1999

An Efficient Implementation of the Transportation Problem

Alissa Michele Sustarsic
University of North Florida

AN EFFICIENT IMPLEMENTATION OF THE TRANSPORTATION PROBLEM

by

Alissa Michele Sustarsic

A thesis submitted to the Department of Mathematics and Statistics
in partial fulfillment of the requirements for the degree of

Masters in Mathematical Science

UNIVERSITY OF NORTH FLORIDA

COLLEGE OF ARTS AND SCIENCES

April, 1999

The thesis of Alissa Michele Sustarsic is approved:	(Date)
Signature Deleted	April 23, 99
Signature Deleted	April 23, 1999
Signature Deleted	4/23/99
Committee Chairperson	
Accepted for the Department:	
Signature Deleted	4/23/99
Chairperson	
Accepted for the College:	
Signature Deleted	4.22.99
Dean	
Accepted for the University:	
Signature Deleted	4/27/99
Dean of Graduate Studies	

ACKNOWLEDGEMENTS

I would like to thank Dr. Adel Boules for his support, guidance, and patience he gave throughout the time spent making this thesis possible. His encouragement and dedication made this thesis possible. I would also like to thank Dr. Champak Panchal and Dr. Mei-Qin Zhan for serving on the thesis committee. Thank you as well to Mrs. Theresa Kleinpoppen who helped edit this paper.

In addition, I would like to thank my parents and family for their encouragement throughout my time at UNF. Finally, I would like to thank my husband Jeff for his support, patience, and encouragement which helped in ways to numerous to mention.

TABLE OF CONTENTS

LIST OF FIGURES

University of North Florida

Abstract

AN EFFICIENT IMPLEMENTATION OF THE
TRANSPORTATION PROBLEM

By Alissa Michele Sustarsic

Chairperson of the Thesis Committee: Dr. Adel Boules
Department of Mathematics and Statistics

The transportation problem is a special type of linear program in which the objective is to minimize the total cost of shipping a single commodity from a number of sources (m) to a number of destinations or sinks (n).

Because of the special structure of the transportation problem, a special algorithm can be designed to find an optimal solution efficiently. Due to the large amount of information in the problem, judicious storage and management of the data are essential requirements of any viable implementation of the transportation algorithm.

Using sparse matrix techniques to store the solution array, and a rooted tree as the labeling method for handling the associated information provides a viable method to solve the transportation problem.

A difficult test problem was designed to test the computer program and demonstrate its efficiency. We were able to successfully implement the transportation algorithm for problems involving one million possible shipping routes. The FORTRAN code developed is included, as well as the results of several runs of the test problem.

Chapter 1

INTRODUCTION AND SOME STANDARD RESULTS

Linear programs are among the most widely used applications of mathematics in industry, business, and government. The objective of **linear programming** is to minimize (or maximize) a linear objective function in *n* real variables subject to a (finite) set of linear constraints, which can be either equations or inequalities.

Definition: The **standard form** of a linear program (LP) is one of the form:

$$\text{Minimize} \quad c^T x \quad \text{Subject to} \quad A\,x = b\,, \quad x \geq 0 \qquad (1.1)$$

where $A = (a_{i,j})$ is a real $m \times n$ matrix, x and c are n-dimensional column vectors, and b is an m-dimensional column vector.

Any linear program can be easily converted to standard form. The details of such conversions can be found in most textbooks on linear programming. (See Taha for details).

Definition: A point x is said to be a **feasible solution** of (1.1) if it satisfies the constraints, i.e., $A\,x = b$ and $x \geq 0$. A feasible point, x_0, is said to be an **optimal solution** of the linear program (1.1) if it satisfies $c^T x_0 \leq c^T x$ for any feasible x. In other words, the objective function attains its minimum value at x_0.

One can always assume that $m \leq n$ and that A has **full rank**, i.e. rank(A) = m. Thus A has m linearly independent columns. This can be assumed because if there are any dependencies among the rows, there is either no solution caused by contradictory constraints or there are redundant equations that can be eliminated.

Definition: Let B be a nonsingular m x m submatrix of A made up of m linearly independent columns. Set all n - m components of x that are not associated with the columns of B equal to zero. The solution to the resulting set of equations is said to be a **basic solution** to $Ax = b$ with respect to the basis B. The components of x associated with the columns of B are called **basic variables**. Because of the full rank assumption, a linear program will always have basic solutions.

Definition: If a feasible solution is also basic, it is referred to as a **basic feasible solution**. If it is also optimal, it is referred to as an **optimal basic feasible solution**.

The basic variables are not necessarily positive. If at least one of the basic variables in a basic solution is zero, then the solution is called a **degenerate basic feasible solution**.

One of the most important theorems in linear programming is the **Fundamental Theorem of Linear Programming** because it gives a criterion for limiting the search for optimal solutions.

Fundamental Theorem of Linear Programming (1.a). Given the standard linear program (1.1):

1) If there is a feasible solution, there is a basic feasible solution.
2) If there is an optimal feasible solution, there is an optimal basic feasible solution.

The above theorem states that the search for optimal solutions must be limited to the set of basic feasible solutions. The proof of The Fundamental Theorem of Linear Programming can be found in Luenberger, on page 18.

The notion of duality is central to both the development of linear programming algorithms and the computational aspects of the subject. Associated with every linear programming problem there is a **dual linear program**, defined as follows:

Definition: The **dual** of the linear program

$$\text{Minimize} \quad c^T x \qquad \text{Subject to} \quad Ax \geq b, \quad x \geq 0 \tag{1.2}$$

is defined as

$$\text{Maximize} \quad \lambda^T b \qquad \text{Subject to} \quad \lambda^T A \leq c^T, \quad \lambda \geq 0. \tag{1.3}$$

The LP (1.2) is referred to as the primal problem and (1.3) is often called the dual problem. λ is called the **dual vector,** and x is called the **primal vector**.

It can be shown, using the above definition, that the standard linear program (1.1) has the following dual program:

$$\text{Maximize} \quad \lambda^T b \qquad \text{Subject to} \quad \lambda^T A \leq c^T \qquad (1.4).$$

The following theorem and its corollary provide the important link between the primal and the dual problem, which will help to solve a linear program. A proof of the **Weak Duality Theorem** can be found in Luenberger, on page 89.

Weak Duality Theorem (1.b). Consider the standard dual pair (1.1) and (1.4). If x and λ are feasible for (1.1) and (1.4) respectively, then $c^T x \geq \lambda^T b$.

This shows that a feasible vector to either problem provides a bound on the value of the other problem. The corollary below gives a condition for the optimality of a solution.

Corollary (1.c). If x_0 and λ_0 are feasible for (1.1) and (1.4) respectively, and if $c^T x_0 = \lambda_0^T b$, then x_0 and λ_0 are optimal for their respective problems.

The above corollary leads to the important necessary and sufficient conditions for optimality (See Taha, pg. 154 for the proof), called the **complementary slackness condition**:

Complementary Slackness Theorem (1.d). Let x and λ be feasible solutions for (1.1) and (1.4) respectively. Then x and λ are optimal for their respective problems if and only if they meet the complementary slackness condition: $(c^T - \lambda^T A)x = 0$.

One method used to solve a linear program is the **simplex algorithm**, which uses the previous theorem as a stopping criterion. The simplex method proceeds from one basic feasible solution to another where the cost, barring degeneracy, is continually decreasing, until an optimal solution (minimum) is reached. The general philosophy behind the **primal simplex method** is to generate a sequence of primal basic feasible solutions and a corresponding sequence of vectors λ (not necessarily dual feasible), such that the complementary slackness conditions are met by each pair x and λ at each iteration. The algorithm terminates once λ becomes feasible for the dual problem.

The simplex method can be performed in tableau form. The first step to the simplex method is to put the problem in **standard canonical form**.

Definition: A standard linear program is said to be in **canonical form** if it has the following properties:

$b_i \geq 0$ for all i, the matrix A contains the columns of the identity matrix and the cost coefficients corresponding to the identity matrix are 0.

The **simplex tableau** in standard canonical form is depicted in Figure (1-1).

a_1	a_2	....	a_m	a_{m+1}	....	a_j	....	a_n	$\boldsymbol{b}$
1	0	....	0	$y_{1,m+1}$	....	y_{1j}	....	y_{1n}	y_{10}
0	1		:	:		:		:	:
:	:		:	:		:		:	:
0	0		:	$y_{i,m+1}$	....	y_{ij}	....	y_{in}	y_{i0}
:	:		:	:		:		:	:
0	0		1	$y_{m,m+1}$		y_{mj}		y_{mn}	y_{m0}
0	0	....	0	r_{m+1}	....	r_j	....	r_n	$-z_0$

Simplex Tableau

Figure (1-1)

The r_j are the reduced cost coefficients, which replace the cost coefficients once the manipulation of the tableau starts. The columns of the identity matrix are not necessarily the leading columns in the tableau, but the above depiction is used for the simplicity of notation. Once a problem is in canonical form, a basic solution can be read directly from the tableau; in the above depiction, x_1 through x_m are the basic variables with values b_1 through b_m. Step two of the simplex algorithm consists of examining the reduced cost coefficients. If all the reduced costs, $r_j \geq 0$, then the current basic feasible solution is optimal. If there exists a column with a negative reduced cost coefficient and all the entries within the column are nonpositive, there is no optimal solution. Otherwise, pick an $r_j < 0$ and pivot around y_{hj} such that $\frac{b_h}{y_{hj}} = \min\left\{\frac{b_i}{y_{ij}} \middle| \; y_{ij} > 0; 1 \leq i \leq m\right\}$. Return to the beginning of step 2 until an optimal solution is determined.

The relationship between the primal and the dual problem defined above can be seen more clearly in the simplex method when it is written in matrix notation. Let B be a **basis matrix**, i.e., a square submatrix of A consisting of the m linearly independent columns of A corresponding to the basic variables x_B, while D consists of the columns of A that correspond to the nonbasic variables x_D. The standard linear program problem can be rewritten, using the partition A=[B,D], x=[x_B,x_D], and c^T=[c_B^T,c_D^T], as

$$\text{Minimize } c_B^T x_B + c_D^T x_D \quad \text{Subject to} \quad B x_B + D x_D = b, \quad x_B \geq 0, x_D \geq 0 \tag{1.5}$$

The basic solution, x=[x_B,0] corresponds to the basis B where $x_B = B^{-1} b$ because x_D=0. For any value of x_D, $x_B = B^{-1} b - B^{-1} D \, x_D$ from (1.5) and thus by substitution, the objective function becomes

$$c_B^T B^{-1} b + \left(c_D^T - c_B^T B^{-1} D\right) x_D \tag{1.6}$$

From (1.6), the reduced cost coefficients for the nonbasic variables x_D are defined as

$$r_D^T = c_D^T - c_B^T B^{-1} D \tag{1.7}$$

The components of r_D^T determine the entering variable into the basis or whether the solution is optimal as described above.

Now it is possible to write the simplex tableau in matrix form as

$$\left[\begin{array}{c|c} A & b \\ \hline c^T & 0 \end{array}\right] = \left[\begin{array}{c|c|c} B & D & b \\ \hline c_B^T & c_D^T & 0 \end{array}\right] \tag{1.8}$$

If the matrix B is used as the basis, then multiplying (1.8) by the matrix, $\left[\begin{array}{c|c} B^{-1} & 0 \\ \hline -\lambda^T & 1 \end{array}\right]$ will result in

$$\left[\begin{array}{c|c|c} I & B^{-1}D & B^{-1}b \\ \hline -\lambda^T B + c_B^T & -\lambda^T D + c_D^T & -\lambda^T b \end{array}\right] \tag{1.9}$$

This corresponds to the one pivoting step in the simplex tableau, with λ^T defined as $\lambda^T = c_B^T B^{-1}$. The vector λ is called the **simplex multiplier.** Substituting the value of λ into (1.9), the resulting tableau is

$$\left[\begin{array}{c|c|c} I & B^{-1}D & B^{-1}b \\ \hline 0 & c_D^T - c_B^T B^{-1}D & -c_B^T B^{-1}b \end{array}\right] \tag{1.10}$$

Note that $(c^T - \lambda^T A)x = (c_B^T - \lambda^T B)x_B + (c_D^T - \lambda^T D)x_D = 0$ because, by definition of λ, the values of the reduced cost coefficients of the basic variables, x_B are 0 and the value of the nonbasic variables, x_D are 0. In other words, the primal simplex method meets the complementary slackness condition for each basic feasible solution, x and the corresponding simplex multiplier λ. By the **Complementary Slackness Theorem** (1.d), the current solution x optimal if and only if λ is dual feasible. But λ is dual feasible if $\lambda^T A \le c^T$, which means $\lambda^T B \le c_B^T$ and $\lambda^T D \le c_D^T$. By construction $\lambda^T B = c_B^T$, so the first inequality holds. Therefore, the necessary and sufficient condition for optimality reduces to $c_D^T - \lambda^T D \;=\; c_D^T - c_B^T B^{-1} D \;\geq\; 0$. Thus, the reduced cost coefficients of the nonbasic variables, r_D^T must be greater than or equal to 0 for optimality to occur.

The simplex method requires the inversion of the basis matrix *B*, and this is done in a number of steps, or iterations, where in each step the matrix *B* differs from the previous in only one column. Thus, the inversion of *B* can be done easily.

Chapter 2

THE TRANSPORTATION PROBLEM

The **balanced transportation problem** is a special type of linear program in which the problem is to minimize the total cost of shipping a single commodity from a number of sources (m) to a number of destinations or sinks (n). The simplex method can be used to solve this problem. However, the special structure of the transportation problem allows for a different technique to be created to solve these problems. This method follows the same basic theory as the simplex method, but will be more computationally efficient and accurate.

Definition: The **balanced transportation problem** is defined as

Minimize $$z(x)=\sum_{i=1}^{m}\sum_{j=1}^{n}c_{i,j}\,x_{i,j} \tag{2.1}$$

Subject to $$\sum_{j=1}^{n}x_{i,j}=a_i \qquad \text{for } i=1 \text{ to } m \tag{2.2}$$

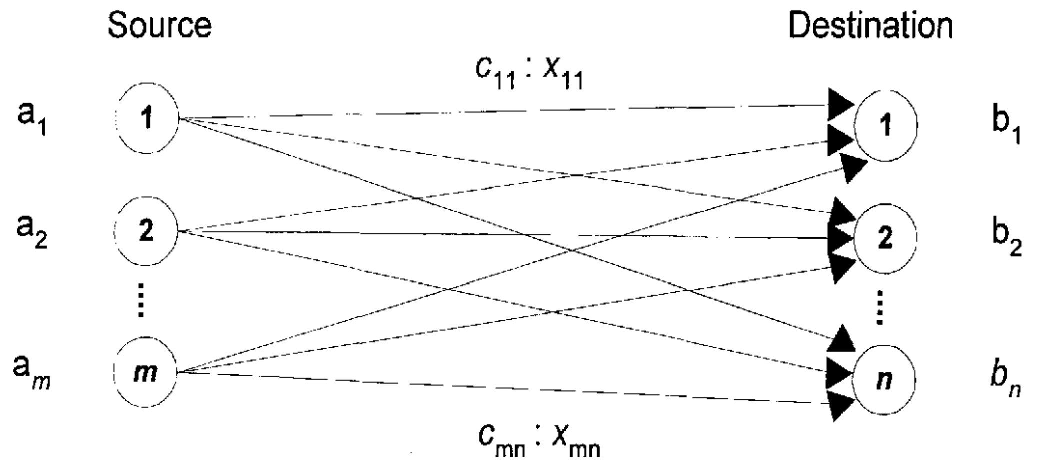
$$\sum_{i=1}^{m}x_{i,j}=b_j \qquad \text{for } j=1 \text{ to } n \tag{2.3}$$

$$x_{i,j}\ge 0 \qquad \text{for } i=1 \text{ to } m, j=1 \text{ to } n$$

where for all i and j, $a_i > 0$, $b_j > 0$ and $\sum a_i = \sum b_j$.

The only data needed for this problem is the cost of transporting the commodity per unit from each source i to each sink j ($c_{i,j}$), the availability of the commodity in source i, a_i, and the demand of sink j, b_j. $x_{i,j}$ represents the amount shipped from source i to sink j. m denotes the number of sources while n denotes the number of sinks. Notice that the number of constraints of the transportation problem is $m+n$ while the number of variables is mn. The first set of constraints (2.2) comes from the fact that the sum of the shipments from source i to all the destinations is equal to the supply available in source i. The second set of constraints (2.3) follows similarly, by considering the sum of the shipments from all the sources to destination j.

The transportation problem is depicted in Figure (2-1).



The Transportation Problem

Figure (2-1)

A node represents a source or a destination, and an arc that joins two nodes (a source and a destination) represents a shipping route through which the commodity is shipped.

The transportation problem is "balanced" because the total supply equals the total demand, $\sum a_i = \sum b_j$. In most applications, this is not the case. However, dummy sources or sinks can always be added to the problem to make it balanced. Being a "balanced" problem is an important feature of the transportation model and as we will now show is the necessary and sufficient condition for the transportation problem to be feasible. To show the sufficiency of this condition, let *S* be equal to the total supply (which is also equal to the total demand), $S = \sum a_i = \sum b_j$. Let $x_{i,j} = \frac{a_i b_j}{S}$ for $i=1,\ldots,m$ and $j=1,\ldots,n$. So $\sum_{j=1}^{n} x_{i,j} = \sum_{j=1}^{n} \frac{a_i\, b_j}{S} = a_i$ and $\sum_{i=1}^{m} x_{i,j} = \sum_{i=1}^{m} \frac{a_i\, b_j}{S} = b_j$. Therefore, *x* is feasible, and a feasible solution always exists for the balanced transportation problem. Conversely, if the transportation problem has a feasible solution *x*, then $\sum_{i=1}^{m}\sum_{j=1}^{n} x_{i,j} = \sum_{j=1}^{n} b_j$ and $\sum_{j=1}^{n}\sum_{i=1}^{m} x_{i,j} = \sum_{i=1}^{m} a_i$. Therefore, $\sum a_i = \sum b_j$, which establishes the necessity of the balance.

The feasible region is also bounded, since $x_{i,j} \le a_i$ and $x_{i,j} \le b_j$ for all *i* and *j*. Thus, $x_{ij} \le \min \{a_i, b_j \mid \forall i, j\}$, and since the feasible region is also closed, it is actually compact. Thus, the objective function will always achieve a minimum value (an optimal solution).

Let us look at the tableau form of the simplex method for the transportation problem, shown in Figure (2-2), where $c_i^T = (c_{i1}, c_{i2}, \ldots\ldots, c_{in})$ for $1 \le i \le m$, 1^T is a row vector of *n* ones, I is a *n* x *n* identity matrix, and $b = (b_1, b_2, \ldots, b_n)^T$.

1^T				a_1
	1^T			a_2
		$\ddots$		$\vdots$
			1^T	a_m
I	I	$\dots$	I	b
c_1^T	c_2^T	$\dots$	c_m^T	

Transportation Problem in the Simplex Tableau

Figure (2-2)

For future reference, let the matrix in the above tableau be denoted by A; thus

$$A = \left[\begin{array}{cccc} \mathbf{1}^T & & & \\ & \mathbf{1}^T & & \\ & & \ddots & \\ & & & 1^T \\ \hline \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} \end{array}\right] \qquad (2.4).$$

The next theorem summarizes the above discussion and determines the (maximum) number of nonzero components in a basic feasible solution.

Theorem (2.a). A balanced transportation problem always has a feasible solution. The rank of the matrix A is $m+n-1$. In other words, there is exactly one redundant constraint and the maximum number of nonzero components in a basic feasible solution is $m+n-1$.

Proof: The existence of a feasible solution was shown above, so it remains to show that the rank of A is as stated. Let R_1, R_2, …,R_m, R_{m+1},……, R_{m+n} denote the $m+n$ rows of the transportation matrix A (2.4). Clearly $R_1+R_2+\ldots+R_m-R_{m+1}-\ldots\ldots-R_{m+n}=0$. Thus, the rank($A$)$< m+n$, so at least one row can be written as a linear combination of the others.

To prove rank(A) $=m+n-1$, it suffices to show that $R_1, R_2, \ldots,R_m, R_{m+1},\ldots\ldots,R_{m+n-1}$ are linearly independent. Suppose not. Let $\alpha_1, \alpha_2,\ldots, \alpha_m$ and $\beta_1, \beta_2,\ldots, \beta_{n-1}$ be coefficients of the rows such that $\alpha_1R_1+\alpha_2R_2+\ldots+\alpha_mR_m+\beta_1R_{m+1}+\ldots\ldots+\beta_{n-1}R_{m+n-1}=0$. This is equivalent to the vector equation

$(\alpha_1+\beta_1,\alpha_1+\beta_2,..,\alpha_1+\beta_{n-1},\alpha_1,\ \alpha_2+\beta_1,..,\alpha_2+\beta_{n-1},\alpha_2,\ldots..,\ \alpha_m+\beta_1,..,\alpha_m+\beta_{n-1},\alpha_m)=0$. It clearly follows that $\alpha_i=\beta_j=0$. Therefore, no nontrivial linear relationship exists between rows R_1, $R_2,\ldots,R_{m+n-1}$, so rank(A)$=m+n-1$.♦

It follows from this theorem that any basis of the transportation problem consists of $m+n-1$ variables.

A direct application of the simplex method in tableau form to Figure (2-2) is computationally inefficient and requires a prohibitive amount of computer storage. For example, in a problem of 1000 sources and as many sinks, the matrix A would have 2,000 x 1,000,000 entries, which is obviously prohibitively large and quite wasteful, since A is very sparse and well structured.

Three questions arise naturally in the development of an algorithm to solve the transportation problem:

1) How do we construct an initial basic feasible solution?
2) How do we determine the optimality of a given basic feasible solution?
3) If the current basic feasible solution is not optimal, how do we construct a new basic feasible solution that is "closer" to the optimal solution than the current solution?

We will answer these three questions in turn, but first let us look at the transportation tableau, which is often used to illustrate paper and pencil calculations. The transportation tableau is depicted in Figure (2-3).

x_{11} c_{11}	x_{12} c_{12}		x_{1n} c_{1n}	a_1
	$\ddots$			a_2
		$\ddots$		$\vdots$
x_{m1} c_{m1}			x_{mn} c_{mn}	a_m
b_1	b_2	....	b_n	

The Transportation Tableau

Figure (2-3)

Each of the boxes, in the above tableau, is called a cell. The unit cost of shipment from source *i* to sink *j*, c_{ij}, is depicted in the center of the cell (*i,j*) (row *i*, column *j*). The bottom row contains the demands and the rightmost column contains the supplies. The location and value of a basic variable is indicated by putting the value of that variable in the top right hand corner of the corresponding cell. Thus, if a cell has only the cost coefficient c_{ij}, the corresponding variable is nonbasic.

We now answer the first of the three questions that were posed previously, by showing how to construct an initial basic feasible solution. There are several different methods for generating an initial basic feasible solution. One of the easiest methods is the **Northwest Corner Rule.** This method can be illustrated using the transportation tableau, Figure (2-3).

Definition: The **Northwest Corner Rule**:

1) Begin with all empty cells.
2) Start with the cell in the upper-left hand corner.
3) Allocate the maximum possible amount consistent with row and column sum requirements. At least one of these requirements will be met, i.e. the supply will be exhausted or the demand will be fulfilled.
4) If the row requirement (supply) is not exhausted, move one cell to the right. If the column requirement (demand) is not met, move one cell down. If both requirements are met simultaneously and the current assignment is not the last, enter a value of 0 in the cell immediately to the right, then move down one cell. (The solution is degenerate in this case.) If more assignments are to be made, go to step 2.

The Figure below is an illustration of the Northwest Corner Rule.

10				10
5	10	5		20
		15		15
		10	20	30
15	10	30	20	

An Example of the Northwest Corner Rule

Figure (2-4)

The solution determined by the Northwest Corner Rule is clearly feasible. The following concept is needed to establish the fact that it is basic.

Definition: A **loop** is an ordered sequence of at least four cells of an array if:

1) Any two consecutive cells lie in either the same row or same column.
2) No three consecutive cells lie in the same row or column.
3) The last cell in the sequence has a row or a column in common with the first cell in the sequence.

The following theorem gives a necessary and sufficient condition for a feasible solution of the transportation problem to be basic:

Theorem (2.b). In a balanced transportation problem, a set of $m+n-1$ variables is basic if and only the corresponding cells in the transportation tableau contain no loops.

Proof: Assume the set of cells contains a loop. Allocate a value of +1 and -1 alternately among the cells in the loop, and entries of 0 in all the rest of the cells not in the loop. Then the sum of all entries in the rows and columns of the array is zero. This corresponds with the multiplication of the constraints of the transportation problems (2.1) that coincide with a cell in the loop with +1 and by –1 respectively. If the columns are summed, the result will be the zero vector. Hence the set of the column vectors is linear dependent. Hence, any set of cells that contain a θ-loop will be linearly dependent. Therefore, the set of cells can not be a basis.

Let Δ be a set of cells corresponding to a basis and assume that Δ contains a loop. As seen from theorem (2.a), the columns corresponding to Δ are linear independent. Thus there does not exist a nonzero linear combination of the column vectors that equal the zero vector. Therefore, there can not be two entries in each row. This leads to a contradiction because a loop must contain two entries within each row that contains an entry of the loop.♦

We now turn to the question of finding a criterion for determining the optimality of a basic feasible solution. The notion of a triangular matrix is needed in order to achieve this. Simply put, a triangular matrix is a nonsingular square matrix that becomes lower triangular after an appropriate permutation of its columns and rows.

Definition: A matrix is said to be a **triangular matrix** if it satisfies the following properties:

1) The matrix has a row that contains exactly one nonzero entry.
2) The submatrix, formed from the matrix by crossing out the row and the column that contains the nonzero entry, also satisfies property (1). This procedure can be repeated until all rows and columns are crossed out.

Clearly, any matrix that satisfies the above 2 properties is a triangular matrix. Therefore, it can be put in lower triangular form by arranging the rows and columns in the order that was determined by the procedure listed above.

The importance of a matrix M being triangular is that the matrix equations, $M\,x = d$, can then be solved by backward substitution. So, if M is a triangular matrix, then after the reordering of the columns and the rows, the system takes the form $M'x = d$, where M' is lower triangular, and can be solved by backward substitution.

An important structural property of the transportation problem is given by the following theorem:

Basis Triangularity Theorem (2.c). Every basis of the transportation problem is triangular.

Proof: Consider the transportation matrix A (2.4). Let us change the sign of the first half of the system that corresponds to the supply constraints. Then, the coefficient matrix of the system will have entries of +1, 0, or -1. By theorem (2.a), one redundant equation can be eliminated. From the resulting matrix M, form a basis B by selecting a square nonsingular submatrix with $m+n-1$ columns.

Each column in A contains two nonzero entries including a +1 and -1, and, hence, each column in B contains at most two nonzero entries also. Thus, the total nonzero entries of B will be at most $2(m+n-1)$. If every column of B contained two nonzero entries, the sum of all the rows in B would be 0 as seen from theorem (2.a). This is a contradiction to B being nonsingular. Therefore, the nonzero entries in B must be less than $2(m+n-1)$. Since B is of order $(m+n-1)$, there must be a row with only one nonzero entry. This verifies the first property of a triangular matrix. A similar argument can be made for the submatrix created from deleting the row and column of B that contained the single nonzero entry; that submatrix will also have a single row with only one nonzero entry. This argument can be repeated, which establishes that the basis B is triangular. ♦

The triangularity of the basis makes it unnecessary to explicitly calculate the inverse of the basis B^{-1}, in order to calculate the simplex multipliers, given by $\lambda^T B = c_B^T$. Therefore, for a transportation problem, because the basis is triangular, the simplex method at this step simplifies to solving for the simplex multipliers directly, using backward substitution.

The next important step of the transportation problem is the form of the dual. The dual of the transportation problem is in the form of (1.4). Let $\lambda^T=(u^T,v^T)$ be partitioned in accordance with the natural partitioning of A. Thus $u^T = (u_1,\ldots,u_m)$ and $v^T = (v_1,\ldots,v_n)$. Remembering that A has two nonzero entries in each column, which can be seen from (2.4), the components corresponding to $c_{i,j}$ in the constraints $\lambda^T A \le c^T$ of the dual can be rewritten as $u_i + v_j \le c_{ij}$. Summarizing, the dual of the transportation problem can be rewritten as

$$\text{Maximize } \sum_{i=1}^{m} a_i u_i + \sum_{j=1}^{n} b_j v_j \text{ subject to } u_i + v_j \le c_{ij} \qquad \text{for } i=1,..m \text{ and } j=1,..n \qquad (2.5).$$

The complementary slackness condition, $(c^T - \lambda^T A)x = 0$, can also be rewritten as $\sum_{i=1}^{m}\sum_{j=1}^{n}(c_{ij} - u_i - v_j)x_{ij} = 0$ (2.6). The nonbasic variables always have a value of 0. Therefore to meet the complementary slackness condition, $(c_{ij} - u_i - v_j)$=0 for all basic variables. If $\lambda^T=(u^T,v^T)$ is also dual feasible, then the solution is optimal by the Complementary Slackness Theorem. Notice, the reduced cost coefficients

$r_{ij} = (c_{ij} - u_i - v_j)$, for the nonbasic cells x_{ij}. So again the criterion for optimality reduces down to whether the reduced costs are nonnegative for the nonbasic variables.

Therefore, after an initial basic feasible solution is found for the primal, the simplex multipliers, $\lambda=(u,v)$ need to be computed and then tested for feasibility. From the primal simplex method, the multipliers are computed from solving $\lambda^T B = c_B^T$. A column from A (2.4), that corresponds to the basic variable x_{ij}, will contain exactly two +1 entries, corresponding to the ith position within the top portion (sources) and to the jth position of the bottom portion (sinks). Thus, each column corresponding to the basic variable x_{ij}, will generate the simplex multiplier equation, $u_i + v_j = c_{ij}$. Remembering that one constraint is redundant, one of the multipliers can be assigned an arbitrary value. For simplicity, set $v_n=0$. The set of equations $u_i + v_j = c_{ij}$ (for all basic variables) can now be solved easily by backward substitution. Notice, by solving these equations, the complementary slackness condition is met.

Therefore, testing whether the simplex multiplier is dual feasible will define the criterion of whether the solutions are optimal. If (u,v) satisfies the inequality, $u_i + v_j \le c_{ij}$ for all i and j, it is dual feasible. Since this inequality is already met for all basic cells (i,j), the inequality, $c_{ij} - u_i - v_j \ge 0$ for the nonbasic cells (i,j) is a necessary and sufficient condition for optimality. This is equivalent to calculating the reduced cost coefficients and if they are all nonnegative, the solution (u,v) is feasible for the dual problem and (u,v) and x are optimal for their respective problems.

Therefore, the next step in the primal transportation algorithm is to calculate the reduced costs, $r_{ij} = c_{ij} - u_i - v_j$. They only need to be calculated for the nonbasic cells because by design, the reduced cost for the basic cells is 0.

If the reduced cost coefficients are not all nonnegative for the basis, then a new basis must be constructed. First, the next theorem allows us to use the reduced cost coefficients from step to step instead of keeping the original cost coefficients.

Theorem (2.d). Let r_{ij} represent the reduced cost coefficients. Then $\sum_{i,j} r_{ij}\, x_{ij}$ differs from the objective function, $\sum_{i,j} c_{ij}\, x_{ij}$ by a constant. Therefore, an optimal vector for $\sum_{i,j} c_{ij}\, x_{ij}$ is also an optimal vector for $\sum_{i,j} r_{ij}\, x_{ij}$.

Proof: $\sum r_{ij}\, x_{ij} = \sum\sum (c_{ij} - u_i - v_j)\, x_{ij} =$

$$= \sum\sum c_{ij}\, x_{ij} - \sum_i (\sum_j x_{ij}) u_i - \sum_j (\sum_i x_{ij}) v_j$$

$$= \sum\sum c_{ij}\, x_{ij} - \sum_i a_i u_i - \sum_j b_j v_j \qquad \blacklozenge$$

Therefore, during the calculations to solve the transportation problem, the reduced cost coefficients can be used to find the optimal solution. For the reason stated above, using the reduced cost coefficients to find an optimal solution will be helpful to show the solution meets the complementary slackness condition.

Once an initial primal feasible solution is recorded in the tableau, *u* and *v* can be recorded in the place allocated for *a* and *b*. From the previous theorem, the reduced costs can be used in place of the original cost coefficients in the rest of the problem. If at least one reduced cost coefficient is negative, a new basis needs to be defined. These alterations of the transportation tableau are shown in Figure (2-5).

x_{11} r_{11}	x_{12} r_{12}		x_{1n} r_{1n}	u_1
	......			u_2
		......		$\vdots$
x_{m1} r_{m1}			x_{mn} r_{mn}	u_m
v_1	v_2	......	v_n	

The Altered Transportation Tableau

Figure (2-5)

Finally, we turn to the last question of how to generate a new basis when the current basic feasible solution is not optimal. A new criterion for finding the location of a variable within the transportation problem needs to be defined. Because of the structure of the tableau displayed in Figure (2-3), the sum of the basic variables in each row and each column must remain the same at each step. Using a loop that contains the entering variable to obtain a new basis will allow for the feasibility of the primal problem to be kept throughout the changes of the basis. A θ-loop will assist in changing the basis.

Definition: A subset of cells is a **θ-loop** if entries of +θ and –θ are put alternately in the cells of the loop, such that if a row or a column contains a cell from the loop with a +θ entry, then it also contains an entry with a –θ.

A basis for the transportation problem has been shown to not contain a loop and also to be triangular. A collection of cells of the transportation array is a **minimal linearly dependent set** if and only if (1) it is linearly dependent and (2) no proper subset of it is linearly dependent. By the definition of a θ-loop, it is clear that a θ-loop is a minimal linearly dependent set. The following theorem will define a criterion of how to find a unique θ-loop.

θ-Loop in $B \cup \{(p,q)\}$ Theorem (2.f). Suppose B is a basic set of $m+n-1$ cells from the mxn transportation array and $\{(p,q)\}$ is a nonbasic cell. Then the collection of cells $B \cup \{(p,q)\}$ contains exactly one θ-loop and this θ-loop contains the nonbasic cell.

Proof: Since B is a basic set, B is linearly independent so it can not contain a θ-loop. Thus, if there is a θ-loop in $B \cup \{(p,q)\}$, the loop must contain $\{(p,q)\}$. Since the rank of A (2.1) is $m+n$-1, no subset of $m+n$ cells is linear independent. So $B \cup \{(p,q)\}$ is linearly dependent. From the previous theorems, $B \cup \{(p,q)\}$ contains at least one θ-loop. From Linear Algebra, a set containing a basis and exactly one nonbasic column vector contains a unique minimal linearly dependent set. Thus, the set of column vectors contained in the set $B \cup \{(p,q)\}$, contains exactly one θ-loop. ♦

To find a θ-loop in $B \cup \{(p,q)\}$, place an entry of +θ in the nonbasic cell (p,q). Then make alternating entries of -θ and +θ among the basic cells, such that each row and column contains a +θ and -θ or none at all. The cells marked by +θ and -θ creates a unique θ-loop. Cells marked +θ are called recipient cells and cells marked -θ are called donor cells. θ-loops can be used to change the basis, which is shown in the next theorem.

Theorem (2.g). Let B be a basis from A (ignoring one row) and let d be another column corresponding to a nonbasic variable which is entering the basis. The vector $y=B^{-1}d$ will give the changes in the current basic variables when the new variable is entered. The components of the vector $y=B^{-1}d$ are +1, -1 or 0.

Proof: Let y be a solution to $By=d$. Then y is the linear combination of the basis that represents d. This can be solved by Cramer's rule as $y_k = \dfrac{\det(B_k)}{\det(B)}$ where B_k is the matrix obtained by replacing the kth column of B by d. Since B is triangular, it may be put into lower triangular form with 1's on the diagonal by a combination of row and column interchanges. Therefore $\det(B)= +1$ or -1. Because any square submatrix of A will only contain entries of 0 or 1 with a maximum of two 1's in each column by the design of the matrix A, every determinant of any submatrix of A will have a value of +1, -1, or 0, so $\det(B_k)= 0, +1,$ or -1. Therefore $y_k=0, +1,$ or -1.♦

The significance of the above theorem is that the current basic variables will change by +1, -1, or 0 when a new variable is entered into the basis, at unit level. If the new variable has a value of θ, then the current basic variables will then change by +θ, -θ, or 0 corresponding to whether it is a recipient cell, donor cell, or a cell not within the loop, respectively.

Therefore, to change the basis, pick a nonbasic variable ($x_{p,q}$) corresponding to a negative cost coefficient (usually the most negative) to enter the basis. Find the unique θ-loop in the set $B \cup \{(p,q)\}$. Place a +θ in the cell (p,q) and the entries -θ and +θ alternatively among the cells within the loop. Let $\widetilde{x}$ represent the current basic feasible solution. Therefore, the values of the new basic feasible solution is $x_{i,j} = \widetilde{x}_{ij} + \theta$, $\widetilde{x}_{ij} - \theta$, or $\widetilde{x}_{ij}$, depending on whether (i,j) is a recipient cell, a donor cell, or a cell not within the loop, respectively and x_{pq}=θ. All other nonbasic variables have a value of zero.

All of the values within the vector x must remain nonnegative so that the solution remains feasible at every step. By choosing θ by the minimum ratio rule, $\theta = \min\{\, x_{rs}\,;\, (r,s)$ a donor cell$\}$, x will always remain feasible. Therefore, θ is determined at each step so that the primal feasibility of x is always retained. The donor cell from which θ is attained, is the cell that is leaving the basis. If more than one donor cell meets this criterion, one is arbitrarily chosen and is replaced in the basis by the cell (p,q). Because $\theta \geq 0$, the objective function at each step will at most be equal to the previous system. The objective function at each step will be $z(\widetilde{x})+r_{pq}\theta$. This can be seen by looking at the value of the objective function at two consecutive steps. At the first step,

$$z(\widetilde{x}) = \sum c_{ij}\,\widetilde{x}_{ij} = \sum\sum r_{ij}\,\widetilde{x}_{ij} + \sum_i a_i u_i + \sum_j b_j v_j$$

with the value of r_{ij} equal to 0 for the current basic variables and the value of $\widetilde{x}_{ij}$ equal to 0 for the nonbasic variables. Therefore, the objective function has a value of

$$z(\widetilde{x}) = \sum_i a_i u_i + \sum_j b_j v_j\,.$$

The objective function corresponding to the new basic feasible solution is

$$z(x)=\sum\sum r_{ij}\,x_{ij}+\sum_{i} a_i\,u_i+\sum_{j} b_j\,v_j=r_{pq}\,x_{pq}+\sum_{i} a_i\,u_i+\sum_{j} b_j\,v_j$$

because the only value changed from the previous objective function, that was not multiplied by 0, is $x_{p,q}=\theta$. Therefore, the objective function differs by $r_{pq}x_{pq}$ where $x_{pq}=\theta$ from the previous step.

Therefore, to summarize the **transportation algorithm**:

1) Compute an initial basic feasible solution.
2) Compute the simplex multipliers and the reduced cost coefficients. If all reduced cost coefficients are nonnegative, stop; the solution is optimal. Otherwise, go to 3.
3) Select a nonbasic variable corresponding to a negative reduced cost coefficient to enter the basis. Find the unique θ-loop and update the solution. Go back to 2.

The following is an example that illustrates the transportation algorithm in tableau form:

2	3	2	20
2	2	2	15
4	1	4	25
10	30	20	

1) Find initial basic feasible solution to the primal. We used the Northwest Corner Rule.

10 2	**10** 3	2	20
2	**15** 2	2	15
4	**5** 1	**20** 4	25
10	30	20	

2) Find the simplex multipliers (put in place of a,b).

10 2	**10** 3	2	$u_1 = 6$
2	**15** 2	2	$u_2 = 5$
4	**5** 1	**20** 4	$u_3 = 4$
$v_1 = -4$	$v_2 = -3$	$v_3 = 0$	

3) Find the reduced cost coefficients (put them in place of cost coefficients).

10 0	**10** 0	-4	6
1	**15** 0	-3	5
4	**5** 0	**20** 0	4
-4	-3	0	

4) Select the most negative cost coefficient. Find the θ-loop containing this. Select $x_{1,3}$ as nonbasic variable entering the basis.

10 0	(10-10) 0	**10**(+10) -4	6
1	**15** 0	-3	5
4	**15**(5+10) 0	**10**(20-10) 0	4
-4	-3	0	

5) Find the simplex multipliers.

10 0	0	**10** -4	$u_1 = -4$
1	**15** 0	-3	$u_2 = 0$
4	**15** 0	**10** 0	$u_3 = 0$
$v_1 = 4$	$v_2 = 0$	$v_3 = 0$	

6) Find the reduced cost coefficients.

10 0	4	**10** 0	-4
-3	**15** 0	-3	0
0	**15** 0	**10** 0	0
4	0	0	

7) Select the most negative cost coefficient. Find the θ-loop. Select $x_{2,3}$.

10 0	4	**10** 0	-4
-3	**5**(15-10) 0	**10**(+10) -3	0
0	**25**(15+10) 0	(10-10) 0	0
4	0	0	

8) Find the simplex multipliers.

10 0	4	**10** 0	$u_1 = 0$
-3	**5** 0	**10** -3	$u_2 = -3$
0	**25** 0	0	$u_3 = -3$
$v_1 = 0$	$v_2 = 3$	$v_3 = 0$	

9) Find the reduced cost coefficients.

10 0	1	**10** 0	0
0	**5** 0	**10** 0	-3
3	**25** 0	3	-3
0	3	0	

This solution is optimal because all reduced cost coefficients are nonnegative.

Chapter 3

LABELING METHODS IN THE PRIMAL TRANSPORTATION ALGORITHM

Although the transportation algorithm described in chapter 2 is an efficient specialization of the simplex method, it is obviously ineffective when dealing with larger problems. Larger problems clearly involve a tremendous amount of data, and keeping track of all the data could be quite difficult. A good example of that is the step of finding the θ-loop, where it is obvious that some kind of a “map” is needed to navigate the transportation tableau.

There is an effective labeling method used to keep track of the basis, which is quite efficient when dealing with larger problems. This method uses graph theory to label the m sources and n destinations creating a directed graph. This method is very useful in the computer implementation of the problem. In order to explain the use of this method, some background in graph theory must be given.

A **graph** $G=(N,A)$ is a pair of sets including a set N of **points** or **nodes** (or vertices) and a set of lines, A , called **edges** or **arcs**, with each edge joining a pair of distinct points in N. The edge denoted by $(i;j)$ is the edge connecting node i to node j. There is at most one edge between two nodes and every edge contains exactly two points of N. A **directed graph** is a graph where every arc has a specific direction. A **path** is a sequence of distinct arcs that join two nodes. The length of the path is defined to be equal to the number of edges in the sequence. A **simple path** is a path where every node along the path appears in the sequence only once. A **cycle** is a path between a node and itself that contains at least two nodes and a **simple cycle** is a cycle where each node appears only once within the cycle. A graph G is a **connected graph** if there exists a path in G between any two of its vertices and is disconnected otherwise. A connected graph that contains no cycles, is a **tree**. Therefore a unique path joins every two distinct points within a tree. A **terminal node** within a tree is a node where there is exactly one edge in A incident at it.

The tree associated with a basis for the transportation tableau is constructed as follows: let the sources $1,2,\ldots,m$ be represented by the nodes with serial numbers $1,2\ldots,m$, and let the sinks $1,2,\ldots,n$ be represented by the nodes with serial numbers $m+1, \ldots m+n$. Therefore, $N=\{1,\ldots,m,m+1,\ldots.m+n\}$ and if a cell (i,j) in the transportation array is basic, then there is a corresponding edge $(i;j+m)$ in the graph. For a subset Δ of a basis B, the set of corresponding edges can be denoted by $A_\Delta=\{(i;j+m)\text{:cell }(i,j)\in\Delta\}$. The graph associated with the subset is then $G_\Delta=(N,A_\Delta)$.

To construct the tree, let node $m+n$ be the root of the tree. The following procedure is repeated until all of the nodes in N are included in the tree: Include in the tree all points $i\in N$ that have not been included yet, satisfying $(i;j)\in A_\Delta$ for some j with the property that j is a point included in the graph at the previous stage.

It follows from the above that the immediate descendants of $m+n$ are the row indices i where x_{in} is a basic variable of the current solution, i.e. cell (i,n) is a basic cell.

Thus, the immediate descendants of the root, *m*+*n* are the sources for which a shipping route to sink *n* exists.

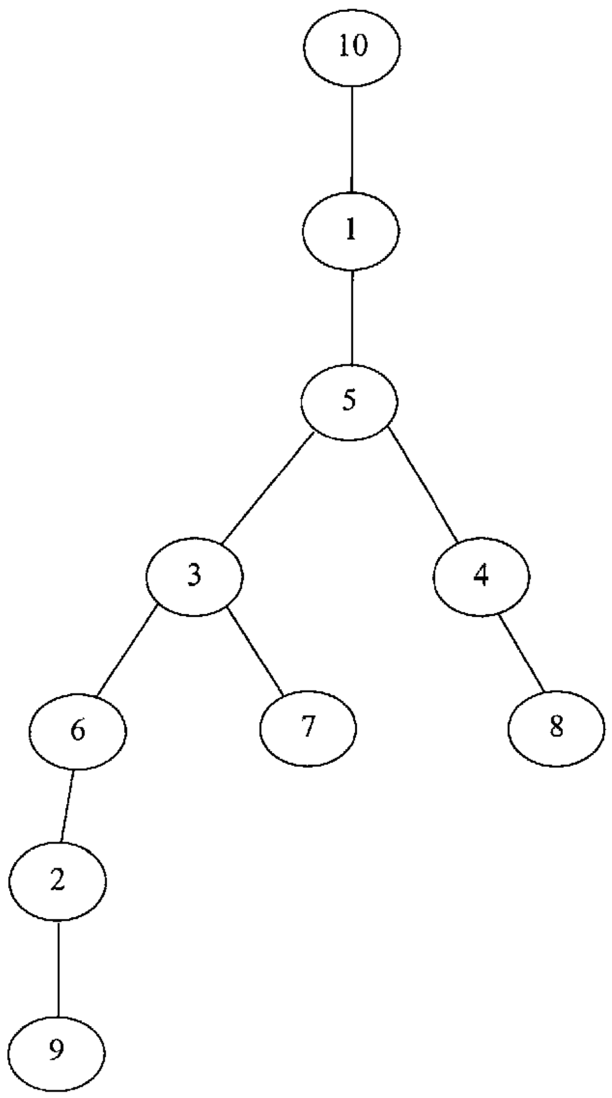
Because these edges within the rooted tree link sources to sinks and vice versa, this graph is directed. All points included in the tree during an even numbered stage are associated with rows of the transportation array (sources) and all points included in the tree during an odd numbered stage are associated with columns of the transportation array (sinks). Therefore at each step the direction alternates so that the arc is pointing from a source to a sink. A basic cell in the transportation array always corresponds to an edge between two nodes, the absence of an edge being equivalent to a cell being nonbasic.

The following example illustrates the correspondence between a basis and the tree describing it.

x					x	**1**
	x			x		**2**
x	x	x				**3**
x			x			**4**
5	**6**	**7**	**8**	**9**	**10**	

This tableau has an x entered in each basic cell, with the rightmost column corresponding to the serial number labeling the rows and the last row corresponding to the serial number labeling the columns.

The tree that corresponds to this basis is depicted in Figure (3-1).



An Example of a Rooted Tree

Figure (3-1)

A set of arrays is needed for the explicit storage of the information needed to describe a tree. Below is a description of the four arrays needed to achieve that goal.

Choose node $m+n$ as the root, then determine all the row indices i such that $(m+n\,;\,i)$ is an arc within the tree. Each such i is called a **immediate successor (or child)** of $m+n$, and $m+n$ is the **predecessor** of each such node. Predecessors and immediate successors of other nodes are defined similarly. The notation for the predecessor index of node j is P(j). If a node is a source, the predecessor index will be the serial number of a sink and vice versa. If a node does not have a unique immediate successor, these successors are considered brothers of each other, which are identified as a sequence ranging from eldest to youngest. Designate the successor index of a node to be the eldest son. Thus, for example if the younger brothers of node j are $\{j_1, j_2, j_3, \ldots, j_r\}$, then designate $S(j) = j_1$ and the **younger brother** of j_t to be j_{t+1} for $1 \le t \le r-1$, denoted as $YB(j_t) = j_{t+1}$. The **elder brother** index is now self-explanatory. The notation for the elder brother of j is EB(j) and for the previous example, $EB(j_{t+1}) = j_t$ for $1 \le t \le r-1$. If one of the relationships does not exist for a certain node, the corresponding value is set to 0. For example, for the root $m+n$, $P(m+n)=0$, $YB(m+n)=0$, and $EB(m+n)=0$. Also for any terminal node j, $S(j)=0$.

The set of younger brothers of j is the union of $\{YB(j)\}$ and the set of younger brothers of YB(j). The set of immediate successors of a point j if $S(j) \neq \varnothing$ is the union of $\{S(j)\}$ and the set of the set of younger brothers of S(j). The **descendants** of i is the union of the set of immediate successors of i and the sets of all descendants of j as j ranges over the set of immediate successors of i. If i is a terminal node, then the set of immediate successors and descendents will be empty.

The predecessor, successor, younger brother, and the elder brother indices help label the basis and are needed to alter the tree when a new basis is to be created.

Continuing with the previous example, the indices for the nodes obtained from the current basis are as follows:

Nodes	1	2	3	4	5	6	7	8	9	10
Predecessor	10	6	5	5	1	3	3	4	2	0
Successor	5	9	6	8	3	2	0	0	0	1
Younger Brother	0	0	4	0	0	7	0	0	0	0
Elder Brother	0	0	0	3	0	0	6	0	0	0

Example of the Indices corresponding to a Rooted Tree

Figure (3-2)

Now we address the procedure of how to change a basis while using a tree as the labeling method. The results of the following theorems are used to find a θ-loop within the tree.

Theorem (3.a). Let i_0 and i_* be a pair of points on a tree G. Then there exists a unique simple path in G from i_0 to i_*.

Proof: G is a tree so it is connected. Therefore there exists at least one path from i_0 to i_*. If more than one path exists then combining these paths would create a cycle from i_0 to i_0. This contradicts the assumption of G being a tree. ♦

In the discussion below, Δ denotes a subset of cells.

Simple Cycles and θ-loops Theorem (3.b). Every θ-loop in Δ corresponds to a simple cycle in G_Δ and vice versa.

Proof: This follows from the definition of a loop and a simple cycle.♦

Therefore Δ contains a θ-loop iff there is a simple cycle in G_Δ. If A_Δ contains $m+n-1$ edges, which is equivalent to a basis for the transportation array, it follows that G_Δ is a tree and therefore contains no cycles. This is equivalent to saying the tree contains no θ-loops, which has already been proven for any basis of a transportation problem. Thus, Δ is a basis for the transportation array iff $G_\Delta=(N,A_\Delta)$ is a tree.

The simple path between a point i and the root $m+n$ is referred to as the **predecessor path** of node i in G_B (defined by the basis B) because only the predecessor indices are used.

Definition: A **simple path** between a point and the root can be defined as follows:

1) The edge $(i;\mathrm{P}(i))$ is the first edge in the path. If $\mathrm{P}(i)$ is the root node, terminate. Otherwise pick $\mathrm{P}(i)$ as the current point j.
2) $(j;\mathrm{P}(j))$ is the next edge in the path.
3) If $\mathrm{P}(j)$ is the root node, terminate. Otherwise, change the current point to $\mathrm{P}(j)$ and return to step 2.

Let B be the basic set of cells for the transportation array and let (p,q) be the nonbasic cell being introduced into the basis. The unique θ-loop in $B \cup \{(p,q)\}$ can be determined by the predecessor indices. The cell (p,q) corresponds to the emerging edge

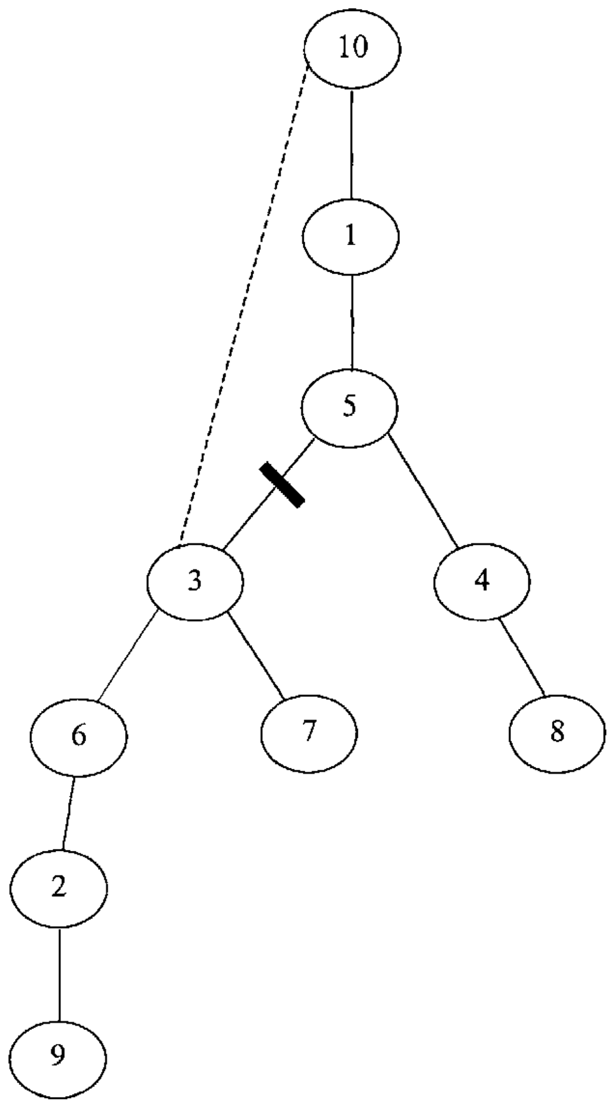
$(p;m+q)$ within the modified tree of the current graph G_B. The θ-loop corresponds to the simple cycle created when the edge $(p;m+q)$ is included in graph, $(N,A_B \cup \{(p;m+q)\})$.

To find the simple cycle, the unique simple path from $m+q$ to the root node as well as the unique simple path from p to the root node needs to be found. Eliminate all common edges between these two simple paths. The last common point of these two paths is known as the **apex** of the simple cycle. Combine what is left from both cycles to create the simple cycle. When i is a node corresponding to a row index, the edges $(i;j)$ in this simple cycle correspond to the cells $(i,j\text{-}m)$ from the transportation array in the θ-loop. Therefore once this simple cycle is found, the new basic feasible solution is formed by adding +θ and -θ as described in chapter 2 and dropping one basic cell from the basis. Assume the dropping cell is (r,s). Then the graph of the new basis B', $G_{B'}$ is obtained from the graph G_B by deleting the edge $(r;m+s)$ and adding the edge $(p;m+q)$.

To illustrate the modification of a tree, consider the example discussed earlier in this chapter. Let the variable (3,6) be the variable selected to enter the basis and the variable (3,1) be the variable leaving the basis (the dotted lines correspond to the loop). The new tableau corresponding to this change is as follows:

X					X	1
	X			X		2
	X	X			**X**	3
X			X			4
5	6	7	8	9	10	

The modifying of the tree in Figure (3-2) corresponding to this new basis is as follows:

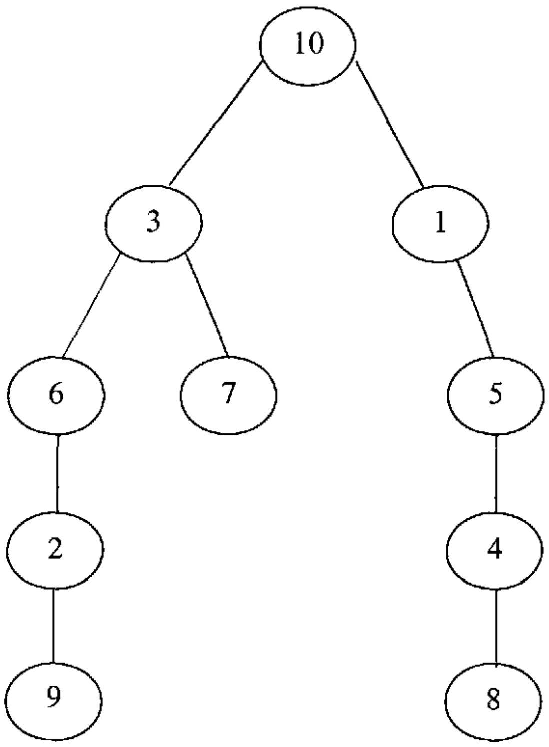


Example of Dropping and Entering Edge of a Tree

Figure (3-3)

The dashed line represents the entering edge and the edge that is slashed represents the edge that is being removed.

Figure (3-4) displays the modified tree that fits the new basis.



Example of a Modified Tree

Figure (3-4)

Chapter 4

SOME IMPLEMENTATION DETAILS

The computer program used to solve the transportation problem essentially follows the steps of the transportation algorithm outlined in the previous chapters. In this chapter, we outline some of the implementation details. Not all the subroutines will be discussed because there is extensive documentation within the program.

The sparsity of the transportation array, x, is an obvious problem that needs to be addressed; Only $m+n-1$ entries can be nonzero while x contains mn entries. In this program, a special method was used to store this matrix. The $m+n-1$ basic variables are stored in a linear array X. Thus, X(1), X(2),..., X($m+n$-1) will always contain the basic variables, arranged by rows. In the vector ROW, the ith entry contains the location in X of the first basic entry from the ith row for$1 \leq i \leq n$, and in the vector COL, the jth entry contains the column index of the basic variables X(j) for $1 \leq j \leq m+n-1$. This allows the program to store the basic feasible solutions more efficiently, since the rest of the values within the matrix x are 0. This method of storage does complicate the program, but the memory space saved by this method far out weighs the expense of adding these pieces.

The following is an example of how the basis is stored.

1					4	
	3			2		
6	8	7				
9			12			

I	1	2	3	4	5	6	7	8	9
X	1	4	3	2	6	8	7	9	12
Column	1	6	2	5	1	2	3	1	4

I	1	2	3	4
Row	1	3	5	8

For example, if we want to find the value of the basic variable x_{ij}, the entries COL(k) should be scanned for Row(i) $\leq$ k $\leq$ ROW(i+1)-1, until COL(k)=j. Subroutine **FIND** performs this task and can be found on page 61. It will output the index value of where the basic variable is located within X.

The first step of this program is to find an initial basic feasible solution to the primal. This is done using the Northwest Corner Rule and is performed by the subroutine **NW,** found on page 54. The design of this method makes it easier to program than other methods. The entries of COL(j) and X(j) are recorded at each step in the order in which the basic variables are allocated, while the value of the i^{th} entry in ROW(i) is assigned j when the first entry of the i^{th} row is assigned. The program will print a statement signaling that the problem is degenerate, if a value of 0 is assigned to any of the basic variables. Once the initial basic feasible solution is found, the initial cost is calculated and stored. As discussed earlier, when a new basic feasible solution is found, adding the reduced cost of the entering variable multiplied by θ gives the cost of the new solution.

An important step of this algorithm is to determine if the solution obtained is optimal. This is where the computer algorithm differs from the hand calculation algorithm. Trees need to be introduced so that a θ-loop can be determined easily. The initial rooted tree is created using the subroutine **TREE** located on page 56 and 57. The predecessor, successor, younger brother, and elder brother indices of each node (j=1 to m+n-1) are saved in the following vectors respectively: PRED(j), SUCC(j), YB(j), and EB(j). The successor index saves the serial number of the eldest son. The younger brother index saves the serial number of the eldest among the set of younger brothers and the eldest brother index saves the serial number of the youngest among the set of the eldest brothers. In order to keep track of which nodes have been labeled so far, a vector SLIST is used. SLIST is initially set so that the i^{th} entry equals i, for i=1 to m+n-1. Once a node has been labeled, the entry corresponding to its serial number in SLIST is changed to 0. The vector NODE is used to store the nodes, which need to be processed at a later stage. The value of the variable KOUNTER is the number of the node currently being processed.

If KOUNTER is a source node, then the column indices associated with the successors of KOUNTER are known because of the way that X is stored. If the node corresponding to the column has not already been processed, its predecessor index is defined to be equal to the KOUNTER. If there is more than one column that corresponds to this row and they have not been processed yet, these will be brothers, letting the eldest brother correspond to the first column obtained and the youngest brother correspond to the last. The same process is carried out if KOUNTER corresponds with a column index, except that the location of the row indices, corresponding to the basic cells in that column, must be determined. This process is repeated until each node is labeled. If a node does not have one of these indices, it is labeled as a 0.

The first step of the main loop of this program is to determine if all reduced costs are positive, and, if not, to determine the entering variable into the basis. This is done by the subroutine **NEWBAS,** which is located on page 58. **NEWBAS** is designed to pick the new variable to enter the basis using the modified row first negative method. This method was used because a study published in "Management Science" in 1974 showed that this method was most efficient, taking into consideration the time it takes to find the pivot variable, the average time per pivot, and the total pivot time (Glover, pg. 801). Modified Row First Negative Method finds the first row with a negative cost coefficient and then scans the rest of that row for any smaller reduced cost, saving the smallest. It saves the row index of the variable entering the basis as NBR, and the column index of the variable entering the basis as NBC. If all reduced costs are nonnegative, this solution is optimal, and the program terminates. If not, the program will print the iteration number and what variable is entering the basis.

The next step in the main iteration loop is to find the θ-loop using the rooted tree. This is done within the subroutine **QLOOP** on page 59 and 60. In order to do this, the simple path from P= NBR to the root and the simple path from Q= NBC+m to the root must be found. These simple paths are found by starting with P or Q, and moving up the tree using the predecessor indices until the root is reached (saving the nodes along the path). PPATH has a designated first entry of NBR and the rest of the entries correspond to the nodes contained in the simple path between P= NBR and the root. QPATH's first two entries correspond to NBR and NBC+*m*, respectively. After these first two entries, the nodes in the simple path between Q = NBC+*m* and the root are stored, with the last entry being the root. After these two paths are determined, they are combined within QPATH, eliminating any duplicated nodes contained within each path except for the APEX of this cycle. At the end of this subroutine, QPATH contains the simple cycle from P=NBR to Q = NBC and PPATH contains the simple path from P = NBR to the APEX.

Now that the θ-loop has been found and stored within QPATH, the value of θ must be determined, and X must be modified. The subroutine **MODFYX** on page 62 and 63, performs these operations. In this subroutine, IIN = NBR and JIN = NBC. The first step of determining the value of θ is to determine the donor and the recipient cells. Because of the way QPATH was constructed and because the tree is a directed graph, the nodes will correspond alternately to a row and a column index, allowing the variables corresponding to the basis to easily be determined. The first two entries within QPATH were defined so that they would correspond to the row and the column of the variable entering the basis. Starting with the third entry, the first basic variable of the loop is equal to X(QPATH(3), QPATH(2)-*m*), and corresponds to a donor cell. For *i*=3 and subsequent odd indices to the end of the array, the next two basic variables are X(QPATH(*i*), QPATH(*i*+1)-*m*), corresponding to a recipient cell and X(QPATH(*i*+2), QPATH(*i*+1)-*m*), corresponding to a donor cell. With the row and column indices known, each basic variable must be found within the X vector using the FIND subroutine.

Once it is found, the location of where it is located within COL and X is saved within LIST. The minimum value of X for donor cell is saved as THETA, and the row index as IOUT, the column index as JOUT, and the location of where the cell is within X as IDROP. Then THETA is subtracted from the donor cells and added to the recipient cells, which can be done easily because the location of each cell was saved within LIST (alternating donor and recipient).

Now the variable leaving the basis must be removed from X and the entering variable must be added. Because of the way that the basic variables were saved in X, it must be decided if the row of the cell entering is greater than the row of the cell that is leaving. The entering variable will be listed at the beginning of the section of COL and X, corresponding to its row. If IIN > IOUT, the indices in COL and X need to be shifted to the left. Otherwise, the indices in COL and X need to be shifted to the right. The indices are only altered between the rows of the entering and the leaving variables.

Once the X values have been modified, RLEFT is given the value of the row and CLEFT is given the value of the column of the variable leaving the basis. The rest of the program is well documented in the following chapter.

Chapter 5

FORTRAN CODE FOR THE TRANSPORTATION PROBLEM

```
!*******************************************************************************************
!OPTRAN IS A PROGRAM THAT CALCULATES THE OPTIMAL SOLUTION FOR A
!TRANSPORTATION PROBLEM
!******************************************************************************************

PROGRAM OPTTRAN
PARAMETER (M=1000,N=1000)
INTEGER     A(M), B(N), C(M,N), X(M+N-1), U(M), V(N), THETA, COST
INTEGER     ROW(M+1), COL(M+N-1), RLEFT, CLEFT, RROW, ROOT
INTEGER     PRED(M+N), SUCC(M+N), EB(M+N), YB(M+N),
INTEGER     NINJC(N), NINJR(M), RINJS(M), CINJS(N)
INTEGER     DESJSTAR(M+N), ALPHA, NDESJSTAR(M+N)
INTEGER     QPATH(M+N+1), PPATH(M+N+1), JPATH(M+N), REDCOST
REAL        AVLENDJ

X=0
ROW=0
COL=0
U=0
V=0
NBR=0
NBC=0
KSTOP=1
COST=0
ROW(M+1)=M+N

!******************************************************************************************
!CALCULATING A COST MATRIX, A,B FOR A TEST PROBLEM WHICH CREATES THE
!OPTIMAL SOLUTION TO BE CREATED BY THE SOUTHEAST CORNER RULE
!     M=  # OF SOURCES
!     N=  # OF SINKS
!     C=  COST MATRIX
!     A=  VALUE AVAILABLE FROM SOURCE
!     B=  VALUE NEEDED AT SINK
!******************************************************************************************

CALL CREATE(M,N,C,A,B)
```

```
!*************************************************************************************
! CALCULATING THE INITIAL BASIC FEASIBLE SOLUTION FOR THE TRANSPORTATION
!METHOD USING THE NORTHWEST CORNER RULE
!       ROW =TELLS WHERE THE FIRST ENTRY FROM ROW I IS STORED AT
!              WITHIN X AND COL
!       COL =CURRENT COLUMN VALUE OF THE BASIC CELLS
!           X =VALUE SENT FROM SOURCE I TO SINK J
!       COST=TOTAL COST OF SHIPMENT WITH THE BFS
!*************************************************************************************

CALL NW(X,A,B,ROW,COL,M,N)

K=1

DO I=1,M+N-1
   IF (ROW(K) == I) THEN
         RROW=K
      K=K+1
   END IF
   COST=COST+C(RROW,COL(I))*X(I)
END DO

!*************************************************************************************
!CALCULATE THE INITIAL DUAL VARIABLES(U,V) WHEN USING THE NWC RULE TO
!FIND A BFS SOLVING THE EQUATION          C(I,J)=U(I)+V(J) FOR ALL BASIC CELLS
!                                    SET V(N)=0 SO U(M)=C(M,N)
!*************************************************************************************

CALL DUAL(M,N,U,V,C,COL)

!*************************************************************************************
!COMPUTING THE INITIAL REDUCED COST COEFFICIENTS AND SAVING THEM IN
!THE COST MATRIX-C
!*************************************************************************************

DO I=1,M
   DO J=1,N
      C(I,J)=C(I,J)-U(I)-V(J)
   END DO
END DO

!*************************************************************************************
!CREATING A ROOTED TREE WHICH WILL ALLOW FOR A θ-LOOP TO BE PICKED
!       PRED=PREDECESSOR INDICES
!       SUCC=SUCCESSOR INDICES       (SERIAL #OF THE ELDEST SON)
!       YB=YOUNG BROTHER INDICES
!              (SERIAL #OF THE ELDEST AMONG THE SET OF YOUNGER BROTHERS)
!       EB=ELDER BROTHER INDICES
!              (SERIAL #OF THE YOUNGEST AMONG THE SET OF ELDEST BROTHERS)
!       (YOUNGEST= RIGHTMOST NODE OF THE TREE ON THAT LEVEL)
!*************************************************************************************

CALL TREE(PRED,SUCC,YB, EB, M, N, ROW, COL)
```

```
!*****************************************************************************************
!START OF LOOP TO FIND OPTIMAL SOLUTIONHAVE BASIC FEASIBLE SOLN,
!       NEED DUAL TO ALSO BE FEASIBLE SO NEED ALL REDUCED COSTS > 0
!       KOUNT=KEEPS TRACK OF HOW MANY ITERATIONS
!       KSTOP=WHEN ALL RC>0 THIS IS SET TO 0 SO THAT IT EXITS LOOP
!*****************************************************************************************


 KOUNT=1

DO WHILE (KSTOP == 1)                                  !BEGINNING OF LARGE LOOP


!*****************************************************************************************
!PICKING THE NEW VARIABLE TO ENTER THE BASIS USING MODIFIED ROW FIRST
!NEGATIVE METHOD. THIS METHOD FINDS THE FIRST ROW WITH A NEG REDUCED
!COST COEFFICIENT AND THEN SCANS THE REST OF THAT ROW FOR ANY OTHER RC
!WHICH IS MORE NEGATIVE.
!       NBR= ROW OF VARIABLE ENTERING THE BASIS
!       NBC= COLUMN OF VARIABLE ENTERING THE BASIS
!*****************************************************************************************


   CALL   NEWBAS(M,N,C,NBR,NBC,KSTOP)

   IF (KSTOP == 0) THEN                                     !EXIT LOOP IF OPTIMAL
      GO TO 1000
   END IF

   PRINT *, 'AT ITERATION',KOUNT ,' THE ENTERING VARIABLE INTO THE BASIS IS'
   PRINT *, 'X(' , NBR, NBC,')'


!*****************************************************************************************
!THIS FINDS THE  θ-LOOP USING THE ROOTED TREE WITH THE NONBASIC
!CELL(P,Q)=(NBR,NBC) TO ENTER THE BASIS.  IN ORDER TO DO THIS -THE SIMPLE
!PATH FROM P TO THE ROOT(M+N) AND THE SIMPLE PATH FROM Q(M+Q) TO THE
!ROOT MUST BE FOUND. THEN WE COMBINE THESE TWO PATHS ELIMINATING ANY
!DUPLICATES, LEAVING ONLY ONE DUPLICATE, WHICH IS THE APEX.
!       QPATH= SIMPLE PATH FROM Q TO THE ROOT
!                  AT END- SIMPLE CYCLE FROM P TO Q
!       LENQ= LENGTH OF QPATH
!       PPATH= SIMPLE PATH FROM P TO ROOT
!                  AT END- SIMPLE PATH FROM P TO APEX
!       LENP= LENGTH OF PATHOFP
!*****************************************************************************************


   CALL QLOOP(NBR,NBC,QPATH,M,N,PRED,LENQ,LENP,PPATH)
```

```
!*************************************************************************************
!THIS CHANGES THE VALUE OF X (THE BASIS) - ADD θ TO RECIPIENT CELLS AND
!SUBTRACT θ FROM THE DONOR CELLS.  CHANGE THE NONBASIC TO A BASIC CELL.
!       IDROP= POSITION OF THE BASIC CELL WHICH IS LEAVING THE BASIS
!       RLEFT=ROW INDEX OF CELL LEAVING THE BASIC FEASIBLE SOLN
!       CLEFT=COLUMN INDEX OF CELL LEAVING THE BASIC FEASIBLE SOLN
!       THETA= VALUE OF NEW BASIC CELL/OLD BASIC CELL
!*************************************************************************************


   CALL MODFYX(QPATH,LENQ,M,N,X,ROW,COL,IOUT,IDROP,JOUT,THETA)
   CLEFT=JOUT
   RLEFT=IOUT

   PRINT *, 'THETA=', THETA


!*************************************************************************************
!THIS IS THE BEGINNING OF UPDATING THE ROOTED TREE
!       RINJS= THE ROW INDICES CONTAINED AS A DESCENDANT OF J*
!       CINJS= THE COLUMN INDICES CONTAINED AS A DESCENDANT OF J*
!       LENR=LENGTH OF RINJS
!       LENC=LENGTH OF CINJS
!       (I1;J1) IS THE EDGE THAT IS ADDED
!       (I2;JSTAR) IS THE EDGE THAT IS LEAVING
!       I2=PRED(JSTAR)
!       ALPHA= USED LATER FOR CHANGING RC COEFFICIENTS
!       KTESTER=KEEPS TRACK OF IF I2,J* ARE ON PPATH TO THE APEX
!*************************************************************************************


   RINJS=0
   CINJS=0
   LENC=0
   LENR=0

   IF (PRED(RLEFT) == CLEFT+M) THEN
      JSTAR=RLEFT
      I2=CLEFT+M
      RINJS(1)=JSTAR
      LENR=1
   ELSE
      JSTAR=CLEFT+M
      I2=RLEFT
      CINJS(1)=JSTAR-M
      LENC=1
   END IF


   KTESTER=0
   IV=1
```

```
    !TESTING WHICH PATH THE EDGE THAT IS LEAVING IS ON
    DO WHILE (KTESTER < 2 .AND. IV < LENP+1)
        IF(RLEFT == PPATH(IV) .OR. CLEFT+M == PPATH(IV)) THEN
            KTESTER = KTESTER +1
        END IF
        IV=IV+1
    END DO

    IF (KTESTER == 2) THEN
        J1=NBR
        I1=NBC+M
        ALPHA=-1
    ELSE
        J1=NBC+M
        I1=NBR
        ALPHA=1
    END IF

!*****************************************************************************************
!DETERMINING THE SIMPLE PATH FROM J1 TO J*
!       JPATH= ARRAY THAT DETERMINES THE SIMPLE PATH FROM J1 TO JSTAR
!       JPATHLEN= COUNTER FOR LENGTH OF JMAT
!*****************************************************************************************

    JPATH=0
    JPATH(1)=J1

    IF (J1 == JSTAR)  THEN                                  !IF J1=JSTAR,IT DOESN'T
        KTESTER=0                                           ! GO INTO THE LOOP
        JPATHLEN=1
    ELSE
        KTESTER=1
        JPATHLEN=2
    END IF

    DO WHILE (KTESTER == 1)                                 !WHEN JSTAR IS REACHED,
    JPATH(JPATHLEN) = PRED(JPATH(JPATHLEN-1))                  ! KTESTER=0
        IF (JPATH(JPATHLEN) == JSTAR) THEN
            KTESTER=0
        ELSE
            JPATHLEN=JPATHLEN+1
        END IF
    END DO

!*****************************************************************************************
!CALCULATING THE DESCENDANTS OF JSTAR AND THE ROW AND COLUMN INDICES
!WITHIN THIS WILL BE  USED TO UPDATE THE REDUCED COSTS
!       DESJSTAR= SAVES INDICES ARE DESCENDANTS OF JSTAR
!       LENDJ=LENGTH OF DESJSTAR
!       RINJS=ROWS THAT ARE DESCENDANTS OF JSTAR
!       CINJS=COLS THAT ARE DESCENDANTS OF JSTAR
!*****************************************************************************************

 CALL DESCEND(SUCC,DESJSTAR,YB,M,N,JSTAR,LENDJ,RINJS,CINJS,LENR,LENC)
```

```
!*************************************************************************************
!NEED TO CUT THE PIECE IN THE TREE WHICH IS LEAVING.  THIS ALTERS THE TREE
!BY CUTTING IT INTO 2 SEPARATE TREES-
!       1 CONTAINING DESCENDANTS OF JSTAR,
!       THE OTHER CONTAINING THE REST OF THE TREE
!THE SUCCESSOR INDEX OF I2=PRED(JSTAR) ALSO IS CHANGED
!*************************************************************************************

    IP=PRED(JSTAR)
    IE=EB(JSTAR)
    IY=YB(JSTAR)
    EB(JSTAR)=0
    YB(JSTAR)=0
    PRED(JSTAR)=0

    IF(IE == 0) THEN
       IF(IY /= 0 ) THEN                !(IE=0,IY/=0)
          EB(IY) = 0
          SUCC(IP)=IY
       ELSE
          SUCC(IP)=0                    !(IE=0,IY=0)
       END IF

    ELSE
       IF (IY /= 0) THEN                !(IE/=0,IY/=0)
          EB(IY)=IE
          YB(IE)=IY
       ELSE                             !(IE/=0,IY=0)
          YB(IE)=0
       END IF
    END IF


!*************************************************************************************
!CALCULATING THE ROW AND COLUMNS INDICES WHICH ARE NOT DESCENDANTS
!OF JSTAR. THIS WILL BE USED TO UPDATE THE REDUCED COSTS.
!       ROOT=ROOT OF TREE
!       NDESJSTAR= INDICES THAT ARE NOT DESCENDANTS OF JSTAR
!       LENNDJ=LENGTH OF NDESJSTAR
!       NINJR=ROWS THAT ARE NOT DESCENDANTS OF JSTAR
!       KK1= LENGTH OF NINJR
!       NINJC=COLS THAT ARE NOT DESCENDANTS OF JSTAR
!       KK2= LENGTH OF NINJC
!*************************************************************************************

    NINJR=0
    NINJC=0
    KK2=1
    KK1=0
    ROOT=M+N
    NINJC(1)=N

    CALL DESCEND(SUCC,NDESJSTAR,YB,M,N,ROOT,LENNDJ,NINJR,NINJC,KK1,KK2)
```

```
!*************************************************************************************
! UPDATE THE RELATIVE COST COEFFICIENTS
!       ALPHA=  1 IF I1 IS A NODE CORRESPONDING TO A ROW IN THE
!               =-1 IF OTHERWISE                            TRANSPORTATION ARRAY
!       REDCOST=REDUCED COST OF THE ENTERING CELL
!*************************************************************************************


   REDCOST=C(NBR,NBC)

   CALL UPDATE(ALPHA,M,N,REDCOST,C,RINJS,CINJS,LENR,LENC,NINJR,NINJC,KK1,KK2)


!*************************************************************************************
! UPDATE THE TREE BY REBUILDING JPATH AND I1
!*************************************************************************************


   CALL UPTREE(M,N,PRED,SUCC,YB,EB,JPATH,JPATHLEN,J1,I1,JSTAR)


!*************************************************************************************
!THIS CALCULATES THE COST OF THE SHIPPING  WHERE C= COST MATRIX
!       COST=COST+THETA*C(NBR,NBC)
!*************************************************************************************

   COST=COST+THETA*REDCOST

   KOUNT=KOUNT+1
   PRINT *, "COST=", COST
END DO                                          !END OF LARGE LOOP


!TESTS TO SEE IF CORRECT ANSWER FOR THE CREATED TEST PROBLEM
1000  ICOST=0
      DO J=1,M+N-1
         ICOST= ICOST+J*(M+N-J)             !CALCULATING COST FOR TEST PROBLEM
      END DO


!SWITCHING COL IN ORDER W/ RESPECT TO ROW (USED TO SEE IF X IS CORRECT)
      KK=1
      DO I=1,M
         IF (COL(ROW(I)) < COL(ROW(I)+1))  THEN
            JJCOL=COL(ROW(I))
            JJX=X(ROW(I))
            COL(ROW(I))=COL(ROW(I)+1)
            COL(ROW(I)+1)=JJCOL
            X(ROW(I))=X(ROW(I)+1)
            X(ROW(I)+1)=JJX
         END IF
         !SUBTRACT VALUE FROM X THAT IT SHOULD HAVE
         X(ROW(I))=X(ROW(I))-KK
         X(ROW(I)+1)=X(ROW(I)+1)-KK-1
         KK=KK+2
      END DO
```

```
     !SUM THE X VALUES AND SHOULD GET ZERO FOR THIS TEST PROBLEM
        IX=0
        DO I=1,M+N-1
           IX=IX+X(I)
        END DO

        PRINT *, 'IX=', IX
        PRINT *, ' OPTIMAL COST SHOULD BE ', ICOST
        PRINT *, 'OPTIMAL COST AT ITERATION', KOUNT, ' IS', COST

  STOP
  END
```

```
!*******************************************************************************
!CREATING A COST MATRIX, A,B FOR A TEST PROBLEM WHICH CREATES THE
!OPTIMAL SOLUTION TO BE CREATED BY THE SOUTHEAST CORNER RULE
!       M=N AND HAVE TO BE EVEN
!       M= # OF SOURCES
!       N=# OF SINKS
!       C=COST MATRIX
!       A= VALUE AVAILABLE FROM SOURCE
!       B= VALUE NEEDED AT SINK
!*******************************************************************************

SUBROUTINE CREATE(M,N,C,A,B)
INTEGER     C(M,N), A(M), B(N)

A=0
B=0

!******CREATING COST
C=5*N

DO J=1,N-1
    C(N-J+1,J) = 2*J-1
    C(N-J,J) = 2*J
END DO

C(1,N)=2*N-1

!*****CREATING A,B
KK=M+N-1
A(M)=M+N-1
B(N)=1

DO  II=1,N-1
    B(II)=2*KK-1
    A(M-II)=2*(KK-1)-1
    KK=KK-2
END DO


RETURN
END
```

```
!****************************************************************************************
! CALCULATING THE INITIAL BASIC FEASIBLE SOLUTION FOR THE TRANSPORTATION
!METHOD USING THE NORTHWEST CORNER RULE
!   THE X VALUES ARE STORED IN THE VECTOR X
!   ROW(I)= TELLS WHERE THE FIRST ENTRY FROM ROW I IS STORED AT
!              WITHIN X AND COL
!   COL(I)= THE COLUMN INDICES THAT CORRESPOND TO EACH BFS
!          X= VALUE SENT FROM SOURCE I TO SINK J
!****************************************************************************************

SUBROUTINE NW(X,A,B,ROW,COL,M,N)
INTEGER     X(M+N-1), A(M), B(N)
INTEGER     ROW(M+1), COL(M+N-1)

I=1
J=1
K=1

ROW(1)=1

DO WHILE (K < M+N)
    IF (A(I) > B(J)) THEN
        X(K)=B(J)
        A(I)=A(I)-B(J)
        COL(K)=J
        J=J+1
    ELSE IF (A(I) < B(J)) THEN
        X(K)=A(I)
        B(J)=B(J)-A(I)
        ROW(I+1)=K+1
        COL(K)=J
        I=I+1
    ELSE
        X(K)=B(J)
        COL(K)=J
        IF (J < N)  THEN
           K=K+1
            X(K)=0
            ROW(I+1)=K+1
            COL(K)=J+1
            J=J+1
            I=I+1
            PRINT *, 'THIS PROBLEM IS DEGENERATE AT X' , I-1, J
        END IF
    END IF
    K=K+1
END DO

RETURN
END
```

```
!*************************************************************************************
!CALCULATES THE INITIAL DUAL VARIABLES(U,V) WHEN USING THE NWC RULE TO
!FIND BFS SOLVING THE EQUATION
!              C(I,J)=U(I)+V(J) FOR ALL BASIC CELLS
!              SET V(N)=0 SO U(M)=C(M,N)
!*************************************************************************************

 SUBROUTINE DUAL(M,N,U,V,C,COL)
 INTEGER    U(M), V(N), C(M,N), COL(M+N-1)

 KU=M
 KV=N
 U(M)=C(M,N)

 DO I=N+M-1,2,-1
    IF (COL(I) == COL(I-1)) THEN
        KU=KU-1
        U(KU)=C(KU,KV)-V(KV)
    ELSE
        KV=KV-1
        V(KV)=C(KU,KV)-U(KU)
    END IF
 END DO


 RETURN
 END
```

```
!************************************************************************************
! CREATING A ROOTED TREE WHICH WILL ALLOW FOR A θ-LOOP TO BE CHOSEN
!       PRED=PREDECESSOR INDICES
!       SUCC=SUCCESSOR INDICES      (SERIAL #OF THE ELDEST SON)
!       YB=YOUNG BROTHER INDICES
!             (SERIAL #OF THE ELDEST AMONG THE SET OF YOUNGER BROTHERS)
!       EB=ELDER BROTHER INDICES
!             (SERIAL #OF THE YOUNGEST AMONG THE SET OF ELDEST BROTHERS)
!       (YOUNGEST= RIGHTMOST NODE OF THE TREE ON THAT LEVEL)
!       KOUNTER= KEEPS TRACK OF WHICH NODE THAT NEEDS TO BE EXPLORED
!       LIST= HELPS KEEP TRACK FOR BROTHERS
!       NODE= KEEPS TRACK OF WHICH NODE WILL BE PROCESSED NEXT
!       SLIST= KEEPS TRACK OF WHICH NODES HAVE BEEN USED SO FAR
!************************************************************************************

SUBROUTINE TREE(PRED,SUCC,YB, EB, M, N, ROW, COL)
INTEGER SUCC(M+N), PRED(M+N), YB(M+N), EB(M+N) , ROW(M+1), COL(M+N-1)
INTEGER     SLIST(M+N), LIST(M+N), NODE(M+N), INDEX


KOUNTER=0
INDEX=1
SUCC=0
PRED=0
YB=0
EB=0
NODE=0
NODE(1)=M+N                                              !(ROOT OF TREE IS M+N)


DO K=1,M+N
   SLIST(K)=K
END DO


DO II=1,M+N
   KOUNTER=NODE(II)
   SLIST(KOUNTER)=0
   K=1
   LIST=0
   IF (KOUNTER <= M)     THEN                            !KOUNTER= A ROW INDEX
      DO I=ROW(KOUNTER),ROW(KOUNTER+1)-1
         J=COL(I)+M
         IF (SLIST(J) == J)     THEN
            PRED(J) = KOUNTER
            LIST(K)=J
            INDEX=INDEX+1
            K=K+1
            NODE(INDEX)=J
         END IF
      END DO
```

```
    ELSE                                                    !KOUNTER=A COLUMN INDEX
        ICOL=KOUNTER-M
        DO J=1,M                                            !J=ROW INDEX
            IF (SLIST(J) == 0)     GO TO 600
            CALL FIND(M,N,ROW,COL,J,ICOL,IFLAG)             !SEE IF A ROW CONTAINS
            IF (IFLAG == 0) GO TO 600                       !THIS COL, IF A ROW DOESN'T
            PRED(J) = KOUNTER                               !CONTAIN THE COL IT GOES
    LIST(K)=J                                               !TO NEXT ROW
            INDEX=INDEX+1
            K=K+1
            NODE(INDEX)=J
600     END DO
    END IF
    SUCC(KOUNTER)=LIST(1)
    ICOUNT=2
    DO WHILE (LIST(ICOUNT)   /= 0)
        EB(LIST(ICOUNT))=LIST(ICOUNT-1)
        YB(LIST(ICOUNT-1))=LIST(ICOUNT)
        ICOUNT=ICOUNT+1
    END DO
END DO


RETURN
END
```

```
!**************************************************************************************
! PICKING THE NEW VARIABLE TO ENTER THE BASIS USING MODIFIED ROW FIRST
!NEGATIVE METHOD. THIS METHOD FIND THE FIRST ROW WITH A NEG REDUCED
!COST COEFFICIENT AND THEN SCANS THE REST OF THAT ROW FOR ANY OTHER RC
!WHICH IS MORE NEGATIVE.
!       NBR= ROW OF VARIABLE ENTERING THE BASIS
!       NBC= COLUMN OF VARIABLE ENTERING THE BASIS
!       KTEST= KEEPS TRACK OF COST SO CAN COMPARE REST OF ROW
!                 & PICK SMALLEST
!       KSTOP= SET TO 0 WHEN ALL RC>=0
!**************************************************************************************

SUBROUTINE NEWBAS(M,N,C,NBR,NBC,KSTOP)
INTEGER    C(M,N) , KTEST

   NBR=0
   NBC=0

   DO I=1,M
      DO J=1,N
         IF(C(I,J) < 0) THEN
             NBR=I
            NBC=J
            KTEST=C(I,J)
            GO TO 40
         END IF
      END DO
   END DO

   KSTOP=0                              !IF ALL RC ≥0 THEN OPTIMAL SOLUTION-EXIT
   GO TO 50

40 NB=NBC
   DO JS=NBC+1,N
      IF(C(NBR,JS) < KTEST) THEN
            KTEST=C(NBR,JS)
         NB=JS
      END IF
   END DO

   NBC=NB

50 RETURN
   END
```

```
!*************************************************************************************
!THIS FINDS THE θ-LOOP USING THE ROOTED TREE AND THE NONBASIC CELL
!(P,Q)=(NBR,NBC) TO ENTER THE BASIS.  IN ORDER TO DO THIS THE SIMPLE PATH
!FROM P TO THE ROOT AND THE SIMPLE PATH FROM Q(M+Q) TO THE ROOT MUST BE
!FOUND.  THEN WE COMBINE THESE TWO PATHS ELIMINATING ANY DUPLICATES,
!LEAVING ONLY ONE DUPLICATE  WHICH IS THE APEX.
!   QPATH=SIMPLE PATH FROM Q TO THE ROOT
!             AT END-SAVES SIMPLE CYCLE FROM P TO Q
!   LENQ= LENGTH OF QTROOT
!   PPATH=SIMPLE PATH FROM P TO ROOT
!             AT END-SAVES SIMPLE PATH FROM P TO APEX
!   LENP= LENGTH OF PTROOT
!   APEX= SAVES THE NUMBER OF THE APEX
!*************************************************************************************


SUBROUTINE QLOOP(NBR,NBC,QPATH,M,N,PRED,LENQ,LENP,PPATH)
INTEGER  QPATH(M+N+1), PPATH(M+N+1),PRED(M+N), APEX


APEX=0
QPATH=0
PPATH=0
QPATH(1)=NBR
QPATH(2)=NBC+M
LENQ=2
LENP=1
PPATH(1)=NBR

!CALCULATING THE SIMPLE PATH FROM Q TO THE ROOT
DO WHILE  (QPATH(LENQ) /= M+N)
   LENQ=LENQ+1
   QPATH(LENQ) = PRED(QPATH(LENQ-1))
END DO

!CALCULATING THE SIMPLE PATH FROM P TO THE ROOT
DO WHILE  (PPATH(LENP) /= M+N)
   LENP=LENP+1
   PPATH(LENP) = PRED(PPATH(LENP-1))
END DO

!THIS ELIMINATES THE DUPLICATES
   QPATH(LENQ)=0
DO WHILE (QPATH(LENQ) == PPATH(LENP))
   APEX=QPATH(LENQ)
   PPATH(LENP)=0
   LENP=LENP-1
   LENQ=LENQ-1
END DO

LENQ=LENQ+1
QPATH(LENQ)=APEX
```

```
!RECORDING THE SIMPLE PATHE FROM P TO Q
DO I=LENP,1,-1
    LENQ=LENQ+1
    QPATH(LENQ) = PPATH(I)
END DO

LENP=LENP+1
PPATH(LENP)=APEX

RETURN
END
```

```
!*************************************************************************************
!THIS FINDS THE LOCATION OF AN X
!*************************************************************************************


 SUBROUTINE FIND(M,N,ROW,COL,III,JJJ,INDEX)
 INTEGER    ROW(M+1),COL(M+N-1)


   INDEX=0


   DO K=ROW(III),ROW(III+1)-1
      IF(COL(K) == JJJ) THEN
         INDEX=K
         GO TO 10
      END IF
   END DO                                   !IF INDEX=0 THEN ITEM NOT FOUND


10 RETURN
   END
```

```
!*********************************************************************************
!THIS CHANGES THE VALUE OF X( THE BASIC FEASIBLE SOLN)
!       IIN=NBR=ROW OF ENTERING VARIABLE
!       JIN=NBC=COL OF ENTERING VARIABLE
!       LIST=LOCATION OF WHERE CELLS ARE THAT ARE WITHIN THE LOOP
!       INDEX=LOCATION OF WHERE THE COL AND X VALUE ARE LOCATED
!       IOUT=ROW OF CELL LEAVING  (RLEFT)
!       IDROP=LOCATION OF COL OF CELL LEAVING
!       JOUT=COL OF CELL LEAVING (CLEFT)
!*********************************************************************************


 SUBROUTINE MODFYX(QPATH,LENQ,M,N,X,ROW,COL,IOUT,IDROP,JOUT,THETA)
 INTEGER    QPATH(M+N+1), ROW(M+1), COL(M+N-1),THETA, LIST(M+N-1)
 INTEGER    X(M+N-1)


!CALCULATING WHERE DONOR,RECIPIENT CELLS ARE LOCATED
IIN=QPATH(1)
JIN=QPATH(2)-M
II=QPATH(3)
JJ=JIN


CALL FIND(M,N,ROW,COL,II,JJ,INDEX)

IOUT=II
THETA=X(INDEX)
LIST(1)=INDEX
IDROP=INDEX


DO I=3,LENQ-2,2
    II=QPATH(I)
    JJ=QPATH(I+1)-M
    CALL FIND(M,N,ROW,COL,II,JJ,INDEX)
    LIST(I-1)=INDEX
    II=QPATH(I+2)
    CALL FIND(M,N,ROW,COL,II,JJ,INDEX)
    LIST(I)=INDEX
    IF (X(INDEX) < THETA) THEN
        IDROP=INDEX
        IOUT=II
        THETA=X(INDEX)
    END IF
END DO

IF (THETA > 0) THEN
    DO I=1,LENQ-2
        THETA=-THETA
        II=LIST(I)
        X(II)=X(II)+THETA
    END DO
END IF
```

```
THETA=-THETA
JOUT=COL(IDROP)


IF (IIN > IOUT) THEN                          !NEED TO SHIFT INDICES IN COL&X TO LEFT
    DO I=IDROP,ROW(IIN)-2
        COL(I)=COL(I+1)
        X(I)=X(I+1)
    END DO
    X(ROW(IIN)-1)=THETA
    COL(ROW(IIN)-1)=JIN
    DO I=IOUT+1,IIN
        ROW(I)=ROW(I)-1
    END DO
ELSE                                          !NEED INDICES SHIFTED TO RIGHT IF
    DO I=IDROP,ROW(IIN)+1,-1                            (NBR ≤RLEFT)   = (IIN ≤ IOUT)
        COL(I)=COL(I-1)
        X(I)=X(I-1)
        END DO
        COL(ROW(IIN))=JIN
        X(ROW(IIN))=THETA
        DO I=IIN+1,IOUT
        ROW(I)=ROW(I)+1
        END DO
END IF

RETURN
END
```

```
!*************************************************************************************
!THIS DETERMINES THE DESCENDANTS OF EITHER J* OR THE ROOT OF THE TREE
!       JSTARR=WHICH NODE WANT DESCENDANTS
!       SUCC= SUCCESSOR INDICES
!       DESCEN= SAVES DESCENDANTS OF JSTARR
!       LEND= LENGTH OF DESCEN
!       YB=YOUNGER BROTHER INDEX
!       INR=ROWS THAT ARE DESCENDANTS
!       INC=COLUMNS THAT ARE DESCENDANTS
!       LENR=LENGTH OF INR
!       LENC=LENGTH OF INC
!       II= CONTROLS WHICH NODE IS CONSIDERED NEXT
!*************************************************************************************

SUBROUTINE DESCEND(SUCC,DESCEN,YB,M,N,JSTARR,LEND,INR,INC,LENRR,LENCC)
INTEGER   SUCC(M+N), DESCEN(M+N), YB(M+N), LEND, TEST, LENRR, LENCC, INR(M),INC(N)

DESCEN=0
DESCEN(1)=JSTARR
TEST=1
II=1
LEND=1

DO WHILE (TEST == 1)
   IF(SUCC(DESCEN(II)) /= 0) THEN
      DESCEN(LEND+1)=SUCC(DESCEN(II))
      IF (DESCEN(LEND+1) < M+1) THEN                    !CORRESPONDS TO A ROW
         INR(LENRR+1) = DESCEN(LEND+1)
         DO WHILE (YB(DESCEN(LEND+1)) /= 0)
            DESCEN(LEND+2) = YB(DESCEN(LEND+1))
            INR(LENRR+2)=DESCEN(LEND+2)
            LENRR=LENRR+1
            LEND=LEND+1
         END DO
         LENRR=LENRR+1
      ELSE                                             !CORRESPONDS TO A COLUMN
         INC(LENCC+1) = DESCEN(LEND+1)-M
         DO WHILE (YB(DESCEN(LEND+1)) /= 0)
            DESCEN(LEND+2) = YB(DESCEN(LEND+1))
            INC(LENCC+2)=DESCEN(LEND+2)-M
            LENCC=LENCC+1
            LEND=LEND+1
         END DO
         LENCC=LENCC+1
      END IF
      LEND=LEND+1
   END IF
   II=II+1
   IF (II > LEND) THEN
      TEST=0
   END IF
END DO

RETURN
END
```

```
!*********************************************************************************
!UPDATING THE BROTHER INDICES  CHANGING JPATH, SUCC(I1)
!ASSUME THAT ANY NEW IMMEDIATE SUCCESSOR OF A POINT JOINS THE PREVIOUS
!IMMEDIATE SUCCESSORS OF THIS POINT AS THEIR ELDEST BROTHER
!       (JOINS AT THE LEFT OF THE SEQUENCE OF BROTHERS)
!       (I1;J1) IS THE EDGE THAT IS ADDED
!       JPATH= SIMPLE PATH FROM J1 TO J* WITH LENGTH=JPATHLEN
!       YB,EB,SUCC=YOUNGER BROTHER, ELDER BROTHER, SUCCESSOR INDICES
!*********************************************************************************


 SUBROUTINE UPBROTHER(YB,EB,JPATH,JPATHLEN,SUCC,M,N,J1,I1)
 INTEGER    YB(M+N), EB(M+N), SUCC(M+N), JPATH(M+N), YBPRIME(M+N)


!BECAUSE YB CHANGES OVERLAP, THE CHANGES MUST BE STORED IN YBPRIME
YBPRIME=0


!REMOVING JPATH OUT OF LIST OF BROTHERS
DO KK=1,JPATHLEN-1
   IF (EB(JPATH(KK)) /= 0)  THEN
      YB(EB(JPATH(KK)))=YB(JPATH(KK))
   END IF
   IF (YB(JPATH(KK)) /= 0) THEN
      EB(YB(JPATH(KK)))=EB(JPATH(KK))
   END IF
END DO


!BECAUSE THESE PTS JOIN AS THE ELDEST AMONG THEIR NEW BROTHERS
DO KK=1, JPATHLEN-1
   EB(JPATH(KK))=0
END DO


!START SHIFT OF BROTHERS TO SIDE OF GRANDPARENT

!ATTACHING THE IMMEDIATE DESCENDANTSOF J1 AS THE BROTHERS OF JPATH(2)
IF (SUCC(J1) /= 0) THEN
   EB(SUCC(J1))=JPATH(2)
END IF
YBPRIME(2)=SUCC(J1)


!ATTACHING THE IMMEDIATE DESCENDANTS OF I1 AS THE BROTHERS OF JPATH(1)=J1
IF (SUCC(I1) /= 0 ) THEN
   EB(SUCC(I1)) = J1
END IF
YBPRIME(1)=SUCC(I1)
```

```
!SHIFTS JPATH AS EB OF GRANDPARENT
DO KK=2,JPATHLEN-1
    IF (SUCC(JPATH(KK)) == JPATH(KK-1)) THEN
        YBPRIME(KK+1) = YB(JPATH(KK-1))
        EB(YBPRIME(KK+1)) = JPATH(KK+1)
    ELSE
        EB(SUCC(JPATH(KK))) = JPATH(KK+1)
        YBPRIME(KK+1) = SUCC(JPATH(KK))
    END IF
END DO


DO KK=1, JPATHLEN
    YB(JPATH(KK))=YBPRIME(KK)
END DO


RETURN
END
```

```
!*************************************************************************************
!THIS SUBROUTINE UPDATES THE TREE INDICES-PRED, SUCC, YB, EB USING JPATH
!       YB,EB,SUCC=YOUNGER BROTHER, ELDER BROTHER, SUCCESSOR INDICES
!       (I1;J1) IS THE EDGE THAT IS ADDED
!        JPATH= SIMPLE PATH FROM J1 TO J* WITH LENGTH=JPATHLEN
!*************************************************************************************

SUBROUTINE UPTREE(M,N,PRED,SUCC,YB,EB,JPATH,JPATHLEN,J1,I1,JSTAR)
INTEGER     PRED(M+N), SUCC(M+N), YB(M+N), EB(M+N) ,JPATH(M+N), YBJT


!UPDATING THE PREDECESSOR INDICES-ONLY ONES THAT CHANGE ARE THE JPATH
!PREDECESSOR INDICES BECOME THE OPPOSITE-REVERSE OF JPATH
   PRED(J1)=I1
   IF (JPATHLEN > 1) THEN
      DO KK=2,JPATHLEN
         PRED(JPATH(KK))=JPATH(KK-1)
      END DO
   END IF


!UPDATING THE BROTHER INDICES-       CHANGING JPATH, SUCC(I1)
! ASSUME THAT ANY NEW IMMEDIATE SUCCESSOR OF A POINT JOINS THE PREVIOUS
! IMMEDIATE SUCCESSOR OF THIS POINT AS THEIR ELDEST BROTHER
!(JOINS AT THE LEFT END OF THE SEQUENCE OF BROTHERS)
   YBJT=0
   IF (JPATHLEN >1) THEN
      YBJT=YB(JPATH(JPATHLEN-1))
   END IF

   CALL UPBROTHER(YB,EB,JPATH,JPATHLEN,SUCC,M,N,J1,I1)


!UPDATING THE SUCESSOR INDICES- ONLY CHANGE JPATH,I1,I2.  SUCCESSORS
!OF JPATH BECOME THEIR OLD PREDECESSOR INDEX(NEXT JPATH ENTRY)
!   YBJT= OLD YB(JPATH(JPATHLEN-1))     BEFORE UPDATED
   SUCC(I1)=J1
   IF (JPATHLEN > 1) THEN
      DO KK=1,JPATHLEN-1
         SUCC(JPATH(KK))=JPATH(KK+1)
      END DO
   END IF


   IF (JPATHLEN > 1) THEN
      IF (SUCC(JSTAR) == JPATH(JPATHLEN-1)) THEN
         SUCC(JSTAR)=YBJT
      !ELSE
      !   SUCC(JSTAR)=SUCC(JSTAR)
      END IF
   END IF

RETURN
END
```

```
!*********************************************************************************
!THIS SUBROUTINE UPDATES THE RELATIVE COST COEFFICIENTS
!  ALPHA=1 IF I1 IS A NODE CORRESPONDING TO A ROW OF THE
!          =-1 IF OTHERWISE                          TRANSPORTATION ARRAY
!      REDCOST= REDUCED COST OF CELL ENTERING THE BASIS
!      RINJS=THR ROW INDICES WITHIN DESJSTAR
!      LENR= CHANGROW LENGTH
!      CINJS=THR COL INDICES WITHIN DESJSTAR
!      LENC= CHANGCOL LENGTH
!      NINJR= ROW INDEX WHICH IS NOT IN DESJSTAR
!      KK1= LENGTH OF NINJR
!      NINJC=COLUMN INDEX WHICH IS NOT IN DESJSTAR
!      KK2= LENGTH OF NINJC
!*********************************************************************************


SUBROUTINE UPDATE(ALPHA,M,N,REDCOST,C,RINJS,CINJS,LENR,LENC,NINJR,NINJC,KK1,KK2)
INTEGER     NINJC(N), NINJR(M), ALPHA, CINJS(M),  RINJS(N), REDCOST ,  C(M,N)


!CHANGING COST COEFFICIENTS
DO I=1,LENR
   DO J=1,KK2
      C(RINJS(I),NINJC(J)) = C(RINJS(I),NINJC(J))+ ALPHA*REDCOST
   END DO
END DO


DO I=1,KK1
   DO J=1,LENC
      C(NINJR(I),CINJS(J)) = C(NINJR(I),CINJS(J)) - ALPHA*REDCOST
   END DO
END DO


RETURN
END
```

Chapter 6

RESULTS AND CONCLUSIONS

In order to test this program, a test problem whose size can be completely controlled was designed. The code for the subroutine **CREATE** is located on page 53. For any integer n, we construct a transportation problem of n sources and n sinks with the following optimal solution:

$X_{n-i+1,i} = 2n-2i+1$	for $1 \leq i \leq n$
$X_{n-i+1,i-1} = 2n-2i+2$	for $2 \leq i \leq n$
$X_{ij} = 0$	otherwise

The vectors A and B will then have the corresponding value

$A_i = \sum_{j=1}^{n} x_{ij} = 4i-1$	for $1 \leq i \leq n$
$B_j = \sum_{i=1}^{n} x_{ij} = 4(n-j)+1$	for $1 \leq j \leq n$

Note, that the solution is concentrated on the skew diagonal of the array and the diagonal above it.

The cost coefficients are defined as follows:

$c_{i,n-i+1} = 2n-2i+1$	for $1 \leq i \leq n$
$c_{i-1,n-i+1} = 2n-2i+2$	for $2 \leq i \leq n$
$c_{ij} = 5n$	otherwise

The tableau corresponding to this test problem is depicted in Figure (6-1) for *n*=6.

30	30	30	30	**2** **10**	**1** **11**	3
30	30	30	**4** **8**	**3** 9	30	7
30	30	**6** **6**	**5** **7**	30	30	11
30	**8** **4**	**7** **5**	30	30	30	15
10 **2**	**9** **3**	30	30	30	30	19
11 **1**	30	30	30	30	30	11
21	17	13	9	5	1	

Example of 6 x 6 Test Problem Optimal Solution

Figure (6-1)

Notice that the example is constructed in such a way that the given X is the unique optimal solution to the problem. This is obvious, but one can also compute the simplex multipliers and the reduced cost coefficients to demonstrate the optimality of the stated solution.

The initial basic feasible solution that is constructed according to the Northwest Corner Rule for this problem is depicted in Figure(6-2).

3 **30**	30	30	30	10	11	3
7 **30**	30	30	8	9	30	7
11 **30**	**0** **30**	6	7	30	30	11
30	**15** **4**	5	30	30	30	15
2	**2** 3	**13** **30**	**4** **30**	30	30	19
1	30	30	**5** **30**	**5** **30**	**1** **30**	11
21	17	13	9	5	1	

6 x 6 Test Problem Initial Basic Feasible Solution-NW Corner Rule

Figure (6-2)

Notice that the Northwest Corner Rule will produce a solution where the basic cells form a staircase, starting at the northwest corner of the array and ending at the bottom right hand corner. It is clear the initial basic feasible solution provided by the Northwest Corner Rule is among the most expensive basic feasible solutions of the test problem.

Thus, we start with the most expensive solution of the problem and generate the least expensive solution. As expected, the test problem turned out to be an excellent one for testing the program.

Figure (6-3) contains the results of four runs of the test problem, corresponding to n=100,300,500, and 1000.

Size of Matrix ***n* x *n***	**Number of Iterations**	**Total Time**	**Total Cost**	**Average Length of DESJSTAR**
100 x 100	3252	1 minute	1333300	44.1916
300 x 300	18709	6 minutes	35999900	90.13561
500 x 500	40383	24 minutes	166666500	149.8887
1000 x 1000	98637	165 minutes	1333333000	230.132

Results From the Computer Program

Figure (6-3)

The results clearly show that our program can efficiently handle problems with a few hundred sources and an equal number of sinks. The number of iterations and the time required to solve the problem as n approaches 1000 explodes quite rapidly as can be seen from the table.

Recommendations: A few features of the current implementation should be examined further in order to improve the performance of this program. One could reprogram the calculation of the reduced cost coefficient doing away with computing the entire set (exactly $\frac{n^2}{2}$ such coefficients must be computed each iteration) and pivot at the first entry a negative cost is detected. This is likely to reduce the calculations significantly, although it is difficult to predict the exact impact of such strategy, since the number of iterations may increase as a consequence of such alterations of the algorithm. The average length of the DESJSTAR was computed in order to have a better understanding of how big a portion of the tree must be modified (on average) at each step. This gave us some understanding of how complex the tree structure is for the current test problem. The average length of DESJSTAR was found to be relatively small compared to the size of the test problem ($2n$ nodes). More difficult test problems, where there are far more (10 to 100 times) sinks than sources, can be easily developed along the same lines of this test problem. The corresponding trees will have a more complex structure because they will have much more lateral width than the trees of the test problem used in this paper. Therefore, this new test problem would further test the efficiency of the subroutines within this program that are concerned with the tree design.

BIBLIOGRAPHY

Glover, Fred, et al. "A Computation Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems." Management Science. Volume 20, Number 5. January 1974: 793-812.

Gould, Ronald. Graph Theory. Menlo Park, California: Benjamin/Cummings Publishing Co., Inc., 1988.

Luenberger, David G. Linear and NonLinear Programming. 2nd ed. Reading, Massachusetts: Addison-Wesley Publishing Co., 1984.

Murty, Katta G. Linear Programming. Upper Saddle River, New Jersey: Prentice Hall, 1995.

Murty, Katta G. Network Programming. Englewood Cliffs, New Jersey: Prentice Hall, 1992.

Murty, Katta G. Operations Research: Deterministic Optimization Models. Upper Saddle River, New Jersey: Prentice Hall, 1995.

Taha, Hardy A. Operations Research: An Introduction. 5th ed. New York: Macmillan Publishing Co., 1992.

VITA

Name of Author:	**Alissa Michele (Andreichuk) Sustarsic**
Place of Birth:	Date of Birth:

EDUCATION:

- MS in Mathematical Science — May 1999
  University of North Florida
- BS in Mathematics with Minor in Biology — May 1997
  Jacksonville University
  Summa Cum Laude with Departmental Honors in Mathematics

TEACHING EXPERIENCE:

- University of North Florida, Mathematics and Statistics Department
  Instructor in College Algebra, Graduate Teaching Assistant
- University of North Florida, Quest Program, Center of Multicultural Affairs
  Mathematics Instructor   Duties- Planning & Teaching Skills for the Clast Exam

ACHIEVEMENTS:

- UNF Outstanding Graduate Student of the Year 1999
- JU Fred E. Noble Gold Medal for Scholarship-Highest Academic Award at Graduation
- JU College of Arts and Sciences 1996-1997 Outstanding Student
- JU Division of Science and Mathematics Student of the Year 1997
- JU Outstanding Mathematics Student of the Year 1996 & 1997
- George Washington University Summer Program for Women In Mathematics 1996
  Chosen as 1/16 College Senior Women, a 1 Month of Mathematics Intensive Study
- Florida Academic Scholar, JU Trustees Scholarship, Robert C. Byrd Scholarship(JU)

PROFESSIONAL ORGANIZATIONS:

- Pi Mu Epsilon National Mathematics Honor Society: President-UNF
- Phi Kappa Phi National Honor Society
- Green Key Honorary Leadership Society
- Alpha Delta Pi Sorority
- Omicron Delta Kappa National Leadership Honor Society

SPORTS AT JACKSONVILLE UNIVERSITY

- Women's Varsity Soccer: Charter Member, Captain, Scholarship Recipient,
  GTE Academic All District III Team, Sun Belt Conference Academic Team 1996
  Women's Club Soccer: Charter Member, President/Treasurer