5-2000

Network flow algorithms and applications

Randy Lee Collins

To the Graduate Council:

I am submitting herewith a thesis written by Randy Lee Collins entitled "Network flow algorithms and applications." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mathematics.

Yueh-er Kuo, Major Professor

We have read this thesis and recommend its acceptance:

William Wade, Charles Collins

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Randy L Collins entitled "Network Flow Algorithms and Applications." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mathematics.

Yueh-er Kuo

Yueh-er Kuo, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor and
Dean of the Graduate School

NETWORK FLOW
ALGORITHMS AND APPLICATIONS

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Randy Lee Collins

May 2000

ACKOWLEGMENTS

At this time, I wish to thank the faculty and staff of the Department of Mathematics at the University of Tennessee, Knoxville for an enjoyable two years. I especially want to give my thanks and appreciation to Dr. Yueh-er Kuo for the advise through out my work on this thesis. Appreciation is also extended to the committee members, Dr. William Wade and Dr. Charles Collins, for their time and assistance.

I would also like to thank my good friends for their words of encouragement and support. I could not of made it without them.

ABSTRACT

This paper looks at several methods for solving network flow problems. The first chapter gives a brief background for linear programming (LP) problems. It includes basic definitions and theorems. The second chapter gives an overview of graph theory including definitions, theorems, and examples.

Chapters 3-5 are the heart of this thesis. Chapter 3 includes algorithms and applications for maximum flow problems. It includes a look at a very important theorem, Maximum Flow/Minimum Cut Theorem. There is also a section on the Augmenting Path Algorithm. Chapter 4 deals with shortest path problem. It includes Dijsksta's Algorithm and the All-Pairs Labeling Algorithm. Chapter 5 includes information on algorithms and applications for the minimum cost flow (MCF) problem. The algorithms covered include the Cycle Canceling, Successive Shortest Path, and Primal-Dual Algorithms. Each of these chapters 3-5 contain definitions, theorems, and algorithms to solve network flow problems.

Throughout the paper the computer program LINDO is used. It serves a couple of functions. First it is a way of checking each solution. The second use is to expose the reader to a very valuable tool in linear programming.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

CHAPTER ONE

LINEAR PROGRAMMING

§ 1.1: Introduction

Mathematical programming (MP) is a branch of mathematics dealing with techniques for maximizing or minimizing an objective function subject to linear, nonlinear, and integer constraints on the variables. MP originated out of World War II [5]. The word "programming" was a specific plan for various military operations. Later MP became part of what is now called Operations Research (OR). Linear programming (LP) is a special case of MP [7]. LP originated back to the work of George Dantzig in 1947 [12]. For the LP problem, the linear objective function is maximized or minimized subject to linear equality or inequality constraints. It is important to know the basics of LP since network flow problems are a type of LP combined with graph theory. There will be more about graph theory in Chapter Two.

The unknown values that we are trying to solve are called *decision variables*. We can represent them as a vector, $\vec{x} = (x_1, x_2, \ldots, x_n)$. The *objective function* is a function in the decision variables. Thus we have $z(x) = (c_1, c_2, \ldots, c_n)(x_1, x_2, \ldots, x_n)^T$, where $(c_1, c_2, \ldots, c_n)$ is a vector of constants and $(x_1, x_2, \ldots, x_n)^T$ is the transpose of the vector. Another way to write the objective function is $z(x) = c_1x_1 + c_2x_2 + \ldots + c_nx_n$. The function z(x) must satisfy certain restrictions on the decision variables called *constraints*.

The goal of a LP problem is to maximize or minimize the objective function subject to the constraints. The basic problem setup for a LP problem is as follows:

$$\text{Optimize } z(x) \tag{1.1}$$

$$\text{subject to } A\vec{x} = \vec{b}, \text{ where } \vec{x}, \vec{b} \in R^n \tag{1.2}$$

$$\text{where } \vec{x} \geq \vec{0}. \tag{1.3}$$

It is often the case that the constraints are inequalities instead of equalities. If it is the intent to maximize z(x) then the standard LP setup is

$$\text{MAX } z(x)$$

$$\text{subject to } A\vec{x} \leq \vec{b}, \text{ where } \vec{x}, \vec{b} \in R^n$$

$$\text{where } \vec{x} \geq \vec{0}.$$

On the other hand, if it is the intent to minimize z(x) then the standard LP setup is

$$\text{MIN } z(x)$$

$$\text{subject to } A\vec{x} \geq \vec{b}, \text{ where } \vec{x}, \vec{b} \in R^n$$

$$\text{where } \vec{x} \geq \vec{0}.$$

Note: A in all of these cases is a mxn matrix.

When the LP problem is solved , if possible, it is said to have a *feasible solution* as long as all the constraints are satisfied. The feasible solution is a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ such that

$$x_1\vec{P}_1 + x_2\vec{P}_2 + \ldots + x_n\vec{P}_n = \vec{P}_0,$$

where $\vec{P}_1, \vec{P}_2, \ldots, \vec{P}_n$ are column vectors of matrix A and $\vec{P}_o$ is column vector of constants. If the goal is to MIN z(x) then the minimum feasible solution is feasible when it minimizes the objective function.

Theorem 1.1: The set of all feasible solutions of a LP problem is a convex set.

Proof: We show that every convex combination of any two feasible solutions is also feasible solution. Let $A\vec{x}_1 = \vec{b}$ and $A\vec{x}_2 = \vec{b}$, where $\vec{x}_1$ and $\vec{x}_2 \geq 0$. For $0 \leq \alpha \leq 1$, let

$\vec{x} = \alpha\vec{x}_1 + (1-\alpha)\vec{x}_2$, $\vec{x} \geq 0$. Thus,

$$\begin{aligned} A\vec{x} &= A(\alpha\vec{x}_1 + (1-\alpha)\vec{x}_2) \\ &= \alpha A\vec{x}_1 + (1-\alpha)A\vec{x}_2 \\ &= \vec{b} \quad [7]. \end{aligned}$$

If there is no solution what satisfies all the constraints then it is termed *infeasible*. An *optimal solution* is the feasible solution that optimizes the objective function. If no solution can be considered optimal because there is always a better solution, then the problem is called *unbounded.*

§ 1.2: Solving LP Problems

There are several methods used to solve LP problems. This section looks at two methods: graphical and simplex method. LINDO (Linear, Interactive and Discrete Optimizer), a computer program, will also be used to solve LP problems. The graphical method is used with simple LP formulations. It involves graphing the constraints on a single axis and identifying the feasible region. The *feasible region* (denoted by K) is the region what satisfies all the constraints. All of the corners points of feasible region are checked for the optimal solution. The corner points are sometimes called the *extreme points*.

Example 1.2.1: A company has 3 plants and makes 2 products. Using the information from Table 1.2.1, what mix of products 1 and 2 will be most profitable?

<u>Solution:</u> Let x_j = number of items of product j.

LP formulation: Max $z = 3x_1 + 5x_2$

ST $x_1 + 0x_2 \leq 4$

$0x_1 + 2x_2 \leq 12$

$3x_1 + 2x_2 \leq 18$

where $x_1, x_2 > 0$

Table 1.2.1 Data for Example 1.2.1

PLANT	PRODUCTS		AVAILIABLE CAPACITY
	1	2	
1	1%	0	4%
2	0	2%	12%
3	3%	2%	18%
UNIT PROFIT	\$3	\$5	

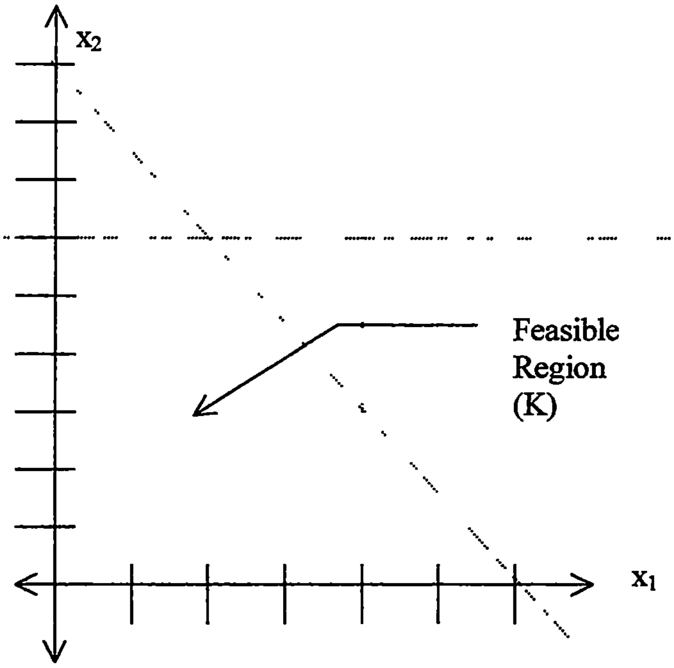


Figure 1.2.1 Graph of LP Formulation Showing Feasible Region

The extreme points are: { (0,0), (4,0), (0,6), (4,3), (2,6) }, as seen in Figure 1.2.1. By inspection the optimal solution is $z^* = 36$ where $x_1 = 2$ and $x_2 = 6$. The company should produce 2 units of product one and 6 units of two to produce a maximum profit of $36.

The graphical method is not useful for large models, thus the simplex method is used. The basic idea is to start with a corner point of the feasible region (usually the origin) and move to adjacent corner point so as to increase z(x) the fastest. This process is continued until z(x) cannot be increased further. The simplex moves to an adjacent corner point feasible solution by increasing one of the x_i's from zero and adjusting the rest so as to satisfy the constraints, that forces one of the nonzero x_i's to become zero [10].

To setup the simplex algorithm we first identify $x_{i0}=b_i$ and $x_{ij}=a_{ij}$ for all $i=1,2,\ldots,m$ and $j=1,2,\ldots,n$. We also define $z_0=\sum_{i=1}^{m} c_i x_{i0}$ and $z_j=\sum_{i=1}^{m} c_i x_{ij}$ for $j=1,2,\ldots,n$. The simplex is setup in a table form called the *tableau*. See Table 1.2.2 for the setup of the Simplex Tableau.

Theorem 1.2: If any basic feasible solution satisfy the condition $z_j - c_j \le 0$ holds for all $j=1,2,\ldots,n$ then (1.1) and (1.2) constitutes a minimum feasible solution. If the problem is to maximize the objective function if all $z_j - c_j \ge 0$ then the maximum optimal solution is found.

Table 1.2.2 Initial Step of Tableau

i	Basis	c	P_0	c_1	c_2							
				P_1	P_2	.	P_l	.	P_m	P_{m+1}	.	P_n
1	P_1	c_1	x_{10}	1	0		0		0	$x_{1,m+1}$		x_{1n}
2	P_2	c_2	x_{20}	0	1		0		0	$x_{2,m+1}$		x_{2n}
.	.	.	.	.	.		.		.	.		.
l	P_l	c_l	x_{l0}	0	0		1		0	$x_{l,m+1}$		x_{ln}
.	.	.	.	.	.		.		.	.		.
m	P_m	c_m	x_{m0}	0	0		0		1	$x_{m,m+1}$		x_{mn}
m+1			z_0	0	0		0		0	$z_{m+1}-c_{m+1}$		z_n-c_n

After setting up the tableau (table for simplex algorithm) do the following steps:

1. Test $z_j - c_j$ to see if $z_j - c_j \leq 0$ for all j (this is the minimum feasible solution).
2. If $z_j - c_j \geq 0$, select largest vector to be introduced into basis (i.e. select vector with $\max_j(z_j - c_j)$).
3. Since a vector is introduced into basis we remove a vector from basis by finding $\theta = \min_i \frac{x_{i0}}{x_{ik}}$, where k corresponds to vectors selected in step 2.
4. Transform the tableau by complete elimination and continue the process.

Example 1.2.2: Use the simplex method to solve Example 1.2.1.

Solution: Introduce three slack variable: x_3, x_4, and x_5 to form a standard LP problem (i.e. equality constraints). There is now 5 variables and 3 equations, so two of the variables are arbitrary. The objective function is rewritten in canocial form as follows: $z - 3x_1 - 5x_2 = 0$. The *pivot* value is determined by the row that has the smallest θ and the column of the most negative coefficient of the objective function. The simplex ends when there are no more negative coefficients of the objective function. The basic variable on the pivot row leaves and tableau and the variable corresponding to pivot column enters the tableau. The simplex tableaus are shown in Table 1.2.3 to Table 1.2.5.

The simplex method ends in Table 1.2.5 since all the z_j - c_j are positive. From the tableau in Table 1.2.5, $x_1 = 2$, $x_2 = 6$, $x_3 = 2$, $x_4 = 0$, and $x_5 = 0$. The maximum z(x) is 36.

Table 1.2.3 First Tableau of Simplex Method (P_4 leaving/P_2 entering)

				3	5	0	0	0	
i	Basis	c	P_0	P_1	P_2	P_3	P_4	P_5	θ
1	P_3	0	4	1	0	1	0	0	----
2	P_4	0	12	0	2	0	1	0	6
3	P_5	0	18	3	2	0	0	1	9
4	----		0	-3	-5	0	0	0	

Table 1.2.4 Second Tableau after One Iteration (P_5 leaving/P_1 entering)

i	Basis	c	P_0	P_1	P_2	P_3	P_4	P_5	θ
1	P_3	0	4	1	0	1	0	0	4
2	P_2	5	6	0	1	0	½	0	----
3	P_5	0	6	3	0	0	-2	1	2
4	----		30	-3	0	0	5/2	0	

Table 1.2.5 Third Tableau after Two Iteration

i	Basis	c	P_0	P_1	P_2	P_3	P_4	P_5	θ
1	P_3	0	2	0	0	1	1/3	-1/3	
2	P_2	5	6	0	1	0	½	0	
3	P_1	3	2	1	0	0	-1/3	1/3	
4	----		36	0	0	0	3/2	1	

The simplex method is a very useful and powerful tool for solving LP problems but an even more powerful tool is the program LINDO. Refer to the appendix for information on how to use LINDO. LINDO is a computer program designed to solve LP problems. It is a very fast and efficient way to solve them [11].

Example 1.2.3: Solve Example 1.2.1 by using LINDO.

From the LINDO Reports Window (Figure 1.2.2) we can see that the objective function value is \$36 with x_1=2 and x_2=6. See Figure 1.2.2 for the output of LINDO.

§ 1.3: Types of LP Problems

There are many applications of LP problems. This section discusses a few such applications. In particular, this section covers the transportation, assignment, and transshipment problem. These types of LP formulations can be found in [12] and [4]. It will be shown later how the problems can be modeled as network flow problems.

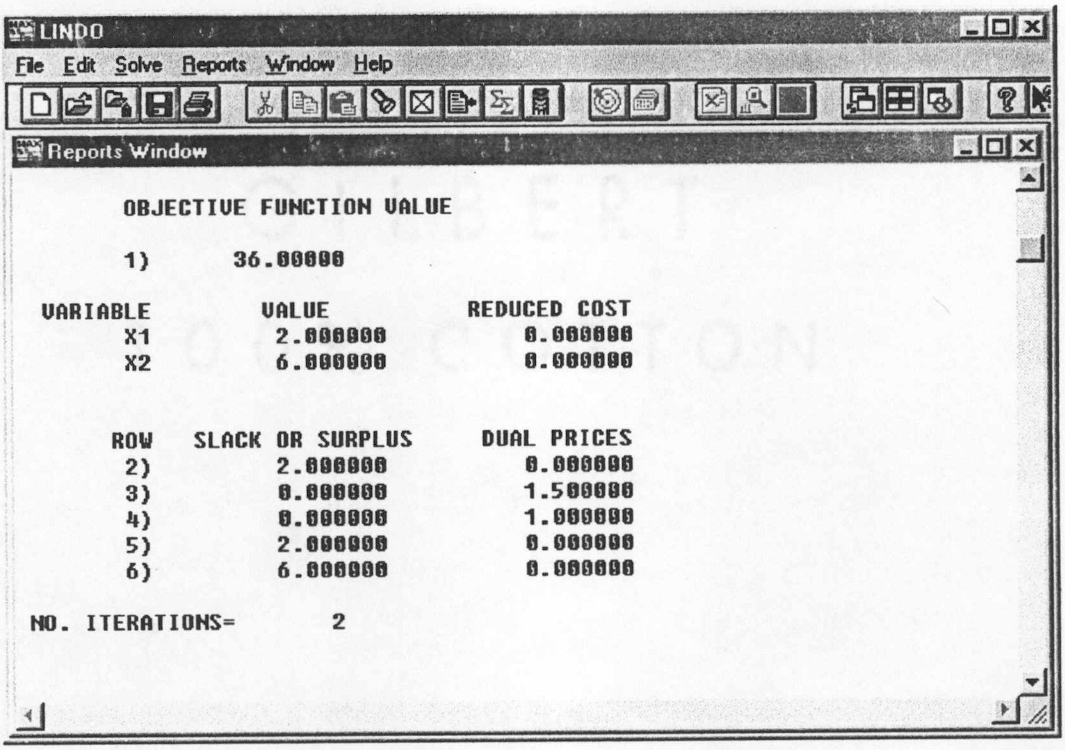


Figure 1.2.2 LINDO Solution of Example 1.2.1

Transportation Problem

The transportation problem involves shipping a product (supply) to a location (demand). The set of m supply points from where a product is shipped can supply at most s_i units. The n demand points can receive at least d_j units of a product. The cost associated with shipping the product is denoted by c_{ij}. The LP formulation is as follows:

$$\text{Transportation Problem:} \quad \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

subject to $$\sum_{j=1}^{n} x_{ij} \le s_i$$

$$\sum_{i=1}^{m} x_{ij} \ge d_j$$

where $x_{ij} \ge 0$

This is a balanced transportation problem if $\sum_{i=1}^{m} s_i = \sum_{j=1}^{n} d_j$.

Assignment Problem

The assignment problem is a subset of the transportation problem. It is a balanced transportation problem. It also has the special property of the x_{ij}'s being 0 or 1. Let x_{ij} = 1 if i is assigned to meet demand j, and let $x_{ij} = 0$ if i is not assigned to meet demand j. The LP formulation is as follows:

Assignment Problem: $$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ for } i = 1,2,\ldots,n$$

$$\sum_{i=1}^{n} x_{ij} = 1 \text{ for } j = 1,2,\ldots,n$$

where $0 \le x_{ij} \le 1$

Transshipment Problem

The transshipment problem is also a special case of the transportation problem, but this time shipments are allowed between supply points or between demand points. A *transshipment point* is a point through which a product can be both received and shipped to other points. The supply point can only ship products, and the demand point can only receive products. The LP formulation is as follows:

Transshipment Problem: $\min \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$

subject to $\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = s_i$

$$\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = 0$$

$$\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = d_i$$

where $0 \leq x_{ij} \leq u_{ij}$

CHAPTER TWO

INTRODUCTION TO NETWORKS

Network theory concerns a class of LP problems having a very special network structure. The combinatorial nature of this structure has resulted in a development of very efficient algorithms that combine ideas on data structures with algorithms from computer science, and mathematics from operations research [9]. Before developing the algorithms the terminology of networks must be covered. Some of the definitions and proofs of some theorems in this chapter may be found in [1], [2], [3], [6], and [7].

§ 2.1: Definitions

A *network*, is made up of nodes and arcs. We will denote the graph or network using G(N,A), where N is the set of nodes and A is the set of arcs. The *nodes* are vertices or points of the network. An *arc* consists of an ordered pair of nodes that represents a possible direction of motion that may occur. It is denoted by (i,j). The *initial node* is a starting node which we will denote the initial node with s. The *terminal node* is a ending node which we will denote the terminal node with t. Often the nodes s and t are called source and sink respectively.

A sequence of arcs such that every arc has exactly one node in common with the previous arc is called a *chain*. For example this set of arcs forms a chain: { (s,i), (j,i), (j,t) }. A *path* is a chain in which the terminal node of each arc is identical to the initial node of the next arc. Note a chain is a path, but a path is not a chain. For example, this set of arcs form a path: { (s,i), (i,j), (j,t) }. A *circuit* has a path from nodes s to t and an arc from t to s. An example of a circuit is { (s,i), (i,j), (j,t), (t,s) }. A *cycle* is a loop in the path. A cycle is a closed path. For example, this set of arcs form a cycle: { (s,i), (i,t), (t,j), (j,s) }. All of these examples can be seen in Figure 2.1.1.

A *connected graph* means there exists a chain between every pair of nodes. If this is the case it is termed weakly connected. A strongly connected graph has a directed path from each node to every other node (see Figure 2.1.2). A graph is termed a *subgraph* , **G'(N',A')**, if N' $\subseteq$ N and A' $\subseteq$ A. A *bipartite graph* is a graph **G=(N,A)** that can be partitioned into two subsets N_1 and N_2.

§ 2.2: Matrix Representations of Networks

Networks can be represented in a matrix form. One such matrix is called the node-arc incidence matrix. It is a n x m matrix. There is one row for each node and one column for each arc (i,j). If there is an arc connecting two nodes we use a +1, but if there is not an arc, a 0 is used.

Only 2m out of the nm entries are nonzero. It is because of this that the node-arc incidence matrix is not an efficient method of storage. The matrix does have some important properties. Each column has entries that are either –1, 0, or +1. Secondly, the number of +1's in a row equals the out-degree (i.e. the number of arcs leaving the node). Finally, the number of –1's in a row equals the in-degree (i.e. the number of arcs entering the node).

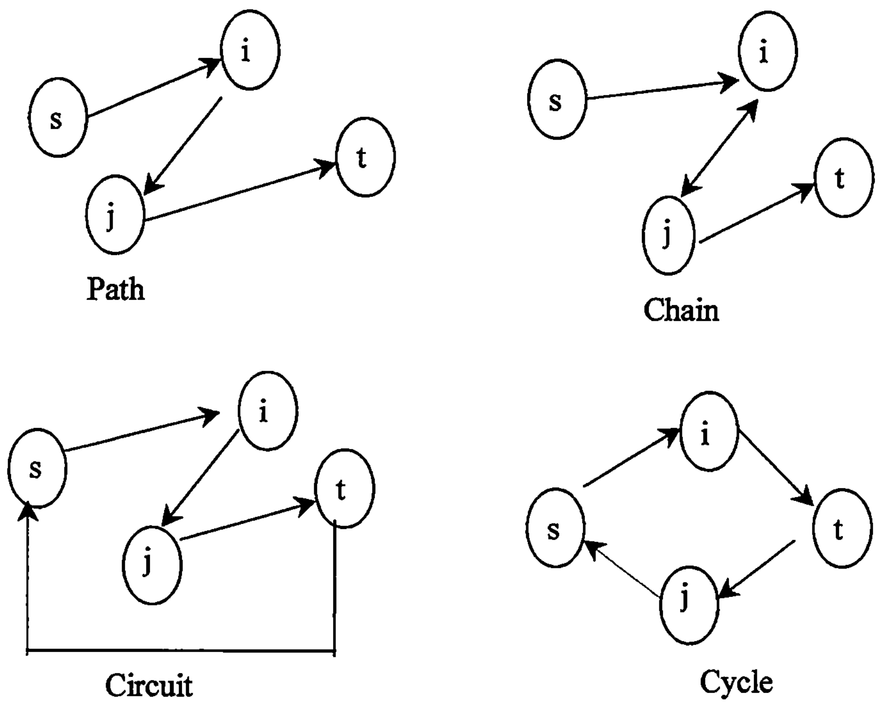


Figure 2.1.1 Examples of Different Network Types

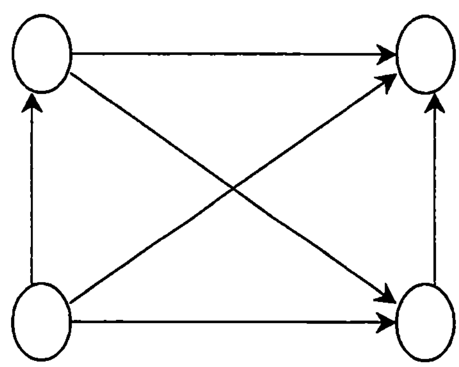


Figure 2.1.2 An Example of Connected Network

Example 2.2.1 Find the corresponding node-arc incidence matrix for the following graph in Figure 2.2.1. The corresponding node-arc incidence matrix is shown in Table 2.2.1.

There are also two important theorems dealing with node-arc incidence matrices, but first we introduce a new definition. A matrix, A, is *unimodular* if the determinant of each basis matrix of A has a value of +1 or −1. The *basis matrix* is a matrix who's columns are linearly independent. An *integer matrix*, A, is a matrix where each a_{ij} entry is an integer.

Theorem 2.2.1 [Unimodularity Theorem]: Let A be an integer matrix with linearly independent rows. Then the following three conditions below are equivalent.

(a) A is unimodular.

(b) Every basic feasible solution defined by the constraints $A\vec{x} = \vec{b}$ and $\vec{x} \geq 0$, is integer for any integer vector $\vec{b}$.

(c) Every basis matrix B of A has an integer inverse B^{-1}.

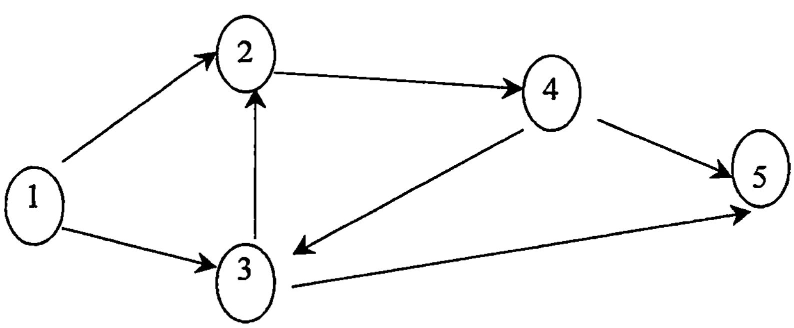


Figure 2.2.1 Network for Example 2.2.1

Table 2.2.1 Node-Arc Incidence Matrix

Nodes \ Arcs	(1,2)	(1,3)	(2,4)	(3,2)	(3,5)	(4,3)	(4,5)
1	+1	+1	0	0	0	0	0
2	-1	0	+1	-1	0	0	0
3	0	-1	0	+1	+1	-1	0
4	0	0	-1	0	0	+1	+1
5	0	0	0	0	-1	0	-1

Proof: (a implies b): Each basis feasible solution $\vec{x}_B$ has an associated basic matrix B for which $B\vec{x}_B = \vec{b}$.

By Cramer's rule, any component x_j of the solution $\vec{x}_B$ will be of the form $x_j = \frac{\det(A)}{\det(B)}$.

We obtain the integer matrix in this formula by replacing the j^{th} column of B with the vector $\vec{b}$. Since, A is unimodular (given), det(B) is ± 1, so x_j is integer.

(b implies c): Let B be a basis matrix of A. Since B has a nonzero determinant, its inverse B^{-1} exists. Let $\vec{e}_j$ denote the j^{th} unit vector. Let $D = B^{-1}$ and $\vec{D}_j$ denote the j^{th} column of D. We will show that the column vector $\vec{D}_j$ is integer for each j whenever condition (b) holds. Select an integer vector $\vec{\alpha}$ so that $\vec{D}_j + \vec{\alpha} \geq 0$. Let $\vec{x} = \vec{D}_j + \vec{\alpha}$.

Notice that $B\vec{x} = B(\vec{D}_J + \vec{\alpha}) = B(B^{-1}\vec{e}_J + \vec{\alpha}) = \vec{e}_J + B\vec{\alpha}$. Multiplying by $D = B^{-1}$, we see that $\vec{x} = \vec{D}_J + \vec{\alpha}$. Since $\vec{e}_J + B\vec{\alpha}$ is integer, condition (b) implies that $\vec{D}_J + \vec{\alpha}$ is integer. We find that $\vec{D}_J$ is integer.

(c implies a): Let B be a basis matrix of A. By assumption, B is an integer matrix, so det(B) is an integer. By condition (c), B^{-1} is an integer matrix; consequently, det (B^{-1}) is also an integer. Since $B B^{-1} = I$, det (B) det (B^{-1}) = 1, which implies that det (B^{-1}) = ± 1 [1].

Theorem 2.2.2 The node-arc incidence matrix A of a directed network is totally unimodular.

<u>Proof:</u> To prove the theorem, we need to show that every square sub-matrix F of A of size k has determinant 0, +1, -1. We establish this result by performing induction on k. Since each element of N is 0, +1, or –1, the theorem is true for k=1. Now suppose it holds for k, we show it is true for k+1. Let F be any (k+1)x(k+1) sub-matrix of A. F satisfies one of 3 properties: (1) F contains a column with no nonzero element, (2) every column of F has exactly two nonzero elements, in which case, one of these must be a +1 and the other a –1, and (3) some column F_J has exactly one nonzero element, in the ith row. We now prove all three cases. The first is trivial, det(F)=0.

<u>Case 2:</u> rows of F are linearly dependent, thus det (F) = 0.

<u>Case 3:</u> let F' be a sub-matrix of F, then F' is obtained by deleting row and column of F. Thus, det (F) = $\pm$ det (F'). Since det (F') = 0,-1, or +1, then det (F) = 0, -1, or +1 [1]

As stated earlier the node-arc incidence matrix is not efficient for storage; thus we have another matrix representation called the node-node adjacency matrix, or simply the adjacency matrix. Here we can store the matrix in a n x n matrix. For each arc we place a 1 at each i,j location and a zero elsewhere. In doing this there are n^2 entries with m being nonzero.

Example 2.2.2: Find the corresponding node-node adjacency matrix for the same graph in Figure 2.2.1. The corresponding adjacency matrix is shown in Table 2.2.2.

Table 2.2.2 Adjacent Matrix

Nodes \ Nodes	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	0
3	0	1	0	0	1
4	0	0	1	0	1
5	0	0	0	0	0

CHAPTER THREE

MAXIMUM FLOW PROBLEM

In some situations the number of quantities that pass through an arc may have a limited capacity so it would be wise to know the most that can be sent at one time. For other situations it would be cheaper to ship the highest capacity to cut down on cost factors. In situations like these we can use the maximum flow method for the most effective flow of quantities. Our goal is to send the most allowable x_{ij} through the network given a limiting capacity. Information used in this chapter was aided by the following sources: [1], [3], and [7].

The amount of *flow* from outside the initial node that flows into the source node is denoted by *f*. There is an associated flow though the network called a flow vector, denoted by x, consisting of all arc flows x_{ij} $\forall$ $(i,j) \in A$. The *maximum capacity* of x is denoted by u_{ij}. We find *f* through algorithms in which we send the largest x_{ij} given the maximum capacity, u_{ij}. The feasible flow is represented by x^o.

It should be noted if an arc (i,j) does not exist we let $u_{ij} = 0$. In other words all arcs are possible but flow is restricted to the arcs in which $u_{ij} > 0$. We also cannot exceed any of the u_{ij}'s. Sometimes we also have a minimum capacity, denoted by l_{ij}. It is often the case that $l_{ij} = 0$.

Theorem 3.1: The flow into the source node s equals the flow out of the terminal node t.

Theorem 3.2: If there exists no chain of arcs, each with positive capacity, joining nodes s to t, then the maximum flow is zero.

Theorem 3.3: The maximum flow is positive if there exists a chain of arcs, each with positive capacity, joining node s to t.

Maximize: f	(3.1)
subject to $\sum_{j=1}^{m} x_{ij} - \sum_{j=1}^{m} x_{ji} = \begin{cases} f & \text{if } i = 1 \\ 0 & \text{if } i \neq 1, m \\ -f & \text{if } i = m \end{cases}$	(3.2)
where $0 \leq x_{ij} \leq u_{ij} \quad \forall \ (i,j) \in A$	(3.3)

A basic fact about the networks is that there is a conservation of flow (i.e. what ever enters a node leaves the node). Thus there is no stockpiling. It is important to note that if (3.2) is equal to f then conservation of flow is at the source; if (3.2) is equal to 0 then conservation of flow is at the intermediate nodes; if (3.2) is equal to –f then conservation of flow is at the sink.

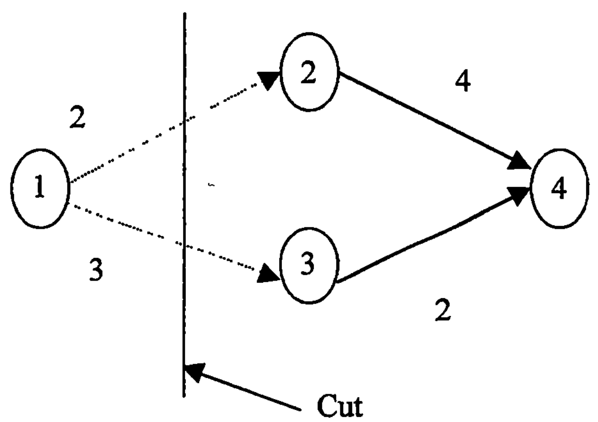
We consider the maximum flow problem subject to the following assumptions:

1. The network is directed.
2. All capacities are nonnegative integers.

3. The network does not contain a directed path from node s to node t composed only of infinite capacity arcs.
4. Whenever arc (i,j) belongs to A, then arc (j,i) belongs to A.
5. The network does not contain parallel arcs (i.e. two or more arcs with the same tail and head nodes).

§ 3.1: Maximum Flow/Minimum Cut Theorem

Next we introduce the Maximum Flow /Minimum Cut Theorem. First we divide the nodes into two distinct sets, X and Y, where $X \in N$ and $Y=N-X$. We let (X,Y) denote the set of arcs from $i \in X$ to $j \in Y$. A cut is a line drawn on the network that separates X from Y. The arcs that are cut are known as the *cut-set*. Figure 3.1.1 shows an example of a cut and the cut-set. The cut-set is denoted by dotted lines in Figure 3.1.1. A network has a finite number of cuts.



X = { 1 }

Y = { 2,3,4 }

Note: The cut-set is (1,2) and (1,3)

Figure 3.1.1 An Example of the Cut-Set

The *capacity* of a cut is defined as $f(X,Y) = \sum_{\substack{i \in X \\ j \in Y}} u_{ij}$ or quite simply the sum of the capacities of the arcs in the cut. Based on the example of Figure 3.1, we see that the capacity of the cut is the sum of the capacities of arc (1,2) and (1,3), which is 5. If an arc is at maximum capacity it is called *saturated* (i.e. $x_{ij} = u_{ij}$).

Theorem 3.1.1 Given any partition of the nodes into two classes, where the first class includes node s and the second class includes node t, then a feasible solution is maximum if every arc is saturated that joins a node of the first class to node of second class.

Lemma 3.1.1 The flow value f of any feasible solution is less than or equal to the capacity f(X,Y) of any cut-set.

Proof: For any cut (X,Y) we sum the first two constraints of (3.2) for all $i \in X$ to obtain

$$f = \sum_{i \in X} \left(\sum_{j} x_{ij} - \sum_{j} x_{ji} \right)$$

This can be rewritten as

$$f = \sum_{\substack{i \in X \\ j \in X}} x_{ij} + \sum_{\substack{i \in X \\ j \in Y}} x_{ij} - \sum_{\substack{i \in X \\ j \in X}} x_{ji} - \sum_{\substack{i \in X \\ j \in Y}} x_{ji}$$

or

$$f = \sum_{\substack{i \in X \\ j \in Y}} x_{ij} - \sum_{\substack{i \in X \\ j \in Y}} x_{ji} + \sum_{\substack{i \in X \\ j \in X}} x_{ij} - \sum_{\substack{i \in X \\ j \in X}} x_{ji}$$

Thus,

$$f = \sum_{\substack{i \in X \\ j \in Y}} x_{ij} - \sum_{\substack{i \in X \\ j \in Y}} x_{ji} \qquad (3.4)$$

Since $x_{ji} \geq 0$ and each $x_{ij} \leq u_{ij}$,

$$f \le \sum_{\substack{i \in X \\ j \in Y}} x_{ij} \le \sum_{\substack{i \in X \\ j \in Y}} u_{ij} = f(X, Y)$$

Thus, the maximum flow is bounded above by the capacity of the arbitrary cut (X,Y), and hence f must be bounded above by the minimal cut capacity [7]

Theorem 3.1.2 [Maximum Flow/Minimum Cut Theorem]: For any network the maximal flow value from node s to node t is equal to the minimum cut capacity.

Proof: Since all arc capacities are finite a maximum flow exists. It could be zero, but let f ' be the value of maximum flow. We now look at the cut (X ', Y ') for f ' and the corresponding arc flow x_{ij}.

Suppose X ' contains both node s and node t. Thus we can find a path from node s to node t such that for any arc in the path either $u_{ij} - x_{ij} > 0$ or $x_{ji} > 0$. Thus we could find a flow greater than f '. Thus $s \in X'$ and $t \in Y'$ and (X ',Y ') is a cut. Note that $x_{ij} = u_{ij}$ when $i \in X'$ and $j \in Y'$, and $x_{ji}=0$ for $i \in X'$ and $j \in Y'$. Thus (3.4) becomes

$$f = \sum_{\substack{i \in X' \\ j \in Y'}} x_{ij} - \sum_{\substack{i \in X' \\ j \in Y'}} x_{ji} = \sum_{\substack{i \in X' \\ j \in Y'}} u_{ij} = f(X', Y').$$

But by Lemma 3.1.1 $f \le f(X,Y)$ for all cuts (X ', Y ') and since f = f(X ', Y '), then (X ', Y ') must yield the minimal cut capacity and thus,

$$\max f = f' = \min f(X,Y) = f(X', Y') \ [7].$$

We now use Theorem 3.1.2 to solve a maximum flow network.

Example 3.1.1: Find the maximum flow for the following network in Figure 3.1.2 using Theorem 3.1.1.

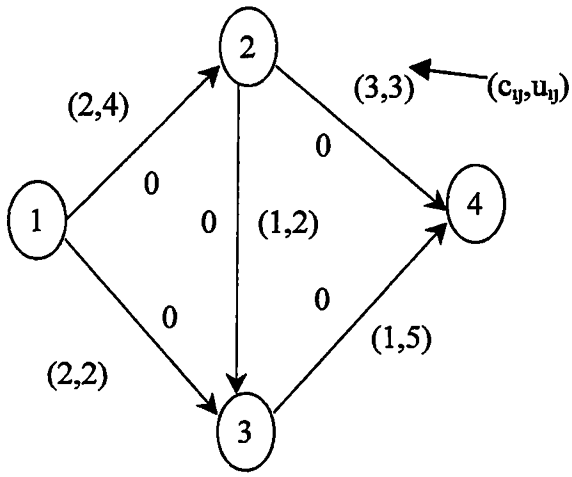


Figure 3.1.2 Network for Example 3.1.1

Table 3.1.1 shows all the possible cut sets for this example. Thus by Theorem 3.1.1, the maximum flow is 6. The problem is that this can become very tedious, thus we must come up with some better methods to find the maximum flow.

§ 3.2: Ford & Fulkerson Algorithm

We now look at a labeling method for finding the maximum flow. It was developed by Ford and Fulkerson in 1956. They were the first to study the maximum flow problem from a computational viewpoint [6]. Below is an algorithm we will use to obtain the maximal flow. It is known as the Ford & Fulkerson Method.

Table 3.1.1 Possible Cut Sets

X	Y	$\sum u_{ij}$
1	2,3,4	6
2	1,3,4	9
3	1,2,4	9
4	1,2,3	8
1,2	3,4	7
1,3	2,4	11

Ford & Fulkerson Algorithm:

1. Set x_{ij}'s to zero.

2. Start with the source node s and label it $[-,\infty]$. This means an unlimited amount of commodities are available. The general labeling convention is defined by the following: $[i\pm, \Delta_j]$ where Δ_j is the positive number representing the change in the capacities and i + represents an increase in flow by an amount Δ_j from node i to j and i – represents a decrease in flow by an amount Δ_j from node j to i.

3. Label the rest of the nodes using the labeling convention and the following two rules:

   (a) If $x_{ij} < u_{ij}$ assign the label $[i+, \Delta_j]$ to node j, where $\Delta_j = \min(\Delta_i, u_{ij}-x_{ij})$ or

   (b) If $x_{ji} > 0$, assign the label $[i-, \Delta_j]$, where $\Delta_j = \min(\Delta_i, x_{ji})$.

4. At node t, the value of Δ_j is the amount we send through the path from s to t.

5. The process ends when a path from the initial node to the terminal node cannot be found.

We now solve the same network from Example 3.1.1 use the labeling method.

Example 3.2.1: Find the maximum flow for the following network in Figure 3.2.1. The iterations for this algorithm is given in Table 3.2.1 to Table 3.2.3. On the tables a * represents a possible path from source to sink for the network.

The algorithm ends when there are no more paths from s to t. We have send a total of six units through the network, this is our maximum flow. The optimal solution is the following: $x_{12}=4$, $x_{13}=2$, $x_{23}=1$, $x_{24}=3$, and $x_{34}=3$.

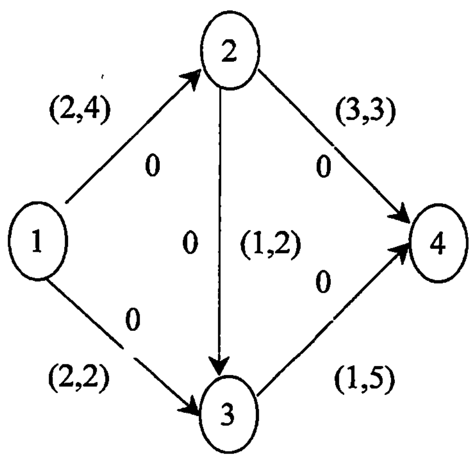


Figure 3.2.1 Network for Example 3.2.1

Table 3.2.1 Initial Setup of Problem

(i,j)	x_{ij}	u_{ij}	Δ_i	$u_{ij}-x_{ij}$	Δ_j	Label
(1,2)	0	4	∞	4	4	[1+,4]
(1,3)	0	2	∞	2	2	[1+,2]*
(2,3)	0	2	4	2	2	[2+,2]
(2,4)	0	3	4	3	3	[2+,3]
(3,4)	0	5	2	5	2	[3+,2]*

Table 3.2.2 Labels after 2 Units of Flow Along Path 1-3-4

(i,j)	x_{ij}	u_{ij}	Δ_i	$u_{ij}-x_{ij}$	Δ_j	Label
(1,2)	0	4	∞	4	4	[1+,4]*
(1,3)	2	2	----	----	----	----
(2,3)	0	2	4	2	2	[2+,2]
(2,4)	0	3	4	3	3	[2+,3]*
(3,4)	2	5	2	3	2	[3+,2]

Table 3.2.3 Labels after 3 Units of Flow Along Path 1-2-4

(i,j)	x_{ij}	u_{ij}	Δ_i	u_{ij}-x_{ij}	Δ_j	Label
(1,2)	3	4	∞	1	1	[1+,1]*
(1,3)	2	2	----	----	----	----
(2,3)	0	2	1	2	1	[2+,1]*
(2,4)	3	3	----	----	----	----
(3,4)	2	5	1	3	1	[3+,1]*

We have seen a labeling method for finding the maximum flow; we now look at the same algorithm but with networks. Ford and Fulkerson developed the algorithm which works by improving upon an initial flow incrementally along some path [9], this approach has become known as the augmenting path algorithm. To solve the augmenting path algorithm we will deal with residual networks. The idea is that we will measure flow in terms of increments. We can think of a residual network as a remaining flow network. So is an arc (i,j) has a flow x_{ij}, then it is still possible to send $u_{ij} - x_{ij}$ units of flow through the arc. We can also send flow backwards, thus canceling the flow (i.e. send a flow of x_{ij} from node j to node i).

The *residual capacity* is the min r_{ij} of any arc in the path, where $r_{ij} = u_{ij} - x_{ij}$. A *residual network G(x)* corresponds to a flow vector x, where x is defined as follows: replace arc (i,j) by two arcs (i,j) and (j,i). The arc (i,j) has cost c_{ij} and residual capacity r_{ij}

. The arc (j,i) has cost $c_{ji} = - c_{ij}$ and residual capacity $r_{ji} = x_{ij}$. An *augmenting path* is a directed path from the source to the sink in the residual network.

Whenever the network contains an augmenting path, we can send additional flow from the source to sink. This is the basic concept behind the augmented path algorithm. When we augment flow we add addition units if we are going in the same direction as the original arc. We subtract units if we are going in the opposite direction as the original arc. The process terminates when G(x) does not contain a directed path from node s to node t.

Theorem 3.2.1: A flow x^o is a maximum flow if and only if the residual network $G(x^o)$ contains no augmenting path.

Augmenting Path Algorithm:

1. Set x_{ij}'s= 0.
2. Identify the augmenting path P from node s to node t.
3. Set $\delta = \min \{ r_{ij} : (i,j) \in P \}$.
4. Augment δ units of flow along P and update the G(x) network.

Example 3.2.2: Find the maximum flow for the following network in Figure 3.2.2 using the augmenting path algorithm. The iterations are shown in Figures 3.2.3 to 3.2.8.

The algorithm now ends by Theorem 3.2.1 since there will be no more augment paths in residual network. We see from the network that f = 6. The optimal x_{ij}'s are as follows: x_{12}=4, x_{13}=2, x_{23}=1,x_{24}=3, x_{34}=3.

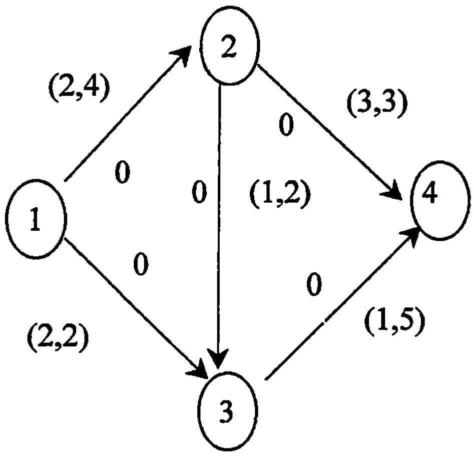


Figure 3.2.2 Network for Example 3.2.2

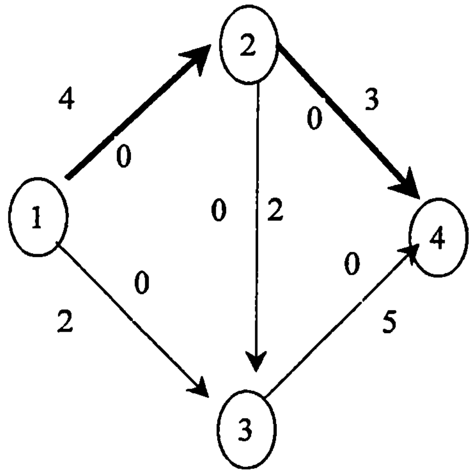


Figure 3.2.3 Residual Network Showing Path For Example 3.2.2

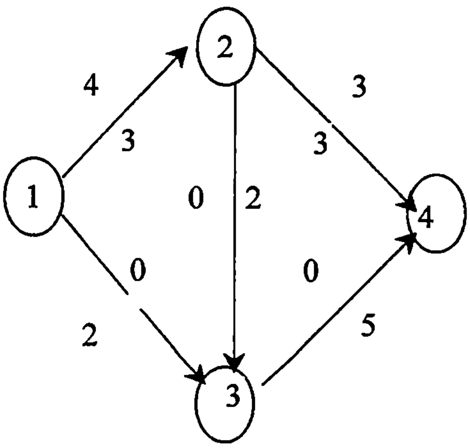


Figure 3.2.4 Updated Network after Augmenting 3 Units

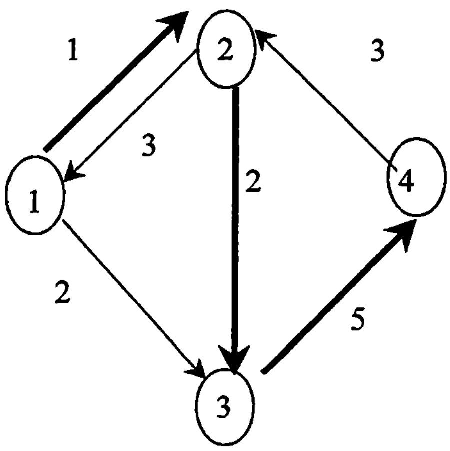


Figure 3.2.5 New Residual Network Showing a Path for Example 3.2.2

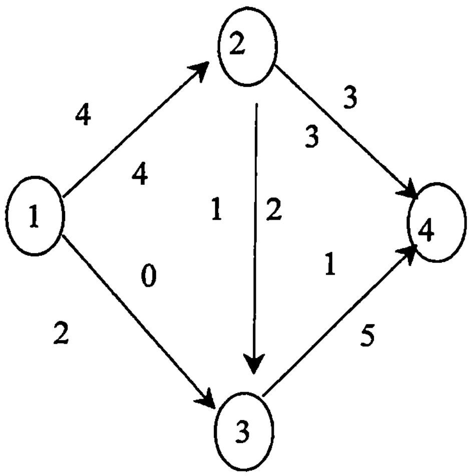


Figure 3.2.6 Updated Network
after Augmenting 1 Unit

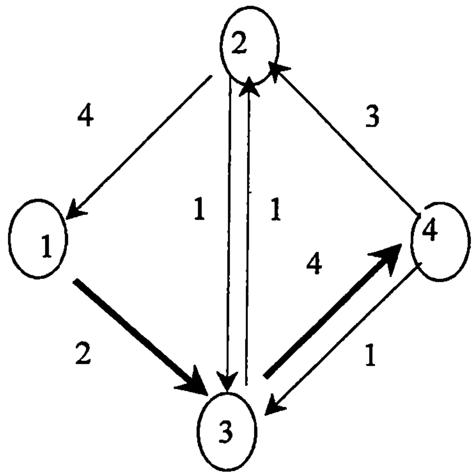


Figure 3.2.7 New Residual Network
Showing a Path for Example 3.2.2

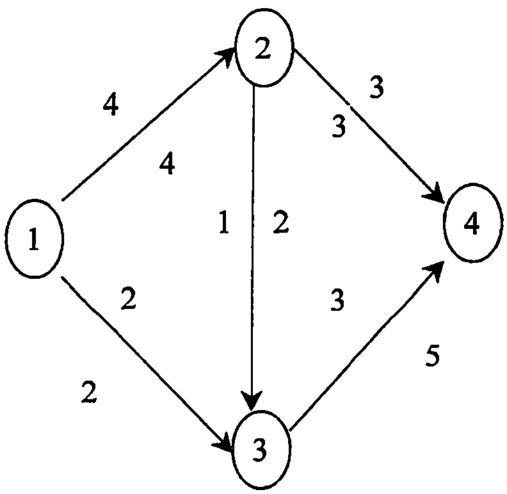


Figure 3.2.8 Updated Network after Augmenting 2 Units

We now look at entering the problem in LINDO to verify the results. As seen in Figure 3.2.9 we must first add a dummy arc with capacity zero to set up LP formulation in Figure 3.2.10. The arc flow of this arc will be the value we wish to maximize since the flow that enters a network leaves the network. From the LINDO Report Window (Figure 3.2.11) on the next page we see that the optimal solution is z*=6 with $x_0=6$, $x_{12}=4$, $x_{13}=2$, $x_{23}=1$, $x_{24}=3$, and $x_{34}=3$.

§ 3.3: Application

A production process indicates the various paths that a product can take on its way to assembly through the plant. The numbers beside each arc represents a the upper limit on items per hour that can be processed at the station. What is the maximum number of parts per hour that the plant can handle? Which operations should be improved?

We solve the residual network by the augmenting path algorithm. We first setup the network for this example (see Figure 3.3.1).

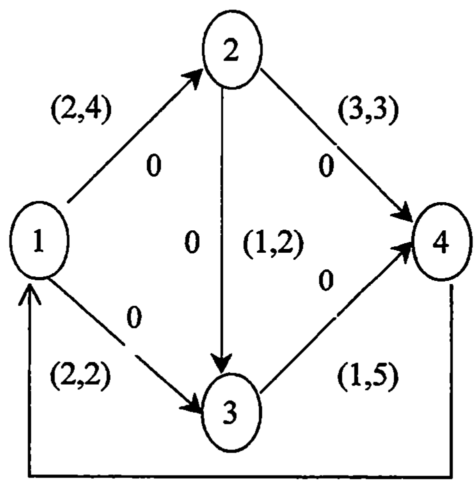


Figure 3.2.9 Network for LINDO

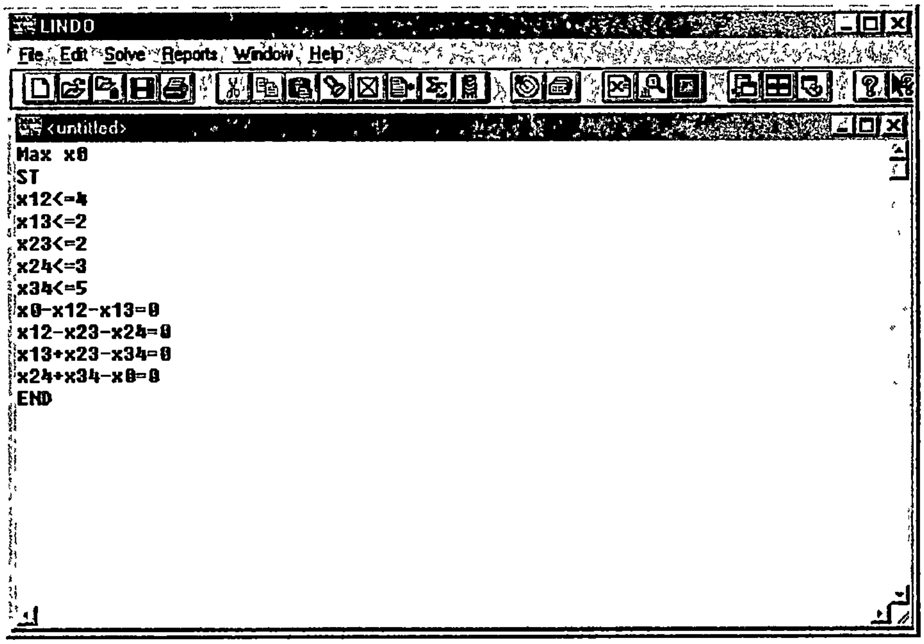


Figure 3.2.10 LP Formulation in LINDO

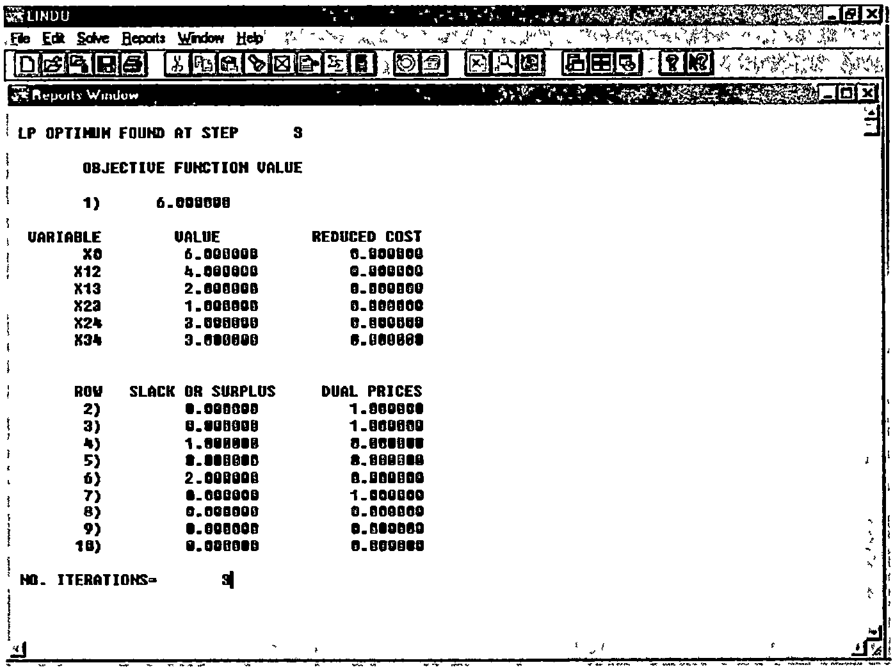


Figure 3.2.11 LINDO Output of Solution

First we choose the path s-1-2-4-t from figure 3.3.1. Thus $\delta = \min\{12,5,4,10\}=4$. We augment the path by 4 units. The residual network is then formed, as seen in Figure 3.3.2. A new path from source to sink is then identified. Thus the path s-1-3-6-t is chosen from Figure 3.3.2. Thus $\delta=\min\{8,9,3,10\}=3$. We augment the path by 3 units now. The new residual network is shown in Figure 3.3.3.

We now choose another path from Figure 3.3.3, say s-1-3-5-t. Thus $\delta = \min\{5,6,7,10\}=5$. We augment the path by 5 units. The algorithm now ends since there are no more augmenting paths from source to sink. Thus we see that the maximum flow is 12 parts per hour. The values of x_{ij}'s are as follows: $x_{s1}=12$, $x_{12}=4$, $x_{13}=8$, $x_{24}=4$, $x_{25}=0$, $x_{35}=5$, $x_{36}=3$, $x_{4t}=4$, $x_{5t}=5$, and $x_{6t}=3$.

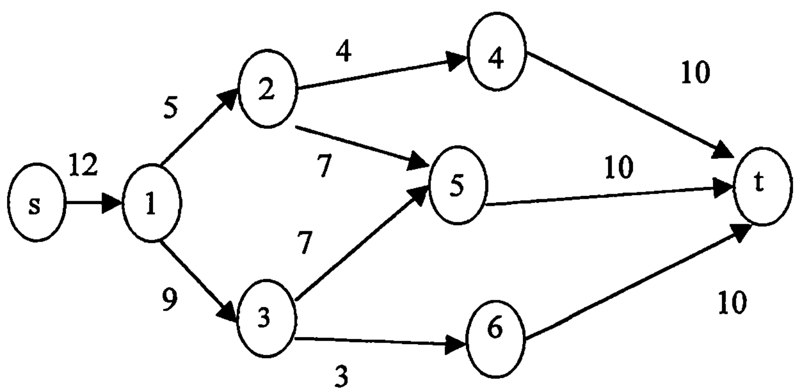


Figure 3.3.1 Network for Example 3.3.1

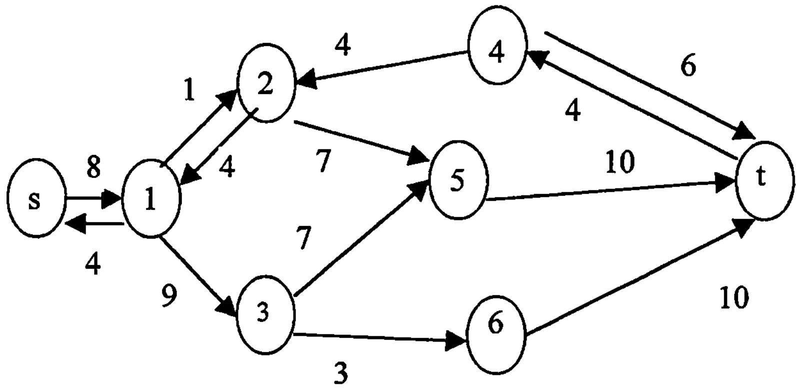


Figure 3.3.2 Residual Network #1 for Example 3.3.1

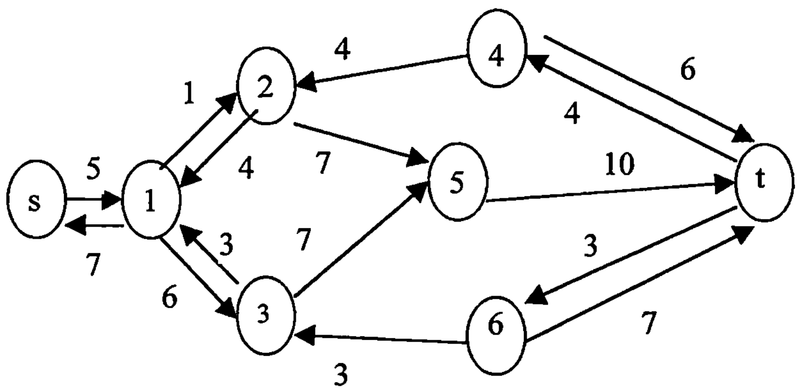


Figure 3.3.3 Residual Network #2 for Example 3.3.1

We now answer the question: What improvements should be made? The operation from 2 to 5 is not necessary. The operations 4-t, 5-t, and 6-t is only at half capacity, thus we need to combine some of these operations for a more efficient system.

We can also solve this problem using LINDO. First, as before, we have to add a dummy arc of capacity zero. Figure 3.3.4 shows the LP formulation and Figure 3.3.5 shows the LINDO output of the solution.

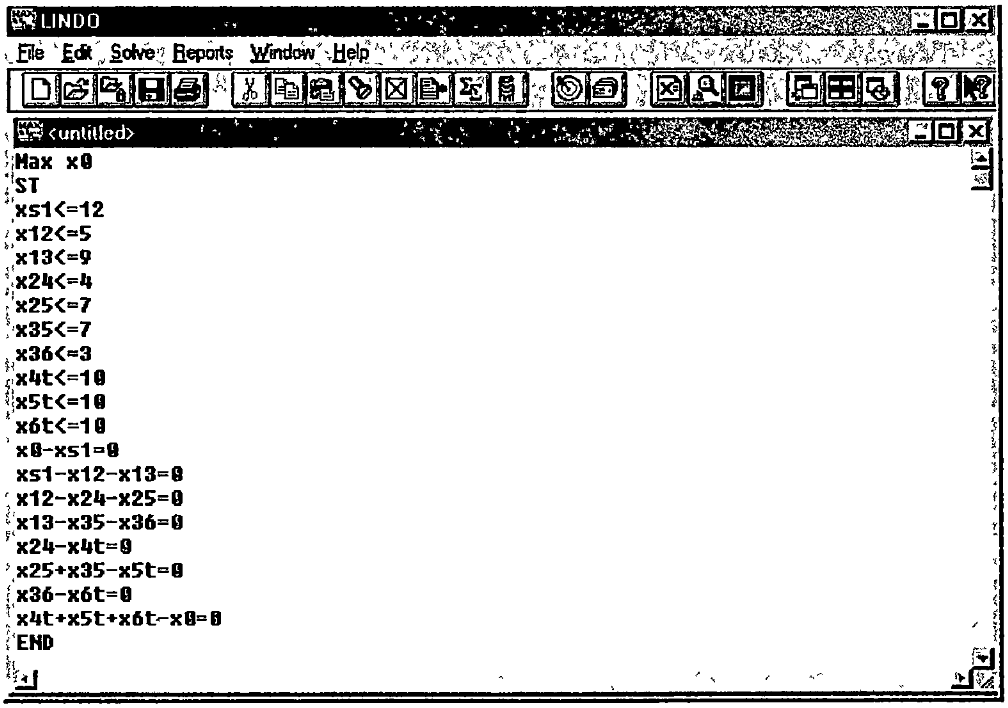


Figure 3.3.4 LP Formulation of Application Problem

```
LP OPTIMUM FOUND AT STEP      1

       OBJECTIVE FUNCTION VALUE

       1)       12.00000

  VARIABLE        VALUE          REDUCED COST
        X0        12.000000          0.000000
       XS1        12.000000          0.000000
       X12         5.000000          0.000000
       X13         7.000000          0.000000
       X24         4.000000          0.000000
       X25         1.000000          0.000000
       X35         4.000000          0.000000
       X36         3.000000          0.000000
       X4T         4.000000          0.000000
       X5T         5.000000          0.000000
       X6T         3.000000          0.000000


       ROW   SLACK OR SURPLUS     DUAL PRICES
        2)         0.000000          1.000000
        3)         0.000000          0.000000
        4)         2.000000          0.000000
        5)         0.000000          0.000000
        6)         6.000000          0.000000
        7)         3.000000          0.000000
        8)         0.000000          0.000000
        9)         6.000000          0.000000
       10)         5.000000          0.000000
       11)         7.000000          0.000000
```

Figure 3.3.5 LINDO Output of Solution for Application Problem

CHAPTER FOUR

SHORTEST PATH PROBLEM

The shortest path problem is used when we must find the least costly flow from node s to node t. The cost is denoted by c_{ij} . The total cost is found by summing the costs of the arcs in the path.

The mathematical notation is as follows:

$$\text{Minimize: } \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij} \tag{4.1}$$

$$\text{Subject to } \sum_{j=1}^{m} x_{ij} - \sum_{j=1}^{m} x_{ji} = \begin{cases} 1 & i=1 \\ 0 & i \neq 1, m \\ -1 & i=m \end{cases} \tag{4.2}$$

$$\text{where } x_{ij} = 1 \text{ or } 0 . \tag{4.3}$$

Note that if x_{ij} is 1 we move along the arc; if x_{ij} is 0 we do not move along the arc. There are four basic assumptions about the shortest path problem. They are as follows:

1. All arc lengths are integers.
2. The network contains a directed path from node s to every other node in the network.
3. The network does not contain a negative cycle.
4. The network is directed.

§ 4.1: Dijkstra's Algorithm

Dijkstra's algorithm finds the shortest path from source to all other nodes in network with nonnegative arc lengths. We denote the distance label as d(i).

Theorem 4.1.1 [Shortest Path Optimality condition]:

For every node $j \in N$, let d(j) denote the length of some directed path from the source node to node j. Then the number d(j) represents shortest path distances iff they satisfy the following shortest path optimality conditions: $d(j) \leq d(i) + c_{ij}$.

Dijkstra's algorithm:

1. Set X = 0 and Y = N.
2. For each node $i \in N$, set $d(i) = \infty$.
3. Set d(s)=0.
4. While $|X| < n$, let $i \in Y$ be a node for which $d(i) = \min\{ d(j): j \in Y\}$. Also let

   $X = X \cup \{i\}$ and $Y = Y - \{i\}$.
5. For each $(i,j) \in A(i)$ do the following: if $d(j) > d(i) + c_{ij}$ then $d(j) = d(i)+c_{ij}$.

Example 4.1.1: Below is an acyclic network in Figure 4.1.1. Use Dijkstra's algorithm to find the shortest path for the network. The iterations of Dijkstra's algorithm are shown in Figure 4.1.2 to Figure 4.1.4.

Another way to solve the shortest path problem is to make a tree diagram. This is an easy but impractical method. Refer to Figure 4.1.5 for an example of a tree diagram.

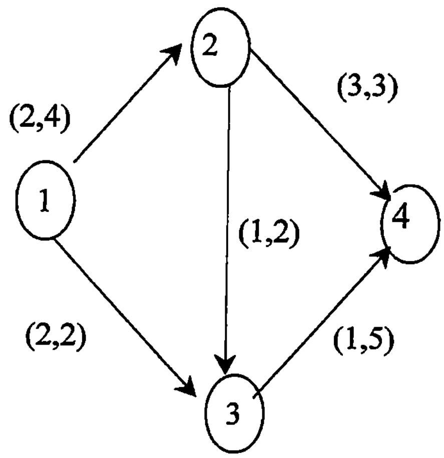


Figure 4.1.1 Network for Example 4.1.1

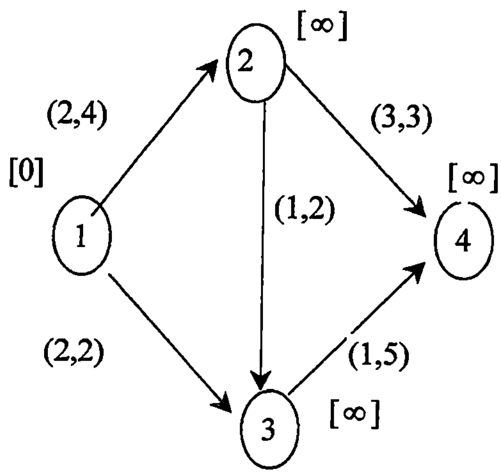


Figure 4.1.2 Initial Labels for Example 4.1.1

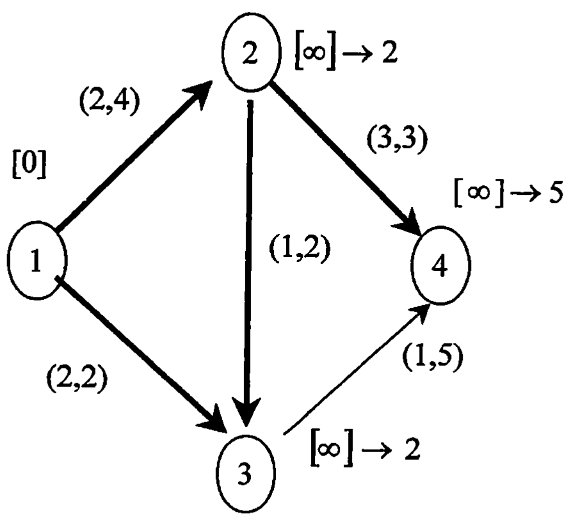


Figure 4.1.3 Label after First and Second Iteration

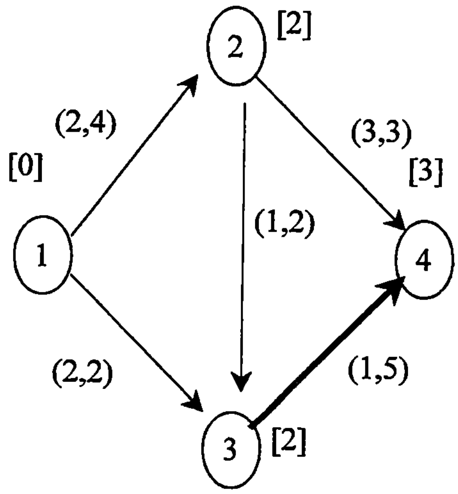


Figure 4.1.4 Final Labels Showing Shortest Distance

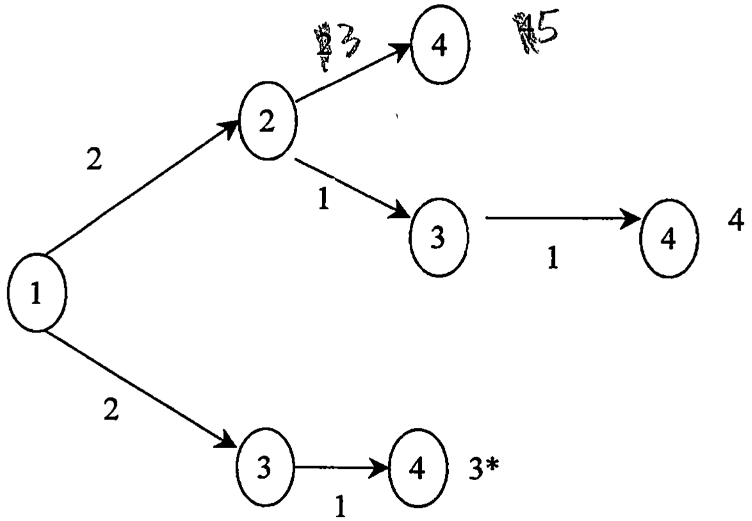


Figure 4.1.5 Tree Diagram for Example 4.1.1 Showing Shortest Path

§ 4.2 : All-Pairs Generic Label-Correcting Algorithm

The all pairs shortest path algorithm requires that we determine the shortest path distances between every pair of nodes in a network. We start with a distance label d[i,j] and update the distance labels until they satisfy Theorem 4.2.1.

Theorem 4.2.1 [All-Pairs Shortest Path Optimality Conditions]:

For every pair of nodes $[i,j] \in N x N$, let d[i,j] represent the length of some directed path from node i to node j satisfying $d[i,j]=0$ for all $i \in N$, and

$d[i,j] \leq c_{ij}$ for all $(i,j) \in A$.

These distances represent all-pairs shortest path distances iff they satisfy the following all-pairs shortest path optimality conditions: $d[i,j] \leq d[i,k] + d[k,j]$ for all nodes i, j, k.

All-Pairs Generic Label Correcting Algorithm:

1. Set $d[i,j]=\infty$ for all $[i,j] \in NxN$.
2. Set $d[i,i]=0$ for all $i \in N$.
3. For all $(i,j) \in A$ set $d[i,j]=c_{ij}$.
4. While the network contains three nodes i,j,k satisfying

   $d[i,j] > d[i,k] + d[k,j]$, set $d[i,j] = d[i,k] + d[k,j]$.

Example 4.2.1: Find the shortest path for the following network in Figure 4.2.1 using the all-pairs label-correcting algorithm.

We first label all the Nx N nodes according to the algorithm (steps 1-3):

$d[1,1]=0$	$d[2,1]=\infty$	$d[3,1]=\infty$	$d[4,1]=\infty$
$d[1,2]=2$	$d[2,2]=0$	$d[3,2]=\infty$	$d[4,2]=\infty$
$d[1,3]=2$	$d[2,3]=1$	$d[3,3]=0$	$d[4,3]=\infty$
$d[1,4]=\infty$	$d[2,4]=3$	$d[3,4]=1$	$d[4,4]=0$

Next we update the nodes according to step 4 of the all-pairs generic label correcting algorithm.

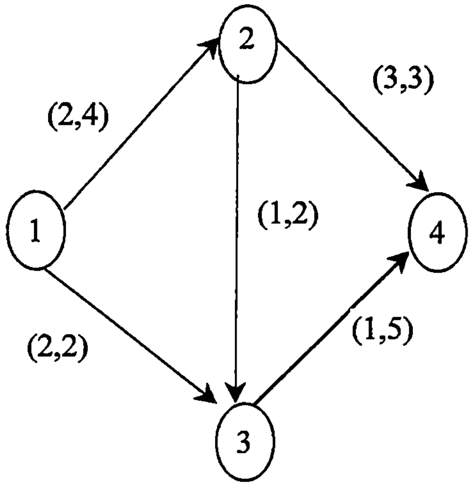


Figure 4.2.1 Network for Example 4.2.1

d[1,2] > d[1,3] + d[3,2]

2 > d[1,3] + d[3,2] (false)

d[1,3] > d[1,2] + d[2,3]

2 > d[1,2] + d[2,3] (false)

d[2,3] > d[2,4] + d[4,3]

2 > d[2,4] + d[4,3] (false)

d[2,4] > d[2,3] + d[3,4]

3 > d[2,3] + d[3,4] (true)

Therefore, d[2,4] = 3.

d[3,4] > d[3,2] + d[2,4]

2 > d[3,2] + d[2,4] (false)

Thus only the distant label of arc (2,4) is updated. The rest of the distant labels stays the same. By inspection we see the shortest path is 1-3-4 (see Figure 4.2.2).

§ 4.3 : Application

A company has warehouses located in different cities, see Figure 4.3.1. Each warehouse produces separate products. These products are then shipped to the other warehouses. For the company we need to cut down on cost and ship products in a fast and efficient way. Suppose the warehouse in Boston needs a product that is located in Los Angeles. Use the network below to find the shortest path. Each of the arcs has the associated cost for shipping one unit through the arc. The solution is shown on Figure 4.3.2. We can see that the shortest path is 1-3-5-9 with a total distance of 186.

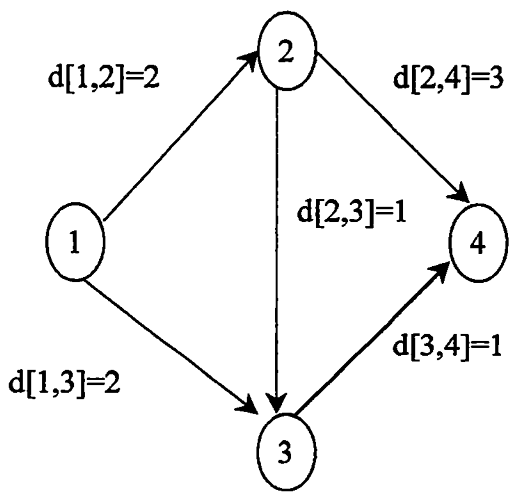


Figure 4.2.2 Network with Updated Distance Labels

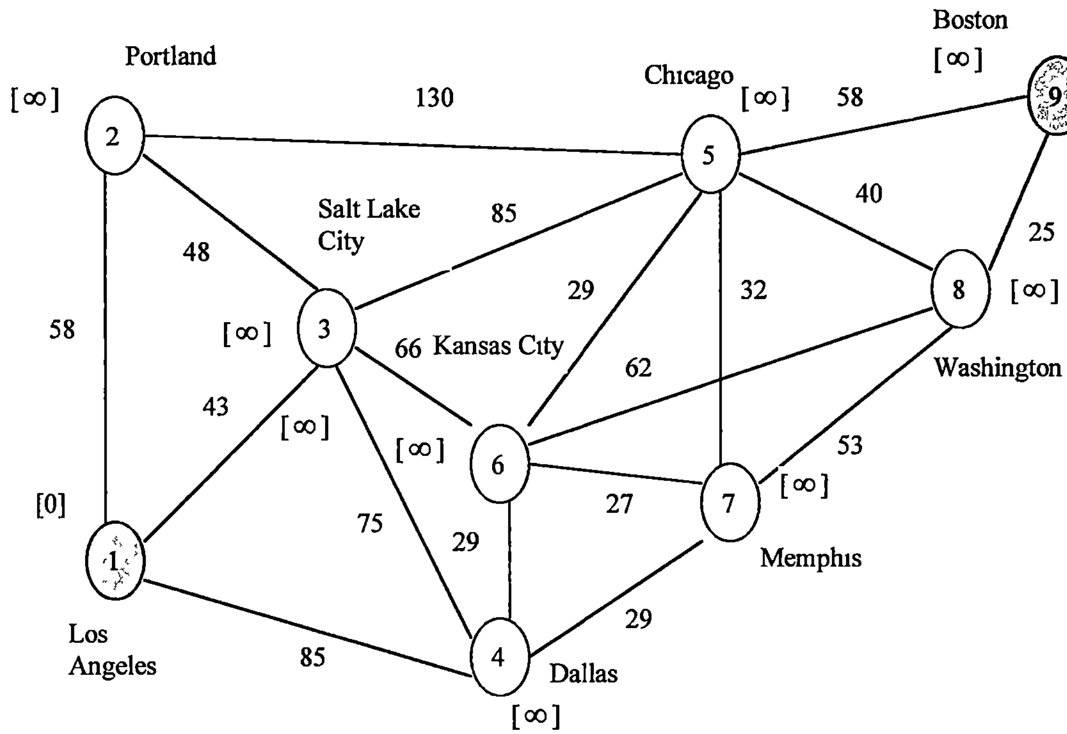


Figure 4.3.1 Network for Shortest Path Application

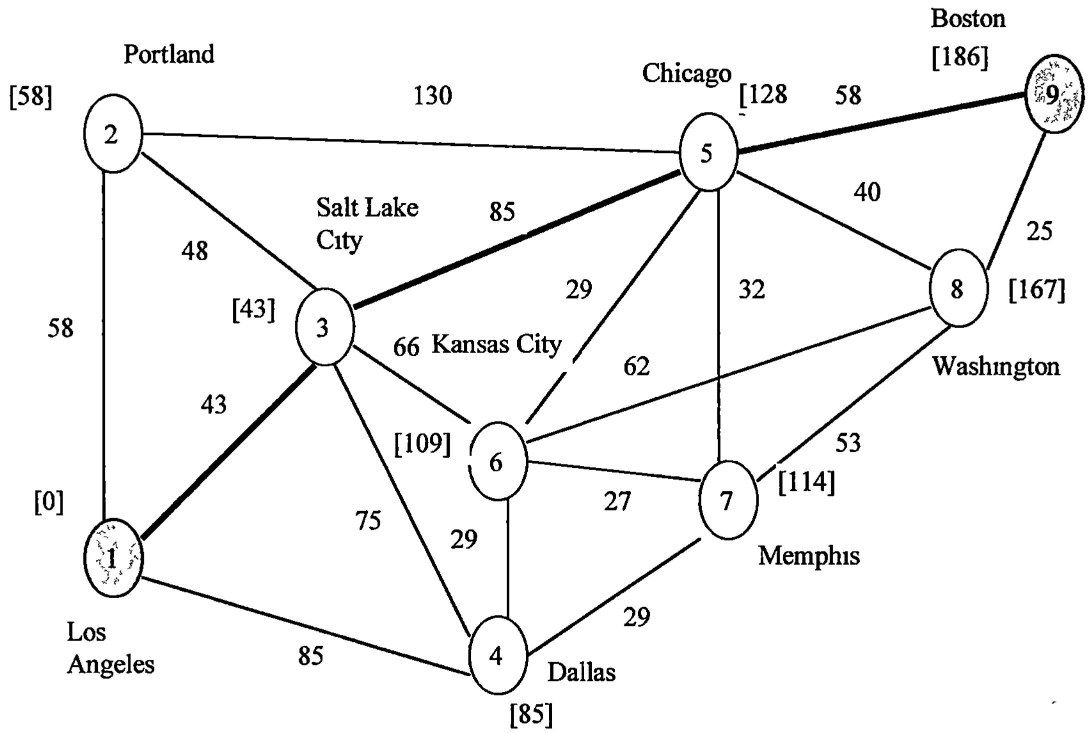


Figure 4.3.2 Network Showing the Shortest Path

CHAPTER FIVE

MINIMUM COST FLOW PROBLEM

Both the shortest path and maximum flow problem is a special case of minimum cost flow problem (MCF Problem). Because of this we sometimes referred to the minimum cost flow problem as the general flow problem. Our goal for the MCF Problem is to find a path so as to ship a commodity from source to sink at minimum cost. Each arc has a cost c_{ij} associated with it, where c_{ij} is the cost of shipping one unit from node i to node j. Information for this chapter can be found in [1], [3], [6], and [12].

Minimize: $$z = \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij} \tag{5.1}$$

subject to $$\sum_{j=1}^{m} x_{ij} - \sum_{j=1}^{m} x_{ji} = \begin{cases} F & \text{if } i = 1 \\ 0 & \text{if } i \neq 1, m \\ -F & \text{if } i = m \end{cases} \tag{5.2}$$

where $0 \leq x_{ij} \leq u_{ij} \quad \forall \ (i, j) \in A.$ (5.3)

In order for the problem to be feasible we must have the following condition: $F \leq f$.

We consider the minimum cost problem under these assumptions:

1. All data are integral.
2. The network is directed.
3. The minimum cost flow problem has a feasible solution.
4. We assume the network G(N,A) contains an uncapacitated directed path between every pair of nodes.
5. All arc costs are nonnegative.

§ 5.1 Cycle Canceling Algorithm

The cycle canceling algorithm uses shortest path computations to find augmenting cycles with negative flow costs; it then augments flow along these cycles and iteratively repeats these computations for detecting negative cost cycles and augmenting flows [1].

Theorem 5.1.1 [Augmenting Cycle Theorem]:

Let x and x^0 be any two feasible solutions of a network flow problem. Then x equals x^0 plus the flow on at most m directed cycles in $G(x^0)$. Furthermore, the cost of x equals the cost of x^0 plus the cost of flow on these augmenting cycles.

Theorem 5.1.2 [Negative Cycle Optimality Conditions]:

A feasible solution x* is an optimal solution of the minimum cost flow problem iff the residual network G(x*) contains no negative cost cycle.

<u>Proof:</u> Suppose that x is a feasible flow and that G(x) contains a negative cycle. Thus x cannot be an optimal flow, since by augmenting positive flow along the cycle we can improve the objective function value. Therefore, if x* is an optimal flow, then G(x*) cannot contain a negative cycle. Suppose x* is a feasible flow and that G(x*) contains no

negative cycle. Let x^0 be an optimal flow and x^* is not equal to x^0. By the augmenting cycle property we can decompose the difference vector $x^0 - x^*$ into at most m augmenting cycles with respect to the flow x^* and the sum of the costs of flows on these cycles equals $cx^0 - cx^* \geq 0$. Since x^0 is an optimal flow, $cx^0 - cx^* \leq 0$. Thus $cx^0 - cx^*=0$. Thus $cx^0 = cx^*$. Thus x^* is also an optimal flow. This augment shows that if $G(x^*)$ contains no negative cycle, then x^* must be optimal [1].

The negative cycle optimality conditions brings us to a method to solve the minimum cost flow problem. The method is called the cycle-canceling algorithm. We maintain a feasible solution at every iteration. We continue the following algorithm until we have no negative cycles.

Cycle Canceling Algorithm:

1. Establish a feasible flow x_{ij} in the network.
2. Change the network into a residual network.
3. Identify any negative cycle. If there is not one then the process ends.
4. For the negative cycle determine the residual capacity (minimum r_{ij}). Augment this cycle by a flow amount equal to the residual capacity.
5. Continue steps 3 and 4 until there are no negative cycles.

Example 5.1.1: Use the cycle-canceling algorithm to solve the minimum cost flow problem from the network below in Figure 5.1.1. Let the flow be 4 units. We will see from this algorithm that our solution will be $x_{12}=2$, $x_{13}=2$, $x_{23}=2$, $x_{24}=0$ and $x_{34}=4$ with the minimum cost of 14. The steps to the algorithm are shown in Figure 5.1.2 to Figure 5.1.8.

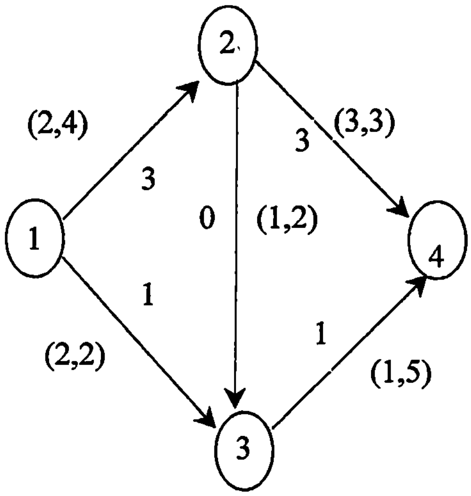


Figure 5.1.1 Network for Example 5.1.1

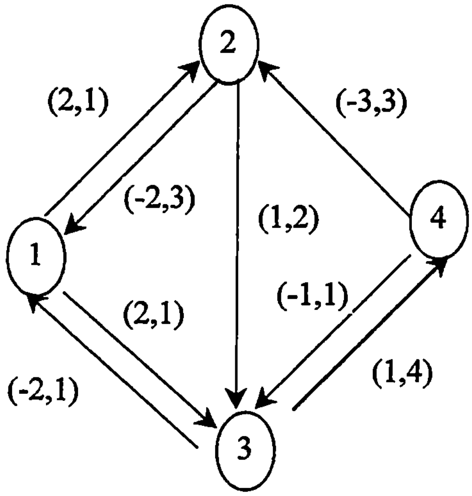


Figure 5.1.2 Residual Network for Example 5.1.1

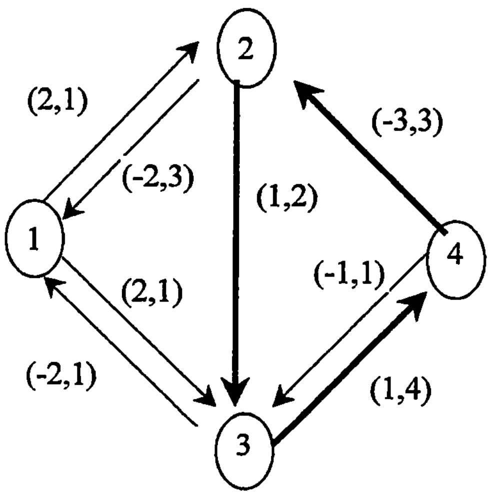


Figure 5.1.3 A Negative Cycle

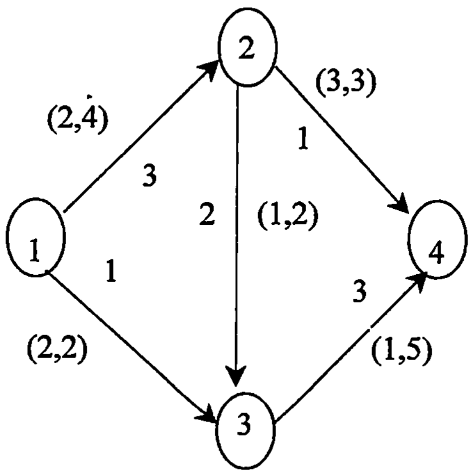


Figure 5.1.4 Updated Network after Augmenting 2 Units

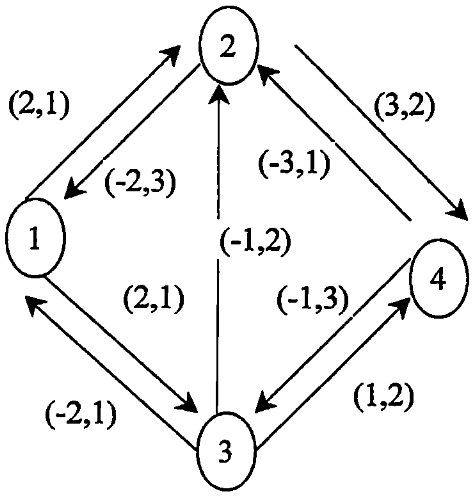


Figure 5.1.5 New Residual Network

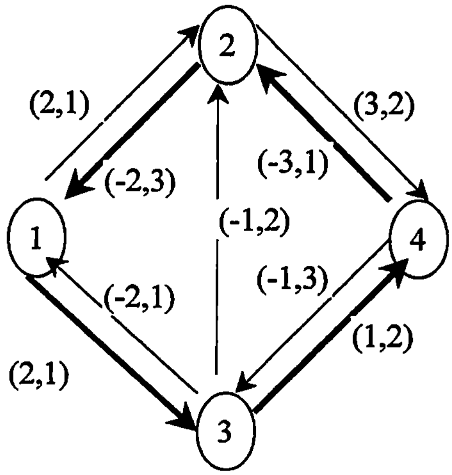


Figure 5.1.6 Second Negative Cycle

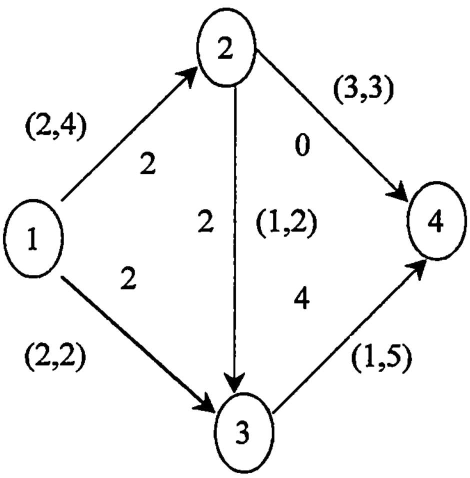


Figure 5.1.7 Updated Network for Example 5.1.1 after Augmenting 1 Unit

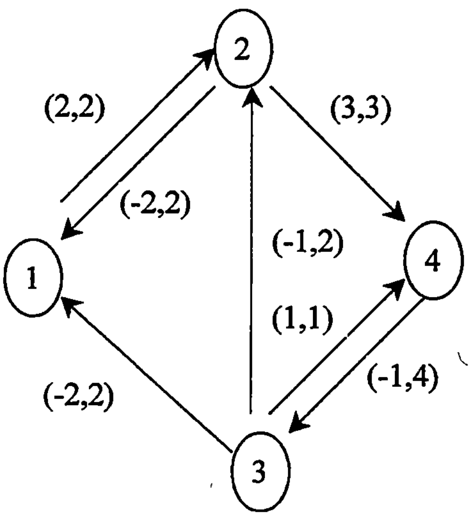


Figure 5.1.8 New Residual Network For Example 5.1.1

§ 5.2 Successive Shortest Path Algorithm

The successive shortest path algorithm selects a node, s, with excess supply and a node, t, with unfulfilled demand and sends a flow from s to t along a shortest path in the residual network.

A *pseudoflow* is a function x: A $\rightarrow$ R+ satisfying the capacity and nonnegativity constraints (not necessarily the mass balance constraints). The imbalance of a node i is denoted by $e(i) = b(i) + \sum_{\{j\ (j,i) \in A\}} x_{ji} - \sum_{\{j\ (i,j) \in A\}} x_{ij}$, where b(i) = F, 0, or -F. The *node potentials*, $\pi(i)$'s, are linear programming dual variables corresponding to the mass balance constraint of node i. The *reduced cost* is given by the following equation: $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$.

If e(i) > 0 for some node i, we say e(i) is the excess of node i; if e(i) < 0, we say e(i) is deficit of node i; if e(i) = 0 we say node i with e(i) is balanced. Let E be the number of excess nodes and D be the number of deficit nodes. If the network contains an excess node it must contain a deficit node.

Theorem 5.2.1 [Reduced Cost Optimality Conditions]:

A feasible solution x* is an optimal solution of the minimum cost flow problem if and only if some set of node potentials satisfy the following reduced cost optimality conditions: $c^{\pi}_{ij} \geq 0$ for every arc (i,j) in G(x*).

Successive Shortest Path Algorithm:

1. Set x = 0 and $\pi = 0$.
2. Set e(i) = F for all n nodes.

3. Set E = { i : e(i) >0 } and D = { i : e(i) < 0 }.

4. Select a node k, element of E, and a node l, element of D.

5. Determine the shortest path from node k to all other nodes in G(x) with respect to the reduced costs c^{π}_{ij}.

6. Let P denote the shortest path from node k to node l.

7. Update π by letting $\pi = \pi - d$.

8. Augment the flow along path P by an amount equal to min[e(k), -e(i), min{r_{ij} : (i,j) element of P}].

9. Finally update x, G(x), E and D, and reduced costs. Continue the process until the solution satisfies the mass balance constraints.

Example 5.2.1: Use the successive shortest path algorithm to solve the minimum cost flow problem in Figure 5.2.1. The labels for each node are shown in Figure 5.2.2.

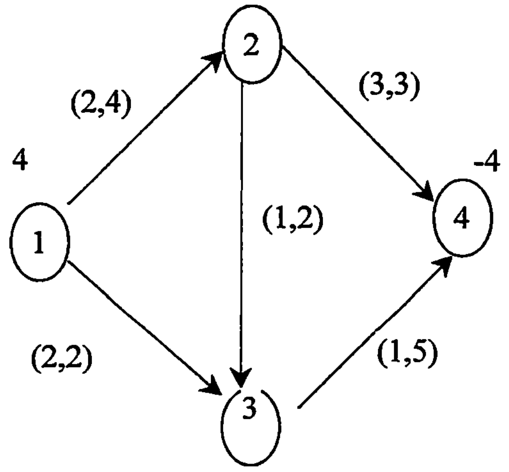


Figure 5.2.1 Network for Example 5.2.1

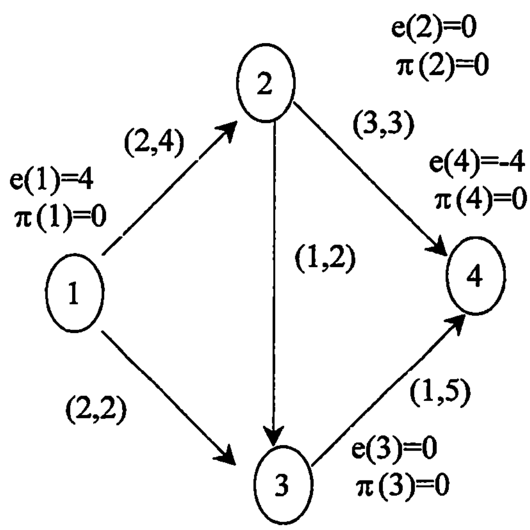


Figure 5.2.2 Node Labels

We set the initial feasible solution to $x_{ij} = 0$. Notice the flow, F, is 4. Also notice since x_{ij}'s are zero the residual network is the same as the original network. E = {1} and D = {4}

We now calculate the shortest distance with respect to reduced costs.

$$d = (0, 2,2,3)$$

Thus the shortest path is 1-3-4. We now update the π's and reduced cost.

$$\pi(1) - d(1) = 0 - 0 = 0$$

$$\pi(2) - d(2) = 0 - 2 = -2$$

$$\pi(3) - d(3) = 0 - 2 = -2$$

$$\pi(4) - d(4) = 0 - 3 = -3$$

$$c^{\pi}_{12} = c_{12} - \pi(1) + \pi(2) = 0$$

$$c^{\pi}_{13} = c_{13} - \pi(1) + \pi(3) = 0$$

$$c^{\pi}_{24} = c_{24} - \pi(2) + \pi(4) = 2$$

$$c^{\pi}_{34} = c_{34} - \pi(3) + \pi(4) = 0$$

We can see from Figure 5.2.3 the new labels for the node potentials and reduced costs. Notice the original network is now updated as seen in Figure 5.2.4. The algorithm now selects a shortest path. The shortest path is 1-3-4, so we augment by the δ =min { 4, -(-4), 2, 5 } = 2. After augmenting by 2 units we obtain a new residual network (see Figure 5.2.5). The algorithm then starts over by updating the reduced costs and node potentials.

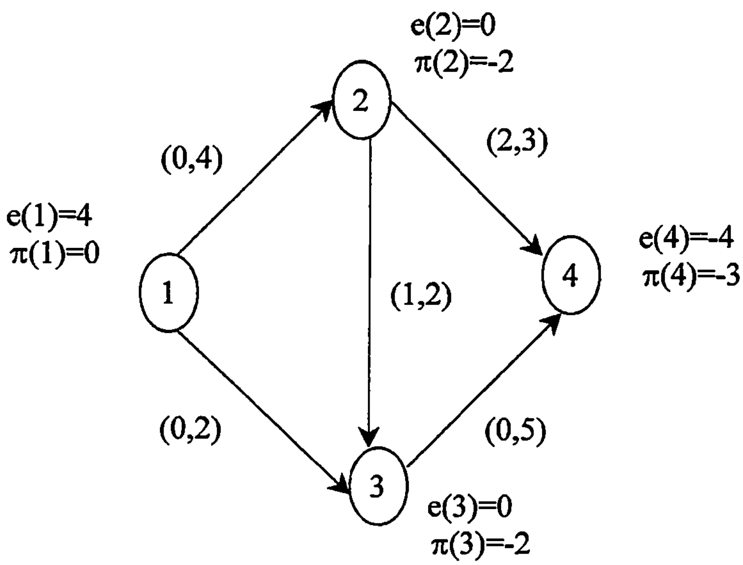


Figure 5.2.3 Node Labels after One Iteration

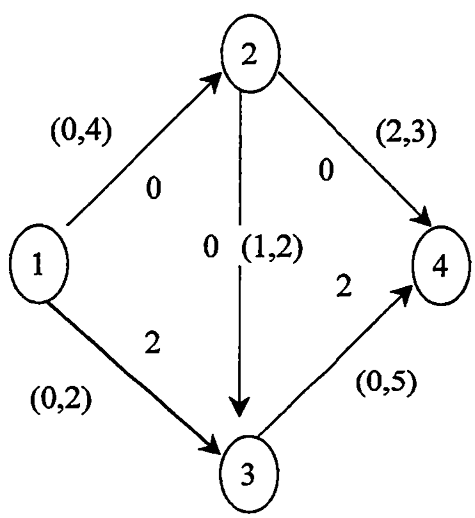


Figure 5.2.4 Updated Network

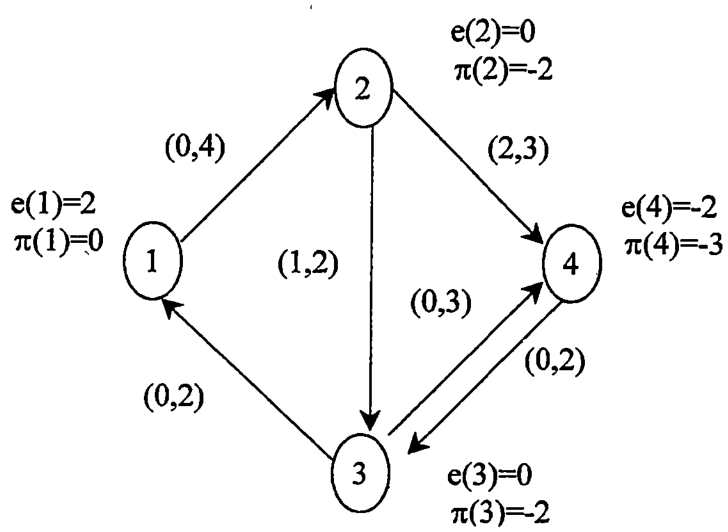


Figure 5.2.5 New Residual Network for Example 5.2.1′

The goal of the algorithm is to have all the nodes balanced, so we again identify any excess and deficient nodes. If there are not any then the algorithm ends. Upon observing Figure 5.2.5 it is clear that there is one excess node and one deficient node, E = {1} and D = {4}. Thus we now calculate the shortest distance with respect to reduced costs:

$$d = (\,0,\,0,1,1\,)$$

Thus the shortest path is 1-2-3-4. We now update the π's and c^{π}_{ij}'s (see Figure 5.2.6).

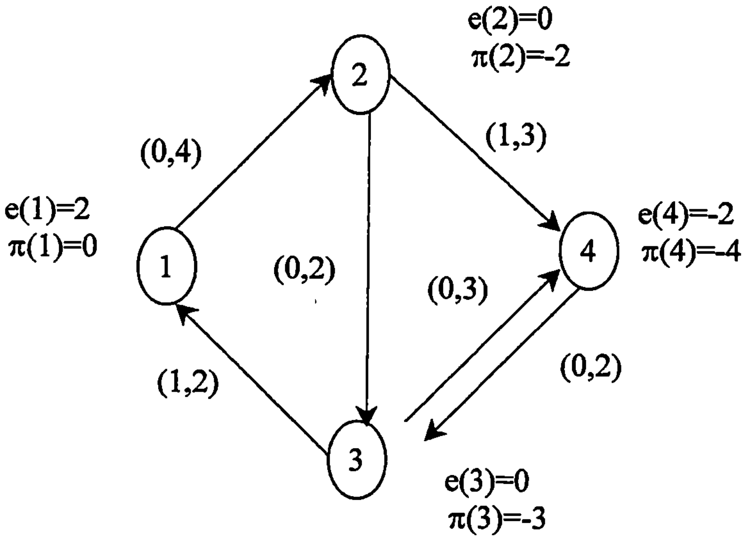


Figure 5.2.6 Updated Residual Network

We now augment by 2 units again. Since e(1)=2 and e(4)=-2 from Figure 5.2.6, then after this augmentation of 2 units there will not be any excess or deficient nodes. The solution is shown in Figure 5.2.7. The minimum cost is 14 with $x_{12}=2$, $x_{13}=2$, $x_{23}=2$, $x_{24}=0$, and $x_{34}=4$.

§ 5.3: Primal-Dual Algorithm

The final algorithm we will look at is the primal-dual algorithm. It is very similar to the successive shortest path algorithm, except it solves a maximum flow problem that sends flow along all shortest paths.

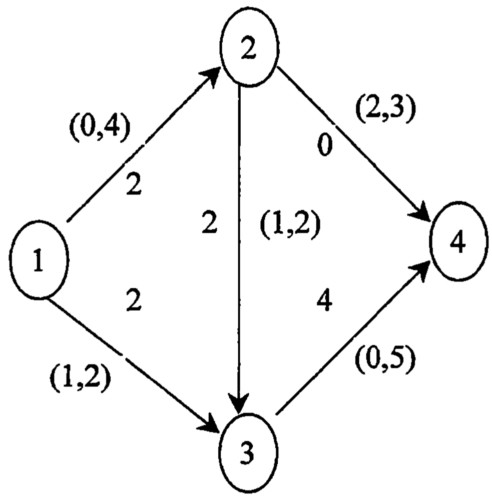


Figure 5.2.7 Updated Network for Example 5.2.1 after Augmenting 2 Units

For each node i with b(i) >0, we add a zero cost arc (s,i) with capacity b(i). For each node i with b(i) < 0, we add a zero cost arc (i,t) with capacity –b(i).

The algorithm transforms the minimum cost problem into a problem with one excess node and one deficit node. The nodes are a source node, s, and sink node, t. The primal-dual solves a maximum flow problem on a subgraph of the residual network G(x). The *admissible network* $G^o(x)$ satisfies the reduced cost optimality conditions for node potentials π. The admissible network contains only those arcs in G(x) with zero reduced cost. We denote $G^o(x)$ as the admissible network, which is an subgraph of G(x).

Primal Dual Algorithm: We continue as long as e(s) > 0

1. Set x = 0 and $\pi = 0$.
2. Set e(s) = b(s) and e(t) = b(t).

3. Determine the shortest path from node s to all other nodes in G(x) with respect to the reduced costs c^{π}_{ij}.

4. Update π by letting $\pi = \pi - d$.

5. Define the admissible network $G^{o}(x)$.

6. Establish a maximum flow from node s to node t in $G^{o}(x)$.

7. Update e(s), e(t), and G(x).

<u>**Example 5.3.1:**</u> Use the primal-dual algorithm to solve the minimum cost flow problem. Notice E = {1} and D = {4}. The network is shown in Figure 5.3.1. Notice each node is given a label, b(i). From the labels we know that flow enters node 1 and exits node 4 with a value of 4. Figure 5.3.2 shows the initial node labels.

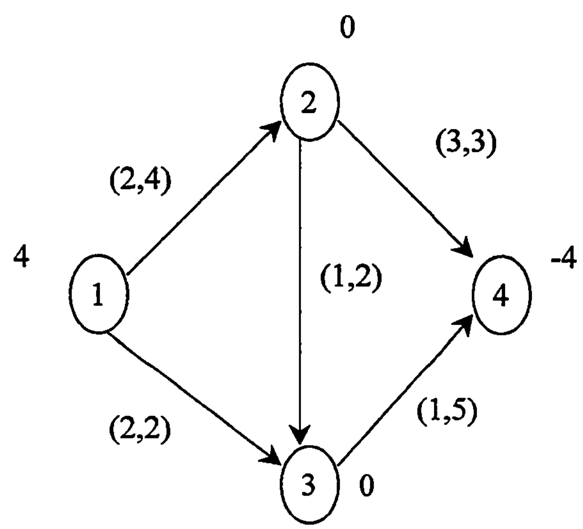


Figure 5.3.1 Network for Example 5.3.1

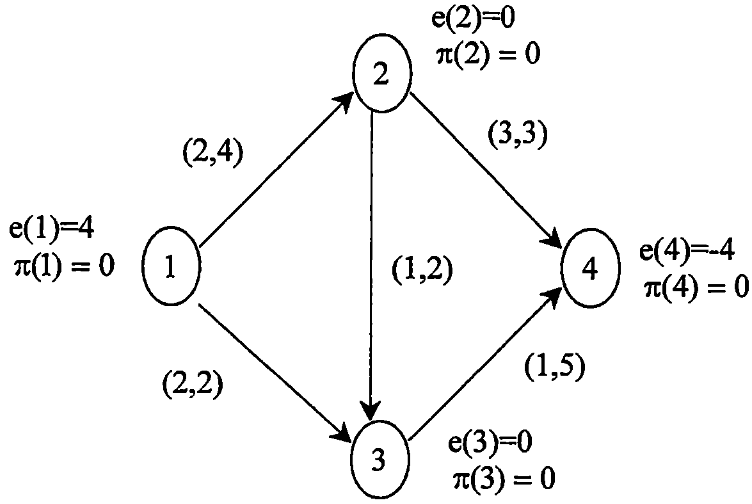


Figure 5.3.2 Node Labels for Example 5.3.1

We now calculate the shortest distance with respect to reduced costs: d = (0, 2,2,3). Thus the shortest path is 1-3-4. We now update the π's (see Figure 5.3.3).

Next identify the admissible network as seen in Figure 5.3.4. We send the maximum flow along the admissible network by using a maximum flow algorithm. We will use the augmenting path algorithm.

We pick the path 1-3-4. Thus $\delta = \min\{ 2,5 \} = 2$. Thus we augment 2 units along path 1-3-4. We now calculate the shortest distance with respect to reduced costs, d= (0, 0,1,1)

Thus the shortest path is 1-2-3-4. We now update the π's and reduced costs (see Figure 5.3.5). After this update a new admissible network is identified and the augmenting path algorithm is invoked. See Figure 5.3.6 for the new admissible network.

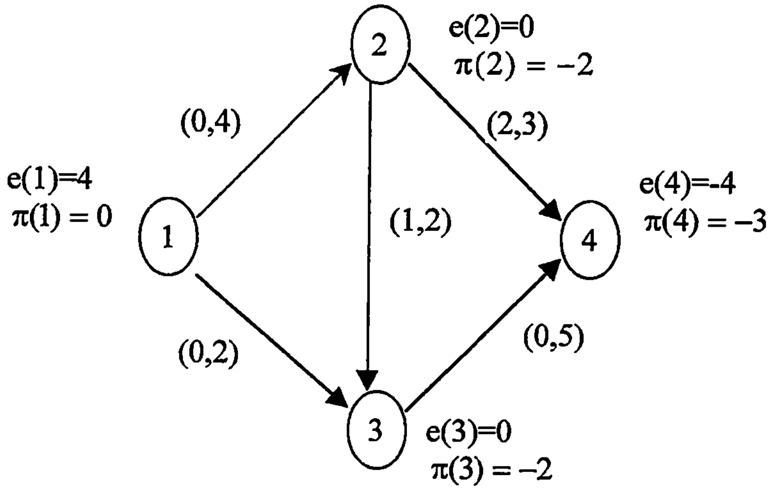


Figure 5.3.3 Updated Residual Network for Example 5.3.1

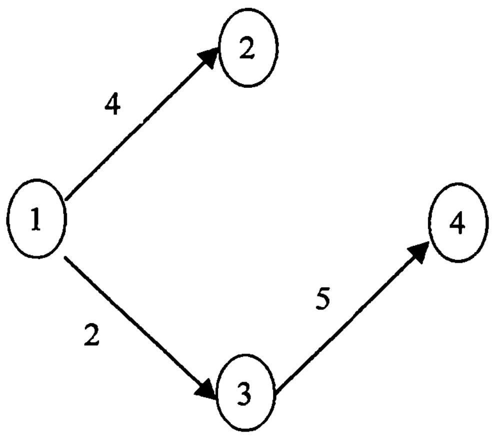


Figure 5.3.4 Admissible Network

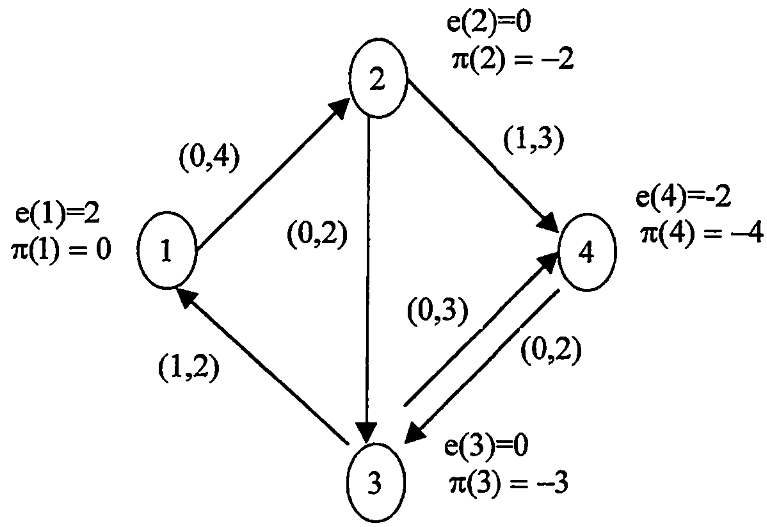


Figure 5.3.5 Updated Node Potentials

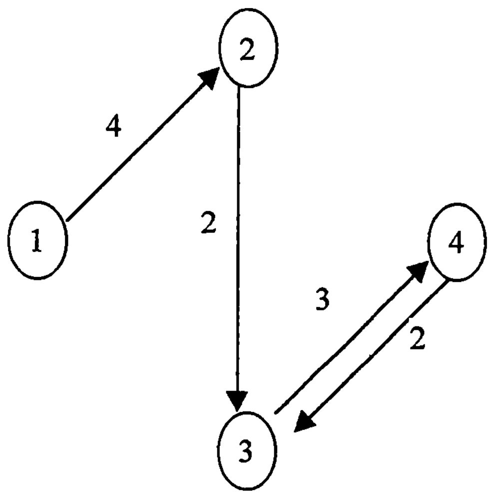


Figure 5.3.6 New Admissible Network

Here again we see Figure 5.3.7 that the minimum cost is 14 with x_{12}=2, x_{13}=2, x_{23}=2, x_{24}=0, and x_{34}=4.

The primal dual algorithm requires a special form in that the algorithm transforms the minimum cost problem into a problem with one excess node and one deficit node. In the above example we started with one excess node and one deficit node. Now lets see what happens with we have more than one excess and deficit nodes as is the case for Figure 5.3.8.

We must add in dummy arcs to transform the network--see the following network in Figure 5.3.9. We combine all the excess nodes into one node and all the deficit nodes into one node. Since node 1 and node 2 have excess of 2 units each, then the new excess node, s, has a total of 4 units. The same is true for the deficient nodes. Thus node t has a total of 4 units deficient.

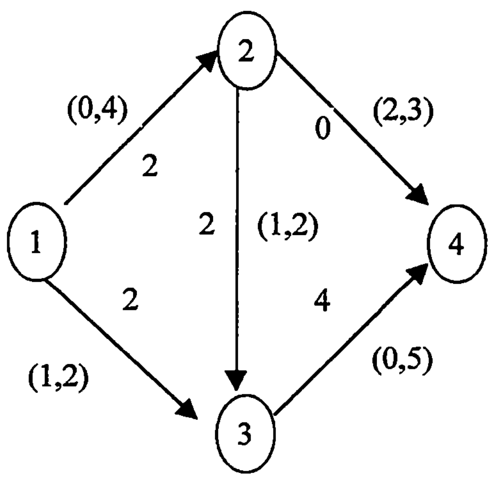


Figure 5.3.7 Updated network After Augmenting 2 Units

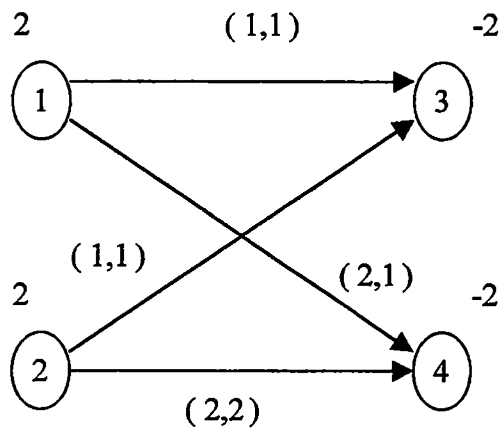


Figure 5.3.8 Network Showing Excess and Deficit Nodes

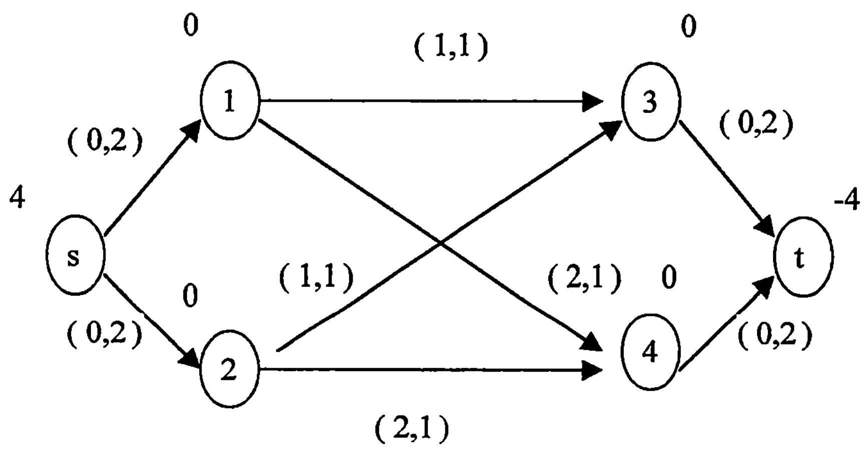


Figure 5.3.9 New Network after Transformation

We now solve the MCF Problem in LINDO. First the LP formulation has to be entered in LINDO. Figure 5.3.10 shows the formulation. From the output in Figure 5.3.11 we see the solution is as follows:

$$z=14 \text{ with } x_{12}=2,\ x_{13}=2,\ x_{23}=2,\ x_{24}=0, \text{ and } x_{34}=4.$$

This verifies our algorithms for the MCF Problem.

§ 5.4: Applications

Example 5.4.1: [Transportation Problem]

A company supplies goods to two customers. The company has two warehouses. The cost of shipping 1 unit from warehouse to customer is shown in the Table5.4.1 below. Find the minimum cost of shipping from the warehouses to customers. Solve using the primal-dual algorithm.

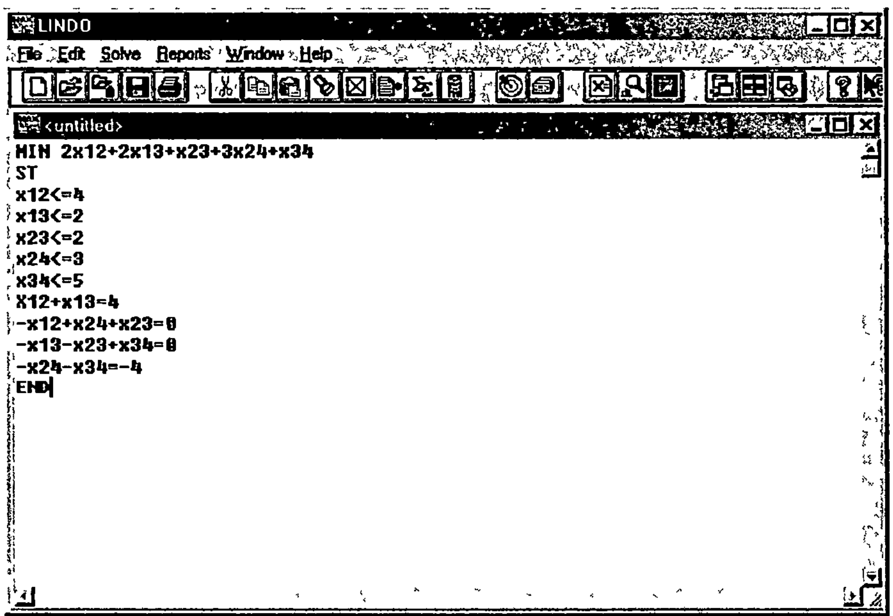
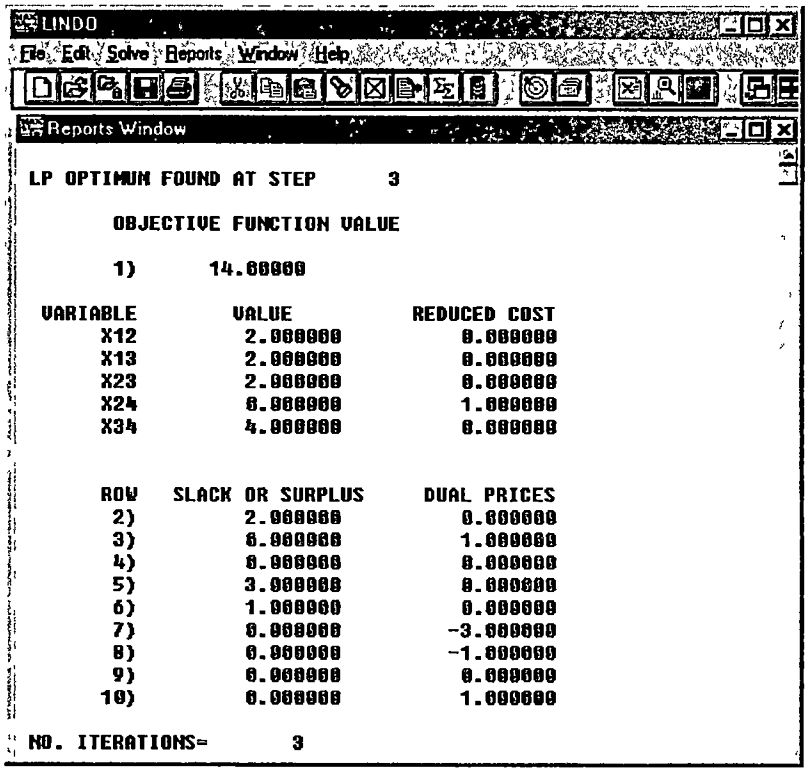


Figure 5.3.10 LP Formulation of MCF Problem



```
LP OPTIMUM FOUND AT STEP        3

        OBJECTIVE FUNCTION VALUE

        1)       14.00000

 VARIABLE         VALUE          REDUCED COST
      X12        2.000000           0.000000
      X13        2.000000           0.000000
      X23        2.000000           0.000000
      X24        0.000000           1.000000
      X34        4.000000           0.000000


      ROW   SLACK OR SURPLUS     DUAL PRICES
       2)        2.000000           0.000000
       3)        0.000000           1.000000
       4)        0.000000           0.000000
       5)        3.000000           0.000000
       6)        1.000000           0.000000
       7)        0.000000          -3.000000
       8)        0.000000          -1.000000
       9)        0.000000           0.000000
      10)        0.000000           1.000000

NO. ITERATIONS=        3
```

Figure 5.3.11 LINDO Output of Solution for Example 5.3.1

The network is given in Figure 5.4.1 for the transportation problem. Since the network has more than one excess and deficient nodes, we transform the network (see Figure 5.4.2). Next all the nodes are given an initial label. These labels can be seen in Figure 5.4.3.

Table 5.4.1 Cost Matrix for Example 5.4.1

	A	B
1	\$15	\$35
2	\$10	\$50

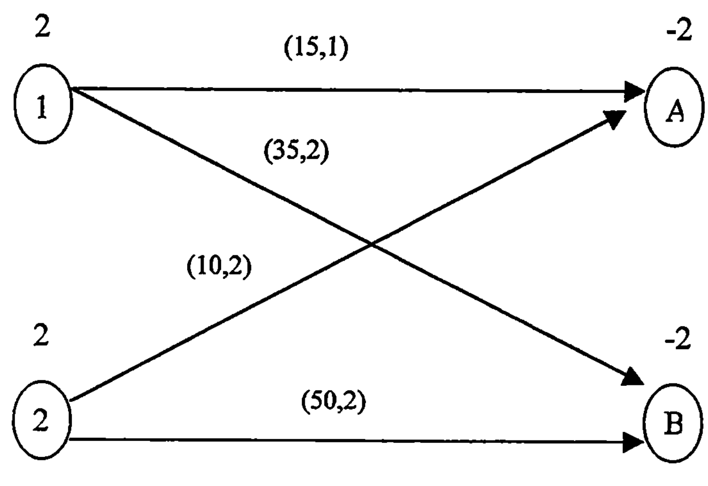


Figure 5.4.1 Network for Transportation Problem

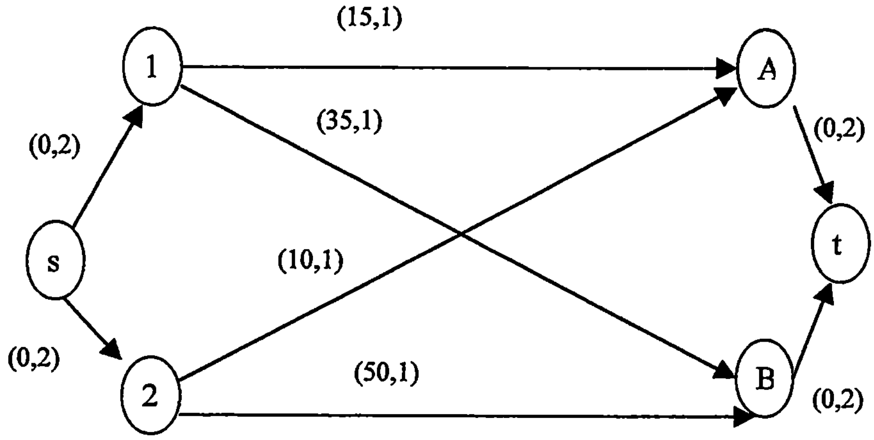


Figure 5.4.2 Transformed Network for Transportation Problem

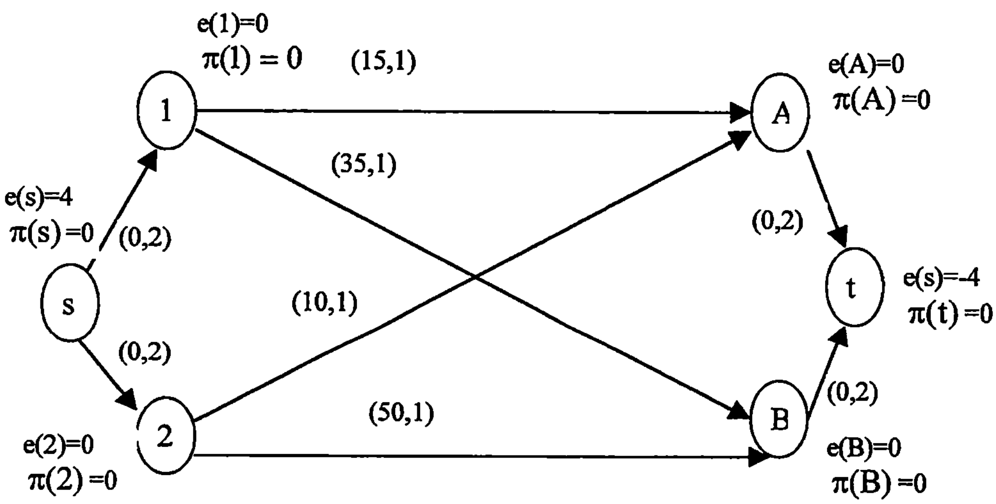


Figure 5.4.3 Transformed Network Showing Node Potentials

We first calculate the distances from the reduced cost:

$$d=(0,0,0,10,35,10).$$

The reduced costs are as follows:

$c^{\pi}_{s1} = 0$	$c^{\pi}_{2A} = 10$
$c^{\pi}_{s2} = 0$	$c^{\pi}_{2B} = 50$
$c^{\pi}_{1A} = 15$	$c^{\pi}_{At} = 0$
$c^{\pi}_{1B} = 35$	$c^{\pi}_{Bt} = 0.$

We can see from the distance labels that a shortest path is s-2-A-t. We now update the node potentials and reduced cost (see Figure 5.4.4):

$\pi(s)-d(s)=0$	$c^{\pi}_{s1} = 0$
$\pi(1)-d(1)=0$	$c^{\pi}_{s2} = 0$
$\pi(2)-d(2)=0$	$c^{\pi}_{1A} = 5$
$\pi(A)-d(A)=-10$	$c^{\pi}_{1B} = 0$
$\pi(B)-d(B)=-35$	$c^{\pi}_{2A} = 0$
$\pi(t)-d(t)=-10$	$c^{\pi}_{2B} =15$
	$c^{\pi}_{At} = 0$
	$c^{\pi}_{Bt} = 25.$

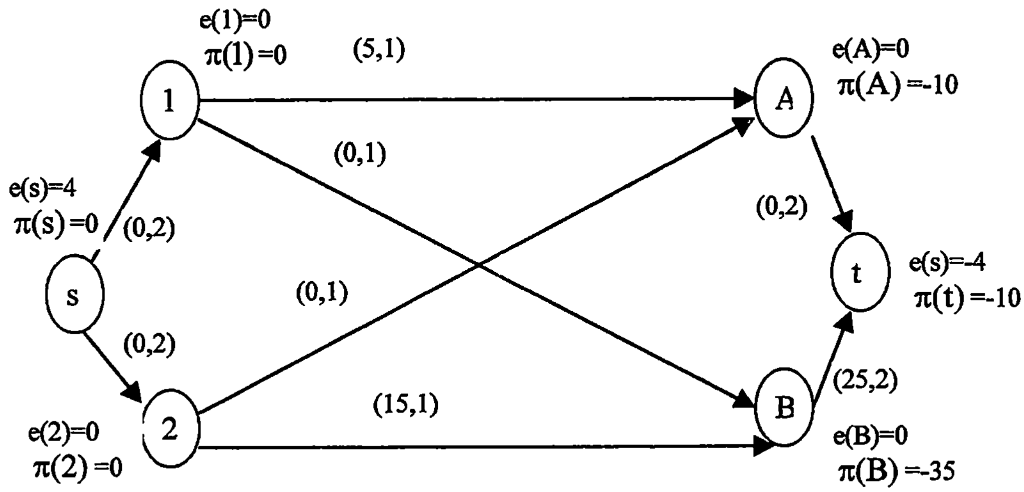


Figure 5.4.4 New Node Labels for Transportation Problem

We next form the admissible network (includes the arcs with zero reduced cost) in Figure 5.4.5. Notice that the augmented path form source to sink is s-2-A-t. We need to send the maximum flow along this path. Thus we use the augmenting path algorithm. Notice the maximum flow occuring along the path s-2-A-t is the minimum of the residuals. We augment by $\delta = \min\{2,1,2\} = 1$, thus one unit of flow is sent along this path. We now calculate the new distance labels: d=(0,0,0,5,0,5).

We now calculate new node potentials and new reduced costs (see Figure 5.4.6):

π(s)-d(s)=0	$c^{\pi}{}_{s1} = 0$
π(1)-d(1)=0	$c^{\pi}{}_{s2} = 0$
π(2)-d(2)=0	$c^{\pi}{}_{1A} = 0$
π(A)-d(A)=-15	$c^{\pi}{}_{1B} = 0$
π(B)-d(B)=-35	$c^{\pi}{}_{2A} = -5$

$\pi(t)-d(t)=-15$ $\qquad\qquad$ $c^{\pi}_{2B} = 15$

$c^{\pi}_{At} = 0$

$c^{\pi}_{Bt} = 20.$

The new updated network is seen in Figure 5.4.7. Again we establish an admissible network (Figure 5.4.8). This time we can augment 1 unit of flow along the path s-1-A-t.

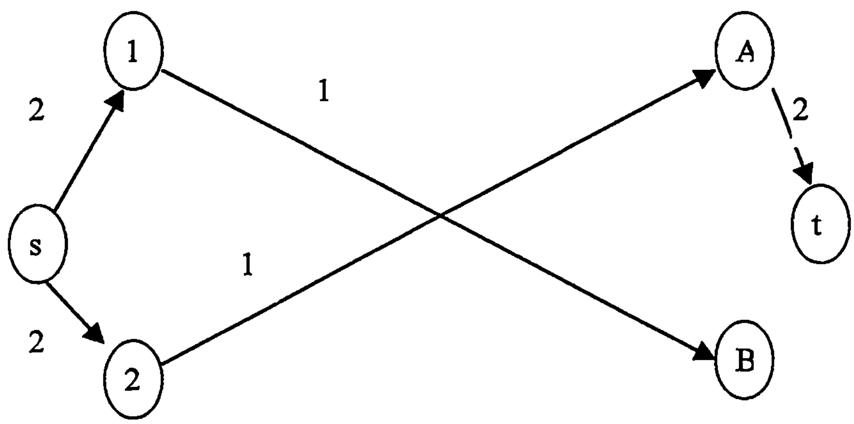


Figure 5.4.5 Admissible Network for Transportation Problem

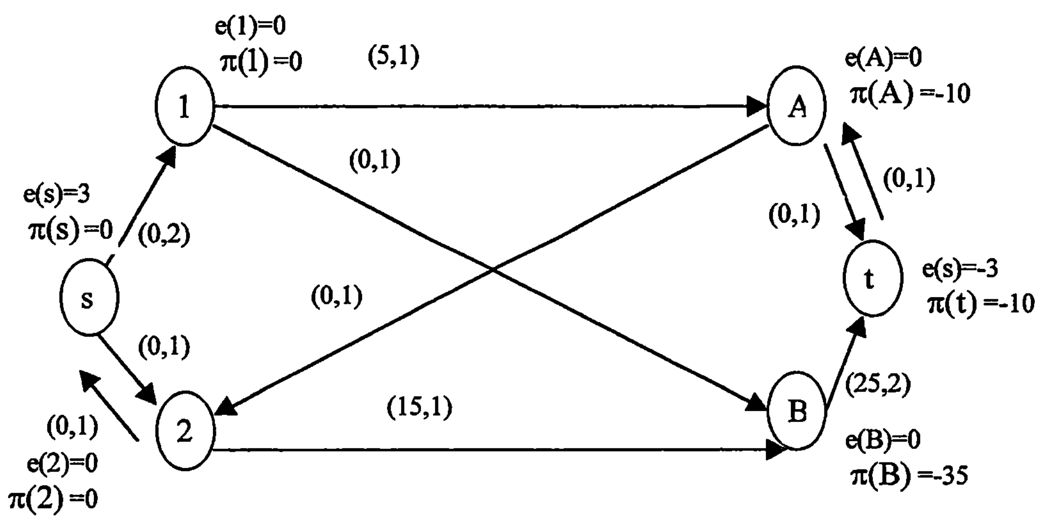


Figure 5.4.6 Updated Network for Transportation Problem

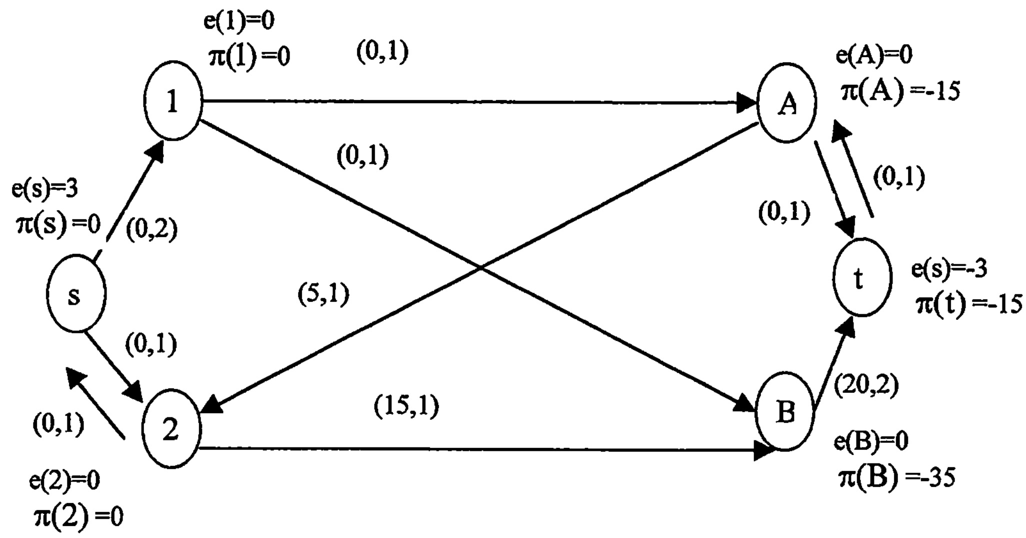


Figure 5.4.7 Updated Network #1 for Transportation Problem

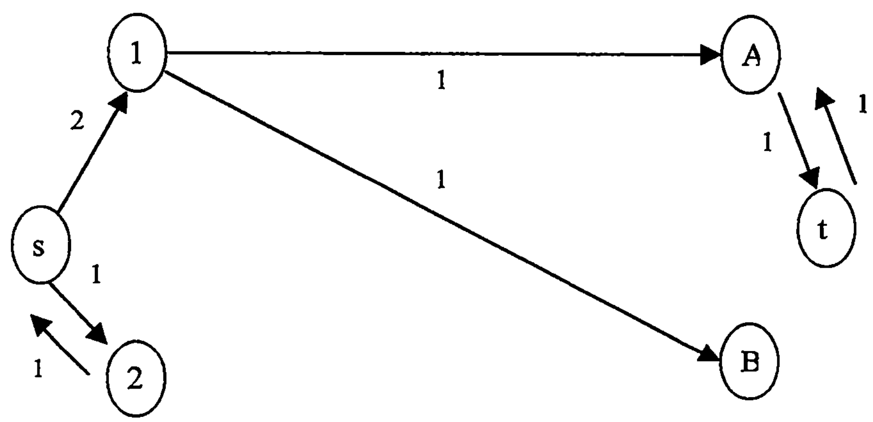


Figure 5.4.8 New Admissible Network #1 for Transportation Problem

We now find the new distance labels: d=(0,0,0,35,0,20). We now compute the new node potentials and reduce cost (see Figure 5.4.9):

$\pi(s)-d(s)=0$	$c^{\pi}_{s1}=0$
$\pi(1)-d(1)=0$	$c^{\pi}_{s2}=0$
$\pi(2)-d(2)=0$	$c^{\pi}_{1A}=35$
$\pi(A)-d(A)=-50$	$c^{\pi}_{1B}=0$
$\pi(B)-d(B)=-35$	$c^{\pi}_{2A}=-40$
$\pi(t)-d(t)=-35$	$c^{\pi}_{2B}=15$
	$c^{\pi}_{At}=15$
	$c^{\pi}_{Bt}=0.$

The updated network is shown in Figure 5.4.10. Next we identify admissible network (Figure 5.4.11). We augment 1 unit of flow through the path s-1-B-t. The new distance labels can be found also: d=(0,15,0,0,15,15).

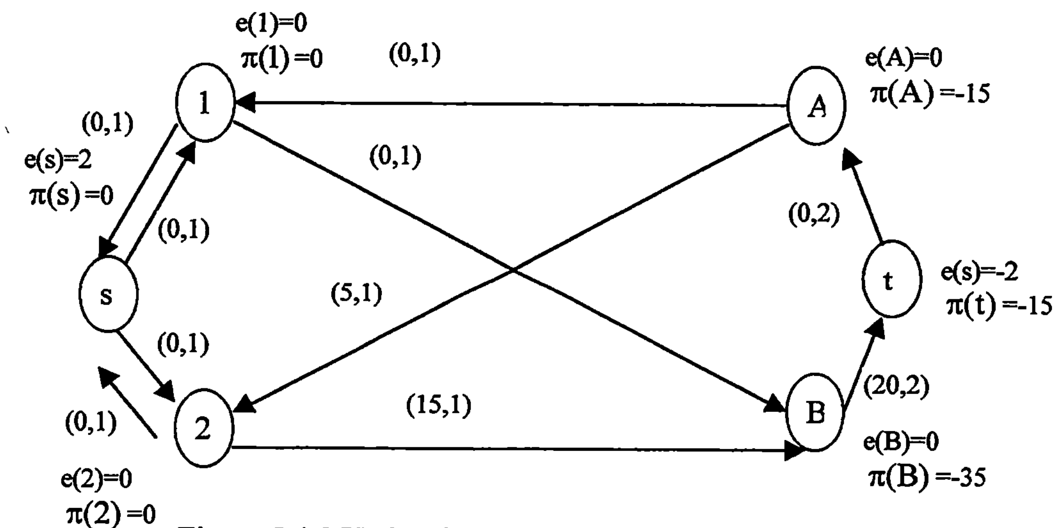


Figure 5.4.9 Updated Network #2 for Transportation Problem

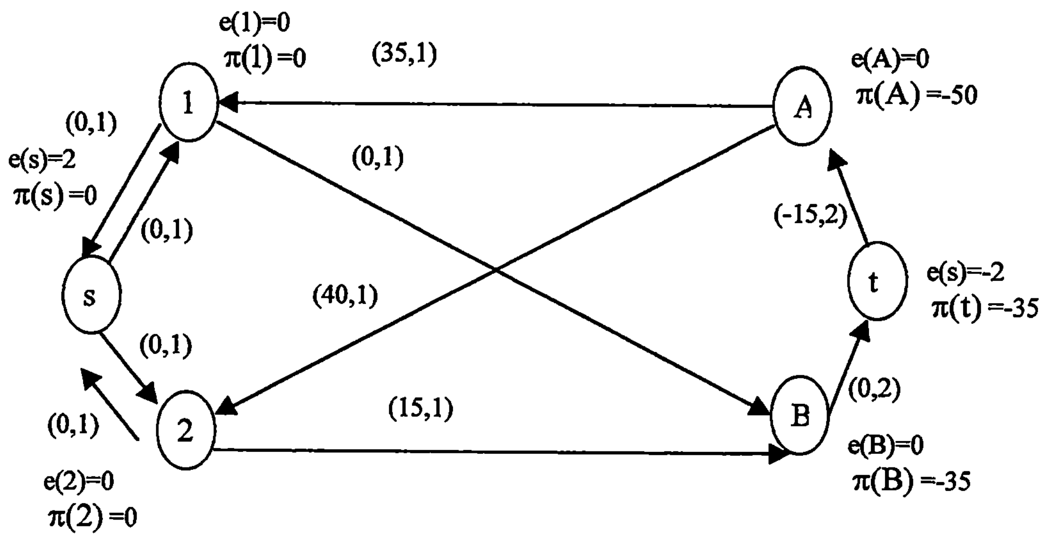


Figure 5.4.10 Updated Network #3 for Transportation Problem

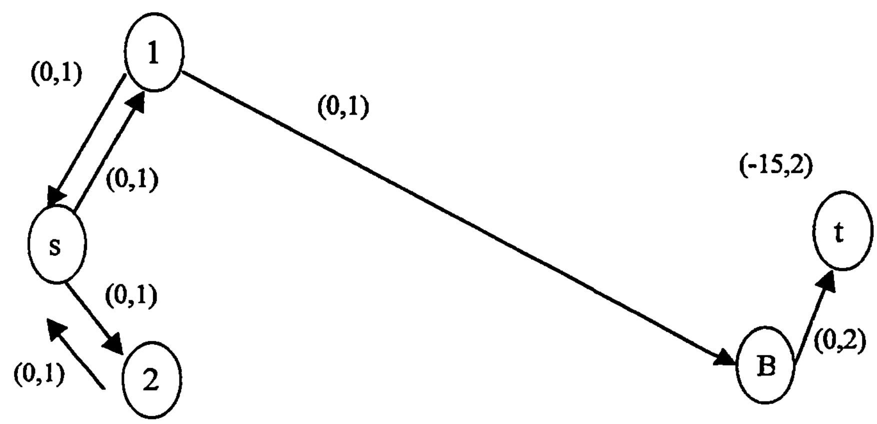


Figure 5.4.11 New Admissible Network #2 for Transportation Problem

We now update the node potentials and reduce cost (Figure 5.4.12):

$\pi(s)-d(s)=0$	$c^{\pi}_{s1}=0$
$\pi(1)-d(1)=-15$	$c^{\pi}_{s2}=0$
$\pi(2)-d(2)=0$	$c^{\pi}_{1A}=35$
$\pi(A)-d(A)=-50$	$c^{\pi}_{1B}=0$
$\pi(B)-d(B)=-50$	$c^{\pi}_{2A}=-40$
$\pi(t)-d(t)=-50$	$c^{\pi}_{2B}=15$
	$c^{\pi}_{At}=15$
	$c^{\pi}_{Bt}=0.$

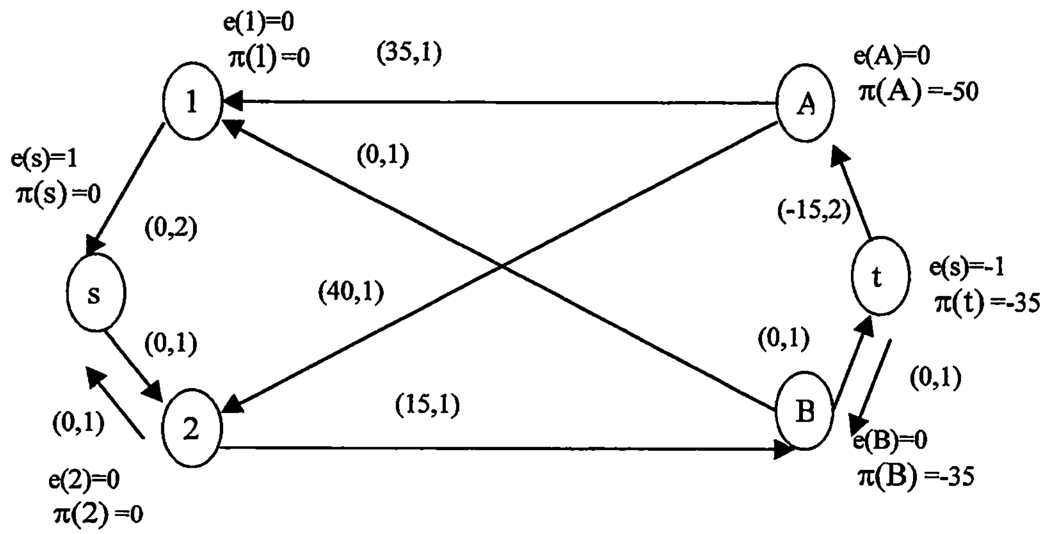


Figure 5.4.12 Updated Network #4 for Transportation Problem

We construct the new admissible network next (see Figure 5.4.13). There will be 1 unit augmented through the path s-2-B-t. The algorithm now ends since there are no more excess nodes. Thus the solution is $x_{s1}=2$, $x_{s2}=2$, $x_{1A}=1$, $x_{1B}=1$, $x_{2A}=1$, $x_{2B}=1$, $x_{At}=2$, $x_{Bt}=2$ and the minimum cost is 110.

As before we now solve the problem using LINDO. We first enter the LP formulation into LINDO (Figure 5.4.14). The solution by LINDO is given in Figure 5.4.15. As we can see the minimum cost is 110. The assignment and transshipment problem can be solved the same way: either by primal-dual algorithm or using LINDO.

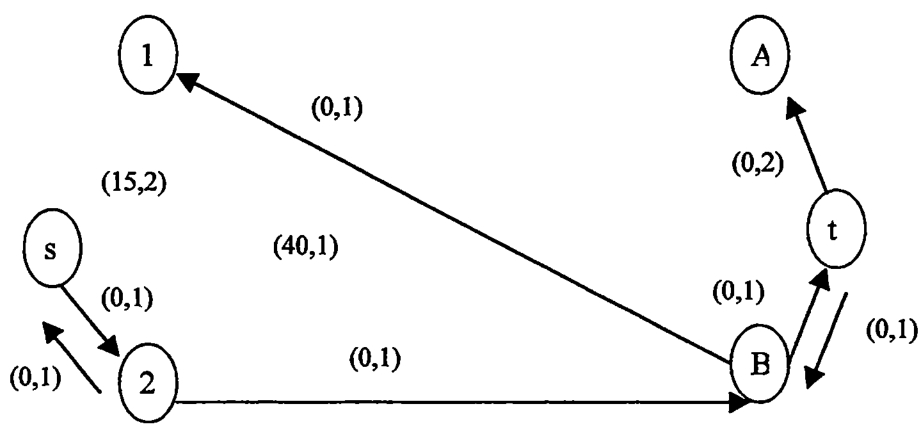


Figure 5.4.13 New Admissible Network #3 for Transportation Problem

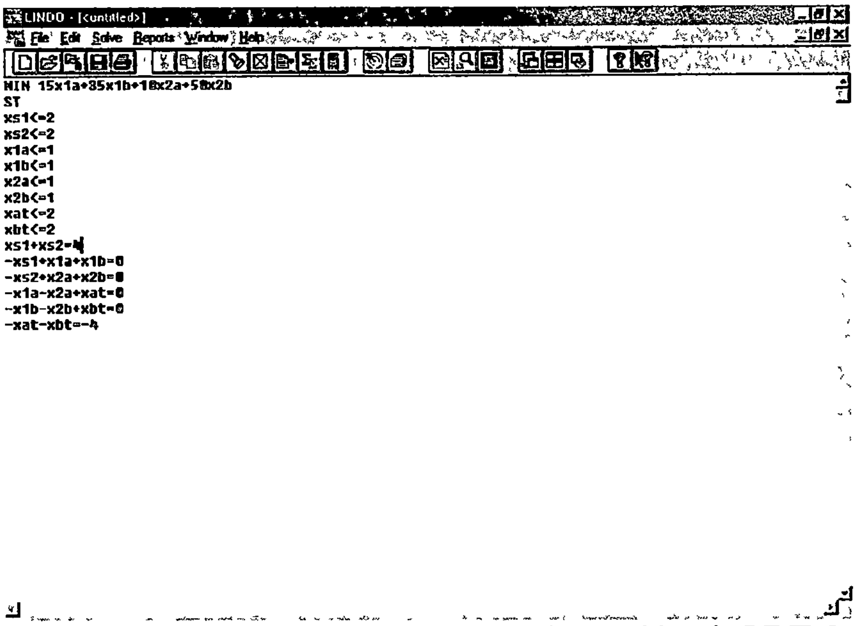


Figure 5.4.14 LP Formulation of Transportation Problem in LINDO

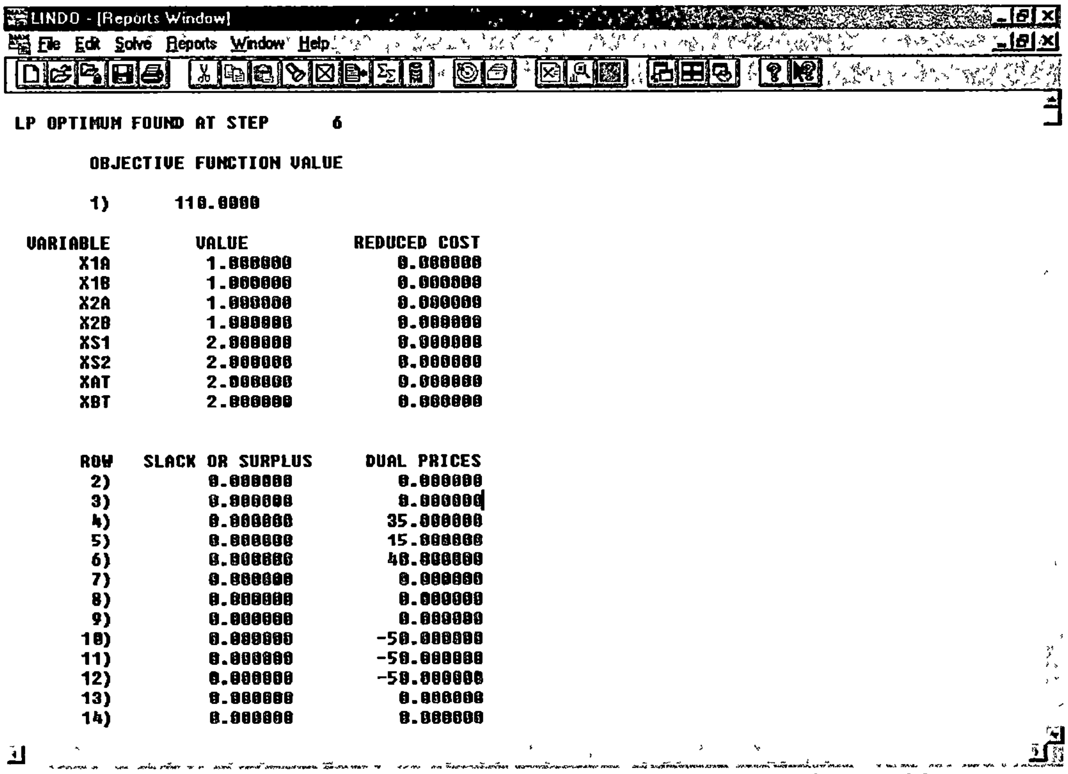


Figure 5.4.15 LINDO Output of Solution for Transportation Problem

REFERENCES

REFERENCES

1. Ahuja, R., Magnanti, T., and Orlin, J., *Network Flows: Theory, Algorithms, and Applıcatıons*, Prentice Hall (1993).

2. American Mathematical Society, Progceeding of Symposia in Applied Mathematics: *The Mathematics of Network Vol 26*, (1981).

3. Bazaraa, M.S., Jarvis, J. J., and Sheralli, H. D., *Linear Programming and Network Flows*, John Wiley & Sons (1990).

4. Bertsekas, D.P., *Linear Network Optimization. Algorithms and Codes*, MIT Press, (1992).

5. Dantzig, G., *Linear Programming and Extentions*, Princeton University Press, (1963).

6. Dantzig, G., and Thapa, M., *Linear Programming 1 Introduction*, Springer, (1997).

7. Gass, S. I., *Linear Programming Methods and Applications*, McGraw Hill, (1985).

8. Gribik, P.R., and Kortanek, K.O., *Extremal Methods of Operations Research*, Marcel Dekker, Inc., (1985).

9. Ho, J., *Linear and Dynamic Programming with Lotus 1-2-3*, Management Information Source Press, (1987).

10. Schrage, L., *Optimization Modeling with LINDO*, Duxbury Press, (1997).

11. Schrage, L., *User's Manual Linear, Integer and Quadratic Programming with LINDO*, The Scientific Press, (1989).

12. Winston, W., *Operations Research: Applications and Algorithms*, Duxbury Press, (1994).

APPENDIX

APPENDIX

LINDO (Linear, Interactive, and Discrete Optimizer) is a powerful tool for solving linear programming problems. It allows the user to input the LP formulation, solve it, assess the correctness and make any necessary changes quickly. Information used in this appendix is from [10] and [11].

Upon opening the program there will be a blank window called the untitled screen (see Figure A1). This is where the problem is typed in as written. First the objective function is entered. Use MAX for maximization and MIN for minimization. On the next line type SUBJECT TO or ST for short. Next all the constraints are entered into the program. If the inequality on a constraint is $\leq$, then enter it as <= in LINDO. The final step is to enter END. This tells LINDO that all the information is entered into the program.

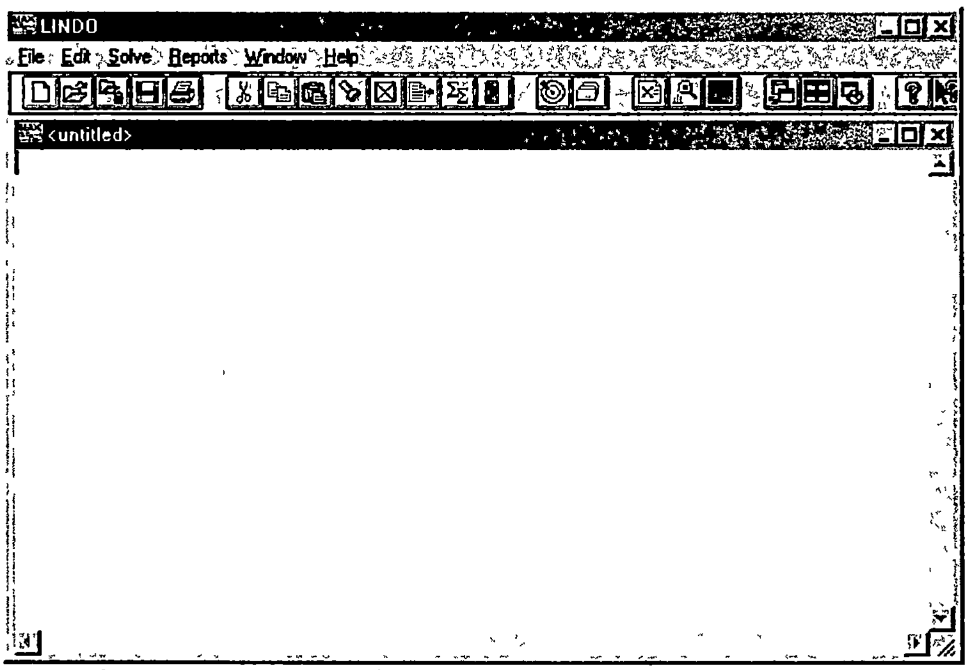


Figure A1: An Example of the Untitled Screen

For example if we wish to maximize the objective function $2x_1+2x_2$ subject to $x_1+x_2 \leq 3$ and $x_1-x_2 \leq 2$, then we type the problem in the blank window (see Figure A2) as follows:

```
MAX 2x1+2x2
ST
x1+x2<=3
x1-x2<=2
END
```

To solve the problem use the solve command under the solve menu or press F5 or Ctrl+s. Once the solve command is implemented LINDO will start to compile the model. If the model is not formulated correctly an error message will appear on the screen.

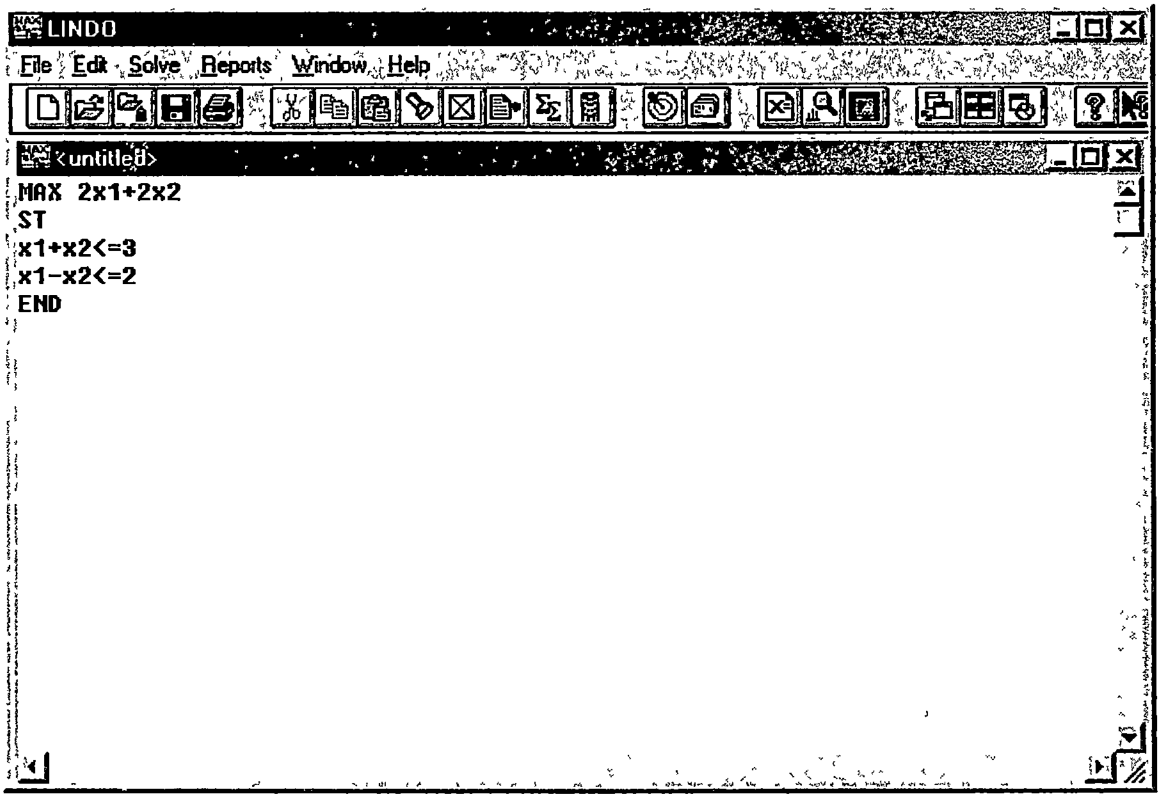


Figure A2: An Example of LP Formulation Entered into LINDO

Corrections will have to be made in order for LINDO to compute the solution. If no errors are detected the LINDO Solver Status window (Figure A3) will appear on the screen. The screen tells information about the solution. As the Table A1 points out the LINDO Solver Status screen mainly is helpful for Inter Programming (IP) formulations. A small window will appear next asking if sensitivity or range analysis needs to be found (see Figure A4). Just click no and close the LINDO Solver Status screen. This option is mainly used by advanced users.

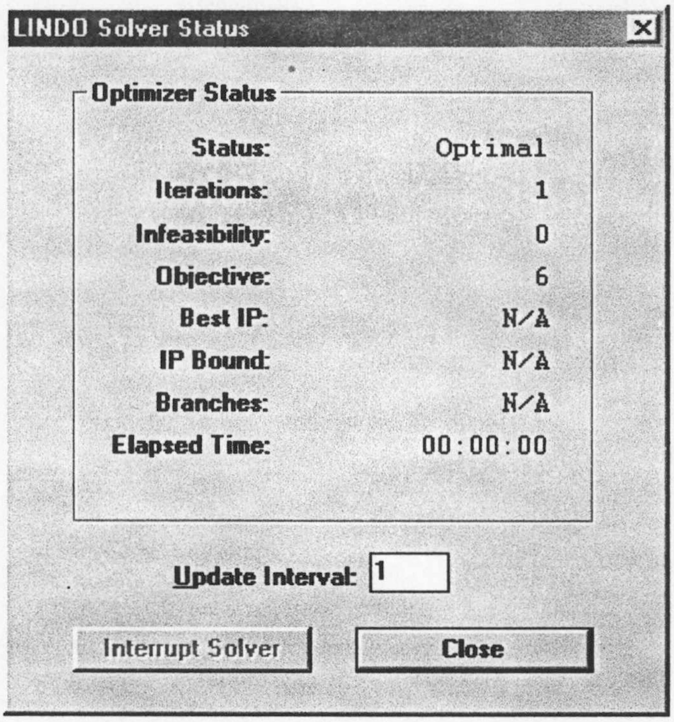


Figure A3: The LINDO Solver Status Window

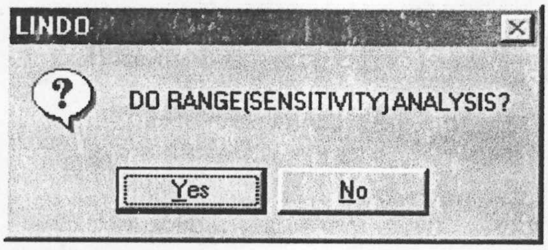


Figure A4: Example of Range Analysis Option Screen

Table A1: Description of LINDO Solver Status Window

Field/Control	Description
Status	Gives status of current solution
Iterations	Number of solver iterations
Infeasibility	Amount by which constraints are violated
Objective	Current value of the objective function
Best IP	Objective value of the best integer solution found (only used for integer programming)
IP Bound	Theoretical bound on the objective for IP*
Branches	Number of integer variables "branched" on By LINDO's IP solver
Elapsed Time	Elapsed time since the solver was invoked
Update Interval	The frequency (in seconds) that the Status Window is updated

Once the option no is clicked on the window asking for range analysis, minimize the LINDO Solver Status window. There will be a new window, the Reports Window (Figure A5). This report gives information about the solution. The report contains the following information: the number of iterations, the optimal solution, the values of the variables, the reduced cost, the dual price and the values of any of the slack/surplus variables. The reduced cost is the rate at which the objective function value will be hurt if a variable currently zero is arbitrarily forced to increase by a small amount. The dual price is the rate at which the objective function value will improve as the right hand side or constant term of the constraint is increased a small amount.

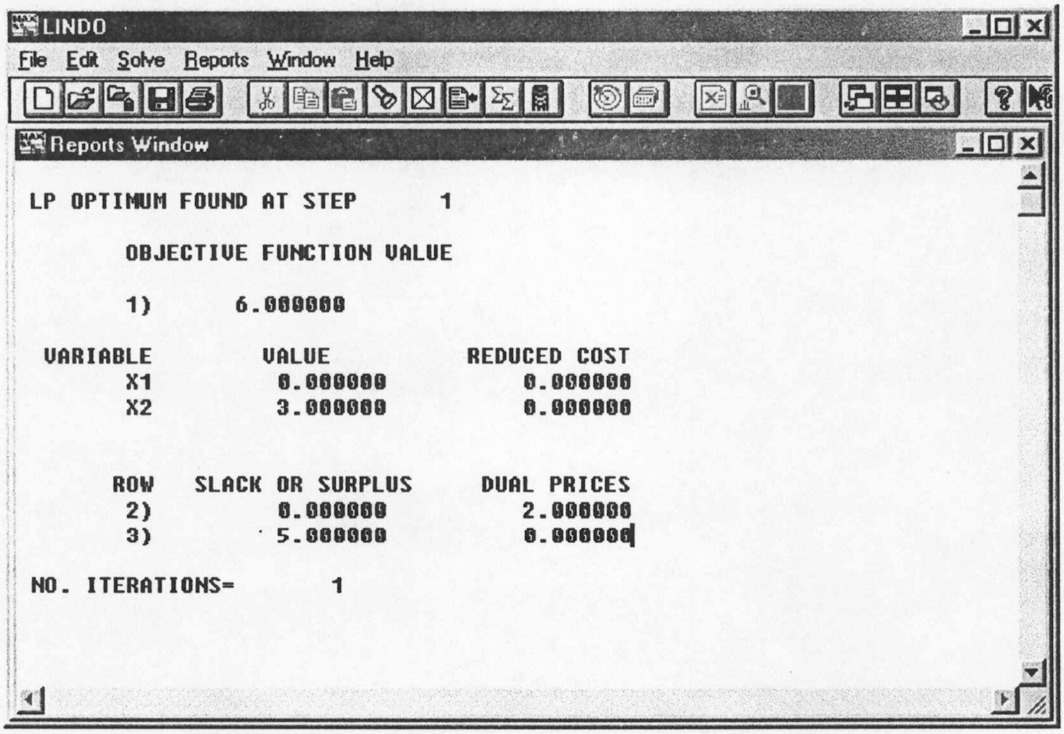


Figure A5: An Example of Report Windows

The Report Window always contains information about different rows. If the row number is 1, it corresponds to the objective function. If the row number is 2 to n, it corresponds to the constraints. For this example, Row 2 corresponds to the first constraint and Row 3 corresponds to second constraint.

For this example, the maximum objective function is z=6. The variables x_1 and x_2 are 0 and 3 respectively. Even though $x_1=0$ the reduced cost is 0, thus, a small increase in this variable will not effect the objective function. The dual price for the first constraint is 2; thus, a small increase in the constant term (3) will increase the objective function by a rate of 2.

Table A2: File Menu Commands

New	F2	Create a new Model Window
Open	F3	Open an existing file
View	F4	Open an existing View Window
Save	F5	Save the active window as text
Close	F7	Close the active window
Print	F8	Send the active window to the printer
Date	Shift+F4	Display the current date and time
Elapsed Time	Shift+F5	Display the elapsed time of current session
Title	Shift+F3	Display the title of active model
Exit	Shift+F6	Exit LINDO

Table A3: Solve Menu Commands

Compile Model	Ctrl+e	Compile the model in the active window
Solve	Ctrl+s	Send the model to LINDO solver
Pivot	Ctrl+n	Perform one iteration on current model

Table A4: Help Menu Commands

Contents	F1	Show the content of LINDO help
How to use Help	Ctrl+F1	Learn how to use the Help Menu

Table A2 to Table A4 gives some short cuts to LINDO's commands. These are only a select few of the many LINDO commands. Further information can be found on the LINDO help menu and from the LINDO website: www.lindo.com. There is also a free downloadable demo version of LINDO.

VITA

Randy Lee Collins was born June 20, 1972 in Camden, Tennessee. After graduating high school in the top ten of his class, he attended the University of Tennessee-Martin where he got a degree in Education with a concentration in secondary mathematics. Upon graduating from UT-Martin in 1995, he had a job teaching high school math in Obion County in Tennessee. After two years of teaching at the high school level, he decided to further his education by attending UT-Martin again to start a degree in Engineering. After a year of engineering classes he decided to attend the University of Tennessee by accepting a teaching assistantship in 1998.

He graduated on May 12, 2000 from the University of Tennessee with a Masters of Science degree in Mathematics -- with a concentrated in applied math.