, to, unit cost, (A8) 2X (A8) 2X (A8) 2X (F10.2) upper bound, lower bound, (F10.2) (F10.2) coefficient in a GUB constraint (F10.2)

The arcs in set 6 are arranged with all arcs that are in the first GUB constraint, followed by all arcs in the second GUB constraint, ..., followed by all arcs in the pth GUB constraint.

Example :

Minimize $10x_1 + 190x_2 + 142\, x_3 + 6x_4 + 53x_5 + 109x_6 + 48x_7 + 60x_8 + 123x_9 + 123x_{10} + 54x_{11} + 67x_{12}$

Subject to

$$
\begin{aligned}
-x_{10} &= -2 \\
-x_1 -x_5 -x_6 &= -1 \\
x_1 -x_{11} +x_{12} &= 0 \\
x_3 +x_4 +x_6 -x_7 -x_9 &= 0 \\
x_4 +x_5 -x_8 -x_{12} &= 0 \\
-x_2 -x_3 +x_7 +x_{10} &= 0 \\
+x_9 +x_{11} &= 2 \\
x_2 +x_8 &= 1 \\
x_3 -4x_4 -3x_5 &\leq 2.60 \\
-x_6 +2x_7 -4x_8 +2x_9 +4x_{10} &\leq 15.28 \\
-x_{11} -x_{12} &\leq 0.05
\end{aligned}
$$

$0 \leq x_1 \leq 7$, $0 \leq x_2 \leq 5$, $0 \leq x_3 \leq 5$,

$0 \leq x_4 \leq 6$, $0 \leq x_5 \leq 17$, $0 \leq x_6 \leq 7$,

$0 \leq x_7 \leq 7$, $0 \leq x_8 \leq 17$, $0 \leq x_9 \leq 5$,

$0 \leq x_{10} \leq 5$, $0 \leq x_{11} \leq 17$, $0 \leq x_{12} \leq 17$,

Input File for the Example

```
NODSEC
      1         -2
      2         -1
      7          2
      8          1
GUBSEC
      3
      3       2.60
      5      15.28
      2       0.05
ARCSEC
      3          6          4     142.00      5.00      0.00      1.00
      4          5          4       6.00      6.00      0.00     -4.00
      5          2          5      53.00     17.00      0.00     -3.00
      6          2          4     109.00      7.00      0.00     -1.00
      7          4          6      48.00      7.00      0.00      2.00
      8          5          8      60.00     17.00      0.00     -4.00
      9          4          7     123.00      5.00      0.00      2.00
     10          1          6     123.00      5.00      0.00      4.00
     11          3          7      54.00     17.00      0.00     -1.00
     12          5          3      67.00     17.00      0.00     -1.00
      1          2          3      10.00      7.00      0.00      0.00
      2          6          8     190.00      5.00      0.00      0.00
ENDATA
ENDPRB
```

APPENDIX B

NETGUB Source Listing

```
C
      PROGRAM NETGUB
C
      INTEGER DIMARC,DIMNOD,DIMGUB,DIMGARC
      DOUBLE PRECISION LBSTEPZ
      PARAMETER(DIMARC=2500)
      PARAMETER(DIMNOD=500)
      PARAMETER(DIMGUB=500)
      PARAMETER(DIMGARC=2500)
      PARAMETER(LBSTEPZ=.5)
C
C  ARRAYS
C
      CHARACTER*8 ARCNAM(DIMARC),NODNAM(DIMNOD),DATAFILE,OUTFILE
      INTEGER BASIS(DIMNOD),CARD(DIMNOD),FROM(DIMARC),
     1       FROMO(DIMARC),LNOD(DIMNOD),ORDER(DIMARC),
     1       NUMVRG(DIMGUB),NVAL(DIMNOD),
     1       PRED(DIMNOD),REDGUB(DIMGUB),GUBADD(DIMARC),
     1       STATUS(DIMARC),THD(DIMNOD),
     1       TO(DIMARC),LSTEP,USTEP,LITER,UITER,FIRSTME
      DOUBLE PRECISION COST(DIMARC),FLOW(DIMNOD),GFLOW(DIMARC),
     1      CTEMP(DIMARC),LGMULT(DIMARC),PI(DIMNOD),
     1      UPPER(DIMARC),YFLOW(DIMARC),ZFLOW(DIMARC),
     1      UTEMP(DIMARC),GCOEF(DIMARC),
     1      GVAL(DIMGUB),LOWER(DIMARC),GARB(2*DIMARC)
C
C  LOCAL VARIABLES
C
      INTEGER DUMMY,ENDATA,PROB,LBOBAS,UBOBAS
C
      EQUIVALENCE (IND,CARD),(LEFT,PRED),(RIGHT,THD),
     *          (NUMOUT,LNOD),(FROMO,ORDER)
C
      INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *       ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *          NGUBS,NODES,ROOT,EPSILON,BIG1
      DOUBLE PRECISION LBSTEP,ILBSTEP,MAXSTEP
      COMMON /STEP/ LBSTEP,ILBSTEP,MAXSTEP
      DOUBLE PRECISION LBND,LBNDPRE,UBND,UBNDPRE
      COMMON /BOUNDS/ LBND,LBNDPRE,UBND,UBNDPRE
      INTEGER FSITER
      COMMON /FEAS/ FSITER
      DOUBLE PRECISION ALPHA
      COMMON /ALP/ ALPHA
      INTEGER NGVLT
      COMMON /NGV/ NGVLT
C
      INTEGER MLITER,MUITER,TLITER,TUITER
C
      DATA ENDATA /'ENDATA '/
C
C
C
      MAXARC=DIMARC
      MAXNOD=DIMNOD
      MAXGUB=DIMGUB
      BIG1=999999999999.0
      BIG=9999999999
      PROB=0
      FIRSTME=1
      MLITER=2000
      MUITER=1000
C
      ALPHA=2.0D0
C
C  SET THE LOWER BOUND STEP SIZE
C
      ILBSTEP=LBSTEPZ
      LBSTEP=LBSTEPZ
C
C  INITIALIZE TOTAL NUMBER OF ITERATION FOR EACH ALGORITHM
C
      LITER=20
      UITER=10
      TLITER=0
      TUITER=0
      FSITER=0
C
C  SET THE TOLERANCE
C
       OPEN(2,file='fort.2')
       REWIND 2
       READ(2,1050) EPSILON
 1050  FORMAT(F10.5)
C
      READ(*,*) DATAFILE
      READ(*,*) OUTFILE
      OPEN(5,file='fort.5')
      REWIND 5
      OPEN(8,file=DATAFILE)
      REWIND 8
      OPEN(6,file=OUTFILE)
C      OPEN(6,file='fort.6')
      REWIND 6
      READ (5,*) ITRMAX,ITROBJ,ITROUT,SAVBAS
C
C  STORE THE FINAL NETWORK BASIS IN THESE FILES
C
      LBOBAS=10
      UBOBAS=14
C
      OPEN(LBOBAS,form='UNFORMATTED',status='SCRATCH')
      OPEN(UBOBAS,form='UNFORMATTED',status='SCRATCH')
      OPEN(11,form='UNFORMATTED',status='SCRATCH')
      OPEN(13,form='UNFORMATTED',status='SCRATCH')
C
      IF (ITRMAX .EQ. 0) THEN
        ITRMAX=100000
      ENDIF
      IF (ITROBJ .EQ. 0) THEN
        ITROBJ=BIG
      ENDIF
      IF (ITROUT .EQ. 0) THEN
        ITROUT=BIG
      ENDIF
C
C  READ IN PROBLEM DATA
C
C
   10 CONTINUE
      PROB=PROB+1
      WRITE (6,1000) PROB
C
C
      CALL INPUT(ARCNAM,COST,CTEMP,FLOW,FROM,GCOEF,GUBADD,GVAL,
     *           IND,LEFT,LOWER,NODNAM,NUMOUT,NUMVRG,NVAL,
     *           ORDER,REDGUB,RIGHT,STATUS,TO,UPPER)

C
C
C  CHECK FOR BAD PROBLEM DATA FILE
C
      IF (FLGERR .EQ. -1) THEN
        WRITE (6,2000)
        GOTO 30
      ENDIF
C
C  DONE WITH ALL PROBLEMS?
C
      IF (FLGEND .EQ. 1) GOTO 30
C
C  ANY ERRORS?
```

```
C
      IF (FLGERR .NE. 0) THEN
        WRITE(6,3000) PROB
C
C  IF THERE WERE FATAL ERRORS CHECK TO SEE IF ENTIRE PROBLEM WAS
C  READ IN.  IF IT WASN'T, MOVE TO START OF NEXT PROBLEM.
C
        IF (FLGERR .EQ. 1) THEN
  20       READ (8,4000) DUMMY
           IF (DUMMY .NE. ENDATA) GOTO 20
        ENDIF
        GOTO 10
      ENDIF
C
C
C  INITIALIZE THE PROBLEM
C
C
      CALL INIT(ARCNAM,BASIS,CARD,COST,CTEMP,FLOW,FROM,FROMO,
     *         GARB,GFLOW,GUBADD,GVAL,GCOEF,LGMULT,LNOD,
     *         LOWER,NUMVRG,NVAL,PI,PRED,REDGUB,STATUS,THD,TO,
     *         UPPER,YFLOW)
C
      IF (FLGITR .EQ. 1) THEN
        WRITE(6,5000)
      ENDIF
C
      IF (FLGOPT .EQ. 1) THEN
        WRITE (6,6000)
        WRITE(6,*) 'TCOST = ',TCOST
        GO TO 70
      ENDIF
C
      IF (FLGINF .EQ. 1) GO TO 70
C
 60   CONTINUE
C
C  INITIALIZE NUMBER OF ITERATION FOR EACH ALGORITHM
C
      LSTEP=0
      USTEP=0
C
 40   CONTINUE
C
C  SOLVE THE LOWER BOUND ALGORITHM
C
      LSTEP=LSTEP+1
      TLITER=TLITER+1
C
      CALL LBALG(ARCNAM,BASIS,CARD,COST,CTEMP,FLOW,FROM,FROMO,
     *         GCOEF,GFLOW,GUBADD,GVAL,LGMULT,LNOD,
     *         LOWER,NUMVRG,PI,PRED,REDGUB,STATUS,THD,TO,
    *         UPPER,YFLOW)
      IF (FLGOPT .EQ. 1 .OR. FLGINF .EQ. 1) THEN
        GO TO 70
      ENDIF
C
      IF(TLITER .GT. MLITER) GO TO 70
      IF ( LSTEP .LT. LITER) GO TO 40
      CALL BASSAV(BASIS,CARD,FLOW,FROMO,GFLOW,LNOD,PRED,STATUS,THD,
    *          LBOBAS)
C
C  INITIALIZE THE UPPER BOUND PROCEDURE
C
      IF (FIRSTME .EQ. 1) THEN
        CALL UINIT(GARB,GFLOW,GUBADD,GVAL,GCOEF,NUMVRG,REDGUB,
    *          UPPER,UTEMP,ZFLOW)
        FIRSTME=0
      ELSE
        CALL BASRED(BASIS,CARD,FLOW,FROMO,GFLOW,LNOD,PRED,STATUS,
    *           THD,UBOBAS)
      ENDIF
C
 50   CONTINUE
C
C  SOLVE THE UPPER BOUND ALGORITHM
C
      USTEP=USTEP+1
      TUITER=TUITER+1
C
      CALL UBALG(ARCNAM,BASIS,CARD,COST,FLOW,FROM,FROMO,GCOEF,
    *         GFLOW,GUBADD,GVAL,LNOD,LOWER,NUMVRG,PI,
    *         PRED,REDGUB,STATUS,THD,TO,UPPER,UTEMP,
    *         ZFLOW,GARB)
      IF (FLGOPT .EQ. 1) THEN
        GO TO 70
      ENDIF
      IF (FLGINF .EQ. 1) THEN
        FLGINF=0
        GO TO 100
      ENDIF
C
      IF (TUITER .GT. MUITER) GO TO 70
      IF (USTEP .LT. UITER) GO TO 50
C
      CALL BASSAV(BASIS,CARD,FLOW,FROMO,GFLOW,LNOD,PRED,STATUS,THD,
    *          UBOBAS)
C
 100  CALL BASRED(BASIS,CARD,FLOW,FROMO,GFLOW,LNOD,PRED,STATUS,THD,
    *          LBOBAS)
      GO TO 60
C

C  OPTIMALITY REACHED
```

```
C
 70   CONTINUE
C
C
C
C  PRINT RESULTS
C
     CALL OUTPUT(ARCNAM,BASIS,CARD,COST,FLOW,FROMO,LNOD,LOWER,
    *           PRED,STATUS,THD,TO,UPPER,YFLOW,ZFLOW,
    *           GFLOW,UTEMP,GCOEF)
C
C
C  PRINT ITERATION COUNTS
C
C     WRITE(6,7000) ITRTOT,ITRREG,ITRBTB,ITRDEG
     WRITE(6,7500) TLITER,FSITER,TUITER
C
C  PRINT TIMING RESULTS
C
C
C
C  ALL DONE WITH THIS PROBLEM
C
     GOTO 10
C
C  ALL DONE WITH PROBLEM SET
C
     CLOSE(11)
     CLOSE(13)
C
  30 CONTINUE
C
C  FORMATS
C
 1000 FORMAT(//1X,'****** PROBLEM NUMBER ',I3,' BEGINS ******')
 2000 FORMAT(//1X,'BAD SECTION HEADER ENCOUNTERED')
 3000 FORMAT(//1X,'EXECUTION NOT ATTEMPTED ON PROBLEM ',I3)
 4000 FORMAT(A8)
 5000 FORMAT (//1X,'****** ITERATION LIMIT EXCEEDED ******'/1X,
    *       'BASIS AND DATA STRUCTURES SAVED')
 6000 FORMAT(//1X,'****** OPTIMAL SOLUTION FOUND ******')
 7000 FORMAT(//1X,'TOTAL NUMBER OF PIVOTS            =',I6/1X,
    *           'NUMBER OF REGULAR PIVOTS          =',I6/1X,
    *           'NUMBER OF BOUND-TO-BOUND PIVOTS   =',I6/1X,
    *           'NUMBER OF DEGENERATE PIVOTS       =',I6)
 7500 FORMAT(//1X,'TOTAL NUMBER OF LOWER BOUND ITERATIONS      =',I6/1X,
    *           'TOTAL NO. OF UB ITERS TO REACH FEASIBILITY =',I6/1X,
    *           'TOTAL NUMBER OF UPPER BOUND ITERATIONS     =',I6)
 8000 FORMAT(//1X,'TIME FOR INPUT    =',F9.2)
 9000 FORMAT(1X,'TIME FOR SOLUTION =',F9.2)
C
     STOP
```

```
      END
C
C
C
C**** ************************************************************
      SUBROUTINE BASSAV (BASIS,CARD,FLOW,FROMO,GFLOW,LNOD,PRED,
     1                   STATUS,THD,UNIT)
C**** ************************************************************
C
C  SUBROUTINE ARGUMENTS
C
      INTEGER BASIS(*),CARD(*),LNOD(*),PRED(*),STATUS(*),THD(*),UNIT,
     *        FROMO(*)
      DOUBLE PRECISION FLOW(*),GFLOW(*)
C
C  LOCAL VARIABLES
C
      INTEGER I
C
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *        ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *              NGUBS,NODES,ROOT,EPSILON,BIG1
C
      REWIND UNIT
C
C  SAVE DATA STRUCTURES
C
      DO 10 I=1,NODES
        WRITE(UNIT) PRED(I),THD(I),CARD(I),LNOD(I),BASIS(I),FLOW(I),
     *              FROMO(I)
   10 CONTINUE
      WRITE(UNIT) ROOT
C
C  SAVE ARC STATUS
C
      DO 20 I=1,ARCS
        WRITE(UNIT) GFLOW(I),STATUS(I)
   20 CONTINUE
      RETURN
      END
C
C
C
C
C
C**** ************************************************************
      SUBROUTINE BASRED (BASIS,CARD,FLOW,FROMO,GFLOW,LNOD,PRED,
     1                   STATUS,THD,UNIT)
C**** ************************************************************
C
```

```
C  SUBROUTINE ARGUMENTS
C
      INTEGER BASIS(*),CARD(*),LNOD(*),PRED(*),STATUS(*),THD(*),UNIT,
     *        FROMO(*)
      DOUBLE PRECISION FLOW(*),GFLOW(*)
C
C  LOCAL VARIABLES
C
      INTEGER I
C
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *        ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *              NGUBS,NODES,ROOT,EPSILON,BIG1
C
C  READ DATA STRUCTURES
C
       REWIND UNIT
      DO 10 I=1,NODES
        READ(UNIT) PRED(I),THD(I),CARD(I),LNOD(I),BASIS(I),FLOW(I),
     *              FROMO(I)
   10 CONTINUE
      READ(UNIT) ROOT
C
C  READ ARC STATUS
C
      DO 20 I=1,ARCS
        READ(UNIT) GFLOW(I),STATUS(I)
   20 CONTINUE
      RETURN
      END
C
C
C
C
C**** ****************************************************************
      SUBROUTINE OUTPUT (ARCNAM,BASIS,CARD,COST,FLOW,FROMO,LNOD,LOWER,
     1                 PRED,STATUS,THD,TO,UPPER,YFLOW,ZFLOW,
     1                 GFLOW,UTEMP,GCOEF)
C**** ****************************************************************
C
C  SUBROUTINE ARGUMENTS
C
      CHARACTER*8 ARCNAM(*)
      INTEGER BASIS(*),CARD(*),FROMO(*),LNOD(*),PRED(*),STATUS(*),
     *        THD(*),TO(*)
      DOUBLE PRECISION  COST(*),FLOW(*),LOWER(*),UPPER(*),YFLOW(*),
     *                 ZFLOW(*),GFLOW(*),UTEMP(*),GCOEF(*)
C
C  LOCAL VARIABLES
C
```

```
      INTEGER BASISI,DUMMY,I,UNIT
C
      INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *        ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *              NGUBS,NODES,ROOT,EPSILON,BIG1
      DOUBLE PRECISION LBND,LBNDPRE,UBND,UBNDPRE
      COMMON /BOUNDS/ LBND,LBNDPRE,UBND,UBNDPRE
      INTEGER NGVLT
      COMMON /NGV/ NGVLT
C
C
C  PRINT LAST SOLUTION FOR THE LOWER BOUND PROCEDURE
C
C
C  READ IN THE  DATA
C
      CALL LBRED(BASIS,CARD,FLOW,GFLOW,LNOD,PRED,STATUS,THD,YFLOW)
C
C
      WRITE(6,9000)
C
C  CHECK FOR FEASIBILITY OF CURRENT SOLUTION
C
C  IN INTERMEDIATE CALLS TO OUTPUT, FEASIBILITY IS CHECKED.
C
C  AT OPTIMALITY, FEASIBILITY HAS ALREADY BEEN DETERMINED
C  AND EITHER FLGINF OR FLGOPT IS 1.
C
C
      IF (FLGINF+FLGOPT .EQ. 0) THEN
        DO 30 I=1,NODES
          IF (IABS(BASIS(I)) .EQ. ARTADD   .AND.  FLOW(I) .NE. 0) THEN
            FLGINF=1
            GOTO 25
          ENDIF
   30   CONTINUE
   25   CONTINUE
      ENDIF
      IF (FLGINF .EQ. 1) THEN
        WRITE (6,1000)
        FLGINF=0
      ENDIF
C
C  WRITE OUT NONZERO ARC INFORMATION
C
      WRITE (6,2000)
      DO 40 I=1,NODES
        BASISI=BASIS(I)
        FLOW(I)=FLOW(I)+LOWER(IABS(BASISI))
        IF (FLOW(I) .NE. 0) THEN
          IF (BASISI .LT. 0) THEN
            K=IABS(BASISI)
            WRITE (6,3000) K,I,PRED(I),FLOW(I),COST(IABS(BASISI))
          ELSE
            WRITE (6,3000) BASISI,PRED(I),I,FLOW(I),COST(BASISI)
          ENDIF
        ENDIF
   40 CONTINUE
C
C  LOOK FOR NONBASIC ARCS AT UPPER BOUNDS
C
      WRITE(6,4000)
      DO 50 I=1,ARCS
        IF (STATUS(I) .EQ. 2) THEN
          UPPER(I)=UPPER(I)+LOWER(I)
          WRITE(6,3000) I,FROMO(I),TO(I),UPPER(I),COST(I)
        ENDIF
   50 CONTINUE
C
C  LOOK FOR NONBASIC ARCS AT LOWER BOUNDS
C
      WRITE(6,5000)
      DO 60 I=1,ARCS
        IF (STATUS(I) .EQ. 1  .AND.  LOWER(I) .NE. 0) THEN
          WRITE(6,3000) I,FROMO(I),TO(I),LOWER(I),COST(I)
        ENDIF
   60 CONTINUE
C
C  LOOK FOR FIXED ARCS
C
      WRITE (6,6000)
      DO 70 I=1,ARCS
        IF (STATUS(I) .EQ. 3) THEN
          WRITE (6,3000) I,FROMO(I),TO(I),LOWER(I),COST(I)
        ENDIF
   70 CONTINUE
C
C  PRINT OBJECTIVE FUNCTION VALUE
C
      WRITE (6,7000) LBNDPRE
C
      WRITE(6,*) 'NUMBER OF VIOLATED GUBS',NGVLT
C
C  PRINT SOLUTION FOR UPPER BOUND PROCEDURE
C
C
C
C  READ IN THE  DATA
```

```
C
     CALL UBRED(BASIS,CARD,FLOW,GFLOW,LNOD,PRED,STATUS,THD,ZFLOW)
C
C
     WRITE(6,9500)
C
C  CHECK FOR FEASIBILITY OF CURRENT SOLUTION
C
C  IN INTERMEDIATE CALLS TO OUTPUT, FEASIBILITY IS CHECKED.
C
C  AT OPTIMALITY, FEASIBILITY HAS ALREADY BEEN DETERMINED IN SOLVE
C  AND EITHER FLGINF OR FLGOPT IS 1.
C
C
     IF (FLGINF+FLGOPT .EQ. 0) THEN
       DO 120 I=1,NODES
         IF (IABS(BASIS(I)) .EQ. ARTADD   .AND.  FLOW(I) .NE. 0.00) THEN
           FLGINF=1
           GOTO 125
         ENDIF
  120    CONTINUE
  125    CONTINUE
     ENDIF
     IF (FLGINF .EQ. 1) THEN
       WRITE (6,1000)
       FLGINF=0
     ENDIF
C
C  WRITE OUT NONZERO ARC INFORMATION
C
     WRITE (6,2000)
     DO 130 I=1,NODES
       BASISI=BASIS(I)
       FLOW(I)=FLOW(I)+LOWER(IABS(BASISI))
       IF (GCOEF(ABS(BASISI)) .LT. 0.00) THEN
         FLOW(I)=FLOW(I)+ZFLOW(ABS(BASISI))
       ENDIF
       IF (FLOW(I) .NE. 0.00) THEN
         IF (BASISI .LT. 0) THEN
           K=IABS(BASISI)
           WRITE (6,3000) K,I,PRED(I),FLOW(I),COST(IABS(BASISI))
         ELSE
           WRITE (6,3000) BASISI,PRED(I),I,FLOW(I),COST(BASISI)
         ENDIF
       ENDIF
  130 CONTINUE
C
C  LOOK FOR NONBASIC ARCS AT UPPER BOUNDS
C
     WRITE(6,4000)
     DO 140 I=1,ARCS
       IF (STATUS(I) .EQ. 2) THEN
```

```
           IF (GCOEF(I) .LT. 0.00) THEN
             UPPER(I)=UPPER(I)+LOWER(I)
           ELSE
             UPPER(I)=ZFLOW(I)+LOWER(I)
           ENDIF
           WRITE(6,3000) I,FROMO(I),TO(I),UPPER(I),COST(I)
         ENDIF
  140 CONTINUE
C
C  LOOK FOR NONBASIC ARCS AT LOWER BOUNDS
C
      WRITE(6,5000)
      DO 150 I=1,ARCS
        IF (GCOEF(I) .LT. 0.00) LOWER(I)=LOWER(I)+ZFLOW(I)
        IF (STATUS(I) .EQ. 1  .AND.  LOWER(I) .NE. 0.00) THEN
          WRITE(6,3000) I,FROMO(I),TO(I),LOWER(I),COST(I)
        ENDIF
  150 CONTINUE
C
C  LOOK FOR FIXED ARCS
C
      WRITE (6,6000)
      DO 160 I=1,ARCS
        IF (STATUS(I) .EQ. 3) THEN
          WRITE (6,3000) I,FROMO(I),TO(I),LOWER(I),COST(I)
        ENDIF
  160 CONTINUE
C
C  PRINT OBJECTIVE FUNCTION VALUE
C
      WRITE (6,7000) UBNDPRE
C
C  FORMATS
C
 1000 FORMAT(//1X,'****************************'/1X,
     *         '*   INFEASIBLE SOLUTION    *'/1X,
     *         '****************************')
 2000 FORMAT(//1X,'NONZERO FLOWS'/1X,'BASIC ARCS'/1X,
     *     '    INDEX     FROM       TO       FLOW',
     *     '     COST')
 3000 FORMAT(1X,3I8,2F15.5)
 4000 FORMAT(/1X,'NONBASIC ARCS AT UPPER BOUND'/1X,
     *     '    INDEX     FROM       TO       FLOW',
     *     '     COST')
 5000 FORMAT(/1X,'NONBASIC ARCS AT LOWER BOUND'/1X,
     *     '    INDEX     FROM       TO       FLOW',
     *     '     COST')
 6000 FORMAT(/1X,'FIXED ARCS'/1X,
     *     '    INDEX     FROM       TO       FLOW',
     *     '     COST')
 7000 FORMAT(//1X,'OBJECTIVE FUNCTION VALUE =',F20.6)
 9000 FORMAT(//1X,'****************************'/1X,
   *          '* LOWER BOUND PROCEDURE     *'/1X,
   *          '*****************************')
 9500 FORMAT(//1X,'*****************************'/1X,
   *          '*  UPPER BOUND PROCEDURE   *'/1X,
   *          '*****************************')
 999  CONTINUE
      RETURN
      END
C
C
C**** **************************************************************
      SUBROUTINE INPUT (ARCNAM,COST,CTEMP,FLOW,FROM,GCOEF,GUBADD,
   *                GVAL,IND,LEFT,LOWER,NODNAM,NUMOUT,NUMVRG,
   *                NVAL,ORDER,REDGUB,RIGHT,STATUS,TO,UPPER)
C**** **************************************************************
C
C  SUBROUTINE ARGUMENTS
C
      CHARACTER*8 ARCNAM(*),NODNAM(*)
C
      INTEGER REDGUB(*),FROM(*),GUBADD(*),IND(*),LEFT(*),
   *        NUMOUT(*),ORDER(*),RIGHT(*),STATUS(*),
   *        TO(*),NUMVRG(*),NVAL(*)
C
       DOUBLE PRECISION COST(*),CTEMP(*),GCOEF(*),GVAL(*),FLOW(*),
   *                LOWER(*),UPPER(*)
C
C  LOCAL VARIABLES
C
      CHARACTER*8 ARCSEC,BLANK,DUMMY,ENDATA,ENDPRB,FREE,FRMNOD,
   *           GUBSEC,LABEL,NAM,NODE,
   *           NODSEC,SLACK,TONODE
C
      INTEGER ARCREC,FRMN,I,INDEX,
   *        NEXT,NEXTAV,NODSM1,
   *        NODSP1,REC,TON,VAL
      DOUBLE PRECISION CST,COEF,LOW,SGVAL,TFLOW,TGVAL,UP
C
      INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
C
      INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
C
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
   *        ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
   *            NGUBS,NODES,ROOT,EPSILON,BIG1
C
      DATA ARCSEC,ENDATA,NODSEC /'ARCSEC ','ENDATA  ','NODSEC  '/
      DATA GUBSEC /'GUBSEC  '/
      DATA BLANK,ENDPRB,FREE/'        ','ENDPRB  ','FREE    '/
      DATA SLACK/'SLACK   '/
C
C  INITIALIZE NODE ARRAYS
C
      DO  5 I=1,MAXNOD
        NODNAM(I)=BLANK
        LEFT(I)=0
        RIGHT(I)=0
        FLOW(I)=0.0D0
        NVAL(I)=0
        IND(I)=0
        NUMOUT(I)=0
    5 CONTINUE
C
C  INITIALIZE ARC ARRAYS
C
      DO 10 I=1,ARCS
        CTEMP(I)=0.0
 10   CONTINUE
C
C  INITIALIZE GUB ARRAYS
C
      DO 15 I=1,MAXGUB
        GVAL(I)=0.0D0
        NUMVRG(I)=0
        REDGUB(I)=0
   15 CONTINUE
C
C  INITIALIZE CONSTANTS
C
      NODES=0
      ARCS=0
      GARCS=0
      NGUBS=0
      NEXTAV=2
      NUMREC=0
      TCOST=0.0D0
      FLGERR=0
      FLGINF=0
      FLGITR=0
      FLGOPT=0
C
C  BEGIN INPUT
C
      READ (8,1000) LABEL
      NUMREC=NUMREC+1
      IF (LABEL .NE. NODSEC  .AND. LABEL .NE. ENDPRB) THEN
        FLGERR=-1
        RETURN
      ELSE
        IF (LABEL .EQ. ENDPRB) THEN
```

```
          FLGEND=1
          RETURN
        ENDIF
      ENDIF
C
C  READ IN RHS VALUES FOR THE NETWORK
C
   20 CONTINUE
        READ(8,2000) NODE,VAL
        NUMREC=NUMREC+1
        IF ( NODE .EQ. GUBSEC) GOTO 30
        IF (VAL .EQ. 0) THEN
          FLGERR=-1
          RETURN
        ENDIF
        I=NODNUM(LEFT,NODE,NEXTAV,NODNAM,RIGHT)
        IF ( FLGERR .EQ. 1) THEN
          RETURN
        ENDIF
        FLOW(I)=VAL
        NVAL(I)=VAL
        IND(I)=-1
        GOTO 20
   30 CONTINUE
C
C  READ IN NO. OF VARIABLES IN EACH GUB AND RHS VALUES FOR THE GUBS
C
      READ(8,1500) NGUBS
      NUMREC=NUMREC+1
      IF (NGUBS .EQ. 0) THEN
        FLGERR=-1
        RETURN
      ENDIF
C
      DO 35 I=1,NGUBS
        READ(8,1500) NUMVRG(I),GVAL(I)
        NUMREC=NUMREC+1
        IF (NUMVRG(I) .LE. 0 ) THEN
          FLGERR=-1
          RETURN
        ENDIF
        GARCS=GARCS+NUMVRG(I)
   35 CONTINUE
      READ(8,1000) LABEL
      NUMREC=NUMREC+1
      IF (LABEL .NE. ARCSEC) THEN
        FLGERR=-1
        RETURN
      ENDIF
C
C  RECORD LOCATION OF ARC DATA
C
```

```
      ARCREC=NUMREC
C
C  MAKE FIRST PASS THROUGH ARC DATA
C
C
C
C  READ IN DATA FOR NEXT ARC
C
   40 CONTINUE
        READ (8,1000) NAM,FRMNOD,TONODE,CST,UP,LOW,COEF
        NUMREC=NUMREC+1
        IF (NAM .EQ. ENDATA) GOTO 50
C
C  DETERMINE NODE NUMBERS FOR THE FROM NODE AND TO NODE OF THE
C  CURRENT ARC
C
        FRMN=NODNUM(LEFT,FRMNOD,NEXTAV,NODNAM,RIGHT)
        IF (FLGERR .EQ. 1) THEN
          RETURN
        ENDIF
        TON=NODNUM(LEFT,TONODE,NEXTAV,NODNAM,RIGHT)
        IF (FLGERR .EQ. 1) THEN
          RETURN
        ENDIF
C
C  CHECK NODE INDICATOR
C
        IF (FLOW(FRMN) .LE. 0.0) THEN
          IND(FRMN)=1
        ENDIF
        IF (FLOW(TON) .GE. 0.0) THEN
          IND(TON)=1
        ENDIF
C
C  CHECK FOR VALID BOUNDS
C
        IF (UP .LT. LOW  .AND.  LOW .NE. 0.0) THEN
          FLGERR=2
          WRITE (6,*) 'ERROR IN BOUNDS'
          WRITE (6,*) NAM,FRMNOD,TONODE,CST,UP,LOW,COEF
          GOTO 40
        ENDIF
        ARCS=ARCS+1
        NUMOUT(FRMN)=NUMOUT(FRMN)+1
C
C  UNRESTRICTED ARC?  IF SO, SET UP COMPLEMENTARY ARC.
C
        IF (UP .LT. 0.0  .AND.  LOW .EQ. 0.0) THEN
          ARCS=ARCS+1
          NUMOUT(TON)=NUMOUT(TON)+1
        ENDIF
      GOTO 40
```

```
C
C  ALL DONE WITH FIRST PASS THROUGH ARC DATA
C
   50 CONTINUE
C
C  CHECK FOR PROBLEM FEASIBILITY
C
      DO 60 I=1,NODES
         IF (IND(I) .EQ. -1) THEN
            FLGERR=2
            IF (FLOW(I) .LT. 0.0) THEN
               WRITE (6,7000) I,FLOW(I)
            ELSE
               WRITE (6,8000) I,FLOW(I)
            ENDIF
         ENDIF
   60 CONTINUE
      IF (FLGERR .EQ. 2) THEN
         RETURN
      ENDIF
C
C  DETERMINE WHETHER DUMMY NODE IS NEEDED
C
      TFLOW=0.0D0
      DO 70 I=1,NODES
         TFLOW=TFLOW+FLOW(I)
   70 CONTINUE
      IF (TFLOW .GT. 0.0) THEN
         WRITE(6,9000) TFLOW
         FLGERR=2
         RETURN
      ENDIF
      IF (TFLOW .LT. 0.0) THEN
         NODES=NODES+1
         IF (NODES .GT. MAXNOD) THEN
            FLGERR=2
            RETURN
         ENDIF
         NODSM1=NODES-1
         DO 80 I=1,NODSM1
            IF (FLOW(I) .LT. 0.0) THEN
               ARCS=ARCS+1
               NUMOUT(I)=NUMOUT(I)+1
            ENDIF
   80    CONTINUE
         FLOW(NODES)=-TFLOW
         NODNAM(NODES)=DUMMY
      ENDIF
C
C  SET UP FOR SECOND PASS THROUGH ARC DATA
C
C  FIRST MOVE POINTER TO BEGINING OF ARC DATA IN FILE 8
```

```
C
      REWIND 8
      DO 801 I=1,ARCREC
        READ(8,2000)
  801 CONTINUE
C
C  SET FROM(.) ARRAY FOR ALL NODSEC
      FROM(1)=1
      NODSP1=NODES+1
      DO 802 I=2,NODSP1
        IM1=I-1
        FROM(I)=FROM(IM1)+NUMOUT(IM1)
  802 CONTINUE
C      WRITE(6,*) ' ARCREC = ',ARCREC
C      WRITE(6,*) ' FROM(I)'
C      WRITE(6,*) (FROM(I), I=1,NODSP1)
C
C  RESET NUMOUT(.) ARRAY TO USE AS POINTER TO FIRST ARC LOCATION
C  FOR EACH NODE I
C
      DO 803 I=1,NODES
        NUMOUT(I)=FROM(I)
  803 CONTINUE
C
C  BEGIN SECOND PASS THROUGH ARC DATA
C
      SGVAL=0.0D0
      TGVAL=0.0D0
      INDEX=1
      NEXT=NUMVRG(INDEX)
      REC=0
   90  CONTINUE
        READ (8,1000) NAM,FRMNOD,TONODE,CST,UP,LOW,COEF
        REC=REC+1
        IF (NAM .EQ. ENDATA) GOTO 100
C
C  RETREIVE NODE NUMBER FOR FROM AND TO NODES
C
        FRMN=NODRET(ERROR,LEFT,FRMNOD,NEXTAV,NODNAM,RIGHT)
        TON=NODRET(ERROR,LEFT,TONODE,NEXTAV,NODNAM,RIGHT)
C
C  UNRESTRICTED ARC? IF SO, STORE COMPLEMENTARY ARC.
C
        IF (UP .LT. 0.0 .AND. LOW .EQ. 0.0) THEN
          LOC=NUMOUT(TON)
          ARCNAM(LOC)=FREE
          TO(LOC)=FRMN
          COST(LOC)=-CST
          LOWER(LOC)=LOW
          UPPER(LOC)=BIG1
          STATUS(LOC)=1
          GCOEF(LOC)=COEF
          UP=BIG1
          NUMOUT(TON)=LOC+1
        ENDIF
C
C  DEFAULT UPPER BOUND?
C
        IF (UP .EQ. 0.0 .AND. LOW .EQ. 0.0) THEN
          UP=BIG1
        ENDIF
C
C  STORE DATA FOR CURRENT ARC
C
        LOC=NUMOUT(FRMN)
        ARCNAM(LOC)=NAM
        TO(LOC)=TON
        COST(LOC)=CST
        LOWER(LOC)=LOW
        GCOEF(LOC)=COEF
C
C  ADJUST FOR NONZERO LOWER BOUNDS
C
        IF (LOW .NE. 0.0) THEN
          TCOST=TCOST+LOW*CST
          FLOW(FRMN)=FLOW(FRMN)+LOW
          FLOW(TON)=FLOW(TON)-LOW
          IF (INDEX .LE. NGUBS) THEN
            GVAL(INDEX)=GVAL(INDEX)-LOW*COEF
          ENDIF
        ENDIF
        UPPER(LOC)=UP-LOW
        STATUS(LOC)=1
C
C  CHECK FOR A FIXED ARC
C
        IF (UP .EQ. LOW) THEN
          STATUS(LOC)=3
        ENDIF
        NUMOUT(FRMN)=LOC+1
C
C  STORE GUB ADDRESS
C
        GUBADD(REC)=LOC
C
C  CHECK FOR GUB CONSTRAINTS REDUNDANCY & FEASIBILITY
C
        IF (INDEX .GT. NGUBS) GO TO 90
        IF (COEF .GT. 0.0) THEN
          TGVAL=TGVAL+UPPER(LOC)*COEF
        ELSE
          SGVAL=SGVAL-UPPER(LOC)*COEF
        ENDIF
        IF (REC .EQ. NEXT) THEN
```

```
          IF (TGVAL .LE. GVAL(INDEX)) THEN
            REDGUB(INDEX)=1
            WRITE(6,*) INDEX,'TH GUB REDUNDANT'
          ENDIF
          IF (GVAL(INDEX) .LT. -SGVAL) THEN
            FLGINF=1
            WRITE(6,3000) INDEX,GVAL(INDEX)
            RETURN
          ENDIF
          IF (GVAL(INDEX) .EQ. 0.0) THEN
            WRITE(6,4000) INDEX
          ENDIF
          CTEMP(GUBADD(NEXT))=SGVAL
          INDEX=INDEX+1
          NEXT=NEXT+NUMVRG(INDEX)
          SGVAL=0.0D0
          TGVAL=0.0D0
        ENDIF
      GOTO 90
C
  100 CONTINUE
C
C
C  ALL DONE WITH INPUT
C
  200 CONTINUE
C
C  SET ADDRESS, COST AND BOUNDS FOR ARTIFICIAL ARCS
C
      ARTADD=ARCS+1
      IF (ARTADD .GT. MAXARC) THEN
        FLGERR=2
        WRITE (6,6000) NUMREC
        RETURN
      ENDIF
      COST(ARTADD)=BIG1
      LOWER(ARTADD)=0.0D0
      UPPER(ARTADD)=BIG1
      GCOEF(ARTADD)=0.0D0
C
C      CALL SORT(FROM,ORDER,ARCS)
      REWIND 12
      DO 220 I=1,NODES
        LOC1=FROM(I)
        LOC2=FROM(I+1)-1
        DO 210 J=LOC1,LOC2
          TON=TO(J)
          WRITE(12) ARCNAM(J),I,TON,COST(J),LOWER(J),UPPER(J),
     *           STATUS(J)
C          WRITE(6,1030) ARCNAM(J),NODNAM(I+1),NODNAM(TON+1),I,
C    *                 TON,COST(J),LOWER(J),UPPER(J),STATUS(J)
C1030      FORMAT(1X,3(A8,2X),4I5,2I10)
```

```
 210    CONTINUE
 220 CONTINUE
     REWIND 12
C
C  FORMATS
C
 1000 FORMAT(3(A8,2X),4F10.2)
 1500 FORMAT(I8,2X,F10.2)
 2000 FORMAT(A8,2X,I10)
 3000 FORMAT(1X,'GUB NO.',I1O,'IS INFEASIBLE WITH RHS VALUE OF',I1O)
 4000 FORMAT(1X,'GUB NO.',I1O,'HAS ZERO RHS VALUE')
 6000 FORMAT(1X,'****** ARC STORAGE HAS BEEN EXCEEDED AT INPUT RECORD',
    *       ' NO. ',I10)
 7000 FORMAT(1X,'NODE NO. ',I5,' HAS SUPPLY OF ',F10.2,
    *       ' BUT NO ARCS OUT OF THE NODE')
 8000 FORMAT(1X,'NODE NO. ',I5,' HAS DEMAND OF ',F10.2,
    *       ' BUT NO ARCS INTO THE NODE')
 9000 FORMAT(1X,'****** PROBLEM INFEASIBLE ******'/1X,
    *       ' DEMAND EXCEEDS SUPPLY BY ',F10.2)
     RETURN
     END
C
C
C
C
C**** ****************************************************************
     INTEGER FUNCTION NODNUM (LEFT,NAM,NEXTAV,NODNAM,RIGHT)
C**** ****************************************************************
C
C  THIS FUNCTION ASSIGNS A NODE NUMBER TO EACH NODE NAME.
C
C
C  FUNCTION ARGUMENTS
C
     CHARACTER*8 NODNAM(*),NAM
     INTEGER LEFT(*),RIGHT(*)
     INTEGER NEXTAV
C
C  LOCAL VARIABLES
C
     INTEGER LOC,TLOC
C
     DOUBLE PRECISION TCOST,EPSILON,BIG1
     INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
    *       ROOT
     COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
    *             NGUBS,NODES,ROOT,EPSILON,BIG1
C
C  START SEARCH FOR EXISTING NODE NAME AT TOP OF LIST
C
C  IF NODE NAME ALREADY EXISTS GET NODE NUMBER
C  OTHERWISE STORE NEW NODE NAME AND ASSIGN A NODE NUMBER
```

```
C
C  THE NODE NAMES ARE STORED IN A BINARY TREE
C  AT EACH NODE IN THE TREE GO LEFT FOR <, RIGHT FOR >
C
      LOC=1
    5 IF (NAM.EQ.NODNAM(LOC)) THEN
C
C  MATCH FOUND
C
        NODNUM=LOC-1
        RETURN
      ENDIF
C
C  KEEP SEARCHING
C
      IF (NAM.LT.NODNAM(LOC)) THEN
C
C  TOOK LEFT BRANCH AT NODE 'LOC'
C
        TLOC=LEFT(LOC)
        IF (TLOC.EQ.0) THEN
C
C  NO SUCCESSOR NODES, ADD NODE NAME TO LEFT OF CURRENT NODE IN TREE
C

          LEFT(LOC)=NEXTAV
          NODNAM(NEXTAV)=NAM
          NODNUM=NEXTAV-1
          NEXTAV=NEXTAV+1
          NODES=NODES+1
          IF (NODES .GT. MAXNOD) THEN
            FLGERR=1
            WRITE(6,1000) NUMREC
          ENDIF
          RETURN
        ELSE
C
C  CONTINUE SEARCH
C
          LOC=TLOC
          GOTO 5
        ENDIF
      ELSE
C
C  TOOK RIGHT BRANCH AT NODE 'LOC'
C
        TLOC=RIGHT(LOC)
        IF (TLOC.EQ.0) THEN
C
C  NO SUCCESSOR NODES, ADD NODE NAME TO RIGHT OF CURRENT NODE IN TREE
C
          RIGHT(LOC)=NEXTAV
          NODNAM(NEXTAV)=NAM
          NODNUM=NEXTAV-1
          NEXTAV=NEXTAV+1
          NODES=NODES+1
          IF (NODES .GT. MAXNOD) THEN
            FLGERR=1
            WRITE(6,1000) NUMREC
          ENDIF
          RETURN
        ELSE
          LOC=TLOC
          GOTO 5
        ENDIF
      ENDIF
C
C  FORMATS
C
 1000 FORMAT(1X,'****** NODE STORAGE HAS BEEN EXCEEDED AT INPUT RECORD',
     *        ' NO. ',I10)
      END
C
C
C
C
C**** *************************************************************
      INTEGER FUNCTION NODRET (ERROR,LEFT,NAM,NEXTAV,NODNAM,RIGHT)
C**** *************************************************************
C
C  THIS FUNCTION RETRIEVES A NODE NUMBER
C
C
C  FUNCTION ARGUMENTS
C
      CHARACTER*8 NODNAM(*),NAM
      INTEGER LEFT(*),RIGHT(*)
      INTEGER ERROR,NEXTAV
C
C  LOCAL VARIABLES
C
      INTEGER LOC,TLOC
C
      INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *        ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *              NGUBS,NODES,ROOT,EPSILON,BIG1
C
C  START SEARCH FOR EXISTING NODE NAME AT TOP OF LIST
C
C  IF NODE NAME ALREADY EXISTS GET NODE NUMBER
```

```
C  OTHERWISE SET ERROR FLAG
C
C  THE NODE NAMES ARE STORED IN A BINARY TREE
C  AT EACH NODE IN THE TREE GO LEFT FOR <, RIGHT FOR >
C
      ERROR=0
      LOC=1
    5 IF (NAM.EQ.NODNAM(LOC)) THEN
C
C  MATCH FOUND
C
        NODRET=LOC-1
        RETURN
      ENDIF
C
C  KEEP SEARCHING
C
      IF (NAM.LT.NODNAM(LOC)) THEN
C
C  TOOK LEFT BRANCH AT NODE 'LOC'
C
        TLOC=LEFT(LOC)
        IF (TLOC.EQ.0) THEN
C
C  NO SUCCESSOR NODES, ILLEGAL NODE NAME PASSED
C

          ERROR=1
          RETURN
        ELSE
C
C  CONTINUE SEARCH
C
          LOC=TLOC
          GOTO 5
        ENDIF
      ELSE
C
C  TOOK RIGHT BRANCH AT NODE 'LOC'
C
        TLOC=RIGHT(LOC)
        IF (TLOC.EQ.0) THEN
C
C  NO SUCCESSOR NODES, ILLEGAL NODE NAME PASSED
C

          ERROR=1
          RETURN
        ELSE
          LOC=TLOC
          GOTO 5
        ENDIF
      ENDIF
C
C  FORMATS
C
      END
C
C
C
C
C**** **********************************************************
      SUBROUTINE SORT(IN,OUT,N)
C**** **********************************************************
C
C  SUBROUTINE ARGUMENTS
C
      INTEGER IN(*),OUT(*)
      INTEGER N
C
C  LOCAL VARIABLES
C
      LOGICAL SORTED
      INTEGER I,IEND,I1,I2,J,NM1
      IEND=N
      DO 5 I=1,IEND
        OUT(I)=I
    5 CONTINUE
      NM1=N-1
      DO 15 J=1,NM1
        IEND=IEND-1
        SORTED=.TRUE.
        DO 10 I=1,IEND
          I1=OUT(I)
          I2=OUT(I+1)
          IF (IN(I1).LE.IN(I2)) GOTO 10
          SORTED=.FALSE.
          OUT(I)=I2
          OUT(I+1)=I1
   10   CONTINUE
        IF (SORTED) RETURN
   15 CONTINUE
      RETURN
      END
C**** **********************************************************
      SUBROUTINE INIT(ARCNAME,BASIS,CARD,COST,CTEMP,FLOW,FROM,
     *          FROMO,GARB,GFLOW,GUBADD,GVAL,GCOEF,
     *          LGMULT,LNOD,LOWER,NUMVRG,NVAL,PI,PRED,
     *          REDGUB,STATUS,THD,TO,UPPER,YFLOW)
C**** **********************************************************
C
C  THE PURPOSE OF THIS ROUTINE IS TO INITIALIZE THE
C  RELAXATION/DECOMPOSITION ALGORITHM
C
```

```
C
C
C  SUBROUTINE ARGUMENTS
C
     CHARACTER*8 ARCNAME
     INTEGER BASIS(*),CARD(*),FROM(*),FROMO(*),GUBADD(*),
    *        LNOD(*),NUMVRG(*),NVAL(*),PRED(*),REDGUB(*),
    *        STATUS(*),THD(*),TO(*)
     DOUBLE PRECISION COST(*),CTEMP(*),FLOW(*),GCOEF(*),
    *                 GFLOW(*),GVAL(*),LGMULT(*),LOWER(*),
    *                 PI(*),UPPER(*),YFLOW(*),
    *                 GARB(*)
C
C  LOCAL VARIABLES
C
     INTEGER I,INDEX,LOC1,LOC2,NEXT,NUM,REC,count
     DOUBLE PRECISION ADJVAL,LBSUBG,NORM,SUM,MAXVLN
C
     INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
     COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
     INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
     COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
     DOUBLE PRECISION TCOST,EPSILON,BIG1
     INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
    *        ROOT
     COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
    *              NGUBS,NODES,ROOT,EPSILON,BIG1
     DOUBLE PRECISION LBSTEP,ILBSTEP,MAXSTEP
     COMMON /STEP/ LBSTEP,ILBSTEP,MAXSTEP
     DOUBLE PRECISION LBND,LBNDPRE,UBND,UBNDPRE
     COMMON /BOUNDS/ LBND,LBNDPRE,UBND,UBNDPRE
C
C  CONSTRUCT STARTING BASIS
C
     DO 10 I=1,ARCS
       GFLOW(I)=0.0D0
       YFLOW(I)=0.0D0
       LGMULT(I)=0.0D0
 10  CONTINUE
C
     FLGOPT=1
C
     CALL START(BASIS,CARD,COST,FLOW,FROM,GFLOW,LNOD,PRED,STATUS,
    *           THD,TO,UPPER,NVAL)
C
C
C SOLVE THE PURE NETWORK PROBLEM
C
     CALL PURNET(ARCNAM,BASIS,CARD,COST,FLOW,FROM,FROMO,GFLOW,LNOD,
    *            LOWER,PI,PRED,STATUS,THD,TO,UPPER)
     IF (FLGINF .EQ. 1) RETURN
C
```

```
      LBND=TCOST
C
C  SOLVE FOR THE INITIAL SCBVLP PROBLEM AND THE DA
C
      count=0
      FLGOPT=1
      MAXVLN=0.0D0
      NORM=0.0D0
      NEXT=0
      SUM=0.0D0
      LOC2=0
      DO 50 INDEX=1,NGUBS
        NEXT=NEXT+NUMVRG(INDEX)
        LOC1=LOC2+1
        LOC2=LOC2+NUMVRG(INDEX)
C
C  IF THE GUB CONSTRAINT IS REDUNDANT THEN LET THE FLOWS BE THE SAME
C  AS THE NETWORK FLOWS
C
         DO 20 I=LOC1,LOC2
           REC=GUBADD(I)
           YFLOW(REC)=GFLOW(REC)
C           LGMULT(I)=0.0D0
C           CTEMP(REC)=0.0D0
           SUM=SUM+GFLOW(REC)*GCOEF(REC)
 20      CONTINUE
         ADJVAL=GVAL(INDEX)+CTEMP(GUBADD(NEXT))
         IF (ABS(ADJVAL) .LE. 1E-15) ADJVAL=0.0D0
C
C  THE FLOWS ARE THE MINIMUM OF THE NETWORK FLOWS AND THE ADJUSTED
C  RIGHT HAND SIDE OF THE GUB
C
C  SET THE SUBGRADIENT FOR THE LOWER BOUND ALG
C
         IF (REDGUB(INDEX) .EQ. 1 .OR. SUM .LE. GVAL(INDEX)) GO TO 45
C
         count=count+1
         SUM=SUM-GVAL(INDEX)
         MAXVLN=MAX(MAXVLN,SUM)
         DO 30 I=LOC1,LOC2
           REC=GUBADD(I)
           IF (GCOEF(REC) .GT. 0.0) THEN
             YFLOW(REC)=0.0D0
           ELSE
             YFLOW(REC)=UPPER(REC)
           ENDIF
           LBSUBG=YFLOW(REC)-GFLOW(REC)
           NORM=NORM+LBSUBG*LBSUBG
C           LGMULT(I)=LBSTEP*LBSUBG
           IF (ABS(LBSUBG) .LE. 1E-15) LBSUBG=0.0D0
           IF (FLGOPT .EQ. 1) THEN
             IF (LBSUBG .NE. 0.00) FLGOPT=0
```

```
          ENDIF
C          IF (ABS(LGMULT(I)) .LE. 1E-15) LGMULT(I)=0.0D0
C          CTEMP(REC)=-LGMULT(I)
 30      CONTINUE
C
 45      SUM=0.0D0
 50   CONTINUE
      LBNDPRE=LBND
      CALL LBSAV(BASIS,CARD,FLOW,GFLOW,LNOD,PRED,STATUS,THD,
     *         YFLOW)
C
C  DETERMINE THE STEP SIZE
C
      LBSTEP=MAXVLN/NORM
C      LBSTEP=MAXVLN/SQRT(NORM)
      MAXSTEP=MAXVLN
C
C  DETERMINE THE LAGRANGEAN MULTIPLIERS
C
      DO 85 I=1,GARCS
        REC=GUBADD(I)
        LGMULT(I)=LBSTEP*(YFLOW(REC)-GFLOW(REC))
        CTEMP(REC)=-LGMULT(I)
 85   CONTINUE
C
C  CHECK FOR OPTIMALITY
C
      IF (FLGOPT .EQ. 1) THEN
        WRITE(6,*) 'OPTIMAL SOLUTION TO THE PURE NETWORK PROBLEM IS
     *            OPTIMAL FOR THE ORIGINAL PROBLEM'
        RETURN
      ENDIF
C
 57   CONTINUE

      DO 60 I=1,NEXT
        REC=GUBADD(I)
        IF ( ABS(CTEMP(REC)) .GT. 1E-15) GO TO 70
 60   CONTINUE
C
C  WE HAVE AN OPTIMAL SOLUTION TO THE ORIGINAL PROBLEM
C
      FLGOPT=1
      RETURN
C
 70   CONTINUE
C
C     FLGOPT=0
C
C
      UBND=BIG1
      UBNDPRE=BIG1
      LBNDPRE=LBND
C
C
      WRITE(6,*) 'TCOST =  ',TCOST
C
      RETURN
      END
C**** ***************************************************************
      SUBROUTINE START(BASIS,CARD,COST,FLOW,FROM,GFLOW,LNOD,PRED,
     1                 STATUS,THD,TO,UPPER,US)
C *** ***************************************************************
C
C     This routine finds a starting basis for the NETGUB
C
c
C  SUBROUTINE ARGUMENTS
C
      INTEGER BASIS(*),CARD(*),FROM(*),LNOD(*),
     1        PRED(*),STATUS(*),THD(*),TO(*),
     1        US(*)
      DOUBLE PRECISION COST(*),FLOW(*),GFLOW(*),UPPER(*)
C
C
C  LOCAL VARIABLES
C
      INTEGER DMNODE,FMNODE,I,L,LOC1,LOC1,N,
     *        TONODE
C
      INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *        ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *              NGUBS,NODES,ROOT,EPSILON,BIG1
c
c  Set the initial undistributed supply.
c  Set cardinality of each node to node one and last node of each
c  node to itself.
c
      DO 10 I = 1,NODES
        IF ( US(I) .NE. 0) THEN
          US(I)=-US(I)
        ELSE
          US(I)=0
        ENDIF
        CARD(I)=1
        LNOD(I)=I
C
C  Initialize node arrays
```

```
C
        PRED(I)=0
        THD(I)=0
        FLOW(I)=0.0D0
c
 10   CONTINUE
C
C  Connect the root node to the demands via artificials with positive
C  flows.
C
C
C  Set ROOT
C
      ROOT=0
      DO 60 I=1,NODES
        IF (US(I) .GT. 0) THEN
          ROOT=I
          GOTO 70
        ENDIF
 60   CONTINUE
      IF (ROOT .EQ. 0) THEN
        FLGERR=-1
        WRITE(6,*) 'NO UNDISTRUBUTED SUPPLY EXISTS'
        RETURN
      ENDIF
 70   FLOW(ROOT)=0.0D0
      CARD(ROOT)=1
      THD(ROOT)=ROOT
      PRED(ROOT)=0
      BASIS(ROOT)=0
      LNOD(ROOT)=ROOT
C
C  First satisfy the demands via artificials
C
      DMNODE=0
      DO 80 I=1,NODES
        IF (US(I) .LT. 0) THEN
          DMNODE=I
          THD(ROOT)=DMNODE
          CARD(ROOT)=CARD(ROOT)+1
          PRED(DMNODE)=ROOT
          BASIS(DMNODE)=ARTADD
          FLOW(DMNODE)=-US(DMNODE)
          TCOST=TCOST+FLOW(DMNODE)*COST(ARTADD)
          LNOD(DMNODE)=DMNODE
          GOTO 90
        ENDIF
 80   CONTINUE
C
C  No demand node exists
C
      GOTO 110
```

```
C
 90   IF (DMNODE .NE. NODES) THEN
        L=DMNODE+1
        DO 100 I=L,NODES
          IF (US(I) .LT.0) THEN
            THD(DMNODE)=I
            CARD(ROOT)=CARD(ROOT)+1
            FLOW(I)=-US(I)
            BASIS(I)=ARTADD
            PRED(I)=ROOT
            TCOST=TCOST+FLOW(I)*COST(ARTADD)
            DMNODE=I
          ENDIF
 100    CONTINUE
      ENDIF
      LNOD(ROOT)=DMNODE
      THD(DMNODE)=ROOT
C
C  Build up chains for the tree
C
C
C  If no undistributed supply exists check for termination
C
 110  IF (ROOT .EQ. NODES) GOTO 160
C
      I=ROOT+1
      DO 150 FMNODE=I,NODES
        IF (US(FMNODE) .LT. 0) GOTO 150
C
C       If no undistributed supply left, connect to tree via
C       artificials
C
        IF (US(FMNODE) .EQ. 0) GOTO 140
C
        LOC1=FROM(FMNODE)
        LOC2=FROM(FMNODE+1)
 120    IF (LOC1 .EQ. LOC2) GOTO 140
        TONODE=TO(LOC1)
C
C       The "to-node" is a demand node
C
        IF (US(TONODE) .LT. 0) THEN
          IF (FLOW(TONODE) .EQ. 0.0 ) THEN
            LOC1=LOC1+1
            GOTO 120
          ENDIF
C
C         A demand node may receive supply
C
          IF ( (UPPER(LOC1) .LE. US(FMNODE)) .AND.
     *         (UPPER(LOC1) .LE. FLOW(TONODE)) ) THEN
C           It is an arc that may be set to upper bound.
```

```
          US(FMNODE)=US(FMNODE)-UPPER(LOC1)
          FLOW(TONODE)=FLOW(TONODE)-UPPER(LOC1)
          TCOST=TCOST+UPPER(LOC1)*(COST(LOC1)-COST(ARTADD))
          STATUS(LOC1)=2
          GFLOW(LOC1)=UPPER(LOC1)
          LOC1=LOC1+1
          IF (US(FMNODE) .EQ. 0) GOTO 140
          GOTO 120
        ENDIF
        IF ( (US(FMNODE) .LT. UPPER(LOC1)) .AND.
     *       (US(FMNODE) .LE. FLOW(TONODE)) ) THEN
C          It is an arc that may become basic
          FLOW(FMNODE)=US(FMNODE)
          FLOW(TONODE)=FLOW(TONODE)-US(FMNODE)
          TCOST=TCOST+US(FMNODE)*(COST(LOC1)-COST(ARTADD))
          GFLOW(LOC1)=FLOW(FMNODE)
          US(FMNODE)=0
C
C          Connect the chain with FMNODE as its highest node to
c          the tree via (FMNODE,TONODE).
C
          CALL CNTDEM(BASIS,CARD,TONODE,FMNODE,LNOD,LOC1,
     *                PRED,STATUS,THD)
          GOTO 150
        ELSE
C          It is an arc that cann't be set to upper bound or made basic.
          LOC1=LOC1+1
          GOTO 120
        ENDIF
C
      ELSE
C
C        The to-node is either a supply node or a transshipment node
C
        IF ( (FMNODE .GT. TONODE .AND. TONODE .GE. ROOT) .OR.
     *       THD(TONODE) .GT.0 ) THEN
          LOC1=LOC1+1
          GO TO 120
        ENDIF
        IF (UPPER(LOC1) .LE. US(FMNODE)) THEN
C          It is an arc that may be set to upper bound
          STATUS(LOC1)=2
          GFLOW(LOC1)=UPPER(LOC1)
          TCOST=TCOST+UPPER(LOC1)*COST(LOC1)
          US(FMNODE)=US(FMNODE)-UPPER(LOC1)
          US(TONODE)=US(TONODE)+UPPER(LOC1)
          LOC1=LOC1+1
          IF (US(FMNODE) .EQ. 0) GOTO 140
          GOTO 120
        ELSE
C          It is an arc that may become basic
          FLOW(FMNODE)=US(FMNODE)
```

```
           GFLOW(LOC1)=FLOW(FMNODE)
           TCOST=TCOST+FLOW(FMNODE)*COST(LOC1)
C
C           Connect FMNODE to TONODE via (FMNODE,TONODE)
C           TONODE becomes the new highest node in the chain
C
           CALL CNTSOT(BASIS,CARD,TONODE,FMNODE,LNOD,LOC1,
     *                PRED,STATUS,THD)
C
           US(TONODE)=US(TONODE)+US(FMNODE)
           US(FMNODE)=0
           GOTO 150
         ENDIF
c
C
        ENDIF
 140     CONTINUE
C
C        Connect FMNODE to tree with an artificial arc.
C
        BASIS(FMNODE)=-ARTADD
        FLOW(FMNODE)=US(FMNODE)
        TCOST=TCOST+FLOW(FMNODE)*COST(ARTADD)
        CALL CNTREE(BASIS,CARD,FMNODE,LNOD,LOC1,PRED,STATUS,THD)
C
 150  CONTINUE
C
C  Connect the tree
C
      IF (ROOT .NE. 1) THEN
 160     N=ROOT-1
        DO 200 FMNODE=1,N
          IF (US(FMNODE) .LT. 0) GO TO 200
C
C          If an undistributed supply is left, connect to tree
C          via artificials
C
          IF (US(FMNODE) .EQ. 0) GO TO 180
C
          LOC1=FROM(FMNODE)
          LOC2=FROM(FMNODE+1)
 170       IF (LOC1 .EQ. LOC2) GO TO 180
          TONODE=TO(LOC1)
C
C
C
C          The "to-node" is a demand node
C
          IF (US(TONODE) .LT. 0) THEN
            IF (FLOW(TONODE) .EQ. 0.0 ) THEN
              LOC1=LOC1+1
              GOTO 170
          ENDIF
C
C          A demand node may receive supply
C
          IF ( (UPPER(LOC1) .LE. US(FMNODE)) .AND.
     *        (UPPER(LOC1) .LE. FLOW(TONODE)) ) THEN
C            It is an arc that may be set to upper bound.
            US(FMNODE)=US(FMNODE)-UPPER(LOC1)
            FLOW(TONODE)=FLOW(TONODE)-UPPER(LOC1)
            STATUS(LOC1)=2
            GFLOW(LOC1)=UPPER(LOC1)
            TCOST=TCOST+UPPER(LOC1)*(COST(LOC1)-COST(ARTADD))
            LOC1=LOC1+1
            IF (US(FMNODE) .EQ. 0) GOTO 180
            GOTO 170
          ENDIF
          IF ( (US(FMNODE) .LT. UPPER(LOC1)) .AND.
     *        (US(FMNODE) .LE. FLOW(TONODE)) ) THEN
C            It is an arc that may become basic
            FLOW(FMNODE)=US(FMNODE)
            FLOW(TONODE)=FLOW(TONODE)-US(FMNODE)
            GFLOW(LOC1)=FLOW(FMNODE)
            TCOST=TCOST+FLOW(FMNODE)*(COST(LOC1)-COST(ARTADD))
            US(FMNODE)=0
C
C            Connect the chain with FMNODE as its highest node to
c            the tree via (FMNODE,TONODE).
C
            CALL CNTDEM(BASIS,CARD,TONODE,FMNODE,LNOD,LOC1,
     *                PRED,STATUS,THD)
C
            GOTO 200
          ELSE
C            It is an arc that cann't be set to upper bound or made basic.
            LOC1=LOC1+1
            GOTO 170
          ENDIF
C
        ELSE
C
C          The to-node is either a supply node or a transshipment node
C
          IF ( (FMNODE .GT. TONODE .AND. TONODE .GE. ROOT) .OR.
     *        THD(TONODE) .GT.0 ) THEN
            LOC1=LOC1+1
            GO TO 170
          ENDIF
          IF (UPPER(LOC1) .LE. US(FMNODE)) THEN
C            It is an arc that may be set to upper bound
            STATUS(LOC1)=2
            GFLOW(LOC1)=UPPER(LOC1)
            TCOST=TCOST+UPPER(LOC1)*COST(LOC1)
```

```
                US(FMNODE)=US(FMNODE)-UPPER(LOC1)
                US(TONODE)=US(TONODE)+UPPER(LOC1)
                LOC1=LOC1+1
                IF (US(FMNODE) .EQ. 0) GOTO 180
                GOTO 170
              ELSE
C                It is an arc that may become basic
                FLOW(FMNODE)=US(FMNODE)
                GFLOW(LOC1)=FLOW(FMNODE)
                TCOST=TCOST+FLOW(FMNODE)*COST(LOC1)
C
C                Connect FMNODE to TONODE via (FMNODE,TONODE)
C                TONODE becomes the new highest node in the chain
C
                CALL CNTSOT(BASIS,CARD,TONODE,FMNODE,LNOD,LOC1,
     *                   PRED,STATUS,THD)
C
                US(TONODE)=US(TONODE)+US(FMNODE)
                US(FMNODE)=0
                GOTO 200
              ENDIF
c
C
            ENDIF
 180         CONTINUE
C
C           Connect FMNODE to tree with an artificial arc.
C
            BASIS(FMNODE)=-ARTADD
            FLOW(FMNODE)=US(FMNODE)
            TCOST=TCOST+FLOW(FMNODE)*COST(ARTADD)
            CALL CNTREE(BASIS,CARD,FMNODE,LNOD,LOC1,PRED,STATUS,THD)
C
 200      CONTINUE
        ENDIF
C
 220  CONTINUE
C
C   Starting basis is complete
C
C
        RETURN
        END
C
C
C
C
C
C**** **********************************************************
        SUBROUTINE CNTSOT(BASIS,CARD,TONODE,FMNODE,LNOD,LOC,
       1            PRED,STATUS,THD)
C**** **********************************************************
```

```
C
C    This routine connects the chain with FMNODE as the highest
C    node in the chain to either a supply node or a transshipment
C    node TONODE via (FMNODE,TONODE).
c
C
c
C  SUBROUTINE ARGUMENTS
C
      INTEGER BASIS(*),CARD(*),TONODE,FMNODE,LNOD(*),LOC,
     1        PRED(*),STATUS(*),THD(*)
C
C  LOCAL VARIABLES
C
      INTEGER LNODP,P,Q
C
      P=FMNODE
      Q=TONODE
      LNODP=LNOD(P)
      STATUS(LOC)=0
      BASIS(P)=-LOC
C
      CARD(Q)=CARD(P)+1
      THD(Q)=P
      LNOD(Q)=LNODP
      PRED(P)=Q
C
      RETURN
      END
C
C
C
C
C
C
C**** ***********************************************************
      SUBROUTINE CNTDEM(BASIS,CARD,DEMAND,FMNODE,LNOD,LOC,
     1                  PRED,STATUS,THD)
C**** ***********************************************************
C
C    This routine connects the chain with FMNODE as the highest
C    node in the chain to the tree via (FMNODE,TONODE).
C
C  SUBROUTINE ARGUMENTS
C
      INTEGER BASIS(*),CARD(*),DEMAND,FMNODE,LNOD(*),LOC,
     1        PRED(*),STATUS(*),THD(*)
C
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *        ROOT
      COMMON/PARM/TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *          NGUBS,NODES,ROOT,EPSILON,BIG1
C
C LOCAL VARIABLES
C
      INTEGER LNODP,LNODQ,P,Q,THDLDQ
C
      P=FMNODE
      Q=DEMAND
      LNODP=LNOD(P)
      LNODQ=LNOD(Q)
      THDLDQ=THD(LNODQ)
      STATUS(LOC)=0
      BASIS(P)=-LOC
C
      CARD(Q)=CARD(Q)+CARD(P)
      THD(LNODQ)=P
      LNOD(Q)=LNODP
C
      PRED(P)=Q
      THD(LNODP)=THDLDQ
C
      CARD(ROOT)=CARD(ROOT)+CARD(P)
      IF (LNOD(ROOT) .EQ. Q) THEN
        LNOD(ROOT)=LNOD(Q)
      ENDIF
C
      RETURN
      END
C
C
C
C
C
C**** **************************************************************
      SUBROUTINE CNTREE(BASIS,CARD,FMNODE,LNOD,LOC,PRED,STATUS,THD)
C**** **************************************************************
C
C    This routine connects node FMNODE to ROOT via (FMNODE,ROOT)
C    and at the same time records the information in a column of
C    the basis matrix.
C
C
C  SUBROUTINE ARGUMENTS
C
      INTEGER BASIS(*),CARD(*),FMNODE,LNOD(*),LOC,PRED(*),
     *       STATUS(*),THD(*)
C
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *       ROOT
      COMMON/PARM/TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *          NGUBS,NODES,ROOT,EPSILON,BIG1
```

```
C
C  LOCAL VARIABLES
C
      INTEGER LNODP,LNODRT,P
C
c
      P=FMNODE
      LNODRT=LNOD(ROOT)
      LNODP=LNOD(P)
      STATUS(ABS(BASIS(P)))=0
C
      CARD(ROOT)=CARD(ROOT)+CARD(P)
      THD(LNODRT)=P
      LNOD(ROOT)=LNODP
C
      PRED(P)=ROOT
      THD(LNODP)=ROOT
C
      RETURN
      END
C**** ****************************************************************
      SUBROUTINE PURNET(ARCNAM,BASIS,CARD,COST,FLOW,FROM,FROMO,GFLOW,
     1                LNOD,LOWER,PI,PRED,STATUS,THD,TO,UPPER)
C**** ****************************************************************
C
C  SUBROUTINE ARGUMENTS
C
      CHARACTER*8 ARCNAM(*)
      INTEGER BASIS(*),CARD(*),FROM(*),FROMO(*),LNOD(*),PRED(*),
     *        STATUS(*),THD(*),TO(*)
      DOUBLE PRECISION COST(*),FLOW(*),GFLOW(*),LOWER(*),PI(*),
     *                 UPPER(*)
C
C  LOCAL VARIABLES
C
      DOUBLE PRECISION DELT,DELTA,FLOWXU,FLOWXV,FLWPRX,
     *                 FX,GUDELT,GVDELT,
     *                 MINCOS,REDCOS
      INTEGER BASISJ,BASISQ,BASISU,BASISV,BASISX,BASPRX,BX,CARDX,
     *        CARDPX, FRMNOD,FRMBEG,BTB,
     *        ENTER,GAMMAU,GAMMAV,
     *        I,INDEX1,INDEX2,J,K,L,MUV,
     *        NODSM1,P,PATHRT,PREDJ,PREDPX,PREDX,Q,RTHD,
     *        THDJ,THDX,THDY1,U,V,W,X,XBAR,XSTAR,XU,XV,X1,X2,X2BAR,
     *        Y1,Y2,Z
C
      INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *       ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *           NGUBS,NODES,ROOT,EPSILON,BIG1
C
C  ITRTOT  = TOTAL NUMBER OF PIVOTS
C  ITRREG = NUMBER OF REGUALR PIVOTS
C  ITRBTB = NUMBER OF BOUND-TO-BOUND PIVOTS
C  ITRDEG = NUMBER OF DEGENERATE PIVOTS
C
      ITRTOT=0
      ITRREG=0
      ITRBTB=0
      ITRDEG=0
C
      FRMNOD=1
C
C
C  DETERMINE DUAL VALUES
C
C
      J=ROOT
      PI(J)=0.0D0
      NODSM1=NODES-1
      DO 90 I=1,NODSM1
        THDJ=THD(J)
        PREDJ=PRED(THDJ)
        BASISJ=BASIS(THDJ)
        IF (BASISJ.GT.0) THEN
          PI(THDJ)=PI(PREDJ)+COST(BASISJ)
        ELSE
          BASISJ=IABS(BASISJ)
          PI(THDJ)=PI(PREDJ)-COST(BASISJ)
        ENDIF
        J=THDJ
   90 CONTINUE
C
C  FIND ENTERING ARC
C
   95 CONTINUE
C
C  THIS ROUTINE WILL FIND AN ELIGIBLE ARC TO ENTER THE CURRENT BASIS
C  BY USING THE CRITERIA OF MOST NEGATIVE (MOST POSITIVE) REDUCED COST
C  OUT OF A NODE
C
C  AS SOON AS A NODE IS FOUND WITH AN ELIGIBLE ARC THE SEARCH STOPS
C
C
C           0 --> ARC BASIC
C  STATUS = 1 --> ARC NONBASIC AT LOWER BOUND
C           2 --> ARC NONBASIC AT UPPER BOUND
C           3 --> ARC FIXED
C
```

```
C  ARC (U,V) WILL BE THE ARC TO ENTER
C
    MINCOS=0.0D0
    FRMBEG=FRMNOD
 800 CONTINUE
       INDEX1=FROM(FRMNOD)
       INDEX2=FROM(FRMNOD+1)-1
       DO 810 K=INDEX1,INDEX2
         IF (STATUS(K) .NE. 0) THEN
           J=TO(K)
           REDCOS=COST(K)+PI(FRMNOD)
           REDCOS=REDCOS-PI(J)
           IF (ABS(REDCOS) .LE. 1E-15) REDCOS=0.0D0
           IF (ABS(REDCOS) .GT. MINCOS) THEN
             IF (REDCOS .LT. 0.0  .AND.  STATUS(K) .EQ. 1) THEN
               U=FRMNOD
               V=J
               ENTER=K
               MINCOS=ABS(REDCOS)
               GOTO 810
             ENDIF
             IF (REDCOS .GT. 0.0  .AND.  STATUS(K) .EQ. 2) THEN
               U=FRMNOD
               V=J
               ENTER=K
               MINCOS=REDCOS
             ENDIF
           ENDIF
         ENDIF
 810   CONTINUE
       IF (MINCOS .NE. 0.0) THEN
         FRMNOD=FRMNOD+1
         IF (FRMNOD .GT. NODES) FRMNOD=1
         GOTO 830
       ENDIF
       FRMNOD=FRMNOD+1
       IF (FRMNOD .GT. NODES) FRMNOD=1
       IF (FRMNOD .NE. FRMBEG) GOTO 800
C
 830 CONTINUE
C
C  IF MINCOS = 0 THEN NO ELIGIBLE ARC WAS FOUND
C
    IF (ABS(MINCOS) .LE. 1E-15) MINCOS=0.0D0
    IF (MINCOS .EQ. 0.0) GOTO 145
C
C
C  FIND THE BASIS EQUIVALENT PATH AND DETERMINE THE LEAVING ARC
C
C  ARC (P,Q) WILL LEAVE
C
    IF (STATUS(ENTER).EQ.1) THEN
```

```
      GAMMAU=-1
      GAMMAV=1
    ELSE
      GAMMAU=1
      GAMMAV=-1
    ENDIF
    XU=U
    XV=V
    DELTA=BIG1
    IF (XU.NE.XV) THEN
10     IF (CARD(XU) .LE. CARD(XV)) THEN
         BASISU=BASIS(XU)
         IF (BASISU*GAMMAU .LE. 0) THEN
           BASISU=IABS(BASISU)
           DELT=UPPER(BASISU)-FLOW(XU)
           IF (ABS(DELT) .LE. 1E-15) DELT=0.0D0
           IF (ABS(DELT-DELTA) .LE. 1E-15) DELTA=DELT
           IF (DELT .LT. DELTA) THEN
             DELTA=DELT
             MUV=U
             Q=XU
             P=PRED(Q)
           ENDIF
         ELSE
           FLOWXU=FLOW(XU)
           IF (ABS(DELTA-FLOWXU) .LE. 1E-15) DELTA=FLOWXU
           IF (FLOWXU.LT.DELTA) THEN
             DELTA=FLOWXU
             MUV=U
             Q=XU
             P=PRED(Q)
           ENDIF
         ENDIF
         XU=PRED(XU)
         IF (XU .EQ. XV) THEN
           GOTO 20
         ELSE
           GOTO 10
         ENDIF
       ENDIF
       BASISV=BASIS(XV)
       IF (BASISV*GAMMAV .LE. 0) THEN
         BASISV=IABS(BASISV)
         DELT=UPPER(BASISV)-FLOW(XV)
         IF (ABS(DELT) .LE. 1E-15) DELT=0.0D0
         IF (ABS(DELTA-DELT) .LE. 1E-15) DELTA=DELT
         IF (DELT.LT.DELTA) THEN
           DELTA=DELT
           MUV=V
           Q=XV
           P=PRED(Q)
         ENDIF
```

```
          ELSE
            FLOWXV=FLOW(XV)
            IF (ABS(DELTA-FLOWXV) .LE. 1E-15) DELTA=FLOWXV
            IF (FLOWXV.LT.DELTA) THEN
              DELTA=FLOWXV
              MUV=V
              Q=XV
              P=PRED(Q)
            ENDIF
          ENDIF
          XV=PRED(XV)
          IF (XU .EQ. XV) THEN
            GOTO 20
          ELSE
            GOTO 10
          ENDIF
        ENDIF
C
C  SAVE INTERSECTION NODE FROM BASIS EQUIVALENT PATH
C
   20 CONTINUE
      PATHRT=XU
C      WRITE(6,*)' ARC ENTERING = ',U,V,ENTER
C      WRITE(6,*)' ARC LEAVING =  ',P,Q,BASIS(Q)
C
C  CHECK FOR DEGENERATE PIVOT (I.E. NONBASIC ARC AT L.B. CHANGING
C  TO NONBASIC AT U.B. OR VICE VERSA).
C
      BTB=0
      IF (ABS(UPPER(ENTER)-DELTA) .LE. 1E-15) DELTA=UPPER(ENTER)
      IF(UPPER(ENTER).LE.DELTA) THEN
        BTB=1
        DELTA=UPPER(ENTER)
      ENDIF
C
C  UPDATE FLOWS ON BASIS EQUIVALENT PATH.
C
      IF (ABS(DELTA) .LE. 1E-15) DELTA=0.0D0
      IF (DELTA .GT. 0.0) THEN
        XU=U
        GUDELT=GAMMAU*DELTA
   35   IF(XU.NE.PATHRT) THEN
          BASISU=BASIS(XU)
          IF(BASISU.LT.0) THEN
            FLOW(XU)=FLOW(XU)+GUDELT
          ELSE
            FLOW(XU)=FLOW(XU)-GUDELT
          ENDIF
          IF (ABS(FLOW(XU)) .LE.1E-15) FLOW(XU)=0.0D0
          BASISU=IABS(BASISU)
          GFLOW(BASISU)=FLOW(XU)
          XU=PRED(XU)
          GO TO 35
        ENDIF
        XV=V
        GVDELT=GAMMAV*DELTA
   36   IF(XV.NE.PATHRT) THEN
          BASISV=BASIS(XV)
          IF(BASISV.LT.0) THEN
            FLOW(XV)=FLOW(XV)+GVDELT
          ELSE
            FLOW(XV)=FLOW(XV)-GVDELT
          ENDIF
          IF (ABS(FLOW(XV)) .LE. 1E-15) FLOW(XV)=0.0D0
          BASISV=IABS(BASISV)
          GFLOW(BASISV)=FLOW(XV)
          XV=PRED(XV)
          GO TO 36
        ENDIF
      ELSE
        ITRDEG=ITRDEG+1
      ENDIF
C
C  BTB = 1 MEANS BOUND-TO-BOUND PIVOT
C  IF IT IS THIS TYPE OF PIVOT THEN A BASIS TREE UPDATE IS NOT DONE
C
      IF (BTB .EQ. 1) THEN
        ITRBTB=ITRBTB+1
        IF (STATUS(ENTER) .EQ. 1) THEN
          STATUS(ENTER)=2
          GFLOW(ENTER)=UPPER(ENTER)
        ELSE
          STATUS(ENTER)=1
          GFLOW(ENTER)=0.0D0
        ENDIF
        GOTO 999
      ELSE
        ITRREG=ITRREG+1
        BASISQ=IABS(BASIS(Q))
        IF (ABS(FLOW(Q)) .LE. 1E-15) FLOW(Q)=0.0D0
        IF (FLOW(Q) .EQ. 0.0) THEN
          STATUS(BASISQ)=1
          GFLOW(BASISQ)=0.0D0
        ELSE
          STATUS(BASISQ)=2
          GFLOW(BASISQ)=UPPER(BASISQ)
        ENDIF
      ENDIF
C
C  REGULAR TREE REROOTING PIVOT
C
C  UPDATE THE TREE NOT CONTAINING NODE Q FIRST
C
C  FIND REVERSE THREAD OF Q
```

```
C
      RTHD=PRED(Q)
   40 IF (THD(RTHD).NE.Q) THEN
        RTHD=LNOD(THD(RTHD))
        GOTO 40
      ENDIF
C
C  UPDATE THE THREAD OF RTHD
C
      THD(RTHD)=THD(LNOD(Q))
C
C  UPDATE LAST NODE
C
      XSTAR=PRED(THD(LNOD(Q)))
      IF (XSTAR.EQ.0) THEN
        XSTAR=ROOT
        LNOD(XSTAR)=RTHD
      ENDIF
      X=P
   50 IF (X.NE.XSTAR) THEN
        LNOD(X)=RTHD
        X=PRED(X)
        GOTO 50
      ENDIF
C      IF (LNOD(XSTAR).EQ.LNOD(Q)) THEN
C        LNOD(XSTAR)=RTHD
C      ENDIF
C
C  UPDATE CARDINALITY
C
      X=P
C      PREDPR=PRED(PATHRT)
C
C  IF THE PATH ROOT IS THE ROOT THEN ADJUST PREDPR TO AVOID
C  UPDATING CARDINALITY OF ROOT
C
C      IF (PREDPR.EQ.0) THEN
C        PREDPR=ROOT
C      ENDIF
   60 IF (X .NE. 0) THEN
        CARD(X)=CARD(X)-CARD(Q)
        X=PRED(X)
        GOTO 60
      ENDIF
C
C  UPDATE THE TREE CONTAINING Q
C
      PRED(Q)=0
      THD(LNOD(Q))=Q
      BASIS(Q)=0
C
C  DETERMINE WHICH TREE WILL BE T1 AND WHICH WILL BE T2.
```

```
C  LET T2 BE THE TREE WITH THE SMALLEST NUMBER OF NODES.
C
     IF (CARD(ROOT) .LT. CARD(Q)) THEN
        X1=Q
        X2=ROOT
        IF (MUV.EQ.U) THEN
           Y1=U
           Y2=V
        ELSE
           Y1=V
           Y2=U
        ENDIF
     ELSE
        X1=ROOT
        X2=Q
        IF (MUV.EQ.U) THEN
           Y1=V
           Y2=U
        ELSE
           Y1=U
           Y2=V
        ENDIF
     ENDIF
C      WRITE(6,*)' X1, Y1, X2, Y2 = ',X1,Y1,X2,Y2
C
C  UPDATE PREDECESSOR
C
     IF (X2 .NE. Y2) THEN
        X=Y2
        W=LNOD(X)
        Z=THD(W)
        CARDX=CARD(Y2)
        CARDPX=CARD(PRED(Y2))
C        IF (X1.EQ.Q) THEN
C           CARD(Y2)=CARD(X2)-CARD(Q)
C        ELSE
          CARD(Y2)=CARD(X2)
C        ENDIF
        PREDX=PRED(X)
        PREDPX=PRED(PREDX)
C
C  UPDATE BASIS AND FLOW.
C
        BASPRX=BASIS(PREDX)
        BASIS(PREDX)=-BASIS(X)
        FLWPRX=FLOW(PREDX)
        FLOW(PREDX)=FLOW(X)
        IF (ABS(FLOW(PREDX)) .LE. 1E-15) FLOW(PREDX)=0.0D0
        L=IABS(BASIS(PREDX))
        GFLOW(L)=FLOW(PREDX)
C
  70    IF (X.NE.X2) THEN
```

```
         PRED(PREDX)=X
C
C  FIND REVERSE THREAD OF X
C
         RTHD=PREDX
  80      IF (THD(RTHD).NE.X) THEN
           RTHD=LNOD(THD(RTHD))
           GOTO 80
         ENDIF
         IF (PRED(Z).EQ.PREDX) THEN
           THD(RTHD)=Z
           THD(W)=PREDX
           W=LNOD(PREDX)
           Z=THD(W)
         ELSE
           THD(W)=PREDX
           W=RTHD
         ENDIF
         CARD(PREDX)=CARD(Y2)-CARDX
         CARDX=CARDPX
         IF (PREDPX.NE.0) THEN
           CARDPX=CARD(PREDPX)
         ENDIF
         IF (PREDX.EQ.X2) THEN
           X2BAR=X
         ENDIF
         X=PREDX
         PREDX=PREDPX
C
C  UPDATE BASIS AND FLOW
C
         BX=BASIS(PREDX)
         BASIS(PREDX)=-BASPRX
         BASPRX=BX
         FX=FLOW(PREDX)
         FLOW(PREDX)=FLWPRX
         IF (ABS(FLOW(PREDX)) .LE. 1E-15) FLOW(PREDX)=0.0D0
         L=IABS(BASIS(PREDX))
         GFLOW(L)=FLOW(PREDX)
         FLWPRX=FX
C
         IF (PREDPX.NE.0) THEN
           PREDPX=PRED(PREDPX)
         ENDIF
         GOTO 70
       ENDIF
C
C  UPDATE LAST NODE
C
       PRED(Y2)=0
       THD(W)=Y2
       IF (LNOD(X2).EQ.LNOD(X2BAR)) THEN
```

```
           LNOD(X2)=W
         ENDIF
         X=PRED(X2)
  100     IF (X.NE.Y2) THEN
           LNOD(X)=LNOD(X2)
           X=PRED(X)
           GOTO 100
         ENDIF
         LNOD(X)=LNOD(X2)
       ENDIF
C
C  IS T(Q) = T1?
C
C        IF (Q .EQ. X1) THEN
C          CARD(Y2)=CARD(X2)-CARD(Q)
C        ENDIF
C      ENDIF
C
C  ATTACH T2 TO T1
C  UPDATE PREDECESSOR
C
       PRED(Y2)=Y1
C
C  UPDATE BASIS AND FLOW OF Y2.
C
       IF(Y2.EQ.U) THEN
         BASIS(Y2)=-ENTER
         PI(Y2)=PI(V)-COST(ENTER)
       ELSE
         BASIS(Y2)=ENTER
         PI(Y2)=PI(U)+COST(ENTER)
       ENDIF
       IF (ABS(PI(Y2)) .LE. 1E-15) PI(Y2)=0.0D0
       IF(STATUS(ENTER).EQ.1) THEN
         FLOW(Y2)=DELTA
       ELSE
         FLOW(Y2)=UPPER(ENTER)-DELTA
       ENDIF
       IF (ABS(FLOW(Y2)) .LE. 1E-15) FLOW(Y2)=0.0D0
       L=IABS(BASIS(Y2))
       GFLOW(L)=FLOW(Y2)
C
C  UPDATE THREAD
C
       THD(LNOD(Y2))=THD(Y1)
       THDY1=THD(Y1)
       THD(Y1)=Y2
C
C  UPDATE LAST NODE
C
       XBAR=PRED(THDY1)
       IF (XBAR.EQ.0) THEN
```

```
C        XBAR=X1
        LNOD(XBAR)=LNOD(Y2)
      ENDIF
      X=Y1
  110 IF (X.NE.XBAR) THEN
        LNOD(X)=LNOD(Y2)
        X=PRED(X)
        GOTO 110
      ENDIF
C      IF (XBAR.EQ.0) THEN
C        LNOD(XBAR)=LNOD(Y2)
C      ENDIF
C
C  UPDATE CARDINALITY
C
      X=Y1
C      IF (X2.EQ.Q) THEN
C 120    IF (X.NE.PATHRT) THEN
C          CARD(X)=CARD(X)+CARD(Y2)
C          X=PRED(X)
C          GOTO 120
C        ENDIF
C        IF (PATHRT.NE.ROOT) THEN
C          CARD(X)=CARD(X)+CARD(Y2)
C        ENDIF
C      ELSE
  130    IF (X.NE.X1) THEN
          CARD(X)=CARD(X)+CARD(Y2)
          X=PRED(X)
          GOTO 130
        ENDIF
        CARD(X)=CARD(X)+CARD(Y2)
C      ENDIF
C
C  UPDATE DUAL VALUES ON REROOTED TREE.
C
      X=Y2
  140 IF(X.NE.LNOD(Y2)) THEN
        THDX=THD(X)
        PREDX=PRED(THDX)
        BASISX=BASIS(THDX)
        IF(BASISX.GT.0) THEN
          PI(THDX)=PI(PREDX)+COST(BASISX)
        ELSE
          BASISX=IABS(BASISX)
          PI(THDX)=PI(PREDX)-COST(BASISX)
        ENDIF
        IF (ABS(PI(THDX)) .LE. 1E-15) PI(THDX)=0.0D0
        X=THDX
        GO TO 140
      ENDIF
C
```

```
C  CHANGE STATUS OF ENTERING ARC
C
      STATUS(ENTER)=0
      ROOT=X1
  999 CONTINUE
C
C  UPDATE OBJECTIVE FUNCTION VALUE
C
      TCOST=TCOST-DFLOAT(DELTA)*DBLE(MINCOS)
C
C  INCREMENT ITERATION COUNT
C
      ITRTOT=ITRTOT+1
      IF (ITRTOT .GT. ITRMAX) THEN
        FLGITR=1
        RETURN
      ENDIF
C
C  CHECK FOR INTERMEDIATE OBJECTIVE FUNCTION VALUE REPORT
C
      IF (MOD(ITRTOT,ITROBJ) .EQ. 0) THEN
        WRITE (6,1000) ITRTOT,TCOST
      ENDIF
C
C  CHECK FOR INTERMEDIATE SOLUTION REPORT
C
      IF (MOD(ITRTOT,ITROUT) .EQ. 0) THEN
        WRITE (6,2000) ITRTOT
      ENDIF
      GOTO 95
C
  145 CONTINUE
C
C  OPTIMALITY INDICATED, ARE WE FEASIBLE?
C
      DO 150 I=1,NODES
        IF (ABS(FLOW(I)) .LE. 1E-15) FLOW(I)=0.0D0
        IF (IABS(BASIS(I)) .EQ. ARTADD  .AND.  FLOW(I) .NE. 0.00) THEN
          FLGINF=1
          RETURN
        ENDIF
  150 CONTINUE
      FLGOPT=1
C
C  FORMATS
C
 1000 FORMAT(1X,'AT ITERATION ',I6,' OBJECTIVE FUNCTION VALUE = ',F15.0)
 2000 FORMAT(//1X,'AT ITERATION ',I6)
      RETURN
      END
C**** ************************************************************
      SUBROUTINE LBALG(ARCNAM,BASIS,CARD,COST,CTEMP,FLOW,
     1              FROM,FROMO,GCOEF,GFLOW,GUBADD,GVAL,
     1              LGMULT,LNOD,LOWER,NUMVRG,PI,PRED,
     1              REDGUB,STATUS,THD,
     1              TO,UPPER,YFLOW)
C**** *************************************************************
C
C  THE PURPOSE OF THIS ROUTINE IS TO CALCULATE A LOWER BOUND
C  FOR THE PROBLEM (NPG)
C
C
C  SUBROUTINE ARGUMENTS
C
      CHARACTER*8 ARCNAM(*)
      INTEGER BASIS(*),CARD(*),FROM(*),FROMO(*),
     1        GUBADD(*),LNOD(*),NUMVRG(*),PRED(*),
     1        REDGUB(*),STATUS(*),THD(*),TO(*)
      DOUBLE PRECISION COST(*),CTEMP(*),FLOW(*),GCOEF(*),GFLOW(*),
     1              GVAL(*),LGMULT(*),LOWER(*),PI(*),
     1              UPPER(*),YFLOW(*)
C
C  LOCAL VARIABLES
C
      INTEGER I,INDEX,K,LOC1,LOC2,REC
      DOUBLE PRECISION NORM,LBSUBG
C
      INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *       ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *            NGUBS,NODES,ROOT,EPSILON,BIG1
      DOUBLE PRECISION LBSTEP,ILBSTEP,MAXSTEP
      COMMON /STEP/ LBSTEP,ILBSTEP,MAXSTEP
      DOUBLE PRECISION LBND,LBNDPRE,UBND,UBNDPRE
      COMMON /BOUNDS/ LBND,LBNDPRE,UBND,UBNDPRE
      INTEGER NGVLT
      COMMON /NGV/ NGVLT
C
      TCOST=0.0D0
      NGVLT=0
C
C  SOLVE THE NETWORK WITH ADJUSTED COST COEFFICIENTS
C
C
C  CALCULATE THE NEW COST COEFFICIENTS
C
      DO 5 REC=1,ARCS
        LOC=GUBADD(REC)
        IF (GCOEF(LOC) .NE. 0.0) THEN
```

```
            IF (ABS(LGMULT(REC)) .LT. 1E-15) LGMULT(REC)=0.0D0
            CTEMP(LOC)=COST(LOC)-LGMULT(REC)
          ELSE
            CTEMP(LOC)=COST(LOC)
          ENDIF
          IF (ABS(GFLOW(LOC)) .LE. 1E-15) GFLOW(LOC)=0.0D0
          TCOST=TCOST+CTEMP(LOC)*GFLOW(LOC)
 5    CONTINUE
C
C
      CALL PURNET(ARCNAM,BASIS,CARD,CTEMP,FLOW,FROM,FROMO,GFLOW,
     *            LNOD,LOWER,PI,PRED,STATUS,THD,TO,UPPER)
C
      IF (FLGINF .EQ. 1) RETURN
C
C
      LBND=TCOST
C
      FLGOPT=1
C
      NORM=0.0D0
      MAXSTEP=0.0D0
      LOC2=0
      DO 10 INDEX=1,NGUBS
        LOC1=LOC2+1
        LOC2=LOC2+NUMVRG(INDEX)
        IF (REDGUB(INDEX) .EQ. 1) THEN
           DO 20 K=LOC1,LOC2
              REC=GUBADD(K)
              YFLOW(REC)=GFLOW(REC)
              LGMULT(K)=0.0D0
              CTEMP(REC)=0.0D0
 20         CONTINUE
           GO TO 10
        ENDIF
        CALL SCBVLP(GVAL(INDEX),GFLOW,GUBADD,GCOEF,
     1              LGMULT,LOC1,LOC2,NUMVRG(INDEX),
     1              UPPER,YFLOW,NORM)
 10   CONTINUE
C
      IF (LBND .GT. LBNDPRE) THEN
        CALL LBSAV(BASIS,CARD,FLOW,GFLOW,LNOD,PRED,STATUS,
     *             THD,YFLOW)
        LBNDPRE=LBND
      ENDIF
      IF (NORM .EQ. 0.00) THEN
        WRITE(6,*) 'OPTIMALITY REACHED ON LOWER BOUND PROCEDURE'
        FLGOPT=1
        RETURN
      ENDIF
C
C
```

```
C     IF ((UBNDPRE-LBND) .LE. .03*UBND) THEN
        LBSTEP=MAXSTEP/NORM
C
C        LBSTEP=MAXSTEP/SQRT(NORM)
C     ENDIF
C
C  CALCULATE THE NEW LAGRANGE MULTIPLIERS
C
      DO 30 INDEX=1,GARCS
        REC=GUBADD(INDEX)
        LBSUBG=YFLOW(REC)-GFLOW(REC)
        IF (ABS(LBSUBG) .LE. 1E-15) LBSUBG=0.0D0
        CTEMP(REC)=-LBSTEP*LBSUBG
        IF (ABS(CTEMP(REC)) .LE. 1E-15) CTEMP(REC)=0.0D0
        LGMULT(INDEX)=LGMULT(INDEX)+LBSUBG*LBSTEP
 30   CONTINUE
C
C  CHECK FOR OPTIMALITY OR NEAR OPTIMALITY
C
      IF (FLGOPT .EQ. 1) THEN
        WRITE(6,*) 'OPTIMALITY REACHED AT LOWER BOUND PROCEDURE'
        RETURN
      ENDIF
C
 40   CONTINUE
C
      IF ((UBNDPRE-LBNDPRE) .LE. EPSILON*ABS(UBND)) THEN
        FLGOPT=1
        LBND=LBNDPRE
        WRITE(6,*)'NEAR OPTIMALITY REACHED AT LOWER BOUND PROCEDURE'
        WRITE(6,*)'SOLUTION IS WITHIN',100*EPSILON,'% '
      ENDIF
C
      RETURN
      END
C
C
C
C**** ************************************************************
      SUBROUTINE SCBVLP(VAL,GFLOW,GUBADD,GCOEF,
     1                 LGMULT,LOC1,LOC2,NUMVRS,
     1                 UPPER,YFLOW,NORM)
C**** ***********************************************************
C
C  SUBROUTINE ARGUMENTS
C
      INTEGER GUBADD(*),LOC1,LOC2,NUMVRS
      DOUBLE PRECISION VAL,GCOEF(*),GFLOW(*),
     1      LGMULT(*),UPPER(*),YFLOW(*),NORM
C
C  LOCAL VARIABLES
C
```

```
      INTEGER INDEX,REC,LASTRC,LOC,NUM,R2
      DOUBLE PRECISION ADJVAL,LBSUBG,LGLB,L1,L2,SUM
C
      INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *       ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *             NGUBS,NODES,ROOT,EPSILON,BIG1
      DOUBLE PRECISION LBSTEP,ILBSTEP,MAXSTEP
      COMMON /STEP/ LBSTEP,ILBSTEP,MAXSTEP
      DOUBLE PRECISION LBND,LBNDPRE,UBND,UBNDPRE
      COMMON /BOUNDS/ LBND,LBNDPRE,UBND,UBNDPRE
      INTEGER NGVLT
      COMMON /NGV/ NGVLT
C
C  DETERMINE THE ADJUSTED  RIGHT HAND SIDE
C  SET ALL THE VARIABLES WITH A ZERO OR POSITIVE RATIO
C
      NUM=NUMVRS
      SUM=0.0D0
      LASTRC=LOC2
      ADJVAL=VAL
      DO 10 REC=LOC1,LOC2
 5       LOC=GUBADD(REC)
         SUM=SUM+GFLOW(LOC)*GCOEF(LOC)
         IF (GCOEF(LOC) .GT. 0.0 .AND. LGMULT(REC) .GE. 0.0) THEN
            YFLOW(LOC)=0.0D0
            LBSUBG=YFLOW(LOC)-GFLOW(LOC)
            NORM=NORM+LBSUBG*LBSUBG
            IF (ABS(LBSUBG) .LE. 1E-15) LBSUBG=0.0D0
C            CTEMP(LOC)=-LBSTEP*LBSUBG
            IF (FLGOPT .EQ. 1) THEN
               LGLB=LGMULT(REC)*LBSUBG
               IF (ABS(LGLB) .GT. 1E-15) FLGOPT=0
            ENDIF
C            LGMULT(REC)=LGMULT(REC)-CTEMP(LOC)
C            IF (ABS(LGMULT(REC)) .LE. 1E-15) LGMULT(REC)=0.0D0
            NUM=NUM-1
            IF (REC .EQ. LOC2) THEN
               LOC2=LOC2-1
               GO TO 15
            ENDIF
            L1=LGMULT(REC)
            L2=LGMULT(LOC2)
            LGMULT(REC)=L2
            LGMULT(LOC2)=L1
            R2=GUBADD(LOC2)
            GUBADD(REC)=R2
```

```
          GUBADD(LOC2)=LOC
          LOC2=LOC2-1
          GO TO 5
        ENDIF
        IF (GCOEF(LOC) .LT. 0.0 ) THEN
          ADJVAL=ADJVAL-GCOEF(LOC)*UPPER(LOC)
          IF (ABS(ADJVAL) .LE. 1E-15) ADJVAL=0.0D0
          IF (LGMULT(REC) .LE. 0.0) THEN
            YFLOW(LOC)=UPPER(LOC)
            LBND=LBND+LGMULT(REC)*YFLOW(LOC)
            LBSUBG=YFLOW(LOC)-GFLOW(LOC)
            NORM=NORM+LBSUBG*LBSUBG
            IF (ABS(LBSUBG) .LE. 1E-15) LBSUBG=0.0D0
C            CTEMP(LOC)=-LBSTEP*LBSUBG
            IF (FLGOPT .EQ. 1) THEN
              LGLB=LGMULT(REC)*LBSUBG
              IF (ABS(LGLB) .GT. 1E-15) FLGOPT=0
            ENDIF
C            LGMULT(REC)=LGMULT(REC)-CTEMP(LOC)
C            IF (ABS(LGMULT(REC)) .LE. 1E-15) LGMULT(REC)=0.0D0
            NUM=NUM-1
            IF (REC .EQ. LOC2) THEN
              LOC2=LOC2-1
              GO TO 15
            ENDIF
            L1=LGMULT(REC)
            L2=LGMULT(LOC2)
            LGMULT(REC)=L2
            LGMULT(LOC2)=L1
            R2=GUBADD(LOC2)
            GUBADD(REC)=R2
            GUBADD(LOC2)=LOC
            LOC2=LOC2-1
            GO TO 5
          ENDIF
         ENDIF
         IF (REC .EQ. LOC2) GO TO 15
 10     CONTINUE
C
 15     CONTINUE
C
       IF (SUM .GT. VAL) THEN
         NGVLT=NGVLT+1
         SUM=SUM-VAL
         MAXSTEP=MAX(MAXSTEP,SUM)
       ENDIF
       IF (NUM .EQ. 0) GO TO 30
       IF (NUM .EQ. 1) GO TO 25
C
       CALL LOCSORT(NUM,LOC1,LOC2,LGMULT,GCOEF,GUBADD)
C
 25    CONTINUE
```

```
C
      DO 20 INDEX=LOC1,LOC2
        REC=GUBADD(INDEX)
        IF (GCOEF(REC) .GT. 0.0) THEN
          YFLOW(REC)=MIN(UPPER(REC),ADJVAL/GCOEF(REC))
          IF (ABS(YFLOW(REC)) .LE. 1E-15) YFLOW(REC)=0.0D0
          ADJVAL=ADJVAL-GCOEF(REC)*YFLOW(REC)
          IF (ABS(ADJVAL) .LE. 1E-15) ADJVAL=0.0D0
        ELSE
          YFLOW(REC)=UPPER(REC)-MIN(UPPER(REC),-ADJVAL/GCOEF(REC))
          IF (ABS(YFLOW(REC)) .LE. 1E-15) YFLOW(REC)=0.0D0
          ADJVAL=ADJVAL+GCOEF(REC)*(UPPER(REC)-YFLOW(REC))
          IF (ABS(ADJVAL) .LE. 1E-15) ADJVAL=0.0D0
        ENDIF
        LBND=LBND+LGMULT(INDEX)*YFLOW(REC)
        LBSUBG=YFLOW(REC)-GFLOW(REC)
C        CTEMP(REC)=-LBSTEP*LBSUBG
C        IF (ABS(CTEMP(REC)) .LE. 1E-15) CTEMP(REC)=0.0D0
        IF (FLGOPT .EQ. 1) THEN
          LGLB=LGMULT(INDEX)*LBSUBG
          IF (ABS(LGLB) .GT. 1E-15) FLGOPT=0
        ENDIF
        NORM=NORM+LBSUBG*LBSUBG
C        LGMULT(INDEX)=LGMULT(INDEX)-CTEMP(REC)
C        IF (ABS(LGMULT(INDEX)) .LE. 1E-15) LGMULT(INDEX)=0.0D0
 20   CONTINUE
 30   LOC2=LASTRC
      RETURN
      END
***** ******************************************************
      SUBROUTINE LOCSORT(N,INDEX1,INDEX2,B,A,LOC)
***** ******************************************************
C
C     The purpose of this routine is to keep track of the position
C     of an ordered array
C
C  SUBROUTINE ARGUMENTS
C
      INTEGER LOC(*),N,INDEX1,INDEX2
      DOUBLE PRECISION B(*),A(*)
C
C  LOCAL ARGUMENTS
C
      DOUBLE PRECISION BSTAR,RSTAR
      INTEGER ID1,ID2,L,L1,LOC1,N1,M
C
      ID1=INDEX1
      ID2=INDEX2
C
      N1=N
      L=1+N/2
 11   L=L-1
```

```
      RSTAR=B(L+ID1-1)
      LOC1=LOC(L+ID1-1)
      BSTAR=RSTAR/A(LOC1)
      GO TO 30
25    LOC1=LOC(ID2)
      RSTAR=B(ID2)
      BSTAR=RSTAR/A(LOC1)
      B(ID2)=B(ID1)
      LOC(ID2)=LOC(ID1)
29    N1=N1-1
      ID2=ID2-1
30    L1=L
31    M=2*L1
      IF (M-N1) 32,33,37
32    IF (B(M+ID1)/A(LOC(M+ID1)) .GE. B(M+ID1-1)/A(LOC(M+ID1-1))) M=M+1
33    IF (BSTAR .GE. B(M+ID1-1)/A(LOC(M+ID1-1)) ) GO TO 37
      B(L1+ID1-1)=B(M+ID1-1)
      LOC(L1+ID1-1)=LOC(M+ID1-1)
      L1=M
      GO TO 31
37    B(L1+ID1-1)=RSTAR
      LOC(L1+ID1-1)=LOC1
      IF (L .GT. 1) GO TO 11
      IF (N1 .GE. 2) GO TO 25
      RETURN
      END
C**** ***********************************************************
      SUBROUTINE UINIT(GARB,GFLOW,GUBADD,GVAL,GCOEF,NUMVRG,
     *                 REDGUB,UPPER,UTEMP,ZFLOW)
C**** ***********************************************************
C
C  THE PURPOSE OF THIS ROUTINE IS TO INITIALIZE THE
C  UPPER BOUND ALGORITHM
C
C
C
C  SUBROUTINE ARGUMENTS
C
      INTEGER GUBADD(*),NUMVRG(*),REDGUB(*)
      DOUBLE PRECISION GARB(*),GCOEF(*),
     *                 GFLOW(*),GVAL(*),UPPER(*),UTEMP(*),ZFLOW(*)
C
C  LOCAL VARIABLES
C
      INTEGER I,INDEX,LOC1,LOC2,NEXT,NUM,OPT,REC
      DOUBLE PRECISION ADJVAL,LBSUBG,SUM
C
      INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *        ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *              NGUBS,NODES,ROOT,EPSILON,BIG1
C
C  FIND OPTIMAL SOLUTION TO THE NETWORK
C  CONSTRUCT STARTING SOLUTION
C
      DO 10 I=1,ARCS
        ZFLOW(I)=UPPER(I)
        UTEMP(I)=UPPER(I)
        GARB(I)=0.0D0
 10   CONTINUE
C
C  SOLVE FOR THE INITIAL SCBVLP PROBLEM AND THE DA
C
      NEXT=0
      SUM=0
      LOC2=0
      DO 50 INDEX=1,NGUBS
        NEXT=NEXT+NUMVRG(INDEX)
        LOC1=LOC2+1
        LOC2=LOC2+NUMVRG(INDEX)
C
C  IF THE GUB CONSTRAINT IS REDUNDANT THEN LET THE FLOWS BE THE SAME
C  AS THE NETWORK FLOWS
C
        IF (REDGUB(INDEX) .EQ. 1) THEN
          DO 20 I=LOC1,LOC2
            REC=GUBADD(I)
            IF (GCOEF(REC) .GT. 0.0) THEN
              ZFLOW(REC)=UPPER(REC)
            ELSE
              ZFLOW(REC)=0.0D0
            ENDIF
            UTEMP(REC)=0.0D0
 20       CONTINUE
          GO TO 50
        ENDIF
        SUM=0.0D0
C
        DO 30 I=LOC1,LOC2
          REC=GUBADD(I)
          ZFLOW(REC)=GFLOW(REC)
          IF (GCOEF(REC) .GT. 0.0) THEN
            UTEMP(REC)=UPPER(REC)
          ELSE
            UTEMP(REC)=0.0D0
          ENDIF
          SUM=SUM+ZFLOW(REC)*GCOEF(REC)
 30     CONTINUE
C
```

```
C  IF THE FLOWS ARE NOT FEASIBLE FOR THE UPPER BOUND DA
C  PROJECT THEM ONTO A FEASIBLE REGION
C
        IF (ABS(SUM - GVAL(INDEX)) .GT. 1E-15) THEN
          NUM=2*NUMVRG(INDEX)
          CALL PROJOP(GARB,GVAL(INDEX),GUBADD,GCOEF,LOC1,LOC2,NUM,
     *              UPPER,UTEMP,ZFLOW)
        ELSE
          DO 40 I=LOC1,LOC2
            REC=GUBADD(I)
            UTEMP(REC)=ZFLOW(REC)-UTEMP(REC)
            IF (ABS(UTEMP(REC)) .LE. 1E-15) UTEMP(REC)=0.0D0
            IF (FLGOPT .EQ. 1) THEN
              IF (UTEMP(REC) .NE. 0.0) FLGOPT=0
            ENDIF
 40       CONTINUE
        ENDIF
C
 50   CONTINUE
C
      RETURN
      END
C**** ***************************************************************
      SUBROUTINE UBALG(ARCNAM,BASIS,CARD,COST,FLOW,FROM,FROMO,
     *              GCOEF,GFLOW,GUBADD,GVAL,
     *              LNOD,LOWER,NUMVRG,PI,PRED,REDGUB,STATUS,
     *              THD,TO,UPPER,UTEMP,ZFLOW,UBSUBG)
C**** ***************************************************************
C
C  THE PURPOSE OF THIS ROUTINE IS TO CALCULATE AN UPPER BOUND
C  FOR THE PROBLEM (NPG)
C
C
C  SUBROUTINE ARGUMENTS
C
      CHARACTER*8 ARCNAM(*)
      INTEGER BASIS(*),CARD(*),FROM(*),FROMO(*),GUBADD(*),
     1        LNOD(*),NUMVRG(*),PRED(*),REDGUB(*),
     1        STATUS(*),THD(*),TO(*)
      DOUBLE PRECISION COST(*),FLOW(*),GCOEF(*),GFLOW(*),GVAL(*),
     1              LOWER(*),PI(*),UBSUBG(*),UPPER(*),
     1              UTEMP(*),ZFLOW(*)
C
C  LOCAL VARIABLES
C
      INTEGER FRMNOD,I,INDEX,INDEX1,INDEX2,J,K,LOC,LOC1,LOC2,
     *        NUM,REC,FIRSTM,PASS
      DOUBLE PRECISION  NORM,UBSTEP,SUM,LTSTEP,LUBND
C
      INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
      INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *        ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *        NGUBS,NODES,ROOT,EPSILON,BIG1
      DOUBLE PRECISION LBND,LBNDPRE,UBND,UBNDPRE
      COMMON /BOUNDS/ LBND,LBNDPRE,UBND,UBNDPRE
      INTEGER FSITER
      COMMON /FEAS/ FSITER
      DOUBLE PRECISION ALPHA
      COMMON /ALP/ ALPHA

      IF (UBND .EQ. BIG1) THEN
        FIRSTM=1
        UBNDPRE=UBND
        LUBND=10.E10
      ENDIF
C
 10   CONTINUE
      FLGINF=0
C
      TCOST=0.0D0
C
C
      CALL REOPT(ARCNAM,BASIS,CARD,COST,FLOW,FROM,FROMO,GCOEF,GFLOW,
     *        LNOD,LOWER,PI,PRED,STATUS,THD,TO,UPPER,UTEMP,ZFLOW)
C
      UTEMP(ARTADD)=BIG1
C
      CALL PURNET(ARCNAM,BASIS,CARD,COST,FLOW,FROM,FROMO,GFLOW,
     1        LNOD,LOWER,PI,PRED,STATUS,THD,TO,UTEMP)
C
      UBND=TCOST
      IF (UBND .LT. UBNDPRE .AND. FLGINF .EQ. 0) THEN
        CALL UBSAV(BASIS,CARD,FLOW,GFLOW,LNOD,PRED,STATUS,THD,ZFLOW)
        UBNDPRE=UBND
        WRITE(6,*) 'UBND =  ',UBND
      ENDIF
C

 22   CONTINUE
C
C
       IF (FLGINF .EQ. 1 ) GO TO 25
C
       IF ((UBNDPRE-LBNDPRE) .LE. EPSILON*ABS(UBNDPRE)) THEN
         UBND=UBNDPRE
         FLGOPT=1
         WRITE(6,*) 'NEAR OPTIMALITY REACHED AT UPPER BOUND PROCEDURE'
         WRITE(6,*) 'SOLUTION IS WITHIN',100*EPSILON,'%OF OPTIMALITY'
         RETURN
      ENDIF
C
 25   CONTINUE
C
C  DETERMINE THE SUBGRADIENT
C
      FLGOPT=1
C
      NORM=0
      FRMNOD=1
 200  CONTINUE
        INDEX1=FROM(FRMNOD)
        INDEX2=FROM(FRMNOD+1)-1
        DO 20 K=INDEX1,INDEX2
          UBSUBG(K)=0.0D0
          J=TO(K)
          IF (GCOEF(K) .NE. 0.0) THEN
            UTEMP(K)=ZFLOW(K)
          ELSE
            UTEMP(K)=0.0D0
          ENDIF
          LOC2=0
          DO 45 INDEX=1,NGUBS
            LOC1=LOC2+1
            LOC2=LOC2+NUMVRG(INDEX)
            IF (REDGUB(INDEX) .EQ. 1) THEN
              DO 55 LOC=LOC1,LOC2
                REC=GUBADD(LOC)
                IF(K .EQ. REC) GO TO 20
 55           CONTINUE
            ENDIF
 45       CONTINUE
          IF ((STATUS(K) .EQ. 2 .AND. GCOEF(K) .GT. 0.0) .OR.
     *        (STATUS(K) .EQ. 1 .AND. GCOEF(K) .LT. 0.0)) THEN
            UBSUBG(K)=COST(K)+PI(FRMNOD)-PI(J)
            IF (ABS(UBSUBG(K)) .LE. 1E-15) UBSUBG(K)=0.0D0
            NORM=NORM+ABS(UBSUBG(K)*UBSUBG(K))
            IF (ABS(NORM) .LE. 1E-15) NORM=0.0D0
          ENDIF
 20     CONTINUE
        FRMNOD=FRMNOD+1
        IF (FRMNOD .LE. NODES) GO TO 200
C
       IF (NORM .EQ. 0.00) THEN
         WRITE(6,*) 'OPTIMALITY REACHED IN UPPER BOUND PROCEDURE'
         FLGOPT=1
         RETURN
       ENDIF
C
       UBSTEP=(UBND-LBND)/(2*NORM)
       IF (FLGINF .EQ. 1) THEN
         IF (UBND .LT. LUBND) UBSTEP=10*UBSTEP
        IF (UBND .GT .LUBND) UBSTEP=UBSTEP/10
      ENDIF
C
 35   CONTINUE
C     UBNDPRE=UBND
      UBNDPRE=MIN(UBND,UBNDPRE)
      LUBND=TCOST
C
      LOC2=0
      DO 30 INDEX=1,NGUBS
        LOC1=LOC2+1
        LOC2=LOC2+NUMVRG(INDEX)
        IF (REDGUB(INDEX) .EQ. 1) THEN
          DO 40 LOC=LOC1,LOC2
            REC=GUBADD(LOC)
            IF (GCOEF(REC) .GT. 0.0) THEN
              ZFLOW(REC)=UPPER(REC)
            ELSE
              ZFLOW(REC)=0.0D0
            ENDIF
            UTEMP(REC)=0.0D0
 40       CONTINUE
          GO TO 30
        ENDIF
        DO 50 LOC=LOC1,LOC2
          REC=GUBADD(LOC)
          ZFLOW(REC)=ZFLOW(REC)-UBSTEP*UBSUBG(REC)
          IF (ABS(ZFLOW(REC)) .LE. 1E-15) ZFLOW(REC)=0.0D0
 50     CONTINUE
 30   CONTINUE
C
C  FIND THE FEASIBLE POINTS
C
      LOC2=0
      DO 60 INDEX=1,NGUBS
        LOC1=LOC2+1
        LOC2=LOC2+NUMVRG(INDEX)
        NUM=2*NUMVRG(INDEX)
        IF (REDGUB(INDEX) .EQ. 1) GO TO 60
        CALL PROJOP(UBSUBG,GVAL(INDEX),GUBADD,GCOEF,LOC1,LOC2,
     *              NUM,UPPER,UTEMP,ZFLOW)
 60   CONTINUE
C
C
C  CHECK FOR INFEASIBILITY
C
      DO 70 I=1,ARCS
        IF (ABS(UTEMP(I)) .LE. 1E-15) UTEMP(I)=0.0D0
        IF(UTEMP(I) .NE. 0.00) GO TO 80
 70   CONTINUE
      IF (FLGINF .NE. 1) FLGOPT=0
      GO TO 90
 80  CONTINUE
     IF (FLGINF .EQ. 1) THEN
       FSITER=FSITER+1
       GO TO 10
     ENDIF
C
 90  CONTINUE
     RETURN
     END
C**** ************************************************************
     SUBROUTINE UBSAV (BASIS,CARD,FLOW,GFLOW,LNOD,PRED,
    1                 STATUS,THD,ZFLOW)
C**** ************************************************************
C
C  SUBROUTINE ARGUMENTS
C
     INTEGER BASIS(*),CARD(*),LNOD(*),PRED(*),STATUS(*),THD(*)
     DOUBLE PRECISION FLOW(*),GFLOW(*),ZFLOW(*)
C
C  LOCAL VARIABLES
C
     INTEGER I
C
     DOUBLE PRECISION TCOST,EPSILON,BIG1
     INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
    *        ROOT
     COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
    *              NGUBS,NODES,ROOT,EPSILON,BIG1
C
     REWIND 11
C
C  SAVE DATA STRUCTURES
C
     DO 10 I=1,NODES
       WRITE(11) PRED(I),THD(I),CARD(I),LNOD(I),BASIS(I),FLOW(I)
  10 CONTINUE
     WRITE(11) ROOT
C
C  SAVE ARC STATUS
C
     DO 20 I=1,ARCS
       WRITE(11) GFLOW(I),ZFLOW(I),STATUS(I)
  20 CONTINUE
     RETURN
     END
C
C
C**** ************************************************************
     SUBROUTINE PROJOP(BRKPNT,VAL,GUBADD,GCOEF,LOC1,LOC2,NUMBKS,
    1                 UPPER,TEMP,ZFLOW)
C**** ************************************************************
C
```

```
C  THE PURPOSE OF THIS ROUTINE IS TO PROJECT AN ALLOCATION ONTO
C  A FEASIBLE REGION
C
C
c
C  SUBROUTINE ARGUMENTS
C
     INTEGER GUBADD(*),LOC1,LOC2,NUMBKS
     DOUBLE PRECISION BRKPNT(NUMBKS),GCOEF(*),TEMP(*),UPPER(*),
    *               VAL,ZFLOW(*)
C
C  LOCAL VARIABLES
C
     INTEGER I,J,L1,M,R1,REC
     DOUBLE PRECISION L2,LAMDA,R2,STERM,SUM,UTERM
C
     INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
     COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
     INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
     COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
     DOUBLE PRECISION TCOST,EPSILON,BIG1
     INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
    *       ROOT
     COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
    *             NGUBS,NODES,ROOT,EPSILON,BIG1
C
C  SET THE BREAKPOINTS
C
     J=1
     L2=0.0D0
     R2=0.0D0
     DO 10 I=LOC1,LOC2
        REC=GUBADD(I)
        BRKPNT(J)=2*(ZFLOW(REC)-UPPER(REC))/GCOEF(REC)
        IF (ABS(BRKPNT(J)) .LT. 1E-14) BRKPNT(J)=0.0D0
        J=J+1
        BRKPNT(J)=2*ZFLOW(REC)/GCOEF(REC)
        IF (ABS(BRKPNT(J)) .LT. 1E-14) BRKPNT(J)=0.0D0
        J=J+1
        IF (GCOEF(REC) .GT. 0.0) THEN
           L2=L2+UPPER(REC)*GCOEF(REC)
        ELSE
           R2=R2+UPPER(REC)*GCOEF(REC)
        ENDIF
 10  CONTINUE
C
     CALL HPSORT(NUMBKS,BRKPNT)
C
     L1=1
     IF (ABS(L2) .LE. 1E-14) L2=0.0D0
     IF (ABS(R2) .LE. 1E-14) R2=0.0D0
     R1=NUMBKS
```

```
20  IF ((R1-L1) .EQ. 1) THEN
      LAMDA=BRKPNT(L1)+((BRKPNT(R1)-BRKPNT(L1))*
   *                  (VAL-L2)/(R2-L2))
      IF (ABS(LAMDA) .LT. 1E-14) LAMDA=0.0D0
      GO TO 40
    ENDIF
    M=(L1+R1)/2
    SUM=0.0D0
    DO 30 I=LOC1,LOC2
      REC=GUBADD(I)
      STERM=GCOEF(REC)*BRKPNT(M)/2
      STERM=(ZFLOW(REC)-STERM)*GCOEF(REC)
      IF (ABS(STERM) .LT. 1E-14) STERM=0.0D0
      IF (GCOEF(REC) .GT. 0.0) THEN
        SUM=SUM+MAX(MIN(STERM,GCOEF(REC)*UPPER(REC)),0.0D0)
      ELSE
        SUM=SUM+MIN(MAX(STERM,GCOEF(REC)*UPPER(REC)),0.0D0)
      ENDIF
      IF (ABS(SUM) .LT. 1E-14) SUM=0.0D0
30  CONTINUE
    IF ( SUM .EQ. VAL) THEN
      LAMDA=BRKPNT(M)
      GO TO 40
    ENDIF
    IF (SUM .GT. VAL) THEN
      L1=M
      L2=SUM
      GO TO 20
    ELSE
      R1=M
      R2=SUM
      GO TO 20
    ENDIF
40  CONTINUE
    DO 50 I=LOC1,LOC2
      REC=GUBADD(I)
      UTERM=2*ZFLOW(REC)/GCOEF(REC)
      STERM=2*(ZFLOW(REC)-UPPER(REC))/GCOEF(REC)
      IF (ABS(UTERM) .LT. 1E-14) UTERM=0.0D0
      IF (ABS(STERM) .LT. 1E-14) STERM=0.0D0
      IF (GCOEF(REC) .GT. 0.0) THEN
        IF (LAMDA .LE. STERM) THEN
          ZFLOW(REC)=UPPER(REC)
        ELSE
          IF (LAMDA .LE. UTERM) THEN
            ZFLOW(REC)=ZFLOW(REC)-GCOEF(REC)*LAMDA/2
            IF (ABS(ZFLOW(REC)) .LE. 1E-14) ZFLOW(REC)=0.0D0
          ELSE
            ZFLOW(REC)=0.0D0
          ENDIF
        ENDIF
        TEMP(REC)=ZFLOW(REC)-TEMP(REC)
```

```
         IF (ABS(TEMP(REC)) .LE. 1E-14) TEMP(REC)=0.0D0
         IF (FLGOPT .EQ. 1 ) THEN
           IF (TEMP(REC) .NE. 0.0) FLGOPT=0
         ENDIF
       ELSE
         IF (LAMDA .LE. UTERM) THEN
           ZFLOW(REC)=0.0D0
         ELSE
           IF (LAMDA .GT. STERM) THEN
             ZFLOW(REC)=UPPER(REC)
           ELSE
             ZFLOW(REC)=ZFLOW(REC)-GCOEF(REC)*LAMDA/2
             IF (ABS(ZFLOW(REC)) .LE. 1E-14) ZFLOW(REC)=0.0D0
           ENDIF
         ENDIF
         TEMP(REC)=ZFLOW(REC)-TEMP(REC)
         IF (ABS(TEMP(REC)) .LT. 1E-14) TEMP(REC)=0.0D0
         IF (FLGOPT .EQ. 1) THEN
           IF (TEMP(REC) .NE. 0.0) FLGOPT=0
         ENDIF
       ENDIF
C
 50   CONTINUE
C
C
C  THE ALLOCATION IS FEASIBLE
C
      RETURN
      END
***** ********************************************************
      SUBROUTINE HPSORT(N,B)
***** ********************************************************
C
C     The purpose of this routine is to sort a linear array
C     into nondecreasing order.
C
C
C  SUBROUTINE ARGUMENTS
C
      DOUBLE PRECISION B(N)
C
C
C  LOCAL ARGUMENTS
C
      DOUBLE PRECISION BSTAR
      INTEGER L,L1,N1,M
C
      N1=N
      L=1+N/2
 11    L=L-1
      BSTAR=B(L)
      GO TO 30
```

```
25  BSTAR=B(N1)
    B(N1)=B(1)
29  N1=N1-1
30  L1=L
31  M=2*L1
    IF (M-N1) 32,33,37
32  IF (B(M+1) .GE. B(M)) M=M+1
33  IF (BSTAR .GE. B(M)) GO TO 37
    B(L1)=B(M)
    L1=M
    GO TO 31
37  B(L1)=BSTAR
    IF (L .GT. 1) GO TO 11
    IF (N1 .GE. 2) GO TO 25
    RETURN
    END
C**** **************************************************************
    SUBROUTINE REOPT(ARCNAM,BASIS,CARD,COST,FLOW,FROM,FROMO,
   1              GCOEF,GFLOW,LNOD,LOWER,PI,PRED,STATUS,THD,
   1              TO,UPPER,UTEMP,ZFLOW)
C**** **************************************************************
C
C  THE PURPOSE OF THIS ROUTINE IS TO AN INITIAL FEASIBLE
C  STARTING POINT FOR THE NETWORK WITH CAPACITY CHANGE
C
C
C  SUBROUTINE ARGUMENTS
C
    CHARACTER*8 ARCNAM(*)
    INTEGER BASIS(*),CARD(*),FROM(*),FROMO(*),
   1       LNOD(*),PRED(*),
   1       STATUS(*),THD(*),TO(*)
    DOUBLE PRECISION COST(*),FLOW(*),GCOEF(*),GFLOW(*),LOWER(*),
   1              PI(*),UPPER(*),UTEMP(*),ZFLOW(*)
C
C  LOCAL VARIABLES
C
    INTEGER DIR,FRMNOD,IJ,INDEX1,INDEX2,J,K,PREDJ,RTHD,TOJ,UP
C
    INTEGER FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
    COMMON /FLAG/ FLGEND,FLGERR,FLGINF,FLGITR,FLGOPT,NUMREC
    INTEGER ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
    COMMON /ITER/ ITRBTB,ITRDEG,ITRMAX,ITROBJ,ITROUT,ITRREG,ITRTOT
    DOUBLE PRECISION TCOST,EPSILON,BIG1
    INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
   *       ROOT
    COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
   *            NGUBS,NODES,ROOT,EPSILON,BIG1
C
C  CONSTRUCT THE VECTOR OF REDUCED REQUIREMENTS
C  PROCEDURE X2D
C
```

```
      FLOW(ROOT)=0.0D0
      J=THD(ROOT)
 10   INDEX1=FROM(J)
      INDEX2=FROM(J+1)-1
      PREDJ=PRED(J)
C
C  IF STILL AT PHASE I
C
      IF (BASIS(J) .EQ. -ARTADD) THEN
        FLOW(PREDJ)=FLOW(PREDJ)+FLOW(J)
        FLOW(J)=-FLOW(J)
        IF (ABS(FLOW(PREDJ)) .LE. .1E-15) FLOW(PREDJ)=0.0D0
        J=THD(J)
        IF (J .EQ. ROOT) GO TO 30
        GO TO 10
      ENDIF
C
      DO 20 K=INDEX1,INDEX2
        TOJ=TO(K)
        IF (TOJ .EQ. PREDJ ) THEN
          IF (STATUS(K) .NE. 0 ) GO TO 20
          FLOW(PREDJ)=FLOW(PREDJ)+FLOW(J)
          IF (ABS(FLOW(PREDJ)) .LE. .1E-15) FLOW(PREDJ)=0.0D0
          FLOW(J)=-FLOW(J)
          IF (ABS(FLOW(J)) .LE. .1E-15) FLOW(J)=0.0D0
          J=THD(J)
          IF (J .EQ. ROOT) GO TO 30
          GO TO 10
        ENDIF
 20   CONTINUE
      FLOW(PREDJ)=FLOW(PREDJ)-FLOW(J)
      IF (ABS(FLOW(PREDJ)) .LE. .1E-15) FLOW(PREDJ)=0.0D0
      J=THD(J)
      IF (J .EQ. ROOT) GO TO 30
      GO TO 10
C
 30   CONTINUE
C
C  ADJUST THE REDUCED REQUIREMENTS FOR THE NONBASIS ARCS
C
      FRMNOD=1
 40   CONTINUE
        INDEX1=FROM(FRMNOD)
        INDEX2=FROM(FRMNOD+1)-1
        DO 50 K=INDEX1,INDEX2
          J=TO(K)
          IF (GCOEF(K) .EQ. 0.0 ) THEN
            UTEMP(K)=UPPER(K)
            GO TO 50
          ENDIF
          IF (STATUS(K) .EQ. 2 .AND. GCOEF(K) .GT. 0.00) THEN
            IF (UTEMP(K) .NE. 0.0) THEN
```

```
              FLOW(FRMNOD)=FLOW(FRMNOD)+UTEMP(K)
              IF (ABS(FLOW(FRMNOD)) .LE. .1E-15) FLOW(FRMNOD)=0.0D0
              FLOW(J)=FLOW(J)-UTEMP(K)
              IF (ABS(FLOW(J)) .LE. .1E-15) FLOW(J)=0.0D0
            ENDIF
            UTEMP(K)=ZFLOW(K)
            GO TO 50
          ENDIF
          IF (STATUS(K) .EQ. 1 .AND. GCOEF(K) .LT. 0.00) THEN
            IF (UTEMP(K) .NE. 0.00) THEN
              FLOW(FRMNOD)=FLOW(FRMNOD)+UTEMP(K)
              FLOW(J)=FLOW(J)-UTEMP(K)
              IF (ABS(FLOW(FRMNOD)) .LE. .1E-15) FLOW(FRMNOD)=0.0D0
              IF (ABS(FLOW(J)) .LE. .1E-15) FLOW(J)=0.0D0
            ENDIF
            UTEMP(K)=UPPER(K)-ZFLOW(K)
            IF (ABS(UTEMP(K)) .LE. .1E-15) UTEMP(K)=0.0D0
            GO TO 50
          ENDIF
          IF (STATUS(K) .EQ. 2 .AND. GCOEF(K) .LT. 0.00) THEN
            UTEMP(K)=UPPER(K)-ZFLOW(K)
            GO TO 50
          ENDIF
          IF (GCOEF(K) .GT. 0.0) THEN
            UTEMP(K)=ZFLOW(K)
          ENDIF
 50     CONTINUE
        FRMNOD=FRMNOD+1
        IF (FRMNOD .LE. NODES) GO TO 40
C
C
C  CONSTRUCT A SET OF BASIC FLOWS FROM THE SET OF REDUCED REQUIREMENTS
C  PROCEDURE D2X
C
      J=LNOD(ROOT)
 60   CONTINUE
        PREDJ=PRED(J)
        FLOW(PREDJ)=FLOW(PREDJ)+FLOW(J)
        IF (ABS(FLOW(PREDJ)) .LE. .1E-15) FLOW(PREDJ)=0.0D0
        IF (BASIS(J) .EQ. -ARTADD) THEN
          FLOW(J)=-FLOW(J)
          GO TO 90
        ENDIF
        INDEX1=FROM(J)
        INDEX2=FROM(J+1)-1
        DO 80 K=INDEX1,INDEX2
          TOJ=TO(K)
          IF (STATUS(K) .EQ. 0 .AND. TOJ .EQ. PREDJ) THEN
            FLOW(J)=-FLOW(J)
            GO TO 90
          ENDIF
 80     CONTINUE
 90      CONTINUE
C
C    FIND REVERSE THREAD OF J
C
       RTHD=PRED(J)
 100     IF (THD(RTHD) .NE. J) THEN
          RTHD=LNOD(THD(RTHD))
          GO TO 100
        ELSE
          J=RTHD
          IF (PRED(J) .EQ. 0) GO TO 105
          GO TO 60
        ENDIF
C
C  CHECK THE BOUNDS FOR THE BASIC ARCS
C
C  CALCULATE THE OBJECTIVE FUNCTION VALUE
C
 105  CONTINUE
C
C  IF STILL AT PHASE I
C
     DO 45 I=1,NODES
       IF (IABS(BASIS(I)) .EQ. ARTADD) THEN
         IF (FLOW(I) .LT. 0.0) THEN
           FLOW(I)=-FLOW(I)
           BASIS(I)=-BASIS(I)
         ENDIF
         TCOST=TCOST+FLOW(I)*COST(ARTADD)
       ENDIF
 45   CONTINUE
C
     FRMNOD=1
 110  CONTINUE
       INDEX1=FROM(FRMNOD)
       INDEX2=FROM(FRMNOD+1)-1
       DO 120 K=INDEX1,INDEX2
         J=TO(K)
         IF (J .EQ. PRED(FRMNOD)) THEN
           IJ=FRMNOD
           DIR=-1
         ELSE
           IJ=J
           DIR=+1
         ENDIF
         IF (STATUS(K) .EQ. 0) THEN
           IF (GCOEF(K) .GE. 0.00 .AND. FLOW(IJ) .GT. UTEMP(K)) THEN
             STATUS(K)=2
             GFLOW(K)=UTEMP(K)
             BASIS(IJ)=DIR*ARTADD
             STATUS(ARTADD)=0
             FLOW(IJ)=FLOW(IJ)-UTEMP(K)
          IF (ABS(FLOW(IJ)) .LE. .1E-15) FLOW(IJ)=0.0D0
          TCOST=TCOST+UTEMP(K)*COST(K)+FLOW(IJ)*COST(ARTADD)
          GO TO 120
        ENDIF
        IF (GCOEF(K) .GE. 0.00 .AND. FLOW(IJ) .LT. 0.0) THEN
          STATUS(K)=1
          GFLOW(K)=0.0D0
          BASIS(IJ)=-DIR*ARTADD
          FLOW(IJ)=-FLOW(IJ)
          TCOST=TCOST+FLOW(IJ)*COST(ARTADD)
          GO TO 120
        ENDIF
        UP=UPPER(K)-ZFLOW(K)+UTEMP(K)
        IF (ABS(UP) .LE. .1E-15) UP=0.0D0
        IF (ABS(FLOW(IJ)-UP) .LE. .1E-15) FLOW(IJ)=UP
        IF (GCOEF(K) .LT. 0.00 .AND. FLOW(IJ) .GT. UP) THEN
          STATUS(K)=2
          GFLOW(K)=UPPER(K)-ZFLOW(K)
          BASIS(IJ)=DIR*ARTADD
          FLOW(IJ)=FLOW(IJ)-UP
          UTEMP(K)=UPPER(K)-ZFLOW(K)
          TCOST=TCOST+UPPER(K)*COST(K)+FLOW(IJ)*COST(ARTADD)
          GO TO 120
        ENDIF
        IF (GCOEF(K) .LT. 0.00 .AND. FLOW(IJ) .LT. UTEMP(K)) THEN
          STATUS(K)=1
          GFLOW(K)=0.0D0
          BASIS(IJ)=-DIR*ARTADD
          FLOW(IJ)=UTEMP(K)-FLOW(IJ)
          UTEMP(K)=UPPER(K)-ZFLOW(K)
          TCOST=TCOST+FLOW(IJ)*COST(ARTADD)+ZFLOW(K)*COST(K)
          GO TO 120
        ENDIF
        IF (GCOEF(K) .GE. 0.0) THEN
          GFLOW(K)=FLOW(IJ)
        ELSE
          TCOST=TCOST+ZFLOW(K)*COST(K)
          FLOW(IJ)=FLOW(IJ)-UTEMP(K)
          IF (ABS(FLOW(IJ)) .LE. .1E-15) FLOW(IJ)=0.0D0
          GFLOW(K)=FLOW(IJ)
          UTEMP(K)=UPPER(K)-ZFLOW(K)
        ENDIF
        TCOST=TCOST+FLOW(IJ)*COST(K)
      ELSE
        IF (STATUS(K) .EQ. 2) THEN
          GFLOW(K)=UTEMP(K)
          TCOST=TCOST+UTEMP(K)*COST(K)
        ENDIF
        IF (GCOEF(K) .LT. 0.00) THEN
            TCOST=TCOST+ZFLOW(K)*COST(K)
        ENDIF
      ENDIF
```

```
120    CONTINUE
       FRMNOD=FRMNOD+1
C
       IF (FRMNOD .LE. NODES) GO TO 110
C
      RETURN
      END
C**** ***********************************************************
      SUBROUTINE LBSAV (BASIS,CARD,FLOW,GFLOW,LNOD,PRED,
     1                 STATUS,THD,YFLOW)
C**** ***********************************************************
C
C  SUBROUTINE ARGUMENTS
C
      INTEGER BASIS(*),CARD(*),LNOD(*),PRED(*),STATUS(*),THD(*)
      DOUBLE PRECISION FLOW(*),GFLOW(*),YFLOW(*)
C
C  LOCAL VARIABLES
C
      INTEGER I
C
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *      ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *           NGUBS,NODES,ROOT,EPSILON,BIG1
C
      REWIND 13
C
C  SAVE DATA STRUCTURES
C
      DO 10 I=1,NODES
        WRITE(13) PRED(I),THD(I),CARD(I),LNOD(I),BASIS(I),FLOW(I)
   10 CONTINUE
      WRITE(13) ROOT
C
C  SAVE ARC STATUS
C
      DO 20 I=1,ARCS
        WRITE(13) GFLOW(I),YFLOW(I),STATUS(I)
   20 CONTINUE
      RETURN
      END
C
C
C
C**** ***********************************************************
      SUBROUTINE LBRED (BASIS,CARD,FLOW,GFLOW,LNOD,PRED,
     1                 STATUS,THD,YFLOW)
C**** ***********************************************************
C
C  SUBROUTINE ARGUMENTS
```

```
C
      INTEGER BASIS(*),CARD(*),LNOD(*),PRED(*),STATUS(*),THD(*)
      DOUBLE PRECISION FLOW(*),GFLOW(*),YFLOW(*)
C
C  LOCAL VARIABLES
C
      INTEGER I
C
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *        ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *              NGUBS,NODES,ROOT,EPSILON,BIG1
C
C  READ DATA STRUCTURES
C
      REWIND 13
      DO 10 I=1,NODES
        READ(13) PRED(I),THD(I),CARD(I),LNOD(I),BASIS(I),FLOW(I)
   10 CONTINUE
      READ(13) ROOT
C
C  READ ARC STATUS
C
      DO 20 I=1,ARCS
        READ(13) GFLOW(I),YFLOW(I),STATUS(I)
   20 CONTINUE
      RETURN
      END
C
C
C**** ************************************************************
      SUBROUTINE UBRED (BASIS,CARD,FLOW,GFLOW,LNOD,PRED,
     1                  STATUS,THD,ZFLOW)
C**** ************************************************************
C
C  SUBROUTINE ARGUMENTS
C
      INTEGER BASIS(*),CARD(*),LNOD(*),PRED(*),STATUS(*),THD(*)
      DOUBLE PRECISION FLOW(*),GFLOW(*),ZFLOW(*)
C
C  LOCAL VARIABLES
C
      INTEGER I
C
      DOUBLE PRECISION TCOST,EPSILON,BIG1
      INTEGER ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,NGUBS,NODES,
     *        ROOT
      COMMON /PARM/ TCOST,ARCS,ARTADD,BIG,GARCS,MAXARC,MAXGUB,MAXNOD,
     *              NGUBS,NODES,ROOT,EPSILON,BIG1
C
C  READ DATA STRUCTURES
```

```
C
       REWIND 11
      DO 10 I=1,NODES
        READ(11) PRED(I),THD(I),CARD(I),LNOD(I),BASIS(I),FLOW(I)
   10 CONTINUE
      READ(11) ROOT
C
C  READ ARC STATUS
C
      DO 20 I=1,ARCS
        READ(11) GFLOW(I),ZFLOW(I),STATUS(I)
   20 CONTINUE
      RETURN
      END
```