OPERATIONS RESEARCH
A GENERALIZED UPPER BOUNDING ALGORITHM
FOR MULTICOMMODITY NETWORK FLOW PROBLEMS

James K. Hartman
Leon S. Lasdon
Technical Memorandum No. 193
June 1970

DEPARTMENT OF MANAGEMENT
CASE WESTERN RESERVE UNIVERSITY
A GENERALIZED UPPER BOUNDING ALGORITHM
FOR MULTICOMMODITY NETWORK FLOW PROBLEMS

James K. Hartman
Leon S. Lasdon

Technical Memorandum No. 193
June 1970

This document may be reproduced in whole or in part for any non-commercial purpose of the United States Government. Its preparation was supported in part by funds allocated to the Department of Operations Research, School of Management, Case Western Reserve University under contract DAHC 15-8-C-0007 (Project Themis) with the U.S. Army Research Office, Durham Army Materiel Command.
A GENERALIZED UPPER BOUNDING ALGORITHM FOR
MULTICOMMODITY NETWORK FLOW PROBLEMS

ABSTRACT

An algorithm for solving min cost or max flow multicommodity flow problems is described. It is a specialization of the simplex method, which takes advantage of the special structure of the multicommodity problem. The only non-graph or non-additive operations in a cycle involve the inverse of a working basis, whose dimension is the number of currently saturated arcs. Efficient relations for updating this inverse are derived.
An algorithm for solving min cost or max flow multicommodity flow problems is described. It is a specialization of the simplex method, which takes advantage of the special structure of the multicommodity problem. The only non-graph or non-additive operations in a cycle involve the inverse of a working basis, whose dimension is the number of currently saturated arcs. Efficient relations for updating this inverse are derived.
linear programming
multicommodity network flows
graph theory
compact inverse method
optimization
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td></td>
<td>TABLE OF CONTENTS</td>
<td>iii</td>
</tr>
<tr>
<td>I</td>
<td>SECTION I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II</td>
<td>SECTION II. PROBLEM STATEMENT</td>
<td>3</td>
</tr>
<tr>
<td>III</td>
<td>SECTION III. THE GENERALIZED UPPER BOUNDING ALGORITHM FOR BLOCK ANGULAR PROBLEMS</td>
<td>6</td>
</tr>
<tr>
<td>IV</td>
<td>SECTION IV. WORKING BASIS STRUCTURE FOR THE MULTICOMMODITY PROBLEM</td>
<td>14</td>
</tr>
<tr>
<td>V</td>
<td>SECTION V. THE ALGORITHM FOR THE MULTICOMMODITY PROBLEM</td>
<td>21</td>
</tr>
<tr>
<td>VI</td>
<td>SECTION VI. UPDATING FORMULAS</td>
<td>26</td>
</tr>
<tr>
<td>VII</td>
<td>SECTION VII. MAX FLOW PROBLEMS AND UNDIRECTED ARCS</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>BIBLIOGRAPHY</td>
<td>35</td>
</tr>
</tbody>
</table>
SECTION I
INTRODUCTION

Multicommodity network flow problems require the selection of optimal flow patterns for each of a number of distinguishable commodities in a capacitated network. The objective can be either to minimize the cost of achieving given flows, or to maximize the sum of the flows. When a node-arc formulation is used, these problems may be written as block diagonal linear programs with coupling rows. In this paper a compact inverse version of the simplex method for solving multicommodity problems is described. By using the special structure of any basis matrix, the simplex method can be performed while maintaining the inverse of a working basis whose dimension is only the number of currently saturated arcs. Aside from multiplication by this inverse, all other simplex computations are performed using addition or graph theoretic operations. The algorithm is a specialization of the generalized upper bounding method for block angular problems [4], [5]. It is similar to Saigal's method [6] which was derived using an arc-circuit formulation.

The approach taken here has two important advantages. First, it presents the algorithm as a direct specialization of a well known general procedure for linear programs. Second, in Saigal's work, at each iteration, several systems of linear equations must be solved and no procedures are given for updating the matrix inverses associated with these equations. Here we show that the only non graph theoretic or non additive operations
required are multiplication by and updating of the working basis inverse. Hence all the nonunimodular aspects of the problem are condensed into a single matrix which appears to be of minimal size. Efficient relations for updating the working basis inverse are derived here as specializations of those in the generalized upper bounding method.
SECTION II

PROBLEM STATEMENT

Consider a network which has nodes 1, 2, ..., ..., N and directed arcs $a_1, a_2, ..., ..., a_M$. The case with undirected arcs will be considered later. Arcs $a_1, ..., a_k (1 \leq k \leq M)$ have capacities $b_1, ..., b_k$. Let there be $K$ commodities and define $x_{km}$ as the flow of commodity $k$ in arc $a_m$. Each commodity $k$ has associated with it a source node $s_k$ and a sink node $t_k$. The constraints are

1. flows are nonnegative

$$x_{km} \geq 0 \quad \text{(all } k \text{ and } m) \quad (1)$$

2. capacity restrictions on arc $a_m$

$$\sum_{k=1}^{K} x_{km} \leq b_m \quad (1 \leq m \leq K) \quad (2)$$

3. flow conservation for commodity $k$ at node $n$.

$$\sum_{a_m \in A_n} x_{km} - \sum_{a_m \in B_n} x_{km} = \begin{cases} -f_k & \text{if } n = s_k \\ 0 & \text{if } n \neq s_k, n \neq t_k \\ +f_k & \text{if } n = t_k \end{cases} \quad (3)$$

where $f_k$ is the amount of flow of commodity $k$ in the network, $B_n$ is the set of arcs terminating at node $n$, and $A_n$ is the set of arcs originating at node $n$. 
For the min-cost problem, the flows $f_k$ are given and the objective is to minimize total cost

$$\min Z = \sum_{k,m} c_{km} x_{km}$$  \hspace{1cm} (4)

The max-flow problem views the $f_k$ as variables and has objective

$$\max \sum_k f_k$$  \hspace{1cm} (5)

Since the max flow problem is a special case of the min cost problem, we will use (4) as the objective.

In matrix form (1) - (4) becomes

minimize $Z$

subject to

$$Z S_1 \cdots S \times_{11} \cdots x_{1M} \times_{21} \cdots x_{2M} \times_{K1} \cdots x_{KM}$$

\[
\begin{array}{cccccccccc}
1 & 0 & -C_{11} & -C_{1M} & -C_{21} & -C_{2M} & \cdots & -C_{K1} & -C_{KM} & = 0 \\
0 & I & I \times I & 0 & I \times I & 0 & \cdots & I \times I & 0 & = b_1 \\
& F & & & & & & & & = d_1 \\
& & & & & & & & = d_2 \\
& & & & & & & & = d_K \\
\end{array}
\]
In the above linear program there are \( k + 1 \) coupling rows and \( K \) identical diagonal blocks. The matrix \( F \) is the node-arc incidence matrix of the network with the last row deleted. Hence \( F \) is \( N-1 \times M \) and has rank \( N-1 \). The variables \( s_i \) are nonnegative slacks for the capacity constraints, and the vector \( d_k \) has \(-f_k\) in position \( s_k\), \(+f_k\) in position \( t_k\), and zeroes elsewhere.
SECTION III
THE GENERALIZED UPPER BOUNDING ALGORITHM FOR BLOCK ANGULAR PROBLEMS

Consider the general block diagonal problem with coupling rows.

\[ \text{minimize } Z \]
\[ \text{subject to } A_0 x_0 + A_1 x_1 + \ldots + A_K x_K = b \] (7)
\[ D_1 x_1 = b_1 \]
\[ \vdots \]
\[ D_K x_K = b_K \]
\[ x_1 \geq 0 \]

where each \( A_i \) is an \( m_i \times n_i \) matrix, each \( D_i \) is \( m_i \times n_i \), and \( Z \) is the first component of \( x_0 \). We assume throughout that the constraint matrix of (7) has full rank. Hence each \( D_i \) has rank \( m_i \). The method is based on the following result proved in [5].

**Theorem 1** Any basis matrix \( B \) for (7) can partitioned to have the form:
where each $B_i$ is an $m_i \times m_i$ nonsingular submatrix of $D_i$.

Using the fact that the $B_i$ are nonsingular we develop a transformation matrix $V$ such that $B V$ is block triangular. The simplest such $V$ has the form

$$
V = - \begin{bmatrix}
B_1^{-1} & 0 \\
B_2^{-1} & \ddots \\
& \ddots & 0 \\
0 & \cdots & B_K^{-1}
\end{bmatrix}
$$

$$
= \begin{bmatrix}
I_1 & 0 \\
V & I_2
\end{bmatrix}
$$
Then

\[
\begin{bmatrix}
B & A_{11} & A_{21} & \cdots & A_{K1} \\
0 & B_1 & B_2 & & B_K
\end{bmatrix}
\]

is the block triangularized basis matrix. The submatrix \( B \), given by

\[
B = \mathbf{\hat{B}} + [A_{11} \ A_{21} \ \cdots \ A_{K1}] \ V
\]

is called the working basis. Since \( B \mathbf{I} \) is nonsingular, \( B \) is also nonsingular.

We now examine how the operations of the revised simplex method may be carried out using quantities associated with the working basis. These operations require only that two sets of linear equations, with coefficient matrices \( B' \) and \( B \), be solved (one for the pricing vector, the other for the transform of the entering vector). Triangularizing \( \mathbf{\hat{B}} \) greatly simplifies their solution.

**Determining the Simplex Multipliers.** Here the vector of simplex multipliers \( \pi = (\pi_0, \pi_1, \ldots, \pi_K) \) is to be computed. These satisfy

\[
\pi B = c_0
\]

or, since only \( z \) has a nonzero objective coefficient, and its column is the leftmost column of \( \mathbf{\hat{B}} \),

\[
\pi \mathbf{\hat{B}} = (1, 0, \ldots, 0)
\]
Multiplying on the right by $T$,

$$\pi (B^T I) = (1, 0, \ldots, 0) I = (1, 0, \ldots, 0)$$ (14)

Since $B^T I$ is triangular, these are easily solved yielding

$$\pi_0 = \text{first row of } B^{-1}$$ (15)

$$\pi_i = -\pi_0 A_{i1} B_{11}^{-1} \quad (i = 1, \ldots, K)$$ (16)

Thus if $B^{-1}$ and $B_{11}^{-1}$ are maintained, the vectors $\pi_0$ and $\pi_i$ are easily computed.

Determining the Column to Enter the Basis. This is done as in the revised simplex method by computing

$$\bar{c}_j = -\pi P_j$$ (17)

for each nonbasic column $P_j$. Note that only 2 partitions of any column $P_j$ are nonzero. If

$$\min \bar{c}_j = \bar{c}_s \geq 0$$

then the current solution is optimal. Otherwise $P_s$ enters the basis. Suppose $P_s$ is a column from the $\sigma$th block so that

$$P_s = [P_{s0}, 0 \ldots 0, P_{s\sigma}, 0 \ldots 0]^T.$$
Finding $\hat{\mathbf{p}}_g = \mathbf{B}^{-1} \mathbf{p}_g$

Here we must solve the linear system

$$\mathbf{B} \hat{\mathbf{p}}_g = \mathbf{p}_g \quad (18)$$

Let

$$\hat{\mathbf{p}}_g = T \mathbf{Z} \quad (19)$$

Substituting (19) into (18) gives

$$(\mathbf{B} I) \mathbf{Z} = \mathbf{p}_g \quad (20)$$

which can be easily solved for $\mathbf{Z} = (Z_0, Z_1, ..., Z_K)'$ since $\mathbf{B} I$ is block triangular:

$$Z_1 = 0 \quad i = 1, ..., K ; i \neq \sigma \quad (21)$$

$$Z_\sigma = \mathbf{B}^{-1}_\sigma \mathbf{p}_{g \sigma} \quad (22)$$

$$Z_0 = \mathbf{B}^{-1}_0 \left( \mathbf{p}_{g0} - \mathbf{A}_0 Z_\sigma \right) \quad (23)$$

Thus $Z_\sigma$ and $Z_0$ can be computed if $\mathbf{B}^{-1}$ and $\mathbf{B}^{-1}_\sigma$ are known.

Then $\hat{\mathbf{p}}_g = (\hat{\mathbf{p}}_{g0}, \hat{\mathbf{p}}_{g1}, ..., \hat{\mathbf{p}}_{gK})'$ is computed from (19) giving

$$\hat{\mathbf{p}}_{g0} = Z_0 \quad (24)$$

$$\hat{\mathbf{p}}_{g1} = V_1 Z_0 \quad i = 1, ..., K ; i \neq \sigma \quad (25)$$

$$\hat{\mathbf{p}}_{g\sigma} = V_\sigma Z_0 + Z_\sigma \quad (26)$$

where $V_i$ is the $i$th partition of $V$. 
Choosing the Column to leave the Basis. This is done according to the standard simplex formulas. If the solution is not unbounded, then column \( r \) of \( B \), \( P_j \), leaves the basis. Assume that this column is from the \( p \)th block of (7). Since computing the new values of the basic variables also proceeds as in the revised simplex method, we now consider updating the matrices \( B^{-1} \), \( B_1^{-1} \) and any other quantities needed for the next iteration.

Updating Formulas. There are two cases which can occur. Only the results are stated here; derivations may be found in [5].

Case 1  The leaving column is non-key. Here the entering column can directly replace the one leaving without destroying the block diagonal structure of \( B \). Then none of the \( B_1^{-1} \) change, and \( B^{-1} \) is transformed to \( *B^{-1} \) by a pivot operation.

\[
*B^{-1} = E B^{-1}
\]

where \( E \) is an \( m_0 \times m_0 \) elementary column matrix equal to the identity, except column \( r \). Let \( \tilde{a}_{1s} \) be the \( i \)th component of \( \tilde{P}_s \). Then column \( r \) of \( E \) has components

\[
\eta_1 = \begin{cases} 
- \frac{\tilde{a}_{is}}{\tilde{a}_{1s}} & i = l, \ldots, m_0; \ i \neq r \\
1/\tilde{a}_{1s} & i = r 
\end{cases}
\]
Case 2  The leaving column is a key column. Here when column  
$P_{j_T}$  leaves the basis, the block  $B_P$  will have only  $m_P - 1$ columns.  
Hence it is necessary to find another basic column from the  $P^{th}$ 
block to restore the basis structure. There are two subcases.

Case 2a  There may be a basic non-key column from the  $P^{th}$ 
block which can be interchanged with  $P_{j_T}$  in the basis. Then the 
leaving column  $P_{j_T}$  will become non-key and Case 1 can be applied.  
Suppose  $P_{j_T}$  is the  $i_2^{th}$ key column in the basis and that it will 
change places with the  $i_1^{th}$ non-key column. Then the working basis 
is updated by

$$ B^{-1} = E B^{-1} $$

where  $E$  is an  $m_0 \times m_0$ elementary row matrix equal to the identity 
except in the  $i_1^{th}$ row. Row  $i_1$ of  $E$  is just the  $i_2^{th}$ row of 
the submatrix  $V$  in the transforming matrix  $T$  in (9). There is a 
non-key column which can be exchanged with  $P_{j_T}$  if and only if there 
is a nonzero element in this row.  $B^{-1}$  will change by a simple pivot, 
and all other  $B^{-1}$  will remain unchanged.

Case 2b  If Case 2a cannot be performed, then by Theorem 1, the 
entering column  $P_{i_2}$  must be from the  $P^{th}$ block and a direct pivot 
is possible. In this case  $B^{-1}$  changes by a simple pivot, and the 
working basis will not change at all.
This completes the description of the algorithm for the general case. Note that at each iteration it is necessary to update at most an $m_0 \times m_0$ working basis inverse and an $m_1 \times m_1$ diagonal block inverse. All updates can be performed using multiplication by an elementary row or column matrix.
SECTION IV
WORKING BASIS STRUCTURE FOR THE MULTICOMMODITY PROBLEM

In the following sections the generalized upper bounding algorithm is applied to the multicommodity problem. Because of the special structure, significant simplifications occur.

Consider any basis matrix $B$ for the multicommodity problem (6). By Theorem 1 the basis matrix can be partitioned as follows:

$$B = \begin{pmatrix}
R_1 & 0 & A_{11} & A_{21} & \cdots & A_{K1} \\
R_2 & I & B_1 & 0 & \cdots & 0 \\
R_3 & 0 & B_2 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
R_{s+1} & 0 & 0 & 0 & \cdots & B_K
\end{pmatrix}$$

(28)
In this basis there are s saturated arcs, and hence s slack variables in the basis. For each of the K commodities there is a diagonal block $B_1$ which, by Theorem 1, is an $N-1 \times N-1$ noneiningular submatrix of the node arc incidence matrix $F$. The remaining $s+1$ columns in $[R_1 R_2 R_3]'$ consist of the cost variable (which is always the first basic variable) and $s$ columns which are excess columns from some of the commodity blocks.

It is well known that the $N-1$ arcs corresponding to the columns of each matrix $B_i$ form a spanning tree in the network [2]. Consequently we will be able to perform all the simplex operations which require $B_i^{-1}$ by graph theoretic means, so it is not necessary to maintain these inverses (or the matrices $B_i$) explicitly.

The only portions of the algorithm which are not "graph theoretic" involve multiplication by the working basis inverse, so we now consider the structure of the working basis. It arises from the submatrix

\[
\begin{bmatrix}
R_1 & 0 \\
R_2 & I
\end{bmatrix}
\]

of $B$ in (28) when $B$ is triangularized by driving $R_3$ to zero. Suppose $P$ is one of the $s$ excess columns in $[R_1 R_2 R_3]'$ of the basis, and that it is from the $k$th commodity block, so
\[ P = (P_0, 0 \ldots 0, P_k, 0 \ldots 0)^t \]

where \( P_0 \) has \( k + 1 \) components and \( P_k \) has \( N - 1 \). The corresponding column in the working basis will then be given by

\[ Q_0 = P_0 - A_k B_k^{-1} F_k \]

(see (12)). Here \( P_k \) is a column of \( F \) not contained in \( B_k \), so it corresponds to an out-of-tree arc for the \( k \)th commodity.

Any such out-of-tree arc forms a unique circuit with the arcs of the spanning tree, and this circuit is described by the vector \(-B_k^{-1} P_k\) whose \( j \)th component is:

+1 if the tree arc corresponding to the \( j \)th column of \( B_k \)
  is in the circuit and oriented the same as the out of tree arc.

-1 if the tree arc corresponding to the \( j \)th column of \( B_k \) is
  in the circuit and oriented in the opposite direction as the
  out of tree arc.

0 if the tree arc corresponding to the \( j \)th column of \( B_k \) is
not in the circuit.

Hence the vector \(-B_k^{-1} P_k\) can be calculated without knowing \( B_k^{-1}\)
by a simple labeling process in the network:

(a) Label the destination node of the out-of-tree arc with
  the label +0. Go to Step b.
(b) Take some node \( n \) which has been labeled but not scanned and scan it. This means that every unlabeled node which is connected to node \( n \) by a tree arc (in the \( k \)th spanning tree) is given a label. If the new node is reached by moving forward on arc \( a_m \), then the new node is labeled \( +m \). If the new node is reached by moving backward on arc \( a_m \), then the new node is labeled \( -m \). Go to Step c.

(c) If the origin node of the out-of-tree arc has been labeled, go to Step d. Otherwise go to Step b.

(d) Backtrack through the tree until the \( +0 \) label is found, recording the vector \( -s_k^{-1} P_k \) as the backtracking is performed.
The submatrix $A_{k1}$ in (29) has columns which contain a cost coefficient as the first component, and either zeroes or a unit vector as the remaining components. Essentially this matrix permutes the arcs of the tree into the order in which they appear in the capacity constraints. Because $B_{k}^{-1}P_{k}$ is all 0 or $+1$, no multiplications are required to compute $A_{k1}B_{k}^{-1}P_{k}$ and hence $Q_{0}$ in (29) is readily computed. This column $Q_{0}$ of the working basis can be interpreted as follows. For $i = 1, \ldots, \ell$ let the $i^{th}$ capacitated arc be the one corresponding to the $i+1^{th}$ row of $B$. Then

a) The first component of $Q_{0}$ is the sum of the cost coefficients of arcs in the circuit for $P$, with a plus sign for arcs oriented as $P$'s arc and minus otherwise.

b) The remaining components are all zero or $\pm$ ones with the $i+1^{th}$ component being

- $+1$ if the $i^{th}$ capacitated arc is the arc associated with the column $P$.
- $+1$ if the $i^{th}$ capacitated arc is in the unique circuit formed in the tree by the addition of $P$ and oriented the same as $P$.
- $-1$ if the $i^{th}$ capacitated arc is in the unique circuit formed in the tree by the addition of $P$, but oriented opposite to $P$.
- 0 otherwise.

As a result of this interpretation, $Q_{0}$ can be computed by a simple extension of the labeling algorithm for finding circuits.
The slack columns in the original basis are not affected by the triangularization. Hence the working basis $B$ will have the form

$$B = \begin{bmatrix} S_1 & 0 \\ S_2 & I \end{bmatrix}_{s+1 \times s+1}$$

(30)

where the columns of $\begin{bmatrix} S_1 \\ S_2 \end{bmatrix}$ have the form of $Q_0$ in (29).

The algorithm presented in Section III involves the inverse of $B$ in several places. In general, the elements of $B^{-1}$ are not integers, and it is necessary to maintain $B^{-1}$ explicitly. The presence of the slack columns lets us write

$$B^{-1} = \begin{bmatrix} S_1^{-1} & 0 \\ -S_2 S_1^{-1} & I \end{bmatrix}$$

(31)

and we will maintain only $S_1^{-1}$ explicitly. Rows of $-S_2 S_1^{-1}$ are just linear combinations of rows of $S_1^{-1}$ with coefficients $-1$, so they are easily obtainable from $S_1^{-1}$ wherever needed.
The result, then, of the special structure of the multicommodity problem is that it suffices to maintain and update a submatrix, $S_l^{-1}$, of the working basis inverse. The dimension of $S_l^{-1}$ is $s+1$, where there are $s$ saturated arcs in the current basis $B$. Thus, considerable savings are obtained whenever the number of saturated arcs is small relative to the total number of capacitated arcs. All other computations are performed by graph theoretic means.
SECTION V
THE ALGORITHM FOR THE MULTICOMMODITY PROBLEM

Assume that at the beginning of some simplex iteration the following quantities are known:

1. The submatrix $S^{-1}$ of $B^{-1}$ in (31)
2. The values and indices of the basic variables
3. The spanning tree for each commodity

In addition it may be desirable to maintain the submatrix $V$ of $I$ in (9) and the submatrix $S_2$ of $B$ in (30) (see Section VI for further discussion). The simplex iteration proceeds as follows.

Determining the simplex multipliers. By (15) the multipliers $\pi_0$ for the capacity constraints are found in the first row of $B^{-1}$.

Referring to (31) multipliers for saturated arcs are found in the first row of $S^{-1}$ and multipliers for unsaturated arcs are zero. Uncapacitated arcs can be assigned a multiplier of zero. The vector $\pi_k$ contains multipliers for the rows intersecting the $k^{th}$ commodity block. By (15) these satisfy

$$\pi_k B_k = -\pi_0 A_{k1} \quad (32)$$
The vector \( \pi_0 \) has a 1 as its first component, and the first row of \( A_k \) contains the negatives of the coefficients for the arcs in the \( k \)th tree. Hence for \( 1 \leq i \leq N - 1 \) the \( i \)th component of 
\[-\pi_0 A_k \] is the cost coefficient of the \( i \)th arc of the \( k \)th tree minus the component of \( \pi_0 \) corresponding to this arc. We will call this the price, \( p_k^i \), of the \( i \)th tree arc. Since \( B_k \) is triangular, equations (32) can be solved by successive elimination. In graph theoretic terms the procedure is:

1. Assign node \( N \) a multiplier of \( 0 \) (the equation for this node has been dropped from \( F \)).

2. Suppose the multiplier \( \pi_k^{n1} \) for node \( n_1 \) has been evaluated and \( n_1 \) is connected to \( n_2 \) by an arc \( a^i \) in the tree with price \( p_k^i \). Then

\[
\pi_k^{n2} = \pi_k^{n1} + p_k^i \quad \text{if the arc is oriented } n_1 \to n_2
\]

\[
\pi_k^{n2} = \pi_k^{n1} - p_k^i \quad \text{if the arc is oriented } n_2 \to n_1
\]

3. Continue branching along the \( k \)th tree until all nodes have been assigned multipliers for the \( k \)th commodity.

Strictly speaking, uncapacitated arcs have no components in \( \pi_0 \). The multiplier for such an arc is taken to be zero.
Determining the column to enter the basis. Let $\tilde{c}_{km}$ be the relative cost factor for $x_{km}$. Referring to (6), $\tilde{c}_{km}$ has at most four non-zero terms:

$$\tilde{c}_{km} = c_{km} - \pi_{0j_m} + \pi_{n1} - \pi_{n2}$$

where

- $\pi_{0j_m}$ is the origin node of arc $a_m$
- $\pi_{nj_m}$ is the destination node of arc $a_m$

and

- $\pi_{0j_m}$ is the component of $\pi_0$ corresponding to arc $a_m$ if the arc is capacitated and zero otherwise.

The slack variable $S_i$ has relative cost factor $-\pi_{0i}$.

Suppose column $P_s = [P_{s0}, 0 \ldots 0, P_{sk}, 0 \ldots 0]'$ from the $k^{th}$ commodity block is chosen to enter the basis. ($k = 0$ implies $P_s$ is a slack column).

Finding $P_s = B^{-1}P_s$ The transformation of the entering column

$P_s$ in terms of the current basis is outlined in equations (21) - (26).

In terms of the multicommodity problem these steps become

$$Z_i = 0 \quad i = 1, \ldots, K \quad i \neq k$$

$$Z_k = B_k^{-1}P_{sk}$$

$$Z_0 = B^{-1}(P_{s0} - A_k Z_k) = B^{-1}Q_{s0}$$

Note that $Z_k$ is just the negative of a circuit vector and $Q_{s0}$ is a column like $Q_{0}$ in (29).
Hence both $Z_k$ and $Q_{s0}$ can be computed using the graph theoretic labeling process described in Section IV. To obtain $Z_0$ it is necessary to multiply by $B^{-1}$, a non-graph operation. The details of the computation are:

$$Z_0 = B^{-1} Q_{s0} = \begin{bmatrix}
    S_1^{-1} & 0 \\
    -S_2 S_1^{-1} & I
\end{bmatrix}
\begin{bmatrix}
    Q_{s0}^{-1} \\
    Q_{s0}^2
\end{bmatrix}
\begin{bmatrix}
    S_1^{-1} Q_{s0} \\
    -S_2 S_1^{-1} Q_{s0} + Q_{s0}^2
\end{bmatrix}
$$

so a matrix multiplication of order $s+1$ must be performed to get $S_1^{-1} Q_{s0}$. Then the rest of the column is generated by additive operations, since $S_2$ is a matrix of zeros and $\pm 1$'s.

Transforming back to $\hat{P}_s$ is accomplished as in (24) - (26) by

$$\hat{P}_{s0} = Z_0$$
$$\hat{P}_{si} = V_i Z_0$$
$$\hat{P}_{sk} = V_k Z_0 + Z_k$$

Here $V_i$ is an $N-1 \times \ell+1$ matrix which is all zero except in columns corresponding to excess columns from commodity block $i$. 
The nonzero columns contain the circuit vectors for those excess columns (see (10)). Thus this transformation from $Z$ to $\hat{P}_b$ is also accomplished using only additive operations. If the entering column is a slack column, then the computations are even simpler - all $Z_i$ are zero ($i \neq 0$), and $Z_0$ is just a column of $B^{-1}$.

Choosing the Column to leave the Basis. This is done according to the standard simplex formulas. Assume that column $r$ of $B$ leaves the basis. Since computing the new values of the basic variables also proceeds as in the standard simplex method, we now consider updating the submatrix $S_1^{-1}$. 
SECTION VI
UPDATING FORMULAS

In previous sections, we have maintained only a submatrix $S^{-1}_1$ of $B^{-1}$. All other quantities are calculated as needed by graph theoretic and additive methods. Hence, in the updating procedures for $B^{-1}$, it suffices to consider updating only $S^{-1}_1$. The cases are the same as in Section III.

Case 1 When the leaving column is non-key, $B^{-1}$ is updated by

$$B^{-1} = E B^{-1}$$

(33)

where $E$ is an elementary column matrix. Since none of the diagonal blocks are affected, the spanning trees are unchanged. There are 4 subcases:

a) The leaving column is a flow column, and the entering column is a flow column.
b) The leaving column is a flow column, and the entering column is a slack column.
c) The leaving column is a slack column, and the entering column is a flow column.
d) The leaving column is a slack column, and the entering column is a slack column.

Consider first Cases 1a and 1b in which the leaving column is a flow column. Then, writing (33) in partitioned form gives...
Hence \( S^{-1}_1 \) is updated by an elementary column matrix.

If the entering column is a flow column, then the updating is complete. If the entering column is a slack column, then \( S^{-1}_1 \) can be reduced in dimension by one, since \( *S^{-1}_1 \) in (74) will contain a unit vector column. To see this, suppose that the leaving column is in position \( r \) in the basis \( (r \leq s+1) \) and that the slack in row \( t \) \( (t \leq s+1) \) is entering. As shown in Section V the first \( r+1 \) components of the transformed entering column are the \( t^{th} \) column of \( B^{-1} \). Updating the working basis is accomplished by pivoting on the \( r^{th} \) element of this column, as illustrated below:
Since the pivot reduces the pivot column to a unit vector, it will also reduce column \( t \) of \( B^{-1} \) to the \( r \)th unit vector. Consequently we can reduce the dimension of \( *S_1^{-1} \) by dropping its \( t \)th column and \( r \)th row.

In Cases 1c and 1d, the leaving column is a slack column in position \( r \) in the basis \((r > s + 1)\), so (33) becomes

\[
E \times B^{-1} = *S^{-1}
\]

In Cases 1c and 1d, the leaving column is a slack column in position \( r \) in the basis \((r > s + 1)\), so (33) becomes

\[
\begin{bmatrix}
I & 0 & \eta_1 & 0 \\
0 & I & \eta_2 & 1
\end{bmatrix}
\begin{bmatrix}
S_1^{-1} & 0 \\
-S_2 S_1^{-1} & I
\end{bmatrix}
= \begin{bmatrix}
*I S_1^{-1} & 0 & \eta_1 & 0 \\
-\eta_2 *S_2^{-1} & I
\end{bmatrix}
\]

(35)

Here

\[
*S_1^{-1} = \begin{bmatrix}
I - \begin{bmatrix}
0 & \eta_1 & 0 \\
\eta_2 & 0 & 0
\end{bmatrix} S_2^{-1}
\end{bmatrix}
\]

(35')

where \( v \) is the \((r-s-1)\)th row of \( S_2 \). As seen from (35), the block triangular structure of \( B^{-1} \) has been destroyed by the presence of the eta column. If the entering column is the slack in row \( t \), (Case 1d) then just as in Case 1b, column \( t \) of \( *B^{-1} \) will contain the \( r \)th unit vector. The structure can then be restored by exchanging the \( r \)th and \( t \)th columns of \( *B^{-1} \). This corresponds to replacing column \( t \) of \( *S_1^{-1} \) (which is a zero column) with the column \( \eta_1 \).
Finally, if the entering column is a flow column (Case 1c), \( *S_1^{-1} \) must increase in size by one since there is one less slack in the basis. To preserve the structure of \( *B^{-1} \), the \( r^{th} \) column and row are moved to position \( s+2 \). Then \( *S_1^{-1} \) is augmented by a border.

\[
\begin{array}{c|c}
*S_1^{-1} & \eta_1 \\
\hline
\omega & \eta \\
\end{array}
\]

where \( \eta \) is the \( r-s-1^{th} \) element of \( \eta_2 \) and \( \omega \) is the \( r-s-1^{th} \) row of \( -*S_2 \cdot S_1^{-1} \). To compute \( \omega \) note from (35) that

\[
-*S_2 \cdot S_1^{-1} = \begin{array}{c|c|c}
1 & \eta_2 \\
\hline
S_2 & S_1^{-1} \\
\end{array}
\]

Hence its \( r-s-1^{th} \) row is

\[
\begin{array}{c|c|c}
0 & \eta & 0 \\
\hline
S_2 & S_1^{-1} = -\eta v S_1^{-1} \\
\end{array}
\]  

(37)

where, as in (36), \( \nu \) is the \( r-s-1^{th} \) row of \( S_2 \). If the calculations in (36) are carried out from right to left (as is clearly preferable), \( \nu S_1^{-1} \) will already have been computed.
Case 2  When the leaving column is a key column, the corresponding arc is an arc in one of the spanning trees (say for commodity k). Removing it from the basis will destroy this tree, so the k\textsuperscript{th} spanning tree must be redefined. As in Case 2a of Section III, we first attempt to exchange the leaving column with a basic non-key column from block k. Consider the basic non-key columns from block k. The arc corresponding to each of these induces a unique circuit in the k\textsuperscript{th} tree. If one of these circuits contains the leaving column, then adding that arc to the tree and removing the leaving column will leave us with a new spanning tree. As in Section III, the working basis is then updated by an elementary row matrix,

$$ *B^{-1} = \begin{pmatrix} \mathbf{1} \\ \mathbf{v} \\ \vdots \\ \mathbf{I} \end{pmatrix} B^{-1}.$$

The vector \( \mathbf{v} \) is a row of the submatrix \( \mathbf{V} \) in (10). It contains zeroes except for +1 in columns corresponding to excess columns from block k whose circuits involve the leaving column. Hence, in particular, \( \mathbf{v} \) is zero in the last \( j-s \) columns, the slack columns. Hence, in partitioned form (38) is

\[
\begin{pmatrix}
*S_2^{-1} & 0 \\
*S_2 \cdot S_1^{-1} & I
\end{pmatrix}
= \begin{pmatrix}
1 & \mathbf{v}_1 \\
\vdots & \vdots \\
0 & I
\end{pmatrix}
\begin{pmatrix}
S_1^{-1} & 0 \\
0 & I
\end{pmatrix}
\begin{pmatrix}
S_2 \cdot S_1 \\
S_2^{-1} \cdot S_1 \\
\mathbf{I}
\end{pmatrix}
\]
where \( v_1 \) contains the first \( s+1 \) components of \( v \). Then

\[
\begin{pmatrix}
  v_1 \\
  *S_1^{-1} \\
  I_1
\end{pmatrix}
= \begin{pmatrix}
  I_1 \\
  v_1 \\
  I_1
\end{pmatrix}
\]

\( S_1^{-1} \)

gives the updating relation for \( *S_1^{-1} \). Note that only one row of
\( S_1^{-1} \) changes, and that row becomes a linear combination of rows of
\( S_1^{-1} \) with coefficients \( 0, \pm 1 \). Hence no multiplication is required
for this update.

If no such exchange is possible, then, as in Case 2b of Section
III, a direct pivot can be performed. A single spanning tree is redefined
(one arc changes). There is no change in the working basis and
hence no change in \( S_1^{-1} \).

In each case, in addition to updating \( S_1^{-1} \) and one of the \( K \)
spanning trees, we may wish to update the submatrices \( V \) and \( S_2 \).
Each column of \( V \) contains at most one nonzero partition, and that
partition is a circuit vector of the form \( -P_k \).

When a non-key column leaves the basis, (Case 1), one column of
\( V \) will change and a new circuit vector must be computed. When a
key column leaves, a spanning tree (say the \( k \)th tree) changes, so all
circuit vectors in \( V_k \) must be recomputed. Since at most two partitions
of \( V_k \) are changed at any iteration, it may be desirable to store the
nonzero columns of \( V_k \) explicitly. Since these contain only zeroes
and ± ones, they can be stored compactly. The alternative is to recompute them at each cycle. The best course of action depends on the amount of high speed storage available.

The matrix $S_2$ is a submatrix of $B$, and as shown in (29) $B$ has columns of the form

$$Q_0 = P_0 - A_k B_k^{-1} P_k$$

As shown in Section IV, $Q_0$ is essentially a permutation of the circuit vector $-B_k^{-1} P_k$. Hence $S_2$ probably should not be stored explicitly; it is easily generated as needed from the columns of $V$. 
SECTION VII
MAX FLOW PROBLEMS AND UNDIRECTED ARCS

To solve the max flow problem, a column for the commodity flow variable \( f_k \) must be added to the \( k^{th} \) block. This column corresponds to a fictitious arc from the sink \( t_k \) to the source \( s_k \) for commodity \( k \). The right hand side vectors \( d_k \) in (6) are all zero, and the cost coefficients are unity for the \( f_k \) and zeroes otherwise. Aside from the change from minimization to maximization, the algorithm proceeds as before.

As shown in [3] problems with undirected arcs can be formulated by defining new variables \( y_{km}^+ \) and \( y_{km}^- \) satisfying

\[
x_{km} = y_{km}^+ - y_{km}^-
\]

\[y_{km}^+ \geq 0, \quad y_{km}^- \geq 0\]

Then the capacity constraint

\[\sum_k |x_{km}| \leq b_m\]

becomes

\[\sum_k (y_{km}^+ + y_{km}^-) \leq b_m\]
provided that

\[ y_{km}^+ y_{km}^- = 0 \]  \hspace{1cm} (39)

If the problem has an optimal solution, then it has a solution in which (39) is satisfied. The constraint matrix then takes the form

\[
\begin{array}{ccccccc}
1 & b & c_1 & c_1 & \cdots & c_k & c_k \\
0 & I & I & 0 & I & 0 & \cdots \\
F & -F & & & & & \\
\end{array}
\]

The algorithm described above can be applied directly to this case. The structure of the working basis is exactly the same. The only change is that the extra columns must be considered in the pricing operation.
BIBLIOGRAPHY


