To the Graduate Council:

I am submitting herewith a thesis written by Fathy Fouad Yassa-Greiss entitled "Optimization in Graphs and Networks." I 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 :

Steven M. Serbin

Vassilio Dougalis

L E Evans Peck

Vice Chancellor
Graduate Studies and Research

b

OPTIMIZATION IN GRAPHS AND NETWORKS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Fathy Fouad Yassa-Greiss

December 1979

To My Wife and Daughter.

ACKNOWLEDGMENTS

The author is grateful for the assistance of Dr. Yueh-er Kuo who directed the preparation of this thesis.

The author also appreciates the suggestions and the services of the other members of his Master's degree committee, Dr. Vassilios A.Dougalis and Dr. Steven M.Serbin.

A special thanks goes to the author's wife for her efforts spent in typing the primary and final copies of this thesis.

ABSTRACT

This thesis is a study of some optimization techniques in graphs and networks.

Chapter I deals with the problem of finding the shortest paths in graphs of relatively small dimensions.

Chapter II discusses and solves the problem of shortest paths in graphs of large dimensions, which can be partitioned into overlapping subgraphs. The method described in Section 2.3. and 2.4., which is due to the author of this thesis, is a means to partition a large graph into its overlapping subgraphs, and to find the cut sets and minimum cut sets in a graph.

Chapter III deals primarily with network flows. In particular, the problems of maximal flows in a network, and minimal cost flows are discussed.

Chapter IV presents flow charts and Fortran IV Computer Programs for the main algorithms presented in this thesis. These flow charts and programs are developed independently by the author.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER I

SHORTEST PATHS IN A NETWORK

1.1. Introduction to Graphs and Networks

Definition 1.1. A graph is a set of two or more different points, with certain pairs of these points joined by one or more lines. The resulting form is called a graph, and is usually denoted by G. The points referred to in the above definition will be called nodes (vertices) of the graph. We shall refer to a node i by N_i, where i is the node number.

A line joining two different nodes will be called an arc (branch or edge) of the graph. We shall refer to an arc joining nodes N_i to N_j by A_{ij}.

The two nodes joined by an arc are referred to as the end points of the arc.

Definition 1.2. An undirected arc is an edge which has no sense of direction (see Figure 1.1.).

Figure 1.1. An Undirected Arc.

A directed arc is an arc which has a sense of direction. The sense of direction is attributed to an arc by stating which node is to be considered the point of origin. Such an arc is called oriented (see Figure 1.2.).

Figure 1.2. A Directed Arc.

Definition 1.3. An undirected graph (unoriented) is simply a graph whose arcs are all undirected (see Figure 1.3.).

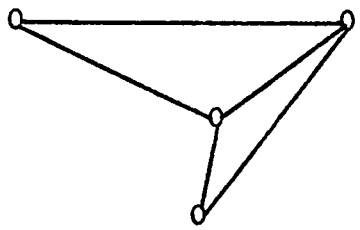


Figure 1.3. Undirected Graph.

Definition 1.4. A directed graph (oriented) is a graph whose arcs are all directed arcs (see Figure 1.4.).

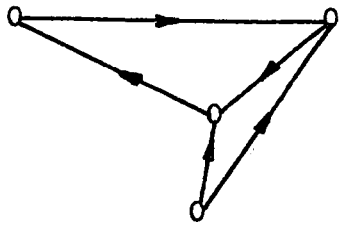


Figure 1.4. Directed Graph.

Definition 1.5. A mixed graph is a graph some of whose arcs are directed and some of which are undirected.

Definition 1.6. A path joining nodes N_i and N_j, is a set of nodes and arcs N_i, A_{ir}, N_r, A_{rk}, ... , A_{qj}, N_j, such that

each node in the set, with the possible exception of the first and last nodes, is the end point for the preceding arc and the starting point for the following arc in the set. The nodes N_i and N_j are called the extremity points of the path. Thus each arc in the path is directed away from N_i and towards N_j (see Figure 1.5. (a)).

Definition 1.7. A route from N_i to N_j is the set of nodes which fall in the path between N_i and N_j in order of occurrence, N_i , N_r , ... , N_q , N_j .

Definition 1.8. A chain is a similar structure to a path except that not all arcs are necessarily directed toward N_j (see Figure 1.5.(b)).

Definition 1.9. If a path is defined so that $i = j$, i.e. the extremity points of the path are one and the same node, then the path is called a circuit (loop) (see Figure 1.5. (c)).

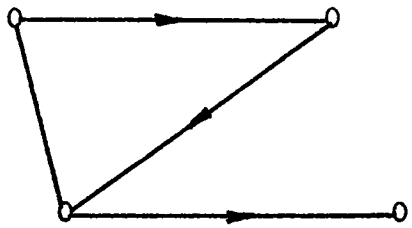
Definition 1.10. A cycle is a closed chain (see Figure 1.5. (d)).

Definition 1.11. A tree is a connected graph which has no loops (see Figure 1.5. (e)).

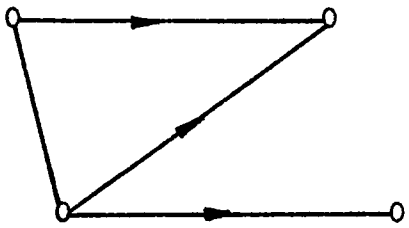
Definition 1.12. A spanning tree is a tree that includes every node of the graph.

Definition 1.13. A graph G is said to be connected if there is a path in G joining any two nodes of the graph. Throughout this paper we shall assume that G is connected.

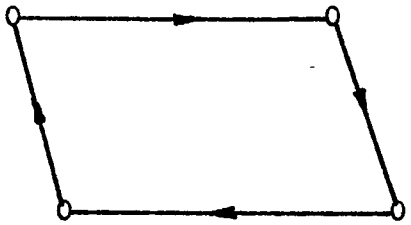
Definition 1.14. A network is a graph such that a flow can take place in the arcs of the graph.



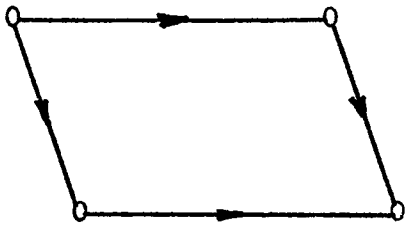
(a) Path.



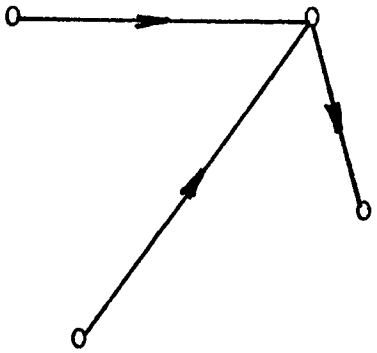
(b) Chain.



(c) Circuit.



(d) Cycle.



(e) Tree.

Figure 1.5. Path, Chain, Circuit, Cycle and Tree.

1.2. The Problem of Shortest Paths

The problem of finding a shortest path from a given starting node to another node in the network is a fundamental problem in network theory. This problem arises very frequently as a sub-problem in other problems of optimization and also has a practical interest of its own.

Consider a network consisting of the nodes N_j, $(j = 1,2,\ldots,n)$ and every arc A_{ij} of the network has associated with it a distance d_{ij}. The problem is to find a path from N_s to N_t with the sum of the distances d_{ij} of arcs in the path a minimum.

If nodes are interpreted as cities then d_{ij} can be interpreted as:

(1) The distance between city i and city j, in which case the problem is to find the shortest distance route to travel from one city to another in the graph.

(2) The time spent to travel from city i to city j, then the problem is to find the shortest time route.

(3) The cost of travelling from city i to city j, then the problem is to find the cheapest route.

There are many other practical problems in which one wants to find an optimum route under other criteria. Some of these problems will be mentioned briefly at the end of this chapter.

We shall not, in general, restrict the d_{ij} to be nonnegative. However, the total distance of any cycle must be nonnegative, otherwise, the shortest path problem is not well defined. Furthermore, the distances d_{ij} are arbitrary and <u>do not have to satisfy</u> the triangular

inequality $d_{ij} + d_{jk} = d_{ik}$, otherwise the problem would be trivial, since the arc A_{st} would be the shortest path from N_s to N_t. Also we assume that in general the distances do not have to satisfy the symmetric relation $d_{ij} = d_{ji}$.

The nonexistence of a negative cycle rules out an undirected arc having a negative length, as we regard an undirected arc as two opposite directed arcs both with the same distance as the undirected arc.

A shortest path from N_i to N_j is a path with minimum total arc lengths. If there is no single arc leading from N_i to a node N_j, we define the length $d_{ij} = \infty$. Also we define $d_{ii} = 0$.

Mathematical Formulation of the Problem

As mentioned before, we are trying to detect a path starting at a certain node, which we will call the source node and denote it by N_s, and ending at another node, which we will call the sink node and denote it by N_t, such that the total distance (cost) travelled from N_s to N_t is a minimum.

The path from N_s to N_t, in general, will include some nodes other than N_s and N_t, together with the arcs connecting these nodes to each other and to N_s and N_t.

From N_s only one of the arcs associated with N_s can be included in this path and it is directed away from N_s, and no arc directed towards N_s will be included in the path. Conversely, for N_t only one arc, of the arcs associated with it, can be included in the path and it is directed towards N_t, and no arc directed away from N_t will be included in the path (see Figure 1.6.).

For the other nodes in the path, exactly two of the arcs associated with each of these nodes will be included in the path, one directed towards it and the other directed away from it.

Let us now associate with each arc A_{ij} a variable x_{ij}, so that if A_{ij} is included in the path then $x_{ij} = 1$, and if A_{ij} is not included in the path then $x_{ij} = 0$ $(i,j = s,1,2,\ldots,n,t)$.

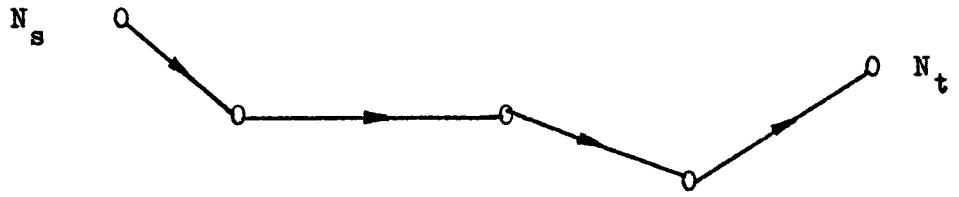


Figure 1.6. Path from N_s to N_t.

We now define the cost function Z_{ij} associated with each arc A_{ij}

$$Z_{ij} = d_{ij}\, x_{ij} \qquad i,j = s,1,2,\ldots,n,t$$

so that $Z_{ij} = d_{ij}$ if A_{ij} is in the path

and $Z_{ij} = 0$ if A_{ij} is not in the path.

Therefore the total cost function associated with the network is given by

$$Z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \cdot x_{ij} \quad ,$$

and we are required to minimize the function Z.

Now, at each node N_j, except the source N_s, only <u>one arc</u> is entering it so that,

$$x_{ij} = 1 \qquad i \neq j \;,\; j \neq s \;,\; j = 1,2,\ldots,n,t$$

and therefore,

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j \neq s \ , \ j = 1,2,\ldots,n,t \quad (a)$$

and

$$\sum_{i=1}^{n} x_{ij} = 0 \qquad j = s \quad (b)$$

Also, each node N_j , except the sink N_t, has only one arc leaving it , so that

$$x_{jk} = 1 \ , \ j \neq k \ , \ j \neq t \ , \ j = s,1,2,\ldots,n$$

and therefore

$$\sum_{k=1}^{n} x_{jk} = 1 \qquad j \neq t \ , \ j = s,1,2,\ldots n, \quad (c)$$

$$\sum_{k=1}^{n} x_{jk} = 0 \qquad j = t \quad (d)$$

Subtracting (a) from (c) and (d) respectively, we get

$$\sum_{k=1}^{n} x_{jk} - \sum_{i=1}^{n} x_{ij} = 0 \quad j \neq s \ , \ j \neq t$$

$$= -1 \quad j \neq s \ , \ j = t$$

Subtracting (b) from (c) we get

$$\sum_{k=1}^{n} x_{jk} - \sum_{i=1}^{n} x_{ij} = 1 \quad j = s \ , \ j \neq t$$

Therefore the problem is as follows :

Minimize

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \cdot x_{ij} \quad (1.1.a)$$

subject to

$$\sum_{k=1}^{n} x_{jk} - \sum_{i=1}^{n} x_{ij} = \begin{cases} 1 & j = s \\ 0 & j \neq s \text{ or } t \\ -1 & j = t \end{cases} \qquad (1.1.b)$$

and $$x_{ij} = 0 \text{ or } 1 \qquad (1.1.c)$$

This problem can be regarded as :

(1) A linear programming problem which can be solved by the simplex method.

(2) A transportation type problem.

(3) An integer linear programming problem.

These techniques will not be discussed in this paper. The methods discussed in this paper for the solution of the shortest path problem are developed specially to solve this problem and others of the same type.

1.3. The Tree Building Algorithm

The algorithm discussed in this section finds a shortest path between two specified nodes N_s and N_t. We first consider the case when all arc lengths are nonnegative and we illustrate the algorithm by an example, Then we consider the case when some or all of the arc lengths are allowed to be negative, where we present the necessary modifications to the algorithm to be valid in this case.

Before we attempt to discuss the algorithm, we first introduce the symbol "is replaced by" denoted by ":=" . Consider now the operation

$$R := \min (R , Q + T) \qquad (1.2)$$

This operation, called the triple operation, means that the value of R is compared to the value of $Q + T$ and if $R \leq Q + T$, then the value of R remains unchanged; and if $R > Q + T$, then R is replaced by $Q + T$.

Graph with Nonnegative Arc Lengths

Instead of finding the shortest path from N_s to N_t, let us find the shortest path from N_s to all other nodes N_i in the network. The reason for doing this is that any node N_i may be an intermediate node on the shortest path from N_s to N_t. If a node N_i is on the shortest path from N_s to N_t, then the subpath from N_s to N_i must be a shortest path from N_s to N_i. Otherwise, we would replace the subpath from N_s to N_i by another path of shorter distance to form a shorter path from N_s to N_t. This new path would be of shorter distance than the original path from N_s to N_t and would contradict the fact that the original path from N_s to N_t is a shortest path.

The main idea is as follows: Suppose we know the m nodes that are closest in total length to N_s in the graph, and also a shortest path from N_s to each of these nodes (a shortest path from N_s to itself is the null path which has length equal to zero).

The node N_s and these m nodes, form a tree and they are called tree nodes. All other nodes are called nontree nodes. Then the $(m + 1)^{st}$ closest node to the node N is found as follows:

For each nontree node N_y, construct m distinct paths from N_s to N_y by joining the shortest path from N_s to N_x by A_{xy} for all tree nodes N_x.

Select the shortest of these m paths and let it temporarily be the shortest path from N_s to N_y.

Then the nontree node with the shortest temporary path from N_s is the $(m+1)^{st}$ new member of the tree. The shortest path from N_s to the $(m+1)^{st}$ node member of the tree uses only tree nodes as its intermediate nodes since all arc lengths are all nonnegative. Starting with $m = 0$, this process can be repeated until the shortest path from N_s to N_t has been found.

We now formally state the tree building algorithm:

Step (1) Initially, all arcs and nodes are nontree arcs and nodes. Assign a value $d(x)$ to each node N_x of the form $d(x) = (a,b)$ where a denotes the length of the shortest path from N_s to N_x and b denotes only a tree node used as intermediate nodes. Initially, set $d(s) = (0,s)$ and $d(x) = (oo\ ,\ x)$ for all $x \neq s$. Let N_y be the last node to become a tree node.

Step (2) Let N_s become a tree node, and let $N_y = N_s$. For each nontree node N_x, redefine $d(x)$ using the triple operation (1.2) as follows:

$$d(x) := \min\ (d(x)\ ,\ d(y) + d_{yx}) \qquad (1.3)$$

In this operation $d(x)$ is of the form $d(x) = (a,b)$ where a is a distance and b is a node number, and when d_{yx}, the length of A_{yx}, enters this operation it transforms to the same form $d_{yx} \longrightarrow (f,y)$ since y is the intermediate tree node where d_{yx} originates, and f is the distance between N_y and N_x,

$$d(x) := \min\ (d(x)\ ,\ d(y) + d_{yx})$$
$$= \min\ ((a,b)\ ,\ (c,r) + (f,y))$$
$$= \min\ ((a,b)\ ,\ (c + f\ ,\ y))$$

then if $c + f \triangleright a$ therefore $d(x) = (c + f\ ,\ y)$, where we see that the total distance from N_s to N_x is $c + f$ and y has become the intermediate node nearest to N_x, and if $c + f \trianglelefteq a$ therefore $d(x) = (a,b)$.

If $d(x) = (\infty,x)$ for all nontree nodes N_x, then stop because no path exists from N_s to any nontree node. Otherwise let the nontree node N_x with the smallest value $d(x)$ become a tree node. Also, the arc directed into node N_x from a tree node that determined the value $d(x)$ in operation (1.3) becomes a tree arc.

Step (3) Let $N_y = N_x$.

If node N_t has become a tree node stop because a shortest path from N_s to N_t has been found. This path consists of the unique path of tree arcs from N_s to N_t. If node N_t has not become a tree node, repeat Step (2).

If we want to find the shortest path from N_s to all other nodes in the graph, Step (3) is modified so that we stop when the tree becomes a spanning tree (if one exists). Otherwise, return to Step (2).

Notice that each tree node has at most one tree arc directed into it (except N_s) and the tree building cannot contain a cycle since nontree arc cannot become a tree arc if both its extremity points have a tree arc incident to it. The unique path from node N_s to any other

node N_x contained in any shortest path tree is a shortest path from N_s to N_x.

Example 1.1. Consider a graph of Figure 1.7. and let us find a shortest path from node N_s to node N_t.

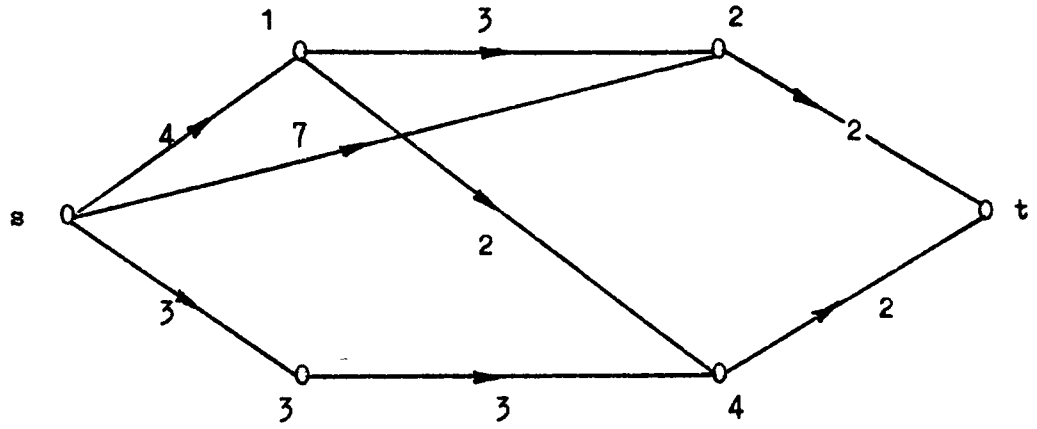


Figure 1.7. Graph for Example 1.1.

Step (1) Only N_s is a tree node $d(s) = (0,s)$ and $d(x) = (\infty, x)$ for all $x \neq s$.

Step (2) $N_y = N_s$ and

$$d(1) := \min\ (d(1)\ ,\ d(s) + d_{s1})$$
$$= \min\ ((\infty,1)\ ,\ (0,s) + (4,s))$$
$$= (4\ ,\ s)$$

$$d(2) := \min\ (d(2)\ ,\ d(s) + d_{s2})$$
$$= \min\ ((\infty,2)\ ,\ (0,s) + (7,s))$$
$$= (7\ ,\ s)$$

$$d(3) := \min\ (d(3)\ ,\ d(s) + d_{s3})$$
$$= \min\ ((\infty,3)\ ,\ (0,s) + (3,s))$$
$$= (3\ ,\ s)$$

$d(4) := \min\ (d(4)\ ,\ d(s) + d_{s4})$

$= \min\ ((\infty,4)\ ,\ (0,s) + (\infty,4))$

$= (\infty,\ 4)$

$d(t) := \min\ (d(t)\ ,\ d(s) + d_{st})$

$= \min\ ((\infty,t)\ ,\ (0,s) + (\infty,t))$

$= (\infty,t)$

then, $\min\ (d(1)\ ,\ d(2)\ ,\ d(3)\ ,\ d(4)\ ,\ d(t))$

$= \min\ ((4,s)\ ,\ (7,s)\ ,\ (3,s)\ ,\ (\infty,4)\ ,\ (\infty,t))$

$= (3,s)$

$= d(3),$

node N_3 is joined to the tree as well as the arc A_{s3} which determined $d(3)$. The current tree is shown in Figure 1.8.(a).

Step (3) N_t has not become a tree node, return to Step (2).

Step (2) $y = 3$

$d(1) := \min\ (d(1)\ ,\ d(3) + d_{31})$

$= \min\ ((4,s)\ ,\ (3,s) + (\infty,1))$

$= (4\ ,\ s)$

$d(2) := \min\ (d(2)\ ,\ d(3) + d_{32})$

$= \min\ ((7,s)\ ,\ (3,s) + (\infty,2))$

$= (7\ ,\ s)$

$d(4) := \min\ (d(4)\ ,\ d(3) + d_{34})$

$= \min\ ((\infty,4)\ ,\ (3,s) + (3,3))$

$= (6\ ,\ 3)$

$d(t) := \min\ (d(t)\ ,\ d(3) + d_{3t})$

$= \min\ ((\infty,t)\ ,\ (3,s) + (\infty,t))$

$= (\infty,t)$

and $\min\ (d(1)\ ,\ d(2)\ ,\ d(4)\ ,\ d(t))$

$= \min\ ((4,s)\ ,\ (7,s)\ ,\ (6.3)\ ,\ (\infty,t))$

$= (4,s)$

$= d(1)\ .$

So node N_1 is joined to the tree as well as arc A_{s1} which determined $d(1)$. The current tree is shown in Figure 1.8.(b).

Step (3) N_t has not become a tree node, return to Step (2).

Step (2) $y = 1$

$d(2) := \min\ (d(2)\ ,\ d(1) + d_{12})$

$= \min\ ((7,s)\ ,\ (4,s) + (3,1))$

$= (7,s)$ or $(7,1)$

$d(4) := \min\ (d(4)\ ,\ d(1) + d_{14})$

$= \min\ ((6,3)\ ,\ (4,s) + (2,1))$

$= (6,3)$ or $(6,1)$

$d(t) := \min\ (d(t)\ ,\ d(1) + d_{1t})$

$= \min\ ((\infty,t)\ ,\ (4,s) + (\infty,t))$

$= (\infty,t)$

We see that there is a tie for both $d(2)$ and $d(4)$. The tie for $d(2)$ will be solved later, but since

$$\min\ (d(2)\ ,\ d(4)\ ,\ d(t))$$
$$= (6,3) \text{ or } (6,1)$$
$$= d(4)$$

we can arbitrarily select one of these two arcs to join the tree. We arbitrarily select $d(4) = (6,3)$. So node N_4 is joined to the tree as well as arc A_{34}. Also we arbitrarily choose $d(2) = (7,s)$.

The current tree is shown in Figure 1.8.(c).

Step (3) N_t has not become a tree node, return to Step (2).

Step (2) $y = 4$

$$d(2) := \min\ (d(2)\ ,\ d(4) + d_{42})$$
$$= \min\ ((7,s)\ ,\ (6,3) + (\infty,2))$$
$$= (7,s)$$

$$d(t) := \min\ (d(t)\ ,\ d(4) + d_{4t})$$
$$= \min\ ((\infty,t)\ ,\ (6,3) + (2,4))$$
$$= (8,4)$$

$$\min\ ((7,s)\ ,\ (8,4))$$
$$= (7,s)$$
$$= d(2)$$

So node N_2 is joined to the tree as well as arc A_{s2}. The current tree is shown in Figure 1.8.(d).

Step (3) N_t has not become a tree node, return to Step (2).

Step (2) $y = 2$

$$d(t) := \min\ (d(t)\ ,\ d(2) + d_{2t})$$
$$= \min\ ((8,4)\ ,\ (7,s) + (2,2))$$
$$= (8,4)$$

Thus node N_t has become a tree node as well as arc A_{4s}. The current tree is shown in Figure 1.8.(e).
(Notice that the tree shown in Figure 1.8.(e) is a spanning tree.)

The total distance of the shortest path from N_s to N_t can be read directly from the node value $d(t) = (8,4)$, where the total distance from N_s to N_t is equal to 8. Also using the labels $d(x)$ we can track the shortest path from N_s to N_t as follows:

$d(t) = (8,4)$ then,

$d(4) = (6,3)$ then,

$d(3) = (3,s)$ so that, the path from N_s to N_t is

$$N_s\ ,\ A_{s3}\ ,\ N_3\ ,\ A_{34}\ ,\ N_4\ ,\ A_{4t}\ ,$$

or simply the route is

$$N_s\ ,\ N_3\ ,\ N_4\ ,\ N_t$$

along tree arcs.

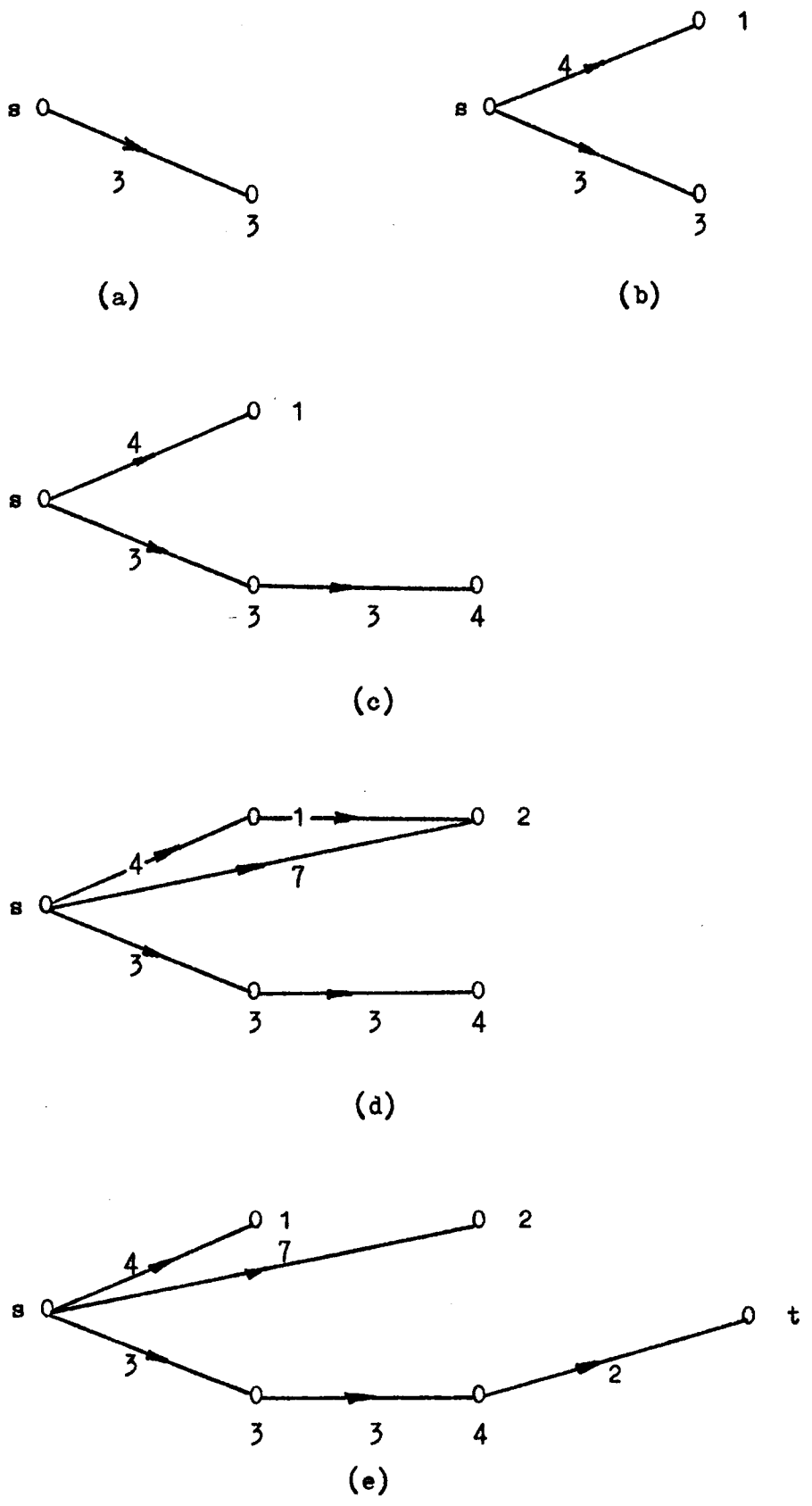


Figure 1.8. Tree Building for Example 1.1.

Graph with Negative Arc Lengths

We assumed in the previous method that all arc lengths were non-negative. What would happen if some of the arc lengths were negative ? For example, consider the graph in Figure 1.9.

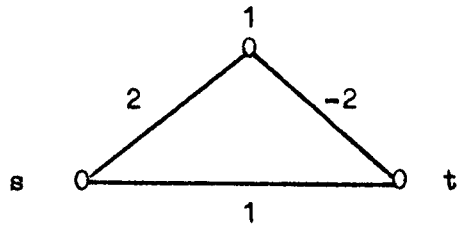


Figure 1.9. Graph with Negative Arc Lengths.

The shortest path from N_s to N_t is

$$N_s \ , \ A_{s1} \ , \ N_1 \ , \ A_{1t} \ , \ N_t \ ,$$

whose length is

$$+ 2 - 2 = 0 .$$

We can easily verify that if the shortest path algorithm presented in the preceding method were applied to this graph, then the path N_s , A_{st} , N_t would erroneously be selected as the shortest path from N_s to N_t .

Fortunately, the tree building algorithm just discussed can be generalized to accommodate arcs with negative lengths. This can be done by three simple changes, namely:

(1) In Step (2), let equation (1.3) be applied to all nodes, not only nontree nodes. Hence, a tree node as well as a nontree node can have its node number $d(x)$ decreased.

(2) If a tree node has its node number decreased, then remove the tree arc incident to it from the tree.

(3) Terminate the algorithm only after all nodes have become tree nodes and Step (2) fails to lower any node numbers.

Proof. The proof is achieved by contradiction. The algorithm cannot terminate unless

$$d(x) + d_{xy} \leq d(y) \quad \text{for all} \quad A_{xy} \tag{1.4}$$

Otherwise, N_y would be a nontree node during the iteration of Step (2) occurring immediately after vertex x has become a tree node.

Suppose that upon termination of the algorithm $d(y)$ does not equal the length of a shortest path from N_s to N_y for some N_y. (If there is more than one such N_y, then take N_y to be the node with the shortest path from N_s that contains the fewest arcs.) Whenever a node number $d(z)$ is finite, it equals the length of some paths from N_s to N_z.

Hence, it follows upon termination $d(y)$ must equal the length of some path from N_s to N_y, and consequently, $d(y)$ must exceed the length of a shortest path from N_s to N_y. Let node N_x be the next-to-last node on the shortest path from N_s to N_x. (If there is more than one such path, select any path with the fewest number of arcs.) Thus, $d(x)$ must equal the length of a shortest path from N_s to N_x, and

$$d(y) \geq d(x) + d_{xy} \tag{1.5}$$

This contradicts (1.4)

Q.E.D.

Can this algorithm fail ? Yes, if the graph contains a circuit whose total length is negative (a negative circuit). In this case, the circuit might be repeated infinitely many times yielding to a nonsimple path whose length is infinitely negative.

To remedy this situation, we apply the algorithm anyway, but we keep count on the number of times each node has entered the tree. As soon as any node has entered the tree for the n^{th} time, where n is the number of nodes in the graph, stop because the graph contains a negative circuit. Otherwise, the algorithm terminates in a finite number of steps with correct results.

To show this, suppose that there are no negative circuits in the graph. When any node N_x receives its final node value $d(x)$, then at worst every other node can enter the tree once more before another node receives its final node value. Hence, no node can enter the tree more than $(n - 1)$ times.

In this case we will not illustrate by an example as the algorithm presented in the next section will cover this case very simply, without worrying whether negative cycles exist or not. In fact, the next algorithm will allow detecting negative cycles very easily.

1.4. Multiterminal Shortest Paths

A Matrix Algorithm

In this section, we will present and discuss an algorithm to find shortest paths between all pairs of nodes. As before all d_{ij} do not have to satisfy the triangular inequality or the symmetric relations, we shall not restrict the d_{ij} to be nonnegative in general.

As we mentioned , the whole problem lies in the fact that

$$d_{ij} + d_{jk} \not\geq d_{ik} ,$$

otherwise, the shortest path between every pair of nodes N_i and N_k would be the arc A_{ik} . However, there must be some arcs A_{ij} for which the shortest path from N_i to N_j is the arc A_{ij}. Let us call these arcs basic arcs. Note that in Section 1.3. all tree arcs are basic but not vice versa.

Suppose we know the shortest path from N_p to N_q , say. Then this path must contain only basic arcs ; otherwise the fact that the path from N_p to N_q is a shortest path will be contradicted. Now if we create an arc A_{pq} with distance equal to the shortest chain, then A_{pq} becomes a basic arc. This problem of multiterminal shortest paths is solved by creating a basic arc (if no such arc already exists) between every pair of nodes. The distance associated with the newly created basic arc is equal to the shortest path formed by basic arcs of the original graph.

Consider now the triple operation defined by (1.2), but in this case it is defined for a given node N_j called the pivot node:

$$d_{ik} := \min (d_{ik} , d_{ij} + d_{jk}) \quad \text{for all } i \neq k \neq j \qquad (1.6)$$

The triple operation as in (1.6) compares the distance of an arc d_{ik} with the distance of a path consisting of two arcs with N_j as intermediate node, and changes the original distance of the arc if it is the larger of the distances.

If we do this operation (1.6) for every node $N_j (j = 1,2,\ldots,n)$, then the new d_{ik} are lengths of the shortest paths between every pair of nodes.

In this algorithm, the amount of computations is $n(n-1)(n-2)$ additions and comparisons.

Proof. Any shortest path must consist of basic arcs of the original graph. The rpoof shows that at the end of computations, a basic arc of distance equal to the sum of the distances of the basic arcs in the shortest path is created. Consider any arbitrary shortest path such as the one shown in Figure 1.10.

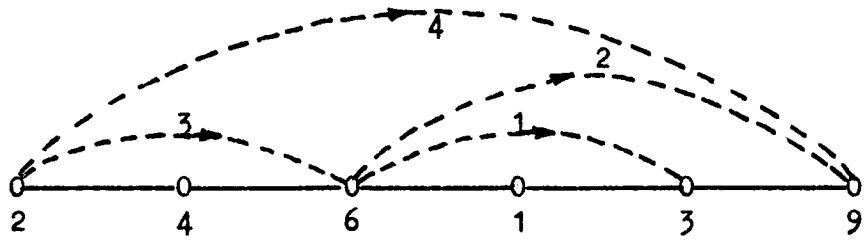


Figure 1.10. Basic Arcs.

The operation (1.6) will create basic arcs successively as shown by the dotted lines. For a graph without negative cycles, the shortest distance between any two nodes is well defined. If the operation (1.6) creates an arc A_{29} that is not equal to the true shortest distance, then that distance will be replaced by the true shortest distance at some later step. Once the true shortest distance is created, then it cannot be replaced, since it is the minimum.

In Figure 1.10. we have an arbitrary shortest path with basic arcs A_{24} , A_{61} , A_{46} , A_{13} and A_{39} . Since in the operation (1.6) j is successively fixed at $1,2,\ldots,9,\ldots$, we create successively the new basic arcs A_{63} , A_{69} , A_{26} and A_{29}.

Q.E.D.

For any graph G with a finite number of nodes n, we shall construct a matrix $D = (d_{ij})$ of dimension nxn called the matrix of distances. The entries of this matrix D will be the distances associated with each arc in the graph. The row number will stand for the node from which an arc originates, and the column number will stand for the node at which an arc terminates.

As mentioned before if no <u>single</u> arc exists between two specific nodes the corresponding entry will be set to ∞.

Also, for the present application we will set $d_{ii} = 0$ for all $i = 1,2,\ldots,n$. If an undirected arc exists say A_{st} , in this case we will set $d_{st} = d_{ts}$. In general D will not be a symmetric matrix. From the proof we can see that when j in operation (1.6) is successively fixed at $1,2,\ldots,n$, then for each of these values, operation (1.6) is carried out for each entry d_{ik} with $i \neq j$ and $k \neq j$, to fill the $(n-1)(n-1)$ matrix obtained be deleting the j^{th} row and column. Since $d_{ii} = 0$ $(i = 1,2,\ldots,j-1,j+1,\ldots,n)$ we need only perform $(n-1)^2 - (n-1) = (n-1)(n-2)$ operations for each fixed value of j. As there are n values of j there are totally $n(n-1)(n-2)$ operations like operation (1.6) to perform, where each operation involves one addition and one comparison.

This number of additions and comparisons is the maximum that can be expected for a network with arbitrarily defined distances.

At the end of this number of operations the resulting matrix D^*, is the matrix of shortest distances.

In all the previous discussions we have defined $d_{ii} = 0$ for all i. If we define $d_{ii} = \infty$ for all i, then operation (1.6), allowing $i = k$, will provide a technique for finding minimum cycles containing N_i. In many applications we are interested in finding negative cycles, and if d_{ii} becomes negative, it shows that a negative cycle exists.

To keep track of the arcs that make up the shortest path, we use the following bookkeeping. In a matrix of dimension $n \times n$, $R = (r_{ij})$, whose entries are calculated along with operation (1.6), the entry in row i and column k indicates the first intermediate node on the shortest chain from N_i to N_k if there is such an intermediate node. Let k be the entry if there is no intermediate node. At the start all entries of the matrix in the (i,k) positions are set to k,

$$r_{ik} = k \tag{1.7}$$

then

$$r_{ik} = \begin{cases} j & \text{if } d_{ik} > d_{ij} + d_{jk} \\ \text{remains the same} & \text{if } d_{ik} \leq d_{ij} + d_{jk} \end{cases} \tag{1.8}$$

At the end of the $n(n-1)(n-2)$ iteration operations (1.8) (which are carried out along with operation (1.6)) the resulting matrix R^*, called the route matrix is the matrix by means of which one can detect any route of shortest distance, if one exists, between any pair of nodes in the graph.

To prove that operation (1.8) works, let N_j be the intermediate node with smallest index in a shortest path from N_p to N_q and N_i and N_k are two neighboring nodes of N_j on the shortest path. During the computations, $d_{ik} > d_{ij} + d_{jk}$, and we set $r_{ik} = r_{ij} = j$.

Since A_{ij} and A_{jk} are basic arcs, $r_{ij} = j$ and $r_{jk} = k$ throughout the computation. This proves the correctness of operation (1.8) for a shortest path with one intermediate node.

Now the original shortest path from N_p to N_q is replaced by another, a path with equal distance but one less intermediate node, since a basic arc A_{ik} with distance equal to $d_{ij} + d_{jk}$ is created. The proof is completed by induction.

In finding the intermediate nodes of a shortest path, or simply a route of shortest distance from N_p to N_q using R^*, we first look at the entry $r_{pq} = k$, which means that N_k is the first intermediate node in the shortest path from N_p to N_q. Then we look at r_{pk} to find the first intermediate node in the shortest path from N_p to N_k. This process is repeated until we find the entry $e_{pi} = i$ which indicates that we have found all the intermediate nodes in the shortest path, namely N_p, N_i, ..., N_k, N_q.

We are now ready to illustrate the above discussed algorithm with an example.

Example 1.2. Consider the graph in Figure 1.11. where the numbers beside the arcs are arc lengths and the numbers beside the nodes are node numbers. We want to find shortest paths between every pair of nodes.

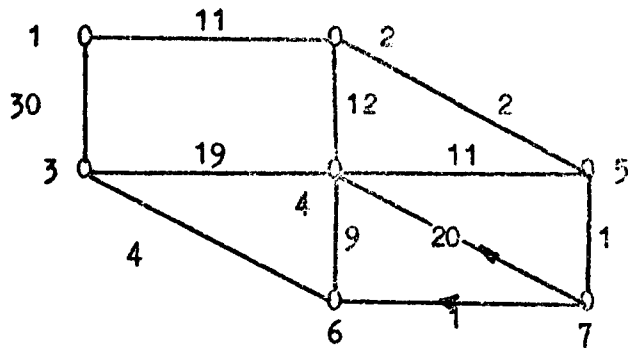


Figure 1.11. Graph for Example 1.2.

The initial distance matrix D and the initial route matrix R are shown in Tables 1.1. and 1.2. respectively.

Table 1.1. Initial Distance Matrix D for Example 1.2.

	1	2	3	4	5	6	7
1	0	11	30	∞	∞	∞	∞
2	11	0	∞	12	2	∞	∞
3	30	∞	0	19	∞	4	∞
4	∞	12	19	0	11	9	∞
5	∞	2	∞	11	0	∞	1
6	∞	∞	4	9	∞	0	∞
7	∞	∞	∞	20	1	1	0

Table 1.2. Initial Route Matrix R for Example 1.2.

	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	1	2	3	4	5	6	7
3	1	2	3	4	5	6	7
4	1	2	3	4	5	6	7
5	1	2	3	4	5	6	7
6	1	2	3	4	5	6	7
7	1	2	3	4	5	6	7

To apply the operation (1.6) for $j = 1$, we check each entry in Table 1.1. which does not lie in the first column or in the first row

$$d_{23} := \min (d_{23} , d_{21} + d_{13})$$
$$= \min (\infty , 11 + 30)$$
$$= 41$$

$$d_{32} := \min (d_{32} , d_{31} + d_{12})$$
$$= \min (\infty , 30 + 11)$$
$$= 41 .$$

All other entries in the $(n-1)(n-1)$ submatrix remain the same. The above will result in Table 1.3. along with the new route matrix R shown in Table 1.4.

Table 1.3. New Distance Matrix for j=1.

	1	2	3	4	5	6	7
1	0	11	30	∞	∞	∞	∞
2	11	0	41	12	2	∞	∞
3	30	41	0	19	∞	4	∞
4	∞	12	19	0	11	9	∞
5	∞	2	∞	11	0	∞	1
6	∞	∞	4	9	∞	0	∞
7	∞	∞	∞	20	1	1	0

Table 1.4. New Route Matrix for j=1.

	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	1	2	1	4	5	6	7
3	1	1	3	4	5	6	7
4	1	2	3	4	5	6	7
5	1	2	3	4	5	6	7
6	1	2	3	4	5	6	7
7	1	2	3	4	5	6	7

For j=2, we apply the operation (1.6) to each of the entries in Table 1.3. which does not lie in the second column or the second row and updating the bookkeeping matrix of node indices (Table 1.4.) then,

$$d_{43} = d_{34} := \min (d_{34} , d_{32} + d_{24})$$
$$= \min (19 , 14 + 12)$$
$$= 19.$$

$$d_{41} = d_{14} := \min (d_{14} , d_{12} + d_{24})$$
$$= \min (\infty , 11 + 12)$$
$$= 23.$$

$$d_{51} = d_{15} := \min (d_{15} , d_{12} + d_{25})$$
$$= \min (\infty , 11 + 2)$$
$$= 13.$$

$$d_{53} = d_{35} := \min (d_{35} , d_{32} + d_{25})$$
$$= \min (\infty , 41 + 2)$$
$$= 43.$$

And if we carry through the computations similar to the previous two steps, we get the pairs of Tables 1.5. to 1.16. where Table 1.15. and 1.16. are the final tables.

Table 1.5. New Distance Matrix for j=2.

	1	2	3	4	5	6	7
1	0	11	30	23	13	∞	∞
2	11	0	41	12	2	∞	∞
3	30	41	0	19	43	4	∞
4	23	12	19	0	11	9	∞
5	13	2	43	11	0	∞	1
6	∞	∞	4	9	∞	0	∞
7	∞	∞	∞	20	1	1	0

Table 1.6. New Route Matrix for j=2.

	1	2	3	4	5	6	7
1	1	2	3	2	2	6	7
2	1	2	1	4	5	6	7
3	1	1	3	4	2	6	7
4	2	2	3	4	5	6	7
5	2	2	2	4	5	6	7
6	1	2	3	4	5	6	7
7	1	2	3	4	5	6	7

Table 1.7. New Distance Matrix for j=3.

	1	2	3	4	5	6	7
1	0	11	30	23	13	34	∞
2	11	0	41	12	2	45	∞
3	30	41	0	19	43	4	∞
4	23	12	19	0	11	9	∞
5	13	2	43	11	0	47	1
6	34	45	4	9	47	0	∞
7	∞	∞	∞	20	1	1	0

Table 1.8. New Route Matrix for j=3.

	1	2	3	4	5	6	7
1	1	2	3	2	2	3	7
2	1	2	1	4	5	3	7
3	1	1	3	4	2	6	7
4	2	2	3	4	5	6	7
5	2	2	2	4	5	3	7
6	3	3	3	4	3	6	7
7	1	2	3	4	5	6	7

Table 1.9. New Distance Matrix for j=4.

	1	2	3	4	5	6	7
1	0	11	30	23	13	32	∞
2	11	0	31	12	2	21	∞
3	30	31	0	19	30	4	∞
4	23	12	19	0	11	9	∞
5	13	2	30	11	0	20	1
6	32	21	4	9	20	0	∞
7	43	32	39	20	1	1	0

Table 1.10. New Route Matrix for j=4.

	1	2	3	4	5	6	7
1	1	2	3	2	2	4	7
2	1	2	4	4	5	4	7
3	1	4	3	4	4	6	7
4	2	2	3	4	5	6	7
5	2	2	4	4	5	4	7
6	4	4	3	4	4	6	7
7	4	4	4	4	5	6	7

Table 1.11. New Distance Matrix for j=5.

	1	2	3	4	5	6	7
1	0	11	30	23	13	32	14
2	11	0	31	12	2	21	3
3	30	31	0	19	30	4	31
4	23	12	19	0	11	9	12
5	13	2	30	11	0	20	1
6	32	21	4	9	20	0	21
7	14	3	31	12	1	1	0

Table 1.12. New Route Matrix for j=5.

	1	2	3	4	5	6	7
1	1	2	3	2	2	4	5
2	1	2	4	4	5	4	5
3	1	4	3	4	4	6	5
4	2	2	3	4	5	6	5
5	2	2	4	4	5	4	7
6	4	4	3	4	4	6	5
7	5	5	5	5	5	6	7

Table 1.13. New Distance Matrix for j=6.

	1	2	3	4	5	6	7
1	0	11	19	23	13	32	14
2	11	0	25	12	2	21	3
3	30	25	0	13	24	4	25
4	23	12	13	0	11	9	12
5	13	2	24	11	0	20	1
6	32	21	4	9	20	0	21
7	14	3	5	10	1	1	0

Table 1.14. New Route Matrix for j=6.

	1	2	3	4	5	6	7
1	1	2	6	2	2	4	5
2	1	2	6	4	5	4	5
3	1	6	3	6	6	6	6
4	2	2	6	4	5	6	5
5	2	2	6	4	5	4	7
6	4	4	3	4	4	6	5
7	5	5	6	6	5	6	7

Table 1.15. D* for Example 1.2.

	1	2	3	4	5	6	7
1	0	11	19	23	13	15	14
2	11	0	8	12	2	4	3
3	30	25	0	13	24	4	25
4	23	12	13	0	11	9	12
5	13	2	6	11	0	2	1
6	32	21	4	9	20	0	21
7	14	3	5	10	1	1	0

Table 1.16. R* for Example 1.2.

	1	2	3	4	5	6	7
1	1	2	6	2	2	7	5
2	1	2	7	4	5	7	5
3	1	6	3	6	6	6	6
4	2	2	6	4	5	6	5
5	2	2	7	4	5	7	7
6	4	4	3	4	4	6	5
7	5	5	6	6	5	6	7

Now if we want the shortest path from N_1 to N_6 say, we look at Table 1.15. where we read $d_{16}^{*} = 15$, and to find the nodes in this path we look at Table 1.16. where :

$$r_{16} = 7 \quad , \quad r_{17} = 5 \quad , \quad r_{15} = 2 \quad , \quad r_{12} = 2 \quad ,$$

and therefore the route is:

$$N_1 \ , \ N_2 \ , \ N_5 \ , \ N_7 \ , \ N_6 \ .$$

1.5. Modifications and Applications

In this section we present some applications of the algorithm discussed in Section 1.4., as well as some modifications to the operation (1.6) to fit other types of applications.

Application 1

An airline is approached by a passenger who wishes to fly from Springfield, Illinois to Ankara, Turkey spending as little time as possible in the air since he is afraid of flying. How should the airline route this passenger ?

The airline should construct a graph whose nodes are the airports between Springfield and Ankara and whose arcs correspond to the flights between the corresponding airports. Let the length of each arc equal the corresponding flight time. The airline should now read the remainder of this chapter to learn how to find a shortest path between the nodes corresponding to Springfield and Ankara.

Application 2

Suppose that you wish to drive from Boston to Los Angeles using only interstate highways. What is the shortest route to take ?

Construct a graph whose nodes correspond to the junctions of the interstate highways joining their respective nodes. Let the length of each arc equal the mileage along the highway that it represents.

The routing problem between Boston and Los Angeles can now be solved by finding the shortest path from the node representing the highway junction in Boston where you would start your journey to the node representing the highway junction in Los Angeles where you would end your journey.

Application 3

You must have an automobile at your disposal for the next five years until your retirement. There are currently a variety of automobiles that you may purchase with different expected lifetimes and costs and there are a variety of leasing arrangements available. What should you do ?

Represent each possible transaction date during the next five years as a node. (To simplify, you might consider only the first day of each month as a possible transaction date.) Represent each possible purchase or lease by an arc from its initial transaction date node to its maturity date node. Let the length of each arc equal the cost of the corresponding transaction. The best purchase/lease combination must correspond to the shortest path from the node representing the current time to the node representing your retirement date.

Application 4

Suppose that a salesman in Boston plans to drive to Los Angeles to visit an important client. He plans to use the interstate highway system described in Application 2, and he plans to visit other clients along his route. He knows on the average how much commission he can earn from each proposed stop along the way from Boston to Los Angeles. What route should he take from Boston to Los Angeles ?

In this situation, the length of each arc in the graph representing the highway system should be set equal to the expected net profit (expected commission less driving expenses) of the corresponding highway segment. The salesman should drive along the longest path from the Boston node to the Los Angeles node.

Observe that in this case some of the arc lengths will be negative where the salesman expects to incur a loss, and positive where the salesman expects to make a profit.

In such a case operation (1.6) is modified to become

$$d_{ik} := \max (d_{ik} , d_{ij} + d_{jk}) \qquad (1.9)$$

and initially $d_{ij} = -\infty$ wherever no single arc exists between N_i and N_j.

Notice that in equation (1.8) "$\wedge$" is replaced by "$\vee$" and vice versa.

Application 5

A small investor must decide how to invest optimally his funds for the coming year. A variety of investments (passbook savings account, certificate of deposit, savings bonds, etc.) are available. For simplicity suppose that investments can only be made and withdrawn on the first of each month. Create a node corresponding to the first day of each month. Place an arc from node N_x to node N_y for each investment that can be made at time x and mature at time y. Notice that this graph contains no circuits. Let the length of each arc equal the profit earned on the corresponding investment.

This case, like Application 4, is a maximization problem and therefore can be solved using equation (1.9).

Application 6

Given a graph with arc capacities b_{ij}, a route from N_p to N_q is called a maximum capacity route if the minimum arc capacity of this route is not less than the minimum arc capacity of any other route from N_p to N_q.

As an application of the triple operations (1.2) presented in this chapter, we can use the following operations to get maximum capacity routes between all pairs of nodes in the network:

$$b_{ik} := \max (b_{ik} , \min (b_{ij} , b_{jk}))$$

for all $i , k \neq j$ and for $j = 1,2,\ldots,n$ successively. Another $n \times n$ matrix $R = (r_{ij})$ similar to the route matrix described in the preceding section, where the entries r_{ik} are all set equal to k initially.

Then,

$$r_{ik} = \begin{cases} j & b_{ik} < \min (b_{ij} , b_{jk}) \\ \text{unchanged if} & b_{ik} \geq \min (b_{ij} , b_{jk}) . \end{cases}$$

Application 7

In many physical applications, it is customary to refer to the state of a system. For example, in Newtonian mechanics one is often interested in the "system" consisting of a mass particle (or some

physical object) moving in a certain environment. Usually one refers to the position and velocity of the particle at a specified time as the "state" of the system at the same time. That is, the physical laws determine how the system passes from one state on to subsequent states.

This terminology can profitably be carried over to the shortest route problem. Think once again of a car and driver moving along certain streets. Let us agree to refer to the city streets, the car, and the driver as the "system" under consideration. The "state" of this system at a specified time could be defined to be the location of the car at that time. The shortest route problem could then be stated as the problem of finding the least time in which the system can pass from the initial state (in which the car is at the starting vertex) to the final state (in which the car is at the terminal vertex).

In Newtonian mechanics there is ordinarily an infinity or "continuum" of states to be considered. For example, a planet moving around the sun is thought of as moving in a continuous way from state to state. It can occupy infinitely many locations in the course of the journey.

In our work, however, we need distinguish only a finite number of states. In fact, we have seen that we only need to consider the times t_{ij} required for a car to go from one vertex to another. Thus, we could define the possible states for a map with N vertices.

Of course when we define the states in this way, we are making an abstract model of the real situation, and in so doing we are ignoring many things which we regard as of minor importance in relation to our objective. For example, non of the variations of speed or other events which occur between two vertices are taken into account. Then, step by step we introduce more realistic features.

CHAPTER II

A DECOMPOSITION ALGORITHM FOR SLIGHTLY OVERLAPPING GRAPHS

2.1. The Problem

In most cases a graph has less than $n(n-1)$ arcs and is far from being completely connected. This means that the associated distance matrix $D = (d_{ij})$ has many entries which are infinity. This fact gives rise to a problem of computer storage and the number of operations performed, when a computer is used to solve the problem of shortest paths specially in large scale graphs ($n = 15 , 20 , 30 , \ldots$).

Before we attempt to find a solution to this problem, let us first introduce some new concepts and definitions.

Definition 2.1. Take a subset of nodes in a graph and denote it by A. Let X be another subset of nodes. We call the set X a cut set of A if it has the following property:

If the set X together with its incident arcs are deleted from the graph, then the graph becomes two disconnected components, one component containing all the nodes of A and no other nodes.

Definition 2.2. A cut set X of A is called a minimum cut set if no proper subset of X has the disconnecting property of X.

Clearly all the neighboring nodes of A constitute a minimum cut set of A.

2.2. A Decomposition Algorithm

The decomposition algorithm described in this chapter takes advantage of the situation mentioned in the former section. We shall first consider the decomposition algorithm in its simplest form where a graph is decomposed into two parts. Next, we shall consider the general case where we have a decomposition of a graph into m overlapping subgraphs.

Decomposition Algorithm for Two Overlapping Graphs

Let us assume that the graph G is partitioned into three sets of nodes

$$A \quad , \quad X \quad , \quad B \quad ,$$

such that,

$$G \quad = \quad A \cup X \cup B \, ,$$

where X is a minimum cut set of A. If the nodes in set A are assigned with the indices $1,2,\ldots, \ |A| \ ,$ (where $|A|$ denotes the cardinality of the set A) and nodes in X are assigned the indices $|A| + 1 \, , \ |A| + 2 \, , \ \ldots \, , \ |A| + |X| \, ,$ then the associated distance matrix is of the form shown in Table 2.1. where

$$D_{AA} \quad = \quad (d_{ij}) \quad \text{with} \quad N_i \in A \quad \text{and} \quad N_j \in A \, ,$$

$$D_{AB} \quad = \quad (d_{ij}) \quad \text{with} \quad N_i \in A \quad \text{and} \quad N_j \in B \, , \quad \text{etc.}$$

Table 2.1. Partitioned Distance Matrix for Two Overlapping Graphs.

	A	X	B
A	D_{AA}	D_{AX}	D_{AB}
X	D_{XA}	D_{XX}	D_{XB}
B	D_{BA}	D_{BX}	D_{BB}

Initially, all entries in D_{AB} and D_{BA} are infinity. In general, we use the notation $D_{\alpha\beta}$ to be the distance matrix (d_{ij}) with $N_i \in \alpha$ and $N_j \in \beta$. Sometimes it is convenient to use $d_{\alpha\beta}$ to represent one of the entries in $D_{\alpha\beta}$.

Definition 2.3. A conditional shortest path subject to the restriction that nodes in the path must be a certain subset of nodes of the graph.

We shall use d^*_{ij} to denote the shortest distance (using any number of arcs) from N_i to N_j. Now we use $d^*_{ij}(\delta)$ to denote the distance of the conditional shortest path from N_i to N_j, where δ is the subset of nodes in which the conditional shortest path must lie. The matrix of conditional shortest distances $(d^*_{ij}(\delta))$ with $N_i \in \alpha$ and $N_j \in \beta$ is denoted by $D^*_{\alpha\beta}(\delta)$, and $d^*_{\alpha\beta}(\delta)$ represents one of its entries. If G is used to denote the whole graph, then $d^*_{ij}(G) = d^*_{ij}$. If we know that the shortest paths lie in the set δ,

then

$$d^{*}_{\alpha\beta}(\delta) \quad = \quad d^{*}_{\alpha\beta}(G) \qquad (2.1)$$

Let $\bar{A}$ denote A U X and $\bar{B}$ denote B U X . We consider the original graph as two graphs overlapping each other. One network, called $\bar{A}$ graph consists of nodes N_i $(N_i \in \bar{A})$ and arcs A_{ij} $(N_i, N_j \in \bar{A})$. The other network, called the $\bar{B}$-graph, consists of nodes N_k $(N_k \in \bar{B})$ and arcs A_{kr} $(N_k , N_r \in \bar{B})$.

The following two theorems give the foundations of the decomposition algorithm for two overlapping graphs.

Theorem 2.1

Let $G = A \cup X \cup B$, where the removal of X will make the graph disconnected. Then the shortest distances between any two nodes in the $\bar{B}$-graph can be obtained by considering only the $\bar{B}$ provided that the conditional shortest distances $D^{*}_{XX}(\bar{A})$ are known.
(note that $\bar{A} = G - B$).

Proof: If the shortest path lies entirely in $\bar{B}$, then,

$$d^{*}_{\bar{B}\bar{B}}(G) \quad = \quad d^{*}_{\bar{B}\bar{B}}(\bar{B}) \quad ,$$

which means it is sufficient to do the triple operations in the $\bar{B}$-graph . Assume that there are many subpaths, of a shortest path, which contain nodes in $\bar{A}$. This is shown symbolically in Figure 2.1.

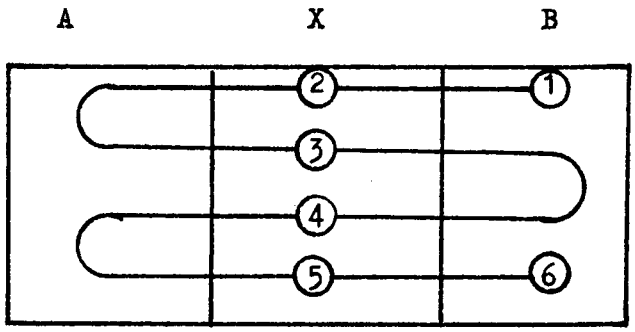


Figure 2.1. Symbolic Representation of Two Overlapping Graphs.

Consider a shortest path from N_1 to N_6. Since both the starting node and the terminal node are in $\bar{B}$ and X is a cut set of B, any subpath that contains nodes in A must begin and end in the set X. In Figure 2.1. the subpath from N_2 to N_3 and the subpath from N_4 to N_5 are two such paths.

If $d_{23}(\bar{A}) = d_{23}(G - B)$

and $d_{45}(G - B) = d_{45}(\bar{A})$

are known, we have effectively in the $\bar{B}$-graph, two arcs A_{23} with distance $d^*_{23}(G - B)$ and A_{45} with distance $d^*_{45}(G - B)$.

The shortest path from N_1 to N_6 then consists of the subpath from N_1 to N_2, A_{23}, the subpath from N_3 to N_4, A_{45}, and the subpath from N_5 to N_6, all of which contain arcs in the $\bar{B}$-graph. Thus, it is sufficient to consider only the $\bar{B}$-graph. Note that this reasoning is independent of the number of subpaths which contain nodes

of $\bar{A}$. Clearly if we interchange A and B and $\bar{A}$ and $\bar{B}$, then the theorem is also true.

Q.E.D.

Theorem 2.2

Let $G = A \cup X \cup B$, where the removal of X will make the graph disconnected. Then

$$d^*_{AB}(G) = \min_X \left(d^*_{AX}(G) + d^*_{XB}(G) \right) \qquad (2.2)$$

$$d^*_{BA}(G) = \min_X \left(d^*_{BX}(G) + d^*_{XA}(G) \right) \qquad (2.3)$$

Proof. To get from any node N_i in A to any node N_k in B , we must pass at least one node N_j in X. If the minimum is taken over all N_j in X, then it is certainly the minimum distance. Let the distance matrix $D^*_{AX}(G)$ be $n_a \times p$ and the distance matrix $D^*_{XB}(G)$ be $p \times n_b$. Then the operation as carried out for a given i and k is

$$d^*_{ik} = \min_j \left(d^*_{ij} + d^*_{jk} \right) \qquad (j = 1,\ldots,p,\ N_i \in A\ ,\ N_k \in B) \qquad (2.4)$$

Q.E.D.

In Theorem 2.2 the total number of additions is $n_a \times n_b \times p$ with the same number of comparisons. Operation(2.4) is like an ordinary matrix multiplication with (+) replacing (x) and (min) replacing ($\sum$) . We shall refer to operation (2.4) as a matrix minisummation .

Based on these two theorems, the triple operations are performed in the $\bar{A}$-graph . This will give the conditional shortest distances between each pair of nodes in the graph $\bar{A}$. In particular, for two nodes in X the conditional shortest distances will replace the original distances between the two nodes. Then we apply the triple operations on the $\bar{B}$-graph where the distances between nodes in X have been replaced by the conditional shortest distances calculated from the $\bar{A}$-graph . After the triple operations in the $\bar{B}$-graph, we apply the triple operations on the $\bar{A}$-graph again.

In fact, the following steps describe completely the decomposition algorithm for two overlapping graphs:

Step (1) Execute the triple operations for each of the entries of the matrix

$$\begin{bmatrix} D_{AA} & D_{AX} \\ D_{XA} & D_{XX} \end{bmatrix}$$

At the end of the step we have $D^*_{AA}(\bar{A})$, $D^*_{AX}(\bar{A})$, $D^*_{XA}(\bar{A})$ and $D^*_{XX}(\bar{A})$.

Step (2) Execute the triple operations on the matrix

$$\begin{bmatrix} D^*_{XX}(\bar{A}) & D_{XB} \\ D_{BX} & D_{BB} \end{bmatrix}$$

At the end of the step we have, from Theorem 2.1 ,

$$D^*_{XX}(G) \quad , \quad D^*_{XB}(G) \quad , \quad D^*_{BX}(G) \quad \text{and} \quad D^*_{BB}(G) \; .$$

Step (3) Execute the triple operations on the matrix

$$\begin{bmatrix} D^*_{AA}(\bar{A}) & D^*_{AX}(\bar{A}) \\ D^*_{XA}(\bar{A}) & D^*_{XX}(G) \end{bmatrix}$$

At the end of this stepwe have, from Theorem 2.1,

$$D^*_{AA}(G) \quad , \quad D^*_{AX}(G) \quad , \quad D^*_{XA}(G) \quad \text{and} \quad D^*_{XX}(G) \; .$$

Step (4) Use the minisummation operation (2.4) to obtain

$$D^*_{AB}(G) \quad \text{and} \quad D^*_{BA}(G) \; .$$

At this point we might try to get an idea about the number of additions (which is the same for comparisons). Let us recall from Chapter I, Section 1.4. that a matrix of order n requires $n(n-1)(n-2)$ additions. Simply we shall take this number to be n^3. So in Step (1) $(|A| + |X|)^3$ additions are needed, and in Step(2) $(|B| + |X|)^3$ additions,$(|A| + |X|)^3$ in Step (3) , and $2 |A| . |X| . |B|$ additions are needed in Step (4).

Therefore the total number of additions needed are

$$2 (|A| + |X|)^3 + (|B| + |X|)^3 + 2 (|A| . |X| . |B|).$$

If the matrix is caluclated as a whole, we need $(|A| + |X| + |B|)^3$ additions. For the sake of comparison let $|A| = |B|$. Then the number of additions that can be reduced from the decomposition is

$$5 |A|^3 + |A|^2 |X| - 3 |A| |X|^2 - 2 |X|^3 .$$

Thus the decomposition reduces computation for $|A| \searrow |X|$.

When $|A| \neq |B|$, then the number of additions that can be reduced is

$$3\,|A| \cdot |B|\,(\,|A| + |B|\,) - |A|^3 - 3\,|A|^2\,|X| + 4\,|A|\,|B|\,|X| - 3\,|A|\,|X|^2 - 2\,|X|^3 .$$

For example if

$$|B| = 5/10\ n \quad , \quad |A| = 4/10\ n \quad , \quad |X| = 1/10\ n$$

the amount of saving is $(494 / 1{,}000)\ n^3$, which is approximately about half of the addition we would have performed using the triple operation without the decomposition algorithm.

Decomposition Algorithm for m Overlapping Graphs

When the graph is extremely large and loosely connected, it is of advantage to decompose the graph into many overlapping graphs. Let the distance matrix of the graph be decomposed into four overlapping graphs as shown in Table 2.2., where the unshaded areas indicate entries that are all infinity originally.

To construct such a distance matrix like Table 2.2., we proceed as follows:

(1) Label any subset of nodes as A.

(2) Then its minimum cut set is X_A .

(3) Let B be a sub set (not necessarily a cut set) connected to $A \cup X_A$, and X_B be the minimum cut set of $A \cup X_A \cup B$. (Notice that the minimum cut set of B is $X_A \cup X_B$, and $B \cup X_B$ is a cut set of $\bar{A}$.)

Table 2.2. The Partitioned Distance Matrix for 4 Overlapping Graphs.

	A	X_A	B	X_B	C	X_C	D
A	D_{AA}	D_{AX_A}	D_{AB}	D_{AX_B}	D_{AC}	D_{AX_C}	D_{AD}
X_A	D_{X_AA}	$D_{X_AX_A}$	D_{X_AB}	$D_{X_AX_B}$	D_{X_AC}	$D_{X_AX_C}$	D_{X_AD}
B	D_{BA}	D_{BX_A}	D_{BB}	D_{BX_B}	D_{BC}	D_{BX_C}	D_{BD}
X_B	D_{X_BA}	$D_{X_BX_A}$	D_{X_BB}	$D_{X_BX_B}$	D_{X_BC}	$D_{X_BX_C}$	D_{X_BD}
C	D_{CA}	D_{CX_A}	D_{CB}	D_{CX_B}	D_{CC}	D_{CX_B}	D_{CD}
X_C	D_{X_CA}	$D_{X_CX_A}$	D_{X_CB}	$D_{X_CX_B}$	D_{X_CC}	$D_{X_CX_C}$	D_{X_CD}
D	D_{DA}	D_{DX_A}	D_{DB}	D_{DX_B}	D_{DC}	D_{DX_C}	D_{DD}

(4) Let C be a set connected to $A \cup X_A \cup B \cup X_B$ and X_C be the minimum cut set of $A \cup X_A \cup B \cup X_B \cup C$.

(5) Continue until no further decomposition is desired.

In Table 2.2., the original graph is decomposed into four overlapping subgraphs:

$\bar{A}$ - graph , ($\bar{A}$ = A U X_A)

$\bar{B}$ - graph , ($\bar{B}$ = X_A U B U X_B)

$\bar{C}$ - graph , ($\bar{C}$ = X_B U C U X_C)

and $\bar{D}$ - graph , ($\bar{D}$ = X_C U D) .

The steps for the general decomposition algorithm for m overlapping graphs:

$\bar{A}$ - graph , $\bar{B}$ - graph , ... , $\bar{K}$ - graph , $\bar{H}$ - graph ,

are as follows:

Step (1) Perform the triple operations on the (m-1) graphs $\bar{A},\bar{B},\ldots,\bar{K}$, successively, where the conditional shortest distances obtained in one graph will replace original distances in the succeeding graph. That is to say $D^*_{X_AX_A}(\bar{A})$ will replace $D_{X_AX_A}$ before we perform the triple operations on B = X_AU B U X_B. (Notice that we can take the sets A, A U X_A U B , ... , successively as the set A in Theorem 2.1. and B U X_B U ... U H , C U X_C U ... U H , ... , correspondingly, as the set B in Theorem 2.1., then we get the following shortest distance matrices.

$$D^*_{\bar{A}\bar{A}}(\bar{A}) \quad , \quad D^*_{\bar{B}\bar{B}}(\bar{A} \cup \bar{B}) \quad , \quad \ldots \quad , \quad D^*_{\bar{K}\bar{K}}(\bar{A} \cup \bar{B} \cup \ldots \cup \bar{K}).$$

Step (2) Perform the triple operations on the m networks, $\bar{H}$, $\bar{K}$, ... , $\bar{B}$, $\bar{A}$, successively, where the distances obtained in one graph will replace the distances of the succeeding graph, that is $D^*_{X_KX_K}(G)$ will replace $D^*_{X_KX_K}(G - H)$. From Theorem2.1. we have

$$D^*_{\bar{H}\bar{H}}(G) \quad , \quad D^*_{\bar{K}\bar{K}}(G) \quad , \quad \ldots \quad , \quad D^*_{\bar{B}\bar{B}}(G) \quad , \quad D^*_{\bar{A}\bar{A}}(G) \quad .$$

Step (3) Find the shortest distances between any two nodes which are not both in one of the sets $\bar{A}$, $\bar{D}$, ... , $\bar{K}$, $\bar{H}$ by minisummation (operation (2.4)).

We shall use the notation $A \oplus X_A \oplus B$ to denote the matrix minisummation with $N_i \in A$, $N_j \in X_A$ and $N_K \in B$. Though both $A \oplus X_A \oplus B$ and $B \oplus X_A \oplus A$ should be calculated, for simplicity write one of them. The order in which minisummation should be executed is as follows:

$$A \oplus X_A \oplus B \cup X_B ,$$

$$A \cup X_A \cup B \oplus X_B \oplus C \cup X_C ,$$

$$A \cup X_A \cup B \cup X_B \cup C \oplus X_C \oplus D \cup X_D ,$$

$$A \cup X_A \cup B \cup X_B \cup C \cup \ldots . \cup F \oplus X_F \oplus K \cup X_K ,$$

$$A \cup X_A \cup B \cup X_B \cup C \cup \ldots . \cup K \oplus X_K \oplus H .$$

In the above matrix minisummation, the $D^*_{AX_B}(G)$ obtained in the first matrix minisummation are used in the second minisummation.

Again we turn our attention at this point to the number of arithmetic operations used in the decomposition algorithm; we assume that $|A| = |B| = \ldots = |H| = t$ and $|X_A| = |X_B| = \ldots = |X_K| = s$. Again we shall calculate the number of additions, since the number of comparisons needed is the same.

In Step (1) the number of additions is $(t+s)^3 + (m-2).(t+2s)^3$. In Step (2) the number of additions is $2(t+s)^3 + (m-2).(t+2s)^3$. In Step (3) the number of additions is

$$2(t+(2t+s) + \ldots + ((m-2)t+(m-3)s))s(t+s)+2((m-1)t+(m-2)s)st$$

$$= m(m-1)t^2s+s(m-1)(m-2)ts^2+(m-2)(m-3)s^3.$$

Therefore the total number of additions is

$$(2m-1)t^3 + (m^2+11m-15)t^2s+(2m^2+18m-35)ts^2+ (m^2+11m-23)s^3 \qquad (2.5)$$

If we do not use the decomposition algorithm but perform the triple operations on the entire matrix, the number of additions is

$$(mt + (m-1)s)^3 \qquad (2.6)$$

For larger m, equation (2.5) approaches $m^2s(t+s)^2$ and equation (2.6) approaches $m^3(t+s)^3$. Thus the ratio of equations (2.5) and (2.6) approaches $(s/(t+s)) / m$ as $m \longrightarrow \infty$.

Each of the subgraphs $\bar{A}, \bar{B}, \ldots, \bar{H}$ can certainly be decomposed as if each one of them was the original graph. Thus the decomposition algorithm can always be used to decompose a graph into finer and finer graphs until decomposition no longer pays.

At this point we should emphasize the fact that each of the subsets of nodes like A, X_A, B, X_B, etc., need not be connected. We can take any subset as A. But, once A is determined, X_A is uniquely determined. The set B can be any set that is connected to $A \cup X_A$, but X_B is the minimum cut set that disconnects $A \cup X_A \cup B$.

The decomposition of the graph described above can be classified as a linear decomposition since the graph is partitioned linearly into m overlapping sets as shown in Figure 2.2.

Consider a graph consisting of m overlapping graphs, which overlap each other arbitrarily as shown in Figure 2.3. In this case we may still decompose the graph linearly by letting $\bar{A}^* = \bar{E}$, say, $\bar{B}^* = \bar{A} \cup \bar{F}$, $\bar{C}^* = \bar{B} \cup \bar{C} \cup \bar{D}$, $\bar{D}^* = \bar{G}$ and $\bar{E}^* = \bar{H}$, for example and then decompose $\bar{B}^*$ into two small graphs and $\bar{C}^*$ into three small graphs.

It is possible to decompose the graph into the graphs $\bar{A}$, $\bar{B}$, ... ,$\bar{K}$, etc., directly , but the number of operations needed would be more than the number needed to decompose the graph linearly.

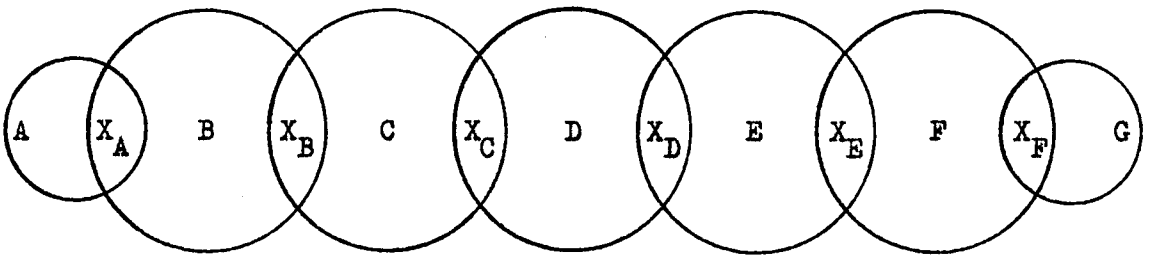


Figure 2.2. Linearly Partitioned Graph.

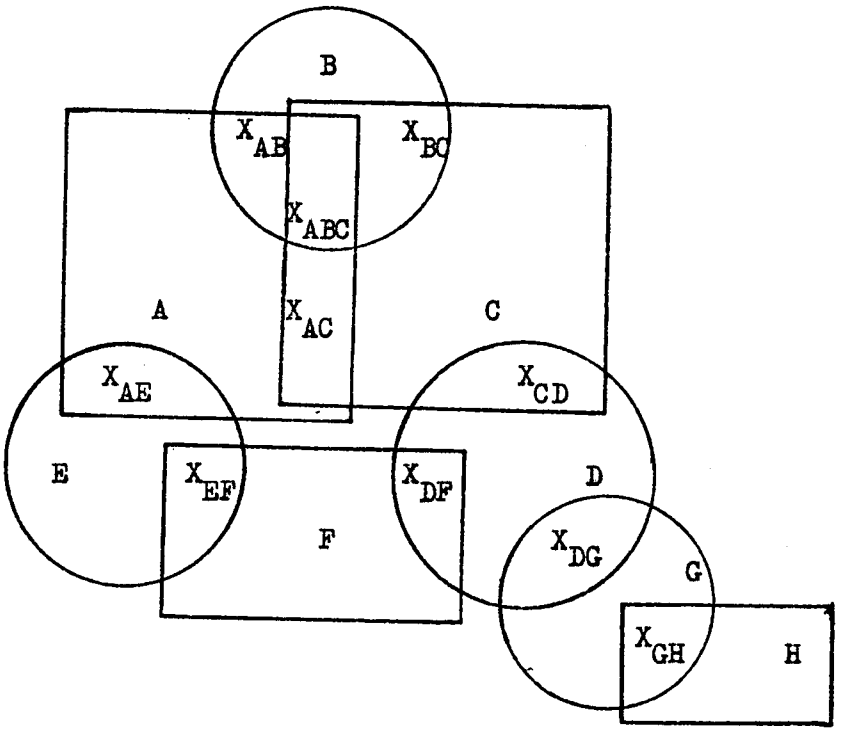


Figure 2.3. Arbitrarily Overlapping Graphs.

At this point it is reasonable to introduce an idea which can be used to explain the decomposition algorithm or can be used together with the decomposition algorithm.

We are given an n-node graph for which we want to find the shortest paths between p nodes , p ≤ n . Then a p-node graph is said to be <u>distance equivalent</u> to the n-node graph if the distance of p(p-1) pairs of shortest paths of the two graphs are the same. In Figure 2.4. the two graphs are distance equivalent with respect to the nodes N_1 , N_2 , N_3 and N_4 .

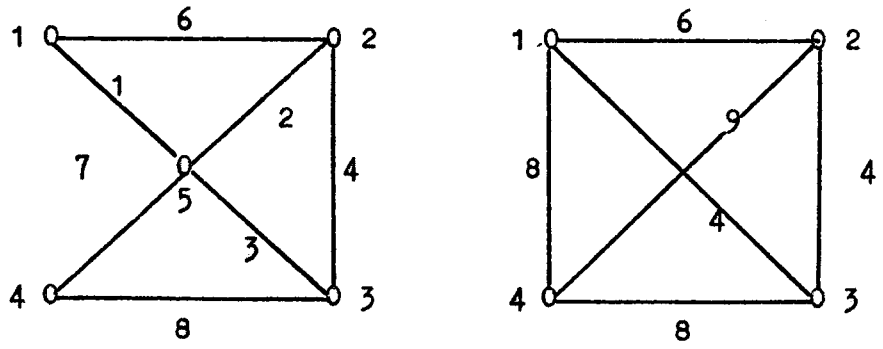


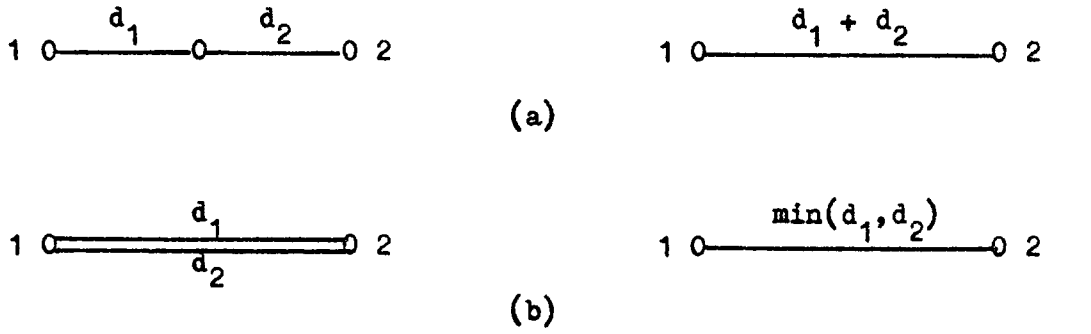
Figure 2.4. A 5 to 4 Nodes Distance Equivalent Graphs.

This idea of distance equivalent graphs can be used locally to eliminate nodes and arcs which are not of concern to us during a particular phase in the calculations, simple formulae can be derived to eliminate these nodes and arcs.

Some graphs and their distance equivalent graphs are shown in Figure 2.5. In Figure 2.5.(c) it is possible that the shortest distance from N_2 to N_3 is $d_4 + d_6$ or $d_2 + d_1 + d_6$, but when we convert the four-node graph to the triangular graph, all shortest distances can be found in the triangular graph. If we apply simple transformationsas those shown in Figure 2.5., we can eliminate (n-p) nodes and construct the distance equivalent p-node graph.

Then we can apply the triple operations or use the decomposition algorithm on the p-node graph.

The proof of Theorem 2.1. can be thought of as eliminating the set A when we obtain $D^{*}_{XX}(A \cup X)$.



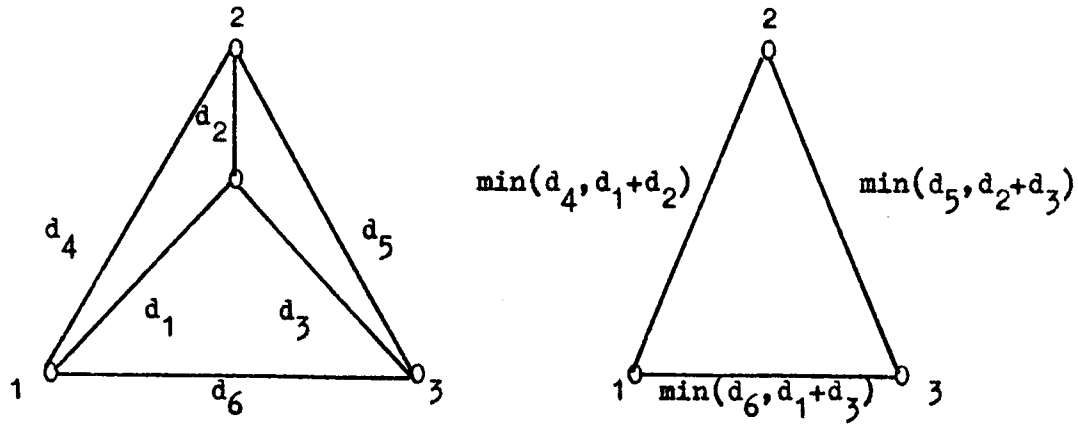


Figure 2.5. Distance Equivalent Graphs.

2.3. Permutation Matrices

This section is devoted to present specific operations in dealing with matrices, which will be useful for the algorithm presented in Section 2.4.

The operations mentioned above are some of the elementary row and column operations. Among all these operations known in the theory of matrices, the ones that concern us are those dealing with rows and column interchanges.

Let I represent the identity matrix of order n

$$I = (\delta_{ij})_{nxn} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

where $$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \qquad i,j = 1,2,\ldots,n.$$

Definition 2.4. A permutation matrix P_{rs}, where $r < s$, is defined to be that matrix obtained by interchanging the r^{th} and s^{th} rows (columns) of the identity matrix I,

$$P_{rs} = \begin{array}{c} \begin{matrix} & & r & & s & \end{matrix} \\ \begin{bmatrix} 1 & & \cdot & & \cdot & \\ & \cdots & \cdot & & \cdot & \\ & & 1\cdot & & \cdot & \\ & \cdots\cdots & 0 & \cdots\cdot & 1 & \cdots\cdot \\ & & \cdot 1 & & & \\ & & \cdot & \cdots & & \\ & & & & 1\cdot\cdot & \\ & \cdots\cdots & 1 & \cdots\cdot & 0 & \cdots\cdot \\ & & \cdot & & \cdot & \\ & & \cdot & & \cdot & 1 \end{bmatrix} \end{array} \begin{matrix} \\ \\ \\ r \\ \\ \\ \\ s \\ \\ \end{matrix} \qquad (2.7)$$

That is P_{rs} has 1 in all the diagonal positions except the (r,r) and (s,s) positions, 1 in the (s,r) and (r,s) positions and 0 elsewhere.

Now consider a matrix A of dimension nxn where

$$A = (a_{ij}) \qquad i,j = 1,2,\ldots,n$$

and let $P_{rs} = (\gamma_{ij})$ of dimension nxn as defined by(2.7).

Proposition 1

If we multiply A by P_{rs} from theleft, this will result in an interchange in positions of the r^{th} and s^{th} rows.

Proposition 2

If we multiply A by P_{rs} from the right, this will result in an interchange in positions of the r^{th} and s^{th} columns.

Proof. (1) Let $B = P_{rs}A$ then by the definition of matrix multiplication, we have

$$b_{ij} = \sum_{k=1}^{n} \gamma_{ik}\, a_{kj} \qquad i,j = 1,2,\ldots,n$$

we have that $\gamma_{ii} = 1 \qquad i \neq r\ ,\ i \neq s\ ,\ i = 1,2,\ldots,n$

and $\gamma_{rr} = \gamma_{ss} = 0 \qquad r \neq s$

$$\gamma_{sr} = \gamma_{rs} = 1$$

and $\gamma_{ij} = 0$ otherwise, so that we have,

$$b_{ij} = a_{ij} \qquad i \neq s\ ,\ i \neq r\ ,\ i = 1,2,\ldots,n$$
$$j = 1,2,\ldots,n$$

$$b_{rj} = \gamma_{rs} a_{sj}$$

$$b_{rj} = a_{sj} \qquad j = 1,2,\ldots,n \qquad (2.8)$$

$$b_{sj} = \gamma_{sr} a_{rj}$$

$$b_{sj} = a_{rj} \qquad j = 1,2,\ldots,n \qquad (2.9)$$

So we have by (2.8) and (2.9) that the r^{th} row of B is identical to the s^{th} row of A , and the s^{th} row of B is identical to the r^{th} row of A .

(2) The proof for Proposition 2 is exactly the same as for Proposition 1 except that

$$C = A P_{rs}$$

and $$C_{ij} = \sum_{k=1}^{n} a_{ik} \gamma_{kj}$$

Q.E.D.

Now for any matrix A we can use the same idea to find a permutation matrix P by means of which we can reorder the rows(columns) of A , (here we want to change the place of more than two rows using only one permutation matrix P). Let us refer to the matrix of row interchange by P_{row} and the matrix of column interchange by P_{col}.

We can find P_{row} by simply referring to the identity matrix I. Once we know the new order for the rows of A, we reorder the corresponding rows of I in the required order. The resulting matrix is the P_{row} we are seeking. Obviously if a row is to be kept in its place, its corresponding identity matrix row remains in its place.

P_{col} can be attained similarly but with the word column substituted for the word row in the above. We illustrate this with an example.

Example 2.1. Consider the matrix A given by

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 6 & 5 & 4 & 7 & 8 & 9 \\ 12 & 10 & 11 & 2 & 3 & 1 \\ 13 & 14 & 15 & 9 & 7 & 8 \\ 10 & 3 & 5 & 1 & 12 & 11 \end{bmatrix}$$

We require that the rows of A be ordered as follows:

$$4 , 6 , 3 , 2 , 5 , 1$$

from top to bottom of matrix.

Then P_{row} is given by

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Then multiplying A by P_{row} from the left we get

$$P_{row}A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 6 & 5 & 4 & 7 & 8 & 9 \\ 12 & 10 & 11 & 2 & 3 & 1 \\ 13 & 14 & 15 & 9 & 7 & 8 \\ 10 & 3 & 5 & 1 & 12 & 11 \end{bmatrix}$$

$$= \begin{bmatrix} 12 & 10 & 11 & 2 & 3 & 1 \\ 10 & 3 & 5 & 1 & 12 & 11 \\ 6 & 5 & 4 & 7 & 8 & 9 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 9 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$$

It can be easily shown that if the same ordering is to be given to the columns of A as well, then simply,

$$P_{col} = P_{row}^{T}$$

where the superscript T refers to matrix transpose.

Also we can show that if we want to restore the original order of the matrix rows (cols) we simply multiply the new matrix by $P_{row}^{T}(P_{col}^{T})$ from the left (right), since the inverse of permutation matrix is again a permutation matrix and $P^{-1} = P^{T}$.

To identify the original rows (columns) order of a permuted matrix we proceed as follows:

(1) Construct a column vector X whose elements are

$$x_i = i \qquad i = 1,2,\ldots,n \tag{1.10}$$

(2) Multiply X by $P_{row}(P_{col})$ from the left (right)

$$y = P_{row} \cdot X \tag{1.11}$$

The resulting column vector y will have as its elements the old row numbers placed in the new positions of the rows.

2.4. Decomposition Algorithm Using Permutation Matrices

In this section we present a systematic method to patition the given graph into its overlapping subgraphs necessary for the decomposition algorithms presented in Section 1.1. and 1.2. With this method we can also find automatically the minimum cut sets needed to disconnect one graph from the other.

Here we give a means to find these overlapping subgraphs, which might be the most difficult thing to find and which is most necessary, before applying the decomposition algorithm.

This method, which is due to the author of this paper, depends mainly on the idea of permutation matrices presented in the former Section 2.3.

We shall first present the above mentioned method and then conclude this chapter with an illustrating example.

The method consists of two main parts, the first is concerned with permuting the columns and rows of the distance matrix D, and the second part is concerned with partitioning the given graph.

Part I. Permutations

Step (1) Start with column 1 and row 1 of the distance matrix D. Check every entry d_{j1} and d_{1j}, $(j = 1,2,\ldots,n)$ if both entries are ∞, the row j of the identity matrix remains unchanged. If anyone of the two entries is non-infinity, when $j=r$, say, interchange row r of the identity matrix with row S_1 of the same matrix, where S_i is given by

$$S_i = \min(j \mid d_{ij} = \infty \text{ and } d_{ji} = \infty) \quad j = 1,2,\ldots,n \qquad (2,12)$$

If two or more rows and/or columns have non-infinity entries in the first column or the first row, check the left tails of the rows and the lower tails of the columns. The row (column) with shortest tail is the one that comes in the first place, so that the row of the identity matrix corresponding with row number equal to the row(column) number of the shortest tail row (column) comes first and then comes the identity matrix row corresponding to the next shortest tail row and so forth, until all rows and columns have been checked for non-infinity entries in the 1^{st} row and column and their corresponding identity matrix rows have come in the upper most positions.

The resulting matrix is the permutation matrix we are seeking. Let this permutation matrix be P_1. Then perform the matrix multiplication

$$D_1 = P_1 \; D \; P_1^T \qquad (2.13)$$

where the subscript 1 refers to row number 1 and column 1 of D.

Step (2) Let $i=2$ and check the entries d_{2j} and d_{j2} of row 2 and column 2 of D_1 starting at the value $j = r_2$ where $d_{1k} = \infty$ and $d_{k1} = \infty$ for all $k = r_2$ and proceed exactly as in Step (1), until all entries of column 2 and row 2 of D_1 have been checked.

The resulting permutation matrix will be P_2. Perform the matrix multiplication $D_2 = P_2 D_1 P_2^T$.

Step (3) Successively let $i = 3,4,\ldots n$ and check the entries d_{ij} and d_{ji} of column i and row i for each value of j starting at the value of $j = r_i$, where

$$d_{(i-1)k} = \infty \text{ and } d_{k(i-1)} = \infty \text{ for all } k = r_i$$

and proceed exactly as in Step (1) until all entries of column i and row i of the matrix D_{i-1} have been checked. At each value of i the resulting permutation matrix is P_i, and perform at each value of i the matrix multiplication

$$D_i = P_i D_{i-1} P_i^T \qquad (2.13)$$

Notice that if $d_{ik} = \infty$ and $d_{ki} = \infty$ for all $k = r_i$ then no permutations are necessary at this value of i, and the permutation matrix for this value of i is

$$P_i = I$$

At the end of the step n the resulting matrix D_n is given by

$$D_n = P_n D_{n-1} P_n^T$$

where it is clear that $P_n = I$ and D_n is the matrix which we will partition to find the overlapping subgraphs of the original graph.

Also we have found throughout the steps n : $P_1, P_2, P_3, \ldots, P_n$. Then the permutation matrices of all permutations performed, P_{row} and P_{col}, are given by

$$P_{row} = P_{col}^T = P_n \, P_{n-1} \, \ldots\ldots\ldots \, P_2 \, P_1 \qquad (2.14)$$

and

$$D_n = P_{row} \, D \, P_{col} \qquad (2.15)$$

Construct the column vector $X = (x_i)$ of dimension $(n \times 1)$ where

$$x_i = i \qquad i = 1, 2, \ldots, n$$

and perform the following multiplication

$$y = P_{row} X \qquad (2.16)$$

The column vector y of dimension $(n \times 1)$ is a vector whose elements are the indices of the nodes in the given graph. An index in the i^{th} position of the vector y implies that the i^{th} row and the i^{th} column of the D_n matrix correspond to the node in the graph with that index.

Also another matrix which will be needed during the computations of the decomposition algorithm is the new route matrix R_n. We first set the original route matrix R following equation (1.7).

Then R_n is obtained by

$$R_n = P_{row} R P_{col} \tag{2.17}$$

At this stage D_n is of the form shown in Figure 2.6.,

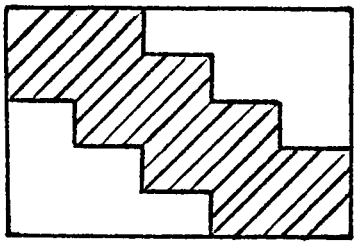


Figure 2.6. The Matrix D_n.

where all white parts refer to ∞ entries, and the shaded part refers to finite as well as some ∞ entries.

Part II. Partitioning the graph

This part describes how to reach a partition of the given graph by partitioning the matrix D_n.

The procedure is quite simple:

Step (1) Choose the set of the first r columns and rows such that all the entries in the first row to the left of the r^{th} column and all the entries in the first column below the r^{th} row are all ∞ .

This set of r columns and rows, chosen this way, is the minimum set that can represent the $\bar{A}$ ($\bar{A} = A \cup X_A$) subgraphs, and it is the first set in the partitioning procedure of the graph.

Obviously one can choose a set of larger cardinality. Refer to the y column vector (2.16) to identify the nodes included in this set $\bar{A}$, these are the elements in the first r positions of y, (See Figure 2.7.(a)).

Step (2) Check the $(r + 1)^{st}$ row and $(r + 1)^{st}$ column for the first non-infinity entry in entry in each of them. Pick the one that comes first in position and this column or row p, say, is the boundary for the set X_A. Draw a horizontal and vertical partitioning lines above the p^{th} row and to the left of the p^{th} column, (See Figure 2.7.(b)).

Step (3) Pick a set B whose cardinal number is s where

$$s \geq 1$$

Draw a horizontal and vertical partitioning line below the row $s + r$ and to the right of column $s + r$. These lines are the left and upper boundaries of the set X_B,(See Figure 2.7.(c)).

Step (4) In column $r + s$ search for the lowest non-infinity entry, and in row $r + s$ search for the non-infinity entry placed most to the right.

Pick the one whichever comes last in position, say, at position q. Draw a horizontal and vertical partitioning line to the right of column q and below row q.

These two lines will give the remaining boundaries for X_B and $\bar{B}$ as well, and they will intersect with the boundaries for X_A (See Figure 2.7.(d)).

Step (5) Proceed along the same lines described in Steps (3) and (4) to find the sets C, X_C, D and so forth, until no further partitioning is necessary.

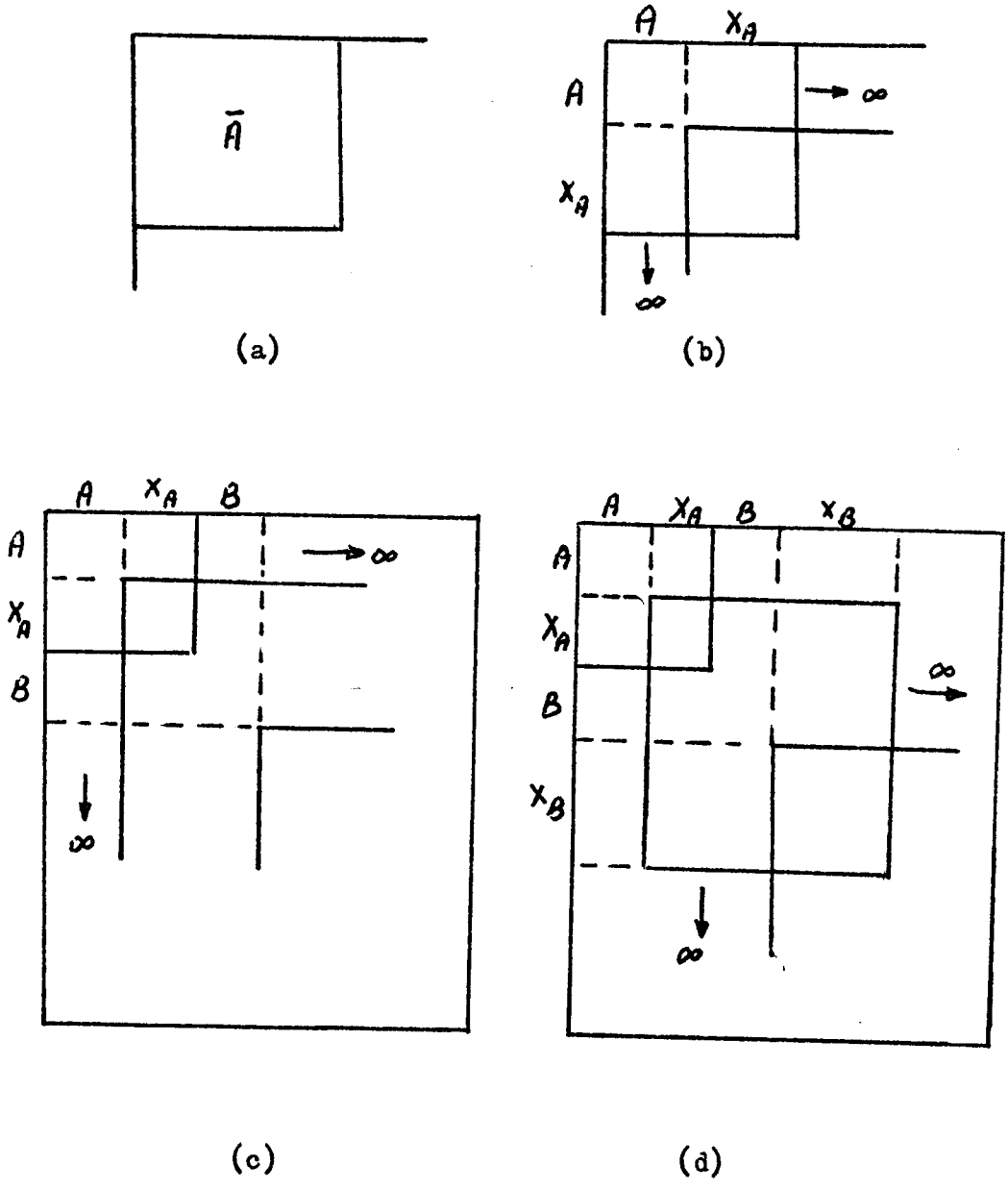
Notice that the optimal partitioning of the matrix D_n, i.e. the partition with the minimum number of ∞ elements inside the $\bar{A}$ and $\bar{B}$, etc., submatrices is achieved by trial and error and therefore we have many partitioning options.

Once the partitioning procedure is done with, the matrix D_n is then ready to be subject to the decomposition algorithm described in Section 2.2. of this chapter.

Notice that along with the decomposition calculations we update the matrix R_n given by (2.17) using the following equations

$$r_{ik} = \begin{cases} r_{jj} & \text{if } d_{ik} > d_{ij} + d_{jk} \\ \text{remains the same} & \text{if } d_{ik} \leq d_{ij} + d_{jk} \end{cases} \qquad (2.18)$$

where i, j, k represent the new numbering of rows and columns of D_n and R_n.



(a) (b) (c) (d)

Figure 2.7. Partitioning the Matrix D_n.

Finally after the decomposition calculations are over we get the matrices D_n^* and R_n^*. Then we can restore the original ordering of both matrices using P_{row} to get D^* and R^* by

$$D^* = P_{row}^T \; D_n^* \; P_{col}^T = P_{col} \; D_n^* \; P_{row}$$

and $$R^* = P_{row}^T \; R_n^* \; P_{col}^T = P_{col} \; R_n^* \; P_{row} \; .$$

There are some remarks to be made before we illustrate the procedure by an example:

(1) This method can be used to partition any given graph into its subgraphs and their minimum cut sets.

(2) The method will insure at all times that cut sets and minimum cut sets are always possible to find using the matrix D_n without any reference to the original graph associated with the distance matrix.

Example 2.2. Consider the graph shown in Figure 2.8. together with its distance matrix given in Table 2.3.

The nodes of the graph represent cities and the entries of the distance matrix represent the time in minutes required to travel from one city to the other on the highway connecting pairs of cities.

Following the steps of Part I of the method, we get P_{row} shown in Table 2.4.

Then we have

$$D_n = P_{row} \; D \; P_{row}^T$$

where D_n is given in Table 2.5.

Now referring to Part II of the method, we partition the matrix D_n which gives the partitioned matrix in Table 2.5.

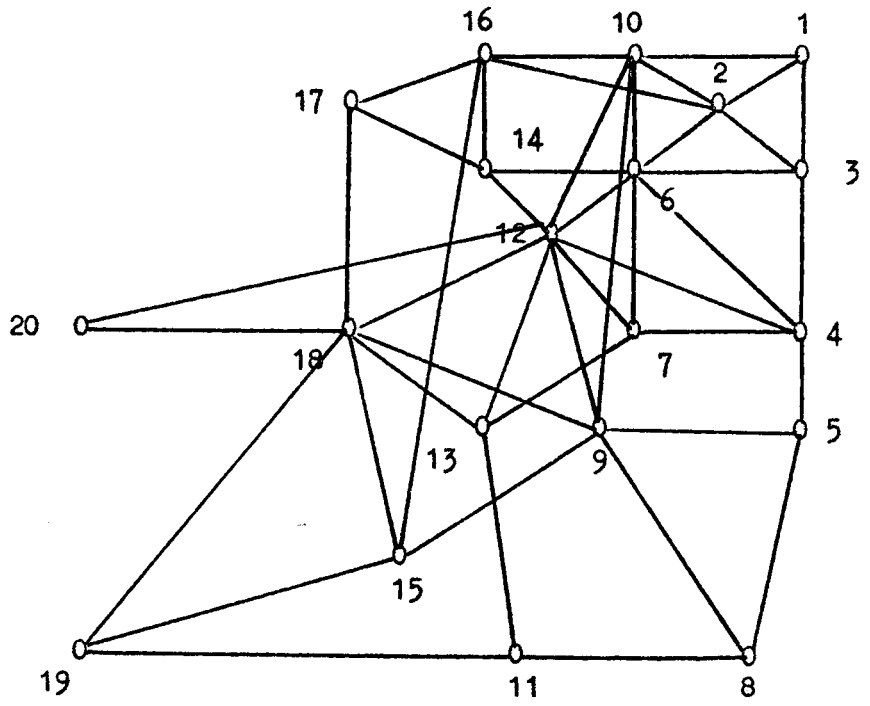


Figure 2.8. Graph for Example 2.2.

Now we apply the decomposition algorithm described in Section 2.2. of this chapter. From Table 2.5. we see that the graph is patitioned to three overlapping graphs, namely $\bar{A}$, $\bar{B}$, $\bar{C}$, where

$$\bar{A} = A \cup X_A$$

$$\bar{B} = X_A \cup B \cup X_B$$

$$\bar{C} = X_B \cup C$$

with X_A is the minimum cut set of A , X_A and X_B are the minimum cut set of B , and X_B is the minimum cut set of C.

Table 2.3. Matrix of Distances for Example 2.2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	220	225							460										
2	220	0	105			240				360						510				
3	225	105	0	130		165														
4			130	0	240	180	195		315			340								
5				240	0			300	310											
6		240	165	180		0	165			360		285		285						
7				195		165	0		225				255							
8					300			0	330		225									
9				315	310		225	330	0	505		255			220			360		
10	460	360				360			505	0		280				200				
11								225			0		330		210				270	
12				340		285			255	280		0	150	70				200		270
13							255				330	150	0		195			210		
14						285						70		0		255	195			
15									220		210		195		0	590		230	105	
16		510								200				255	590	0	110			
17														195		110	0	275		
18									360			200	210		230		275	0	195	170
19											270				105			195	0	
20												270						170		0

Table 2.4. The Permutation Matrix P_{row}

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
12	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
15	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
19	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Table 2.5. Permuted and Partitioned Distance Matrix D_n

	1	2	3	10	6	16	4	9	12	7	14	8	17	15	5	18	13	20	11	19	
1	0	220	225	460																	
2	220	0	105	360	240	510															A
3	225	105	0		165		130														
10	460	360		0	360	200		505	280												
6		240	165	360	0		180		285	165	285										X_A
16		510		200		0					255		110	590							
4			130		180		0	315	340	195					240						
9				505			315	0	255	225		330		220	310	360					B
12				280	285		340	255	0		70					200	150	270			
7					165		195	225		0							255				
14					285	255			70		0		195								
8								330				0			300				225		X_B
17						110					195		0			275					
15						590		220						0		230	195		210	105	
5							240	310				300			0						
18								360	200				275	230		0	210	170		195	
13									150	255				195		210	0		330		
20									270							170		0			
11												225		210			330		0	270	C
19														105		195			270	0	
	A			X_A				B						X_B					C		

The sets A , B , C , X_A , X_B are given by

$$A = (1,2,3) \qquad |A| = 3$$

$$X_A = (10,6,16,4) \qquad |X_A| = 4$$

$$B = (9) \qquad |B| = 1$$

$$X_B = (12,14,7,18,17,15,5,18) \qquad |X_B| = 8$$

$$C = (13,20,11,9) \qquad |C| = 4$$

Also from Table 2.5. we have $D_{\overline{AA}}$, $D_{\overline{BB}}$, $D_{\overline{CC}}$ given in Tables 2.6. , 2.7. and 2.8. respectively. Notice that all blank entries represent infinite entries.

Applying the decomposition algorithm we finally get

$$D^*_{\overline{AA}}(G) \quad , \quad D^*_{\overline{BB}}(G) \quad , \quad D^*_{\overline{CC}}(G)$$

shown in Tables 2.9. , 2.10. and 2.11.

Also we use the final tables to replace the portions of the partitioned matrix 2.5. where all entries were infinite, namely,

$$D_{AB} \ , \ D_{AX} \ , \ D_{AC} \ , \ D_{X_A C} \ , \ D_{BC} \ ,$$

$$D_{BA} \ , \ D_{XB^A} \ , \ D_{CA} \ , \ D_{CX_A} \ , \ D_{CB} \ .$$

Table 2.6. $D_{\overline{AA}}$

		A						
	0	220	225	460				
A	220	0	105	360	240	510		
	225	105	0		165		130	
	460	360		0	360	200		
		240	165	360	0		180	
X_A		510		220		0		
			130		180		0	

Table 2.7. $D_{\overline{BB}}$

		X_A			B				X_B			
	0	360	200		505	280						
	360	0		180		285	165	285				
X_A	220		0					255		110	590	
		180		0	315	340	195					240
B	505			315	0	255	225		330		220	310	360
	280	285		340	255	0		70					200
		165		195	225		0						
		285	255			70		0		195			
					330				0			300	
X_B			110					195		0			275
			590		220						0		230
				240	310				300			0	
				360	200					275	230		0

Table 2.8. $D_{\overline{CC}}$

		X_B						C				
		0		70				200	150	270		
			0						255			
		70		0		195						
					0		300				225	
X_B				195		0		275				
							0	230	195		210	105
					300			0				
		200				275	230	0	210	170		195
		150	255				195	210	0		330	
		270						170		0		
C					225		210		330		0	270
							105	195			270	0

Table 2.9. $D^{*}_{\overline{AA}}$ (G)

		A			X_A		
	0	220	225	460	390	660	355
A	220	0	105	360	240	510	235
	225	105	0	465	165	615	130
	460	360	465	0	360	200	540
	390	240	165	360	0	540	180
X_A	660	510	615	200	540	0	665
	355	235	130	540	180	665	0

Table 2.10. $D^{*}_{\overline{BB}}$ (G)

		X_A			B				X_B				
	0	360	200	540	505	280	525	350	835	310	625	780	480
	360	0	540	180	390	285	165	285	720	480	610	420	485
X_A	200	540	0	665	580	325	705	255	910	110	590	890	385
	540	180	665	0	315	340	195	410	540	605	535	240	540
B	505	390	580	315	0	255	225	325	330	520	220	312	360
	280	285	325	340	255	0	405	70	585	265	345	565	200
	525	165	705	195	225	405	0	450	555	645	445	435	465
	350	285	255	410	325	70	450	0	655	195	415	635	270
X_B	835	720	910	540	330	585	555	655	0	850	435	300	665
	310	480	110	605	520	265	645	195	850	0	505	830	275
	625	610	590	535	220	345	445	415	435	505	0	530	230
	780	420	890	240	310	565	435	635	300	830	530	0	670
	480	485	385	540	360	200	465	270	665	275	230	670	0

Table 2.11. $D^{*}_{\overline{CC}}$ (G)

	X_B								C			
	0	405	70	585	265	234	265	200	150	270	480	395
	405	0	450	555	570	445	435	465	255	635	585	550
	70	450	0	655	195	415	635	270	220	340	550	465
	585	555	655	0	850	435	300	665	555	835	225	495
X_B	265	570	195	850	0	505	830	275	415	445	715	470
	345	445	415	435	505	0	530	230	195	400	210	105
	565	435	635	300	830	530	0	670	690	835	525	635
	200	465	270	665	275	230	670	0	210	170	440	195
	150	255	220	555	415	195	690	210	0	380	330	300
	270	635	340	835	445	400	835	170	380	0	610	365
C	480	585	550	225	715	210	525	440	330	610	0	270
	395	550	465	495	470	105	635	195	300	365	270	0

We get at the end of these computations

$$D^*_{AB}(G) \quad , \quad D^*_{AX_B}(G) \quad , \quad D^*_{AC}(G) \quad , \quad D^*_{X_{AC}}(G) \quad , \quad D^*_{BC}(G) \quad , \text{ etc.}$$

Therefore we have the matrix of shortest distances $D^*_n(G)$ given in Table 2.12.

To restore the original ordering of the matrix rows and columns, we perform the matrix multiplication

$$D^*(G) \quad = \quad P^T_{row} \quad D^*_n(G) \quad P_{row} \quad ,$$

which is given in Table 2.13. Also the route matrix R^* necessary to detect any shortest path in the graph is given in Table 2.14.

Table 2.12. D_n^* (G)

	A			X_A				B	X_B								C			
	0	220	225	460	390	660	355	670	675	550	675	895	770	890	595	875	805	945	1100	995
A	220	0	105	360	240	510	235	550	525	405	525	775	620	770	475	725	660	795	980	875
	225	105	0	465	165	615	130	445	450	325	450	670	645	665	370	650	580	720	875	770
	460	360	465	0	360	200	540	505	280	525	350	835	310	625	780	480	430	550	760	675
X_A	390	240	165	360	0	540	180	390	285	165	285	720	480	610	420	485	420	555	750	680
	660	510	615	200	540	0	665	580	325	705	255	910	110	590	890	385	475	555	800	580
	355	235	130	540	180	665	0	315	340	195	410	540	605	535	240	540	450	610	745	640
B	670	550	445	505	390	580	315	0	255	225	325	330	520	220	310	360	405	525	430	325
	675	525	450	280	285	325	340	255	0	405	70	585	265	345	565	200	150	270	480	395
	550	405	325	525	165	705	195	225	405	0	450	555	645	445	435	465	255	635	585	550
	675	525	450	350	285	255	410	235	70	450	0	655	195	415	635	270	220	340	550	465
X_B	895	775	670	835	720	910	540	330	585	555	655	0	850	435	300	665	555	835	225	495
	770	620	645	310	480	110	605	520	265	645	195	850	0	505	830	275	415	445	715	470
	890	770	665	625	610	590	535	220	345	445	415	435	505	0	530	230	195	400	210	105
	595	475	370	780	420	890	240	310	565	435	635	300	830	530	0	670	690	835	525	635
	875	725	650	480	485	385	540	360	200	465	270	665	275	230	670	0	210	170	440	195
	805	660	580	430	420	475	450	405	150	255	220	555	415	195	690	210	0	380	330	300
C	945	795	720	550	555	555	610	525	270	635	340	835	445	400	835	170	380	0	610	365
	1100	980	875	760	750	800	745	430	480	585	550	225	715	210	525	440	330	610	0	270
	995	875	770	675	680	580	640	325	395	550	465	495	470	105	635	195	300	365	270	0

Table 2.13. D^* (G)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	220	225	355	595	390	550	895	670	460	1100	675	805	675	890	660	770	875	995	945
2	220	0	105	235	475	240	405	775	550	360	980	525	660	525	770	510	620	725	875	795
3	225	105	0	130	370	165	325	670	445	465	875	450	580	450	665	615	645	650	770	720
4	355	235	130	0	240	180	195	540	315	540	745	340	450	410	535	665	605	540	640	610
5	595	475	370	240	0	420	435	300	310	780	525	565	690	635	530	890	830	670	635	835
6	390	240	165	180	420	0	165	720	390	360	750	285	420	285	610	540	480	485	680	555
7	550	405	325	195	435	165	0	555	225	525	585	405	255	450	445	705	645	465	550	635
8	895	775	670	540	300	720	555	0	330	835	225	585	555	655	435	910	850	665	495	835
9	670	550	445	315	310	390	225	330	0	505	430	255	405	325	220	580	520	360	325	525
10	460	360	465	540	780	360	525	835	505	0	760	280	430	350	625	200	310	480	675	550
11	1100	980	875	745	525	750	585	225	430	760	0	480	330	550	210	800	715	440	270	610
12	675	525	450	340	565	285	405	585	255	280	480	0	150	70	345	325	265	200	395	270
13	805	660	580	450	690	420	255	555	405	430	330	150	0	220	195	475	415	210	300	380
14	675	525	450	410	635	285	450	655	325	350	550	70	220	0	415	255	195	270	465	340
15	890	770	665	535	530	615	445	435	220	625	210	345	195	415	0	590	505	230	105	400
16	660	510	615	665	890	540	705	910	580	200	800	325	475	255	590	0	110	385	580	555
17	770	620	645	605	830	480	645	850	520	310	715	265	415	195	505	110	0	275	470	445
18	875	725	650	540	670	485	465	665	360	480	440	200	210	270	230	385	275	0	195	170
19	995	875	770	640	635	680	550	495	325	675	270	395	300	465	105	580	470	195	0	365
20	945	795	720	610	835	555	635	835	525	550	610	270	380	340	400	555	445	170	365	0

Table 2.14. R^* (G)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	2	3	3	4	3	4	5	4	10	15	6	7	6	9	10	16	12	15	12
2	1	2	3	3	4	6	6	5	4	10	15	6	7	6	9	16	16	12	15	12
3	1	2	3	4	4	6	4	5	4	2	15	6	7	6	9	2	14	12	15	12
4	3	3	3	4	5	6	7	5	9	6	15	12	7	12	9	14	14	12	15	12
5	4	4	4	4	5	4	4	8	9	6	8	9	7	12	9	14	14	9	15	12
6	3	2	3	4	4	6	7	5	7	10	13	12	7	14	9	14	14	12	18	12
7	4	6	4	4	4	6	7	9	9	6	13	13	13	6	9	14	14	13	15	18
8	5	5	5	5	5	5	9	8	9	9	11	9	11	12	11	14	14	15	11	18
9	4	4	4	4	5	7	7	8	9	10	15	12	12	12	15	14	14	18	15	12
10	1	2	2	6	6	6	6	9	9	10	13	12	12	12	13	16	16	12	18	12
11	15	15	15	15	8	13	13	8	15	13	11	13	13	13	15	15	18	16	19	18
12	6	6	6	4	9	6	13	9	9	10	13	12	13	14	13	14	14	18	18	20
13	7	7	7	7	7	7	7	11	12	12	11	12	13	12	15	14	14	18	15	18
14	6	6	6	12	12	6	6	12	12	12	13	12	12	14	13	16	17	12	18	12
15	9	9	9	9	9	9	9	11	9	13	11	13	13	13	15	16	18	18	19	18
16	10	2	2	14	14	14	14	14	14	10	15	14	14	14	15	16	17	17	18	18
17	16	16	14	14	14	14	14	14	14	16	18	14	14	14	18	16	17	18	18	18
18	12	12	12	12	9	12	13	15	9	12	15	12	13	12	15	17	17	18	19	20
19	15	15	15	15	15	18	15	11	15	18	11	18	15	18	15	18	18	18	19	18
20	12	12	12	12	12	12	18	18	12	12	18	12	18	12	18	18	18	18	18	20

2.5. Discussion

This section is devoted to discuss and make comments on the paper by T.C.Hu. [7]

The partitioning method for m overlapping graphs described on pages 46 and 47 of this thesis is different from the method presented by Hu in his book.

He suggested the following steps for partitioning the graph into its m overlapping subgraphs:

(1) Label any subset of nodes as A.

(2) Then its minimum cut set is X_A.

(3) Let B be a cut set (not necessarily a minimum cut set) $A \cup X_A$ and X_B be the minimum cut set of $A \cup X_A \cup B$. (Notice that the minimum cut set of B is $X_A \cup X_B$.)

(4) Let C be a cut set of $A \cup X_A \cup B \cup X_B$ and X_C be the minimum cut set of $A \cup X_A \cup B \cup X_B \cup C$.

(5) Continue until no further partitioning is necessary.

The difference between the method described on pages 46-47 and the above mentioned method is that in Steps (3) and (4) the sets B,C and all similar sets do not have to be cut sets, they only have to be sets connected to the sets found in previous steps,i.e. B needs only be a set connected to $A \cup X_A$, C needs only be connected to $A \cup X_A \cup B X_B$ and so forth. This agrees with the picture shown in Table 2.2. on page 47, which is the same picture given in Hu's paper presented in his book (Chapter 10,pp.165) and which is less restricting than his suggested partitioned method.

The following is a justification of this claim:

From the definition of a cut set, according to Hu's method the removal of B,C,etc.,should disconnect $A \cup X_A$,$A \cup X_A \cup B \cup X_B$, etc., respectively, from the rest of the graph. This means that the removal of the rows and columns of the partitioned matrix corresponding to these sets, which is equivalent to the removal of their corresponding nodes from the graph, should leave the partitioned matrix with only infinite entries at places representing arc lengths connecting $A \cup X_A$, $A \cup X_A \; B \cup X_B$, etc., to $(G - A \cup X_A \cup B)$, $(G - A \cup X_A \cup B \cup X_B \cup C)$,etc., respectively.

But referring to Table 2.2., we can see that the removal of the columns and rows corresponding to the sets B,C,etc., will not disconnect the graph. This is because when we remove $D_{\alpha B}$, $D_{B\alpha}$, $D_{\alpha C}$, $D_{C\alpha}$, etc., this will leave $D_{X_A X_B}$, $D_{X_B X_C}$, etc., in the partitioned matrix which are non-infinite entries all of them (α stands for A , X_A , B , X_B , C , X_C , etc.)

This means that $\bar{A}$, $\bar{B}$, etc., will still be connected to the rest of the graph, even after the removal of the sets B , C , etc., respectively and therefore showing that B,C,etc., are not cut sets of $\bar{A}$, $\bar{B}$, etc., respectively.

The steps presented on page 46 are a remedy for this situation, and, they are justified as follows:

In Theorem 2.1. and 2.2., we only needed that X_A , X_B , etc. be minimum cut sets of $A \cup X_A$, $A \cup X_A \cup B$, etc., respectively, and therefore B,C,etc.,need not be cut sets.

The sets B,C,etc., need only be connected sets to $A \cup X_A$, $A \cup X_A \cup B \cup X_B$, etc., respectively, so that when we apply Theorem 2.1. we take $B \cup X_B$, $C \cup X_C$, etc., to be the set B in Theorem 2.1.

CHAPTER III

OPTIMIZATION IN NETWORK FLOWS

3.1. Network Flows

In this chapter we discuss techniques to solve some network flow problems.

The following are examples of problems that can be regarded as flows in networks:

(1) The shipment of finished goods from a manufacturer to a distributor.

(2) The movement of people from their homes to places of employment.

(3) The delivery of letters from their point of posting to their destination.

(4) A system of chemical processing units in which some product is undergoing change. The nodes correspond to the various units such as heat exchange, reactors and so forth. The arcs correspond to pipelines between units.

(5) A computer network in which the nodes correspond to computer systems or terminals and the telephone lines or microwaves transmission lines correspond to the arcs of the network.

(6) A transportation system in which the nodes are cities and the arcs are roads which material is transported.

There are many real world situations similar to the above which can be profitably regarded as networks. For example, one might wish to maximize the amount transported from one place to another by the delivery system, or determine the least costs way to send a given number of objects from one place to another via the system, or determine the quickest way to deliver a shipment through the system.

If the original flow problem can be modeled in terms of a network flow problem with a high degree of accuracy, then the results and algorithms for network flows can be applied to the original problem.

To be more specific, we deal with two problems:

(1) Finding maximal flows in networks.

(2) Finding minimal cost flows in networks.

Before we discuss the above two problems, we first introduce some new definitions.

Defintition 3.1. A flow in a network is a way of sending objects from one node to another by travelling along the arcs in their directions.

The objects that flow from the source to the sink in the network are called flow units or units.

Definition 3.2. A cut $(X,\bar{X})$, where X is a subset of nodes of the network and $\bar{X}$ is its complement, is a set of all arcs A_{ij} with either $N_i \in X$ and $N_j \in \bar{X}$ or $N_j \in X$ and $N_i \in \bar{X}$. A cut set is therefore a set of arcs the removal of which will disconnect the network.

A cut separating the source N_s and the sink N_t is a cut $(X,\bar{X})$ with $N_s \in X$ and $N_t \in \bar{X}$.

Definition 3.3. The upper limit on the folw in an arc A_{ij} is the capacity of the arc and is denoted by C_{ij}. The flow capacity of an arc is a positive number and may be infinite, in which case the arc is said to be uncapacitated.

If an arc has a flow capacity of zero, removing the arc from the network does not alter the potential flow in the network.

Definition 3.4. The capacity of a cut $(X,\bar{X})$, denoted by $C(X,\bar{X})$ is $\sum_{i,j} C_{ij}$ with $N_i \in X$ and $N_j \in \bar{X}$.

Note that in defining a cut, we count all the arcs that are between the sets X and $\bar{X}$, but in calculating its capacity, we count only the capacity of the directed arcs from X to $\bar{X}$, but not the directed arcs from $\bar{X}$ to X. Therefore, $C(X,\bar{X})$ is not equal to $C(\bar{X},X)$ in general.

Definition 3.5. A cut separating N_s and N_t with minimum capacity is called a minimum cut.

Definition 3.6. A flow augmenting path is a path along which the flow in the arcs can be increased, so that the flow does not exceed the minimum of all capacity of arcs in the path.

3.2. Maximal Flow Problem

The maximal flow problem is to find a feasible equilibrium or steady state flow pattern through a network such that the total flow from the source to the sink is maximized. Should there exist more

than a single source (sink) node, then a super source (super sink) node is connected to each of the sources (sinks). The arcs making the connection are uncapacitated.

Let us consider an oriented network with a single source, a single sink and a set of intermediate connected nodes. For each node, with the exception of the source and sink, we shall assume that conservation of flow is maintained, i.e., flow into a given node equal flow out of that node. Suppose the network has n node, where node N_1 is the source and node N_n is the sink. We will designate the flow along arc A_{ij} by x_{ij}.

Mathematical Formulation of the Problem

It is clear that no flow in a given arc may exceed the capacity of that arc. Hence we require constraints of the form:

$$0 \leq x_{ij} \leq C_{ij} \qquad i,j,= 1,2,\ldots,n \quad (3.1)$$

In addition, as noted above there is conservation of flow at each node, i.e. the flow into each node, except the source and sink, equals the flow out of that node. Hence we have:

$$\sum_i x_{ki} = \sum_j x_{jk} \qquad k = 2,3,\ldots,n-1 \quad (3.2)$$

Also we have for the source node N_1

$$\sum_i x_{1i} = v \qquad (3.3)$$

and for the sink node N_n

$$\sum_j x_{jn} = v \qquad (3.4)$$

Our objective is to maximize flow. Hence, the objective function may be stated as maximizing the sum of flows originating at the source or flowing into the sink.

To summarize the maximal flow problem is given by:

$$\text{Maximize} \quad v \tag{3.5}$$

subject to

$$\sum_i x_{ki} - \sum_j x_{jk} = \begin{cases} v & k=1 \\ 0 & k\neq 1 \ , \ k\neq n \\ -v & k=n \end{cases} \tag{3.6}$$

$$0 \leq x_{ij} \leq C_{ij} \qquad i,j = 1,2,\ldots,n.$$

It is easily seen that this problem is a linear programming problem. However, for a network with many nodes and arcs, the problem can be quite large. Instead of this, we shall present a <u>labeling</u> method devised by Ford and Fulkerson [5] which is quite efficient for solving our problem.

Before we present this method we first make a final notice: If the flows in the arcs of the network have an upper bound C_{ij}, as well as a lower bound l_{ij} such that the flow in an arc x_{ij} is governed by the inequality

$$l_{ij} \leq x_{ij} \leq C_{ij} \qquad i,j=1,2,\ldots,n,$$

then by a simple change of variable, the problem can be reduced to the problem stated in equations (3.5) and (3.6) namely,

$$y_{ij} = x_{ij} - l_{ij} \qquad i,j=1,2,\ldots,n$$

Therefore we have

$$0 \leq y_{ij} \leq u_{ij} \qquad i,j=1,2,\ldots n$$

where $u_{ij} = C_{ij} - l_{ij}$ $\qquad i,j=1,2,\ldots,n.$

Let us state without proof some theorems which form the foundations for network. (For a detailed proof of these theorems refer to Ford and Fulkerson [5] .)

Lemma 3.1. Let f be a flow from the source N_s to the sink N_t in a network, and let f have value v . If $(X,\bar{X})$ is a cut set separating N_s and N_t then

$$v = f(X,\bar{X}) - f(\bar{X},X) \leq C(X,\bar{X})$$

Theorem 3.2. (Max Flow - Min Cut Theorem). For any network the maximal flow from N_s to N_t is equal to the minimal cut capacity of all cuts separating N_s and N_t.

Corollary 3.3. A flow f is maximal if and only if there is no flow augmenting path with respect to f.

Corollary 3.4. A cut $(X,\bar{X})$ is minimal if and only if every maximal flow f saturates all arcs of $(X,\bar{X})$ whereas all arcs of $(\bar{X},X)$ are flowless with respect to f.

Corollary 3.5. Let $(X,\bar{X})$ and $(Y,\bar{Y})$ be minimal cuts. Then $(X \cup Y , \overline{X \cup Y})$ and $(X \cap Y , \overline{X \cap Y})$ are also minimal cuts.

The Labeling Method

In order to make our description of this method perfectly general, let us assume that each arc A_{ij} has associated with it capacities C_{ij} and C_{ji}. In the event that flows are allowed in one direction only, one of these capacities would be zero, e.g. $C_{ij} \geq 0$, $C_{ji} = 0$.

Definition 3.6. The excess capacity, denoted by e_{ij}, is defined by

$$e_{ij} = C_{ij} - x_{ij} + x_{ji}$$
$$e_{ji} = C_{ji} - x_{ji} + x_{ij} \qquad (3.7)$$

where x_{ij} and x_{ji} is any set of flows in A_{ij}.

We now describe the labeling method for solving maximal flow problems in the following steps:

Step (0) We initially set all flows to zero. We also calculate for each arc its excess capacities e_{ij} and e_{ji} given by equation (3.7).

Step (1) We begin at the source node, N_1, and consider all nodes which are connected to the source by arcs of positive excess capacities. We designate the nodes which are connected to the source this way with index p. Call this set of nodes P.

We then label each node $N_p \in P$ on our network with two labels (α_p, β_p). These labels are defined as follows:

α_p = excess capacity from source to node p,

β_p = the node index from which we came (source).

Therefore we have

$$(\alpha_p , \beta_p) = (e_{1p} , 1), N_p \epsilon P \qquad (3.8)$$

If we had labeled the sink in the first step, which is theoretically possible but highly unlikely in any practical situation, we move to the final part of this method.

Step (2) Generally, the sink is not labeled in the first step. Hence, from the set of nodes $N_p \epsilon P$, we choose $p = p_1$. We then seek out nodes, not yet labeled, which are joined to node $N_{p1} \epsilon P$, by arcs of positive excess capacity. If there are none we go to $N_{p_2} \epsilon P$ and repeat. If we can reach any unlabeled nodes in this fashion, let us designate this set of nodes with index q and this set of nodes will be called Q. We now proceed to label the nodes $N_q \epsilon Q$ as follows:

$$\alpha_q = \min (e_{pq} , \alpha_p) \qquad (3.9)$$

$$\beta_q = p$$

The label α_q represents the minimum excess capacity of the two arcs A_{1p} and A_{pq}. The label β_q tells us the node from which we came to reach nodes N_q. We carry out this

labeling process for all nodes $N_p \in P$. Again, if we have labeled the sink during this step, we proceed to the latter part of this method.

Step (3) (General Step) This is basically the same as the previous step. If we have a set of labeled nodes $N_r \in R$, we look for nodes not yet labeled which are connected to nodes N_r by arcs which have positive excess capacities. We do this for each $N_r \in R$. If we find a set of unlabeled nodes N_t T, we label these nodes as follows:

$$\alpha_t = \min (e_{rt} , \alpha_r) \qquad (3.10)$$

$$\beta_t = r$$

We repeat this general step. Since the network has a finite number of nodes and arcs we must reach one of the two following conditions in a finite number of steps:

(1) The sink is labeled, or

(2) We cannot label the sink and no other nodes can be labeled.

If we reach condition (1), we can increase the flow. If we reach condition (2), the existing flow is the maximal flow and the problem is solved.

Step (4) (Flow Change) Consider now the case where the sink has been labeled. The label on the sink, α_n, indicates how much positive excess capacity exists from source to sink over the particular path traversed, and therefore how much the

flow can be increased. Furthermore, it is a simple matter to traverse this path, since the second label of the nodes, β_j , indicates the preceding node leading to N_j.

Let us designate by e_{uv} the excess capacities of the arcs in this particular path from source to sink, which enabled us to label the sink. We can increase the flow by α_n and we do this by recalculating the excess capacities for each arc. These new excess capacities are given by:

$$\bar{e}_{uv} = e_{uv} - \alpha_n$$
$$\bar{e}_{vu} = e_{vu} + \alpha_n \qquad (3.11)$$

and $\bar{e}_{ij} = e_{ij}$ for arcs not in the sink labeling path.

Step (5) Repeat the entire labeling process for the network with the new excess capacities starting again at the source N_1 . It is clear that since we are dealing with finite maximal flows in a finite network, we must reach condition (2) of the sink in a finite number of steps.

At each stage we can calculate the net flows in each arc using the definition of excess capacities given in equation (3.7). The net flow in arc A_{ij} is

$$x_{ij} = C_{ij} - e_{ij} \quad \text{if} \quad e_{ji} = 0.$$

If both $C_{ij} \neq 0$, $C_{ji} \neq 0$ then either:

$$x_{ij} = C_{ij} - e_{ij} \quad \text{and} \quad x_{ji} = 0$$

$$x_{ji} = C_{ji} - e_{ji} \quad \text{and} \quad x_{ij} = 0 ,$$

depending upon which of them is positive.

The Labeling Method : A Proof

We prove in the next few lines that the above method, when it terminates under the conditions stated, does indeed yield the maximal flow.

It is obvious that at each stage the constraints are satisfied since the capacity restrictions are satisfied by the definition of excess capacity and calculation of flows. Further, we are obviously satisfying the conservation of flow at each node. Hence, what we need to show is that the final flow is the maximal flow.

Now, let us consider the situation when we terminate our calculation: we see that two disjoint sets of nodes (nodes we have been able to label and those we have not) exist. Let us designate the nodes we have been able to reach in the final labeling process by S_1 , and those we have not been able to label by S_u . If we consider the set of oriented arcs that connect members of S_1 to members of S_u , denoted by J_c , is clearly a cut of the network because unless each path from N_1 to N_n contains at least one of these arcs, we cannot reach the sink.

Combining equations (3.6) and (3.7), we see that for all nodes, except N_1 and N_n , we can write

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} (C_{ij} - e_{ij}) = 0 \qquad i = 2,\ldots,n-1 \qquad (3.12)$$

For the source N_1 we have

$$\sum_{\substack{j=1 \\ j\neq 1}}^{n} (C_{ij} - e_{ij}) = \sum_{j=2}^{n} x_{1j} = \text{total flow} \qquad (3.13)$$

If we now sum the left hand side of equation (3.12) over labeled nodes, i.e. over nodes $N_i \in S_1$, and recall that the source is labeled, we see from equations (3.12) and (3.13) that

$$\sum_{i\in S_1} \sum_{\substack{j=1 \\ j\neq i}}^{n} (C_{ij} - e_{ij}) = \sum_{j=2}^{n} x_{1j} = \text{total flow} \qquad (3.14)$$

Notice that if both $j \; S_1$ and $i \; S_1$, then we have terms for both $C_{ij} - e_{ij}$ and $C_{ji} - e_{ji}$ present in the summation and they cancel each other. Hence, for $j \in S_1$ all terms cancel, and only terms for $j \in S_u$ are present. However, if $i \in S_1$ and $j \in S_u$, and since S_1 and S_u are disjoint sets, we know that $e_{ij} = 0$. Hence, we have from (3.14)

$$\sum_{i\in S_1} \sum_{j\in S_u} C_{ij} = \text{total flow} \qquad (3.15)$$

From the definition of a cut, we see that a set of arcs such as that given by equation (3.15) is clearly a cut. However, equation (3.15) says that this particular cut equals the total flow. We have already noted that the total flow cannot exceed the sum of the capacities in any cut. However, in equation (3.15) we have found a cut

in which the total flow equals the capacities of the arcs in the cut. Hence, for this particular cut which is minimal, the flow must be maximal.

Q.E.D.

Computational Algorithm

Here we show how to organize the maximal flow algorithm into a compact matrix format. For a network with n nodes we merely require an n^{th} order matrix of the excess capacities e_{ij}. We also require two additional columns for the labels. We will also note the order of labeling in a third additional column. The matrix format is more convenient than performing the calculations on the network diagram. The maximal flow matrix format is shown in Table 3.1.

Table 3.1. Matrix Format for the Maximal Flow Problem.

Nodes	1	2 ..... j ..... n	Labels	Step
1	-	e_{12} ..... e_{1j} ..... e_{1n}	$\underline{\alpha}$ $\underline{\beta}$	
2	e_{21}	- ..... e_{2j} ..... e_{2n}	α_2 β_2	
.				
.				
.				
i	e_{i1}	e_{i2} ..... e_{ij} ..... e_{in}	α_i β_i	
.				
.				
.				
n	e_{n1}	e_{n2} ..... e_{nj} ..... -	α_n β_n	

In order to use Table 3.1. we begin with the matrix $E = (e_{ij})$ of excess capacities. Initially, $e_{ij} = C_{ij}$, we start in row 1 and look for columns with positive excess capacities, i.e. $e_{1j} > 0$. For all such columns we set $\alpha_j = e_{1j}$ and $\beta_j = 1$. Assuming that we have not labeled the sink, i.e. a value has not been given to α_n, we look at the first of the rows i that have been labeled at the previous step. We keep track of these in the final column marked "Step". In row i we look for $e_{ij} > 0$ for which row j has not been labeled. For all such rows, we set $\alpha_j = \min (e_{ij}, \alpha_i)$, $\beta_i = i$.

We continue this for each of the rows labeled at the first step as long as α_n is not computed. If all rows are labeled and we still have not labeled the sink, we now examine the rows labeled in the second step. We repeat the entire procedure. Since the algorithm is finite, we must either label the sink or find that we cannot reach the sink. If α_n is not labeled, the flow is a maximal flow. Otherwise, we know we can increase the flow.

Suppose we have some value α_n and $\beta_n = r$. We need to construct a new matrix. In order to do so, we first calculate the e_{uv} in the path from the source to the sink. Since $\beta_n = r$, we know that the final row was reached from row r. Therefore, we need to calculate $\bar{e}_{rn}$ given by

$$\bar{e}_{rn} = e_{rn} - \alpha_n \qquad (3.16)$$

Now suppose that for row r, $\beta_r = S$. Hence, we next calculate e_{sr} in row s as

$$\bar{e}_{sr} = e_{sr} - \alpha_n \tag{3.17}$$

If $\beta_s = t$, then we next change e_{ts}, etc. Next we recalculate e_{vu}, i.e. for each e_{uv} that was changed by subtracting α_n, we recalculate the corresponding e_{vu} by adding α_n

$$\begin{aligned} \bar{e}_{nr} &= e_{nr} + \alpha_n \\ \bar{e}_{rs} &= e_{rs} + \alpha_n \qquad \text{etc.} \end{aligned} \tag{3.18}$$

All other e_{ij} are unchanged in the matrix. Once the new matrix is constructed we repeat the entire process.

When we reach a matrix in which α_n cannot be given a value we have found the maximal flow.

In order to find the total flow for any matrix we need only add all the entries in column 1 or all the entries in row n. Therefore

$$\text{Total Flow} = \sum_{i=1}^{n} e_{i1} = \sum_{j=1}^{n} e_{nj} \tag{3.19}$$

We illustrate the algorithm by an example.

Example 3.2.1. Consider the network shown in Figure 3.1. The entries shown on the oriented arcs are the capacities C_{ij} in the direction of the arrows.

The initial matrix of capacities is shown in Table 3.2. Initially all $e_{ij} = C_{ij}$.

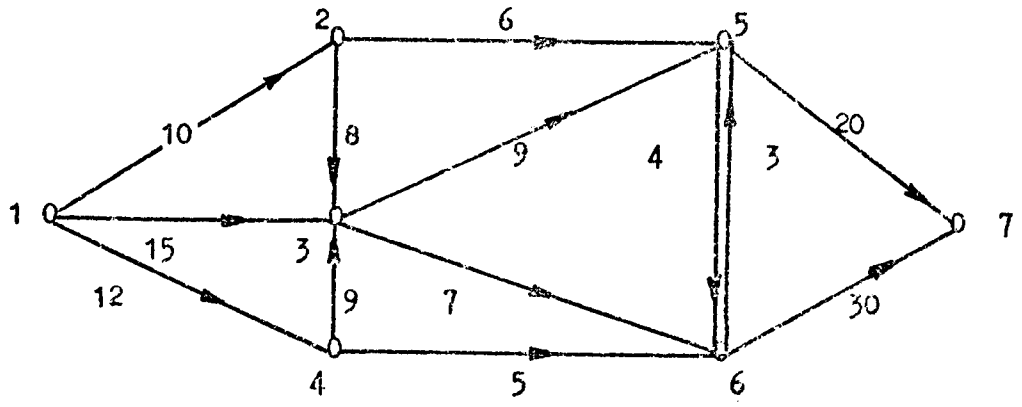


Figure 3.1. Network for Example 3.2.1.

In Step (1) we can label α_2 , α_3 , α_4 with 10,15,12 respectively. Then we examine row 2 we can see that we can label α_5

$$\alpha_5 = \min (e_{25} = 6 , \alpha_2 = 10) = 6$$
$$\beta_5 = 2$$

Table 3.2. Initial Labeling Matrix for Example 3.2.1.
Flow = 0 , $\alpha_7 = 6$.

Nodes	1	2	3	4	5	6	7	Labels		Step
1	-	10	15	12	0	0	0	α	β	
2	0	-	8	0	6	0	0	10	1	1
3	0	0	-	0	9	7	0	15	1	1
4	0	0	0	-	0	5	0	12	1	1
5	0	0	0	0	-	4	20	6	2	2
6	0	0	0	0	3	-	30	7	3	2
7	0	0	0	0	0	0	-	6	5	3

and no other nodes can be labeled from row 2.

Then we move to row 3 , we see that we can label 6

$$\alpha_6 = \min (e_{36} = 7 , \alpha_3 = 15) = 7$$

$$\beta_6 = 3$$

and no other nodes can be labeled from row 3.

Then we move to row 4, we find that no new nodes can be labeled from row 4.

Therefore we move to row 5 we see that we can label the sink N_7 from row 5

$$\alpha_7 = \min(e_{57} = 20 , \alpha_5 = 6) = 6$$

$$\beta_7 = 5.$$

We have labeled the sink and $\alpha_7 = 6$. Hence, we can increase the flow by 6 units. Since $\beta_7 = 5$, we know that e_{57} will be decreased by $\alpha_7 = 6$. Hence,

$$\bar{e}_{57} = e_{57} - \alpha_7 = 20 - 6 = 14$$

and $$\bar{e}_{75} = e_{75} + \alpha_7 = 0 + 6 = 6.$$

Since $\beta_5 = 2$ therefore

$$\bar{e}_{25} = e_{25} - \alpha_7 = 6 - 6 = 0$$

and $$\bar{e}_{52} = e_{52} + \alpha_7 = 0 + 6 = 6.$$

Since $\beta_2 = 1$ therefore

$$\bar{e}_{12} = e_{12} - \alpha_7 = 10 - 6 = 4$$

and $$\bar{e}_{21} = e_{21} + \alpha_7 = 0 + 6 = 6.$$

All other values of excess capacities e_{ij} are unchanged. The new table is hsown in Table 3.3. We now repeat the labeling in Table 3.3., until we label the sink N_7 . The results of labeling stages are shown in Tables 3.4. to 3.7.

Table 3.3. Flow = 6 , $\alpha_7 = 9$.

Nodes	1	2	3	4	5	6	7	Labels		Step
1	-	4	15	12	0	0	0	$\underline{\alpha}$	β	
2	6	-	8	0	0	0	0	4	1	1
3	0	0	-	0	9	7	0	15	1	1
4	0	0	9	-	0	5	0	12	1	1
5	0	6	0	0	-	4	14	9	3	2
6	0	0	0	0	3	-	30	7	3	2
7	0	0	0	0	6	0	-	9	5	3

Table 3.4. Flow = 15 , $\alpha_7 = 6$.

Nodes	1	2	3	4	5	6	7	Labels		Step
1	-	4	6	12	0	0	0	$\underline{\alpha}$	β	
2	6	-	8	0	0	0	0	4	1	1
3	9	0	-	0	0	7	0	6	1	1
4	0	0	9	-	0	5	0	12	1	1
5	0	2	9	0	-	4	5	3	6	3
6	0	0	0	0	3	-	30	6	3	2
7	0	0	0	0	15	0	-	6	6	3

Table 3.5. Flow = 21 , $\alpha_7 = 5$.

Nodes	1	2	3	4	5	6	7	Labels		Step
1	-	4	0	12	0	0	0	$\underline{\alpha}$	$\underline{\beta}$	
2	6	-	8	0	0	0	0	4	1	1
3	15	0	-	0	0	1	0	9	4	2
4	0	0	9	-	0	5	0	12	1	1
5	0	2	9	0	-	4	5	3	6	3
6	0	0	6	0	3	-	24	5	4	2
7	0	0	0	0	15	6	-	5	6	6

Table 3.6. Flow = 26 , $\alpha_7 = 1$.

Nodes	1	2	3	4	5	6	7	Labels		Step
1	-	4	0	7	0	0	0	$\underline{\alpha}$	$\underline{\beta}$	
2	6	-	8	0	0	0	0	4	1	1
3	15	0	-	0	0	1	0	4	2	2
4	5	0	9	-	0	0	0	7	1	1
5	0	2	9	0	-	4	5	1	6	4
6	0	0	6	5	3	-	19	1	3	3
7	0	0	0	0	15	11	-	1	6	4

In Table 3.7. we cannot label the sink and therefore the maximal flow has been attained. The total flow is

$$\text{Maximal Flow} = \sum_{i=1}^{n} e_{i1} = \sum_{j=1}^{n} e_{1j}$$

$$\text{Maximal Flow} = 27.$$

Table 3.7. Final Labeling Matrix for Example 3.2.1.
Flow = 27 , $\alpha_7 = 0$.

Nodes	1	2	3	4	5	6	7	Labels		Step
1	-	3	0	7	0	0	0	$\underline{\alpha}$	$\underline{\beta}$	
2	7	-	7	0	0	0	0	3	1	1
3	15	1	-	0	0	0	0	3	2	2
4	4	0	9	-	0	0	0	7	1	1
5	5	0	6	9	0	4	5			
6	0	0	7	5	3	-	18			
7	0	0	0	0	15	12	-			

Also from Table 3.7. we can read the excess capacities in each arc A_{ij} , in other words the flow units that did not make it to the sink. Also we can calculate the flow in each arc using these excess capacities as follows:

$$x_{ij} = C_{ij} - e_{ij} \qquad i,j = 1,2,\ldots,n \quad i \neq j$$

$$e_{12} = 3 \qquad x_{12} = 7$$
$$e_{13} = 0 \qquad x_{13} = 15$$
$$e_{14} = 7 \qquad x_{14} = 5$$
$$e_{23} = 7 \qquad x_{23} = 1$$
$$e_{25} = 0 \qquad x_{25} = 6$$
$$e_{35} = 0 \qquad x_{35} = 9$$
$$e_{36} = 0 \qquad x_{36} = 7$$
$$e_{43} = 9 \qquad x_{43} = 0$$

$e_{46} = 0$ $\quad x_{46} = 5$

$e_{56} = 4$ $\quad x_{56} = 0$

$e_{57} = 5$ $\quad x_{57} = 15$

$e_{65} = 3$ $\quad x_{65} = 0$

$e_{67} = 18$ $\quad x_{67} = 12$

The optimal flow is shown in Figure 3.2. where we have deleted the arcs with zero flow.

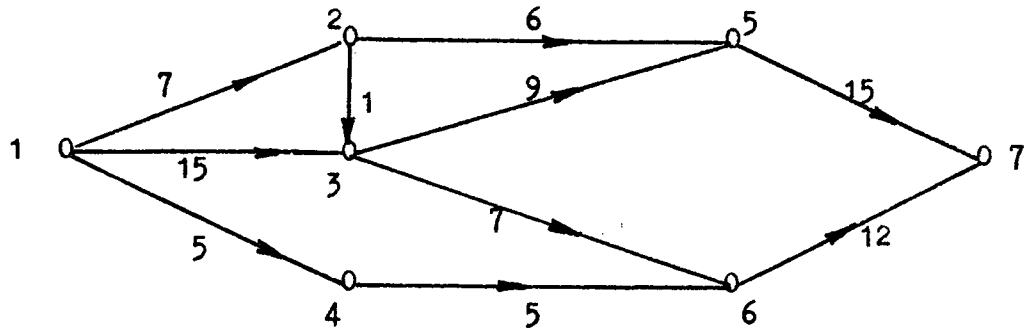


Figure 3.2. Maximal Flow for Example 3.2.1.

3.3. Minimal Cost Flows

In Section 3.2. every arc of the network was assigned a given capacity C_{ij} which limits the amount of flow that can pass through the arc. In this section every arc is also assigned a cost d_{ij}, which can be taken as the cost of shipping one unit of flow along that arc. Now we are interested in shipping a given amount of flow from the source to the sink at minimal cost. If there are no capacity restrictions on the arcs, then the problem becomes one of finding

shortest path from N_s to N_t and then shipping all the flow along that path. To state the problem formally, we want to

$$\text{Minimize } Z = \sum_i \sum_j d_{ij} \cdot x_{ij} ,$$

subject to

$$\sum_j x_{ij} - \sum_k x_{ki} = \begin{cases} v & i = s \\ 0 & i \neq s \text{ or } t \\ -v & i = t \end{cases}$$

and

$$0 \leq x_{ij} \leq C_{ij} \qquad i,j = 1,2,\ldots,n,$$

where d_{ij} is the cost of shipping one unit of flow along the arc A_{ij}, and x_{ij} is the flow in that arc. It is assumed implicitly that v is not greater than the maximal flow from N_s to N_t, otherwise the problem is not feasible.

Most of the algorithms for solving the minimal cost flows use the primal-dual approach of linear programming. We here present two algorithms for solving this problem which do not use the linear programming concept and also are quite efficient computationally.

Algorithm 3.3.1

The first algorithm can be stated as follows:

Step (0) Start with all arc flows equal to zero and the flow value equals zero.

Step (1) Define the modified costs d^*_{ij} with respect to the given flow that exists in the network as follows:

$$d^*_{ij} = d_{ij} \quad \text{if} \quad x_{ij} < C_{ij} \quad (0 \leq x_{ij})$$

$$d^*_{ij} = \infty \quad \text{if} \quad x_{ij} = C_{ij}$$

$$d^*_{ij} = -d_{ji} \quad \text{if} \quad x_{ji} > 0.$$

Step (2) Find the shortest path or the minimal cost path from N_s to N_t using the current modified costs d^*_{ij} obtained in Step (1). Then ship the flow along this path until the path is no longer the shortest path. Replace the old flow value by the old flow value plus the flow along the path. If the new flow value is v, stop. Otherwise, return to Step (1).

This algorithm has the interesting feature that when the flow in Step (2) is p, it gives the minimum cost flow of p units from N_s to N_t. Therefore, we get the minimal cost flow for $p = 1,\ldots,v$. This should be classified as a dual algorithm since we do not attain the optimal solution (the total required flow) except at the final step.

We now illustrate this algorithm with two simple examples, each presenting a different case. For illustration purposes we define the following two matrices:

(1) The matrix of excess capacities $E = (e_{ij})$,

(2) The matrix of arc flows $X = (x_{ij})$.

The elements of these two matrices satisfy the relations

$$e_{ij} + x_{ij} = C_{ij} \qquad i,j = s,1,2,\ldots,t.$$

Example 3.3.1. Consider the network of Figure 3.3., where all arcs are directed arcs. We want to find the maximal flow at minimal cost.

Each pair of numbers (x,y) beside an arc represents the arc capacity and the cost of unit flow respectively. The initial capacity and cost matrices are shown in Table 3.8. and Table 3.9. respectively.

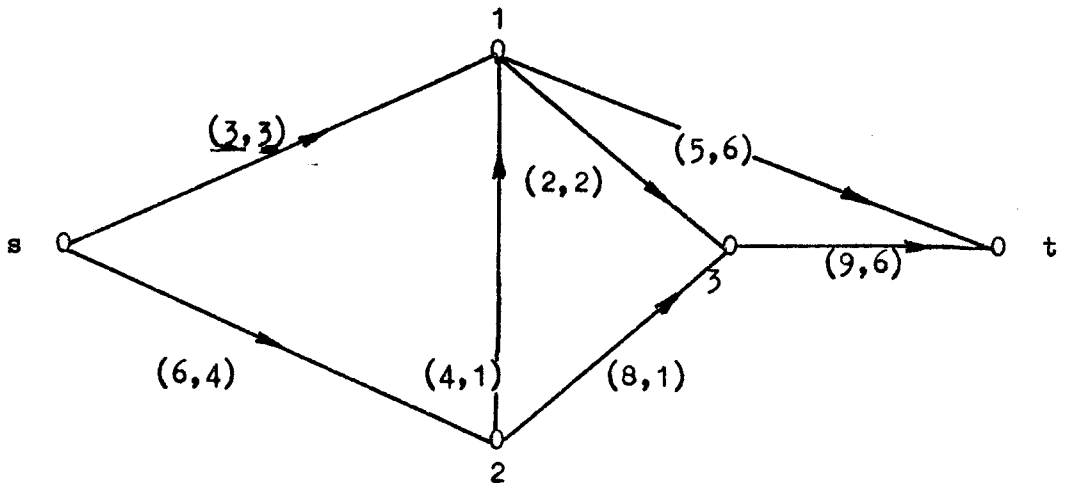


Figure 3.3. Network for Example 3.3.1.

Table 3.8. Capacity Matrix for Figure 3.3.

	s	1	2	3	t
s	0	3	6	0	0
1	0	0	0	2	5
2	0	4	0	8	0
3	0	0	0	0	9
t	0	0	0	0	0

Table 3.9. Cost Matrix for Figure 3.3.

	s	1	2	3	t
s	0	3	4	∞	∞
1	∞	0	∞	2	6
2	∞	1	0	1	∞
3	∞	∞	∞	0	6
t	∞	∞	∞	∞	0

Step (0) The arc flows $x_{ij} = 0$. The flow is $f^{o}_{st} = 0$, and the flow cost is $Z^{o} = 0$.

Step (1) Define $d^{*}_{ij} = d_{ij}$.

Step (2) Solve for the shortest distance (minimal cost) from N_s to N_t. The route of shortest distance is N_s , N_1 , N_t and the maximum flow in this path is

$$f^{1}_{st} = \max (C_{st} , \min(C_{s1} , C_{1t})) = \max (0 , \min(3,5)) = 3 ,$$

and the flows in the arcs are $x_{s1} = 3$, $x_{1t} = 3$, and all the other flows are unchanged ($x_{ij} = 0$).

Therefore the cost for this flow is

$$Z^{1} = 3(9) = 27 .$$

Return to Step (1).

Step (1) Define the modified costs d^{*}_{ij} as follows

$$d^{*}_{s1} = \infty , \quad d^{*}_{1s} = -3$$

$$d^{*}_{1t} = d_{1t} , \quad d^{*}_{t1} = -6$$

and all other costs remain unchanged. At this stage the matrices of excess capacities E , modified costs D^{*} and arc flows X are given by

$E =$

	s	1	2	3	t
s	0	0	6	0	0
1	0	0	0	2	2
2	0	4	0	8	0
3	0	0	0	0	9
t	0	0	0	0	0

$D^* =$

	s	1	2	3	t
s	0	∞	4	∞	∞
1	-3	0	∞	2	6
2	∞	1	0	1	∞
3	∞	∞	∞	0	6
t	-6	∞	∞	∞	0

$X =$

	s	1	2	3	t
s	0	3	0	0	0
1	0	0	0	0	3
2	0	0	0	0	0
3	0	0	0	0	0
t	0	0	0	0	0

Step (2) Solve for the shortest distance from s to t using the current cost matrix. To do this we use Figure 3.4. where an infinity is equivalent to the deletion of a saturated arc.

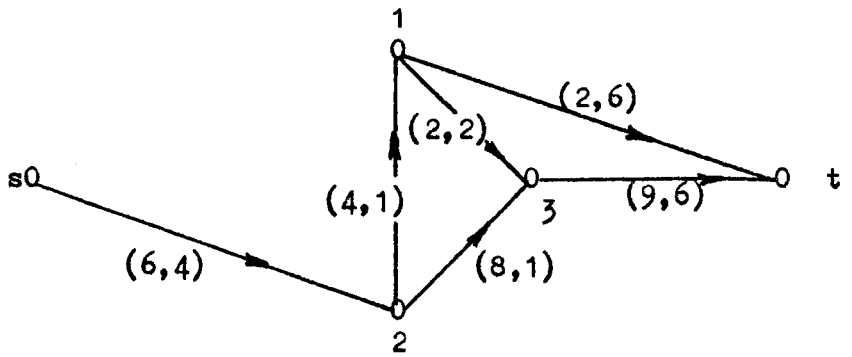


Figure 3.4. Iteration 1 Network.

The shortest path is seen to be either A_{s2} , A_{23} , A_{3t} , or A_{s2} , A_{21} , A_{1t} with total cost in both cases 11 . To solve for this tie we pick the path with maximum flow. The maximum flow for the path A_{s2} , A_{23} , A_{3t} is

$$\max (e_{st}, \min (e_{s2}, e_{23}, e_{3t}))$$

$$= \max (0, \min (6, 8, 9))$$

$$= 6 .$$

The maximum flow for the path A_{s2} , A_{21} , A_{1t} is

$$\max (e_{st}, \min (e_{s2}, e_{21}, e_{1t}))$$

$$= \min (6, 4, 2)$$

$$= 2 .$$

So we pick $f^2_{st} = 6$ and the path is A_{s2} , A_{23} , A_{3t} with total cost $Z^2 = 6 (4 + 1 + 6) = 66$.

Return to Step (1).

Step (1) Define the modified costs d^*_{ij}

$$\begin{aligned} d^*_{s2} &= \infty , & d^*_{2s} &= -4 \\ d^*_{23} &= d_{23} , & d^*_{32} &= -1 \\ d^*_{3t} &= d_{3t} , & d^*_{t3} &= -6 \end{aligned}$$

and all other costs remain unchanged. At this stage the matrices of excess capacities E , modified costs D^* and arc flows X are given by

$$E = \begin{array}{c|ccccc} & s & 1 & 2 & 3 & t \\ \hline s & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 2 & 2 \\ 2 & 0 & 0 & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 & 0 & 3 \\ t & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$D^{*} = \begin{array}{c|ccccc} & s & 1 & 2 & 3 & t \\ \hline s & 0 & \infty & \infty & \infty & \infty \\ 1 & -3 & 0 & \infty & 2 & 6 \\ 2 & -4 & 1 & 0 & 1 & \infty \\ 3 & \infty & \infty & -1 & 0 & 6 \\ t & -6 & \infty & \infty & -6 & 0 \end{array}$$

$$X = \begin{array}{c|ccccc} & s & 1 & 2 & 3 & t \\ \hline s & 0 & 3 & 6 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 3 \\ 2 & 0 & 0 & 0 & 6 & 0 \\ 3 & 0 & 0 & 0 & 0 & 6 \\ t & 0 & 0 & 0 & 0 & 0 \end{array}$$

The network at this stage is shown in Figure 3.5., where we notice that the arcs deleted correspond to saturated arcs. The pairs of numbers on the arcs show the present excess capacities and the costs of one unit of flow respectively.

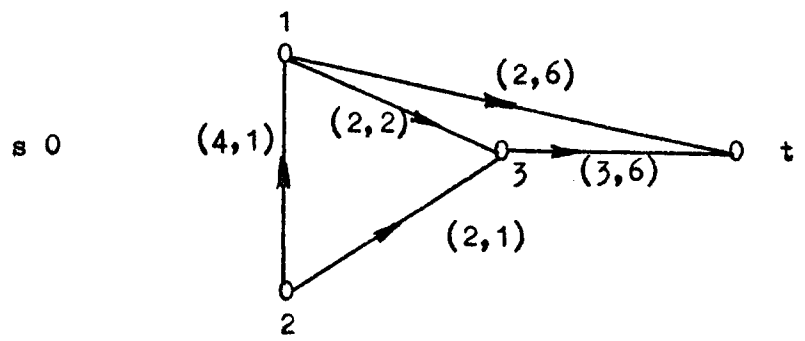


Figure 3.5. Iteration 2 Network.

We notice that the source N_s and the sink N_t became disconnected at this stage, therefore we have an optimal flow. The total cost of the flow is

$$Z = Z^1 + Z^2$$

$$Z = 27 + 66$$

Total Cost $Z = 93$.

And the present flow is

$$f_{st} = f_{st}^1 + f_{st}^2$$

$$= 3 + 6$$

Optimal Flow $= 9$ units.

The network for the minimal cost flow is shown in Figure 3.6.

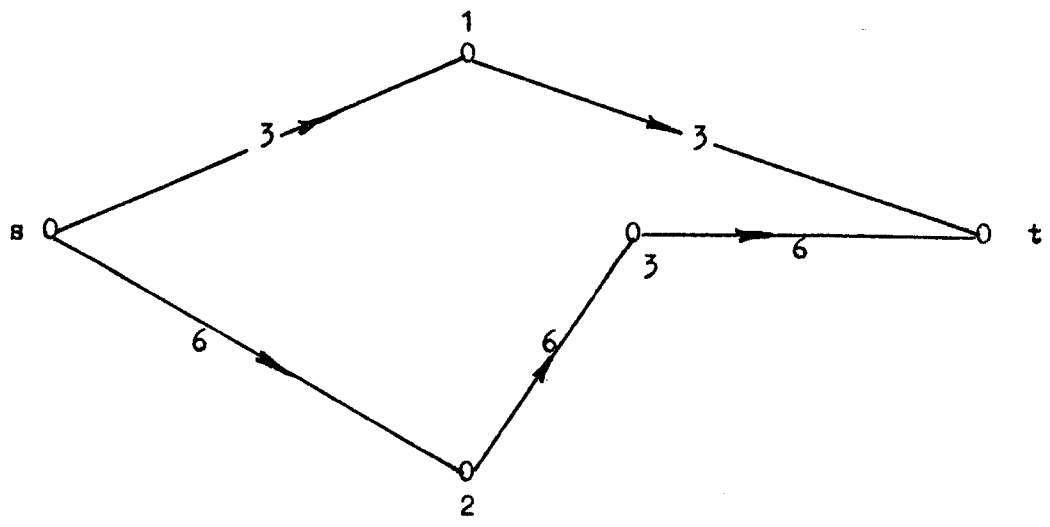


Figure 3.6. Minimal Cost Flow for Example 3.3.1.

We now give some comments in order:

(1) When we find an arc flow $x_{ij} = C_{ij}$ the corresponding modified cost $d^*_{ij} = \infty$ is equivalent to the deletion of the corresponding arc from the network which is also the same as having an excess capacity 0.

(2) When we solve the tie for equivalent routes by taking the route of larger flow we saved one more step of computation and we reached the solution quicker.

Example 3.3.2. Consider the network shown in Figure 3.7. where all arcs are undirected and we want to find a flow of five units at minimum cost.

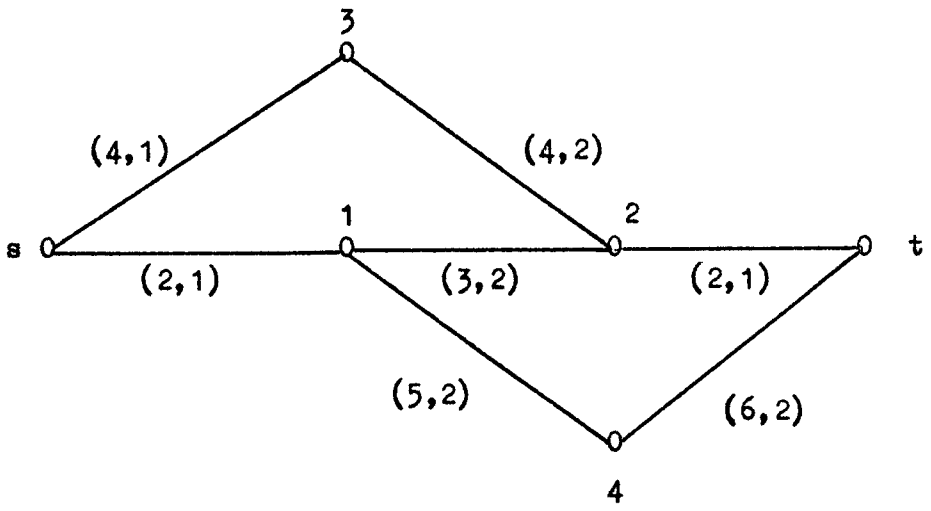


Figure 3.7. Network for Example 3.3.2.

Here we will use the same set of matrices defined in Example 3.3.1 and which are given in Tables 3.10.,3.11. and 3.12.

Step (0) Set $x_{ij} = 0$ therefore $e_{ij} = C_{ij}$.

Table 3.10. Capacity Matrix for Figure 3.7.

	s	1	2	3	4	t
s	0	2	0	4	0	0
1	4	0	3	0	5	0
2	0	3	0	4	0	2
3	4	0	4	0	0	0
4	0	5	0	0	0	6
t	0	0	2	0	6	0

Table 3.11. Cost Matrix for Figure 3.7.

	s	1	2	3	4	t
s	0	1	∞	1	∞	∞
1	1	0	2	∞	2	∞
2	∞	2	0	2	∞	1
3	1	∞	2	0	∞	∞
4	∞	2	∞	∞	0	2
t	∞	∞	1	∞	2	0

Table 3.12. Flow Matrix for Figure 3.7.

	s	1	2	3	4	t
s	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
t	0	0	0	0	0	0

Step (1) Define $d^*_{ij} = d_{ij}$.

Step (2) Solve for the shortest path from N_s to N_t using the modified costs defined in Step (1). We find that we have two shortest paths namely A_{s1}, A_{12}, A_{2t} with total cost = 4 and A_{s3}, A_{32}, A_{2t} with total cost = 4. To solve the tie we find the maximum flow in each path.

For the first path the maximum flow is

$$\begin{aligned} & \max \ (e_{st} , \min (e_{s1} , e_{12} , e_{2t})) \\ = \ & \max \ (0 \ , \ \min (4 , 3 , 2)) \\ = \ & 2 . \end{aligned}$$

And for the second path the maximum flow is

$$\begin{aligned} & \max \ (e_{st} , \min (e_{s3} , e_{32} , e_{2t})) \\ = \ & \max \ (0 \ , \ \min (4 , 4 , 2)) \\ = \ & 2 . \end{aligned}$$

Again we have a tie for the maximum flow, so we pick arbitrarily the path A_{s1} , A_{12} , A_{2t} so that

$$x_{s1} = x_{12} = x_{2t} = 2$$

and $x_{ij} = 0$ otherwise.

Hence, $f^1_{st} = 2$

and $Z^1 = 2(1 + 2 + 1)$

$= 8.$

Return to Step (1).

Step (1) Define the modified costs d^*_{ij} as follows:

$$d^*_{s1} = \infty \ , \quad d^*_{1s} = -1$$

$$d^*_{12} = d_{12} \ , \quad d^*_{21} = -2$$

$$d^*_{2t} = \infty \ , \quad d^*_{t2} = -1$$

and $d^*_{ij} = d_{ij}$ otherwise. At this stage the matrices of excess capacities E, modified costs D^* and arc flows X are given by

$E =$

	s	1	2	3	4	t
s	0	0	0	4	0	0
1	4	0	1	0	5	0
2	0	3	0	4	0	0
3	4	0	4	0	0	0
4	0	5	0	0	0	6
t	0	0	2	0	0	0

$D^* =$

	s	1	2	3	4	t
s	0	∞	∞	1	∞	∞
1	-1	0	2	∞	2	∞
2	∞	-2	0	2	∞	∞
3	1	∞	2	0	∞	∞
4	∞	2	∞	∞	0	2
t	∞	∞	-1	∞	2	0

$X =$

	s	1	2	3	4	t
s	0	2	0	0	0	0
1	0	0	2	0	0	0
2	0	0	0	0	0	2
3	0	0	0	0	0	0
4	0	0	0	0	0	0
t	0	0	0	0	0	0

Step (2) We then find the shortest path using the current modified costs. The shortest path is found to be A_{s3}, A_{32}, A_{21}, A_{14}, A_{4t}, with total cost = 5,(1 + 2 - 2 + 2 + 2), and the maximal flow is

$$\max (e_{st}, \min (e_{s3}, e_{32}, e_{21}, e_{14}, e_{4t}))$$

$= \max (0 , \min (4 , 4 , 3 , 5 , 6))$

$= 3 ,$

so we have

$$x_{s3} = x_{32} = x_{21} = x_{14} = x_{4t} = 3$$

and

$$x_{ij} = 0 \text{ otherwise.}$$

Hence,

$$f^2_{st} = 3$$

$$z^2 = 3 \times 5$$

$$= 15.$$

We see that we reached the required flow of 5 units. Hence the total flow = 5 units with total cost = 23. The present flow is shown in Figure 3.8. where the numbers indicate arc flows in the shown directions

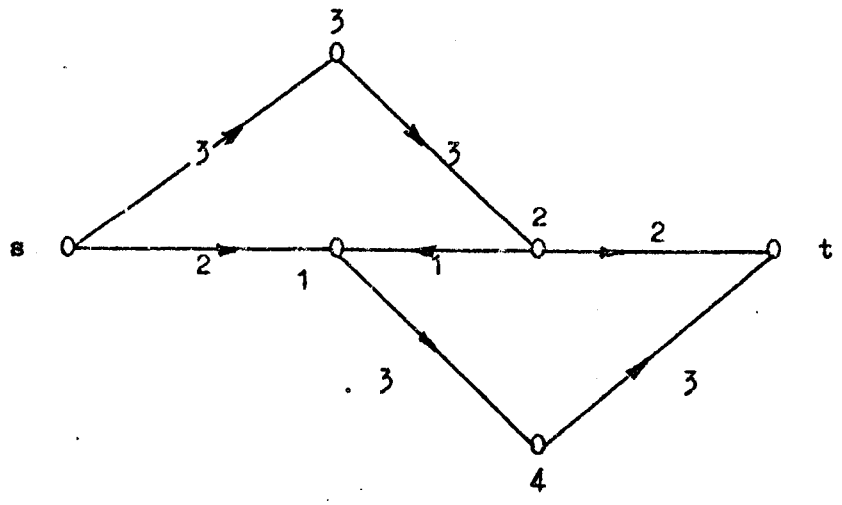


Figure 3.8. Minimal Cost Flow for Example 3.3.2.

Algorithm 3.3.2

This second algorithm can be stated as follows:

Step (1) Find any feasible flow of v units from N_s to N_t. This can be done by guessing or using the maximal flow algorithm of Section 3.2. We stop when the flow reaches v.

Step (2) Define the modified costs d^*_{ij} as follows:

$$d^*_{ij} = d_{ij} \text{ if } x_{ij} < C_{ij} \quad (0 \leq x_{ij})$$

$$d^*_{ij} = \infty \text{ if } x_{ij} = C_{ij}$$

$$d^*_{ij} = -d_{ji} \text{ if } x_{ji} > 0.$$

Step (3) Using the d^*_{ij} as distances, find negative cycles in the network. Recall that negative cycles can become detected in the shortest path problem by allowing d_{ii} to become negative, and if d_{ii} becomes negative then a negative cycle exists. If none exists, the current flow is optimum. Notice that if at the end of an iteration step of the shortest path algorithm one or more d_{ii} becomes negative we stop at this step and we don't proceed to the next, because negative cycle(s) have been found and can be deleted using the route matrix. This is because if we proceed to the next stage all information we have about the present negative cycle(s) will be eliminated by the computations of the next iteration step.

If such a negative cycle exists, superimpose on the cycle a flow of amount δ, where

$\delta = \min (C_{ij} - x_{ij})$

in the negative cycle and return Step (2). (If the negative cycles are disjoint, we superimpose similar flows on each of them.) Then define again new modified costs as in Step (1) and repeat until no more negative cycles are found.

Since this algorithm gives a feasible flow of v units to start, it should be classified as a primal algorithm. It is easy to see that both algorithms will give a flow of v units if v units is not greater than the maximal flow.

It is slightly harder to see that the algorithm will give optimum flow when completed. The proof of the two algorithms depend on the following theorem, which can be considered as the central theorem in minimal cost flow and which we state without proof:

Theorem 3.6. A flow value v is optimum if and only if, based on the modified costs, there exists no negative cycles with respect to the flow.

Example 3.3.3. Consider the network of Figure 3.9. with arc capacities and costs written beside the arcs. We want to find the maximal flow at minimum cost from N_s to N_t. The matrices of capacities and costs are given in Tables 3.13. and 3.14. repectively.

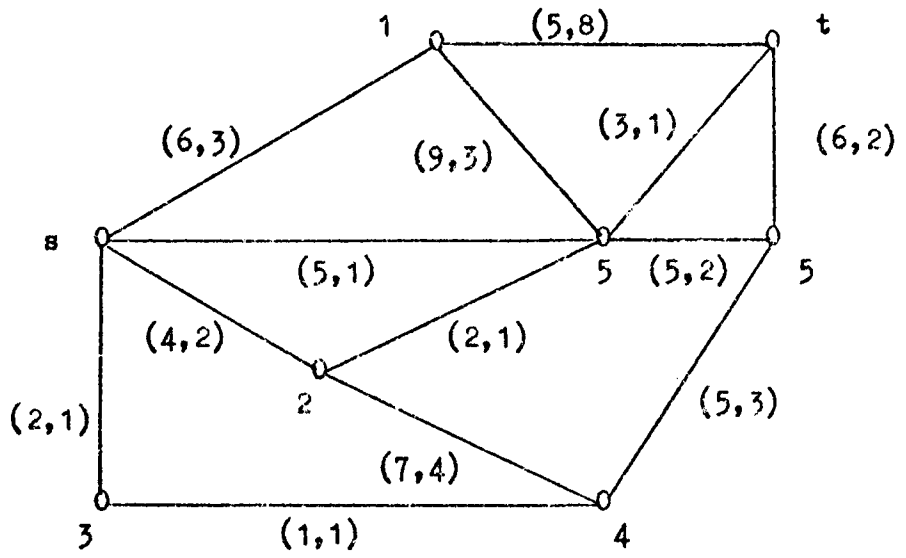


Figure 3.9. Network for Example 3.3.3.

Table 3.13. Capacity Matrix for Example 3.3.3.

	s	1	2	3	4	5	6	t
s	-	6	4	2	0	5	0	0
1	6	-	0	0	0	9	0	15
2	4	0	-	0	7	2	0	0
3	2	0	0	-	1	0	0	0
4	0	0	7	1	-	0	5	0
5	5	9	2	0	0	-	5	3
6	0	0	0	0	5	5	-	6
t	0	15	0	0	0	3	6	-

Table 3.14. Cost Matrix for Example 3.3.3.

	s	1	2	3	4	5	6	t
s	0	3	2	1	∞	1	∞	∞
1	3	0	∞	∞	∞	3	∞	8
2	2	∞	0	∞	4	1	∞	∞
3	1	∞	∞	0	1	∞	∞	∞
4	∞	∞	4	1	0	∞	3	∞
5	1	3	1	∞	∞	0	2	1
6	∞	∞	∞	∞	3	2	0	2
t	∞	8	∞	∞	∞	1	2	0

Step (1) Find the maximal flow from the source N_s to the sink N_t using the maximal flow algorithm of Section 3.2.

The maximal flow is found to be $f_{st}^{max} = 16$ units, as shown

in Figure 3.10. where the numbers beside the arcs represent the arc flows which constitute the maximal flow (an arc with no number has zero flow).

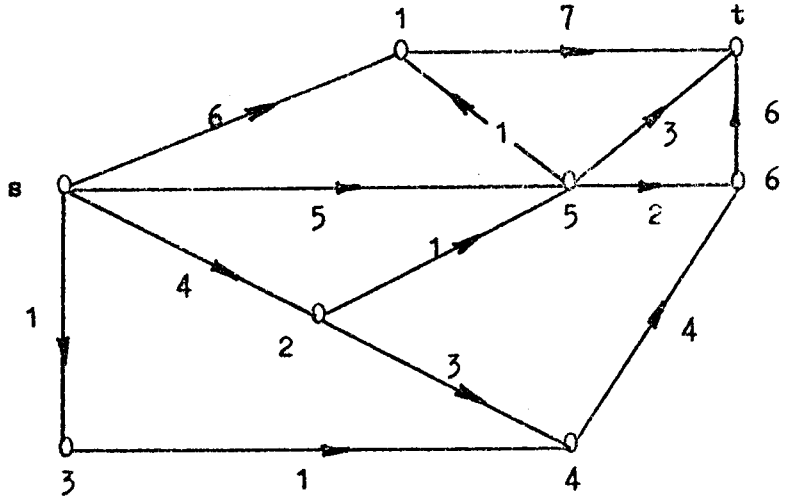


Figure 3.10. Maximal Flow for Example 3.3.3.

Step (2) Define the modified costs d^*_{ij} as follows:

$$
\begin{aligned}
d^*_{s1} &= \infty, & d^*_{1s} &= -3 \\
d^*_{s2} &= \infty, & d^*_{2s} &= -2 \\
d^*_{s3} &= 1, & d^*_{3s} &= -1 \\
d^*_{s5} &= \infty, & d^*_{5s} &= -1 \\
d^*_{1t} &= 8, & d^*_{t1} &= -8 \\
d^*_{15} &= -3, & d^*_{51} &= 3 \\
d^*_{24} &= 4, & d^*_{42} &= -4 \\
d^*_{25} &= 1, & d^*_{52} &= -1 \\
d^*_{34} &= \infty, & d^*_{43} &= -1 \\
d^*_{46} &= 3, & d^*_{64} &= -3 \\
d^*_{56} &= 2, & d^*_{65} &= -2
\end{aligned}
$$

$$d^*_{5t} = \infty \quad , \quad d^*_{t5} = -1$$
$$d^*_{6t} = \infty \quad , \quad d^*_{t6} = -2$$

and the matrix of modified costs D^* is given by

$D^* =$

	s	1	2	3	4	5	6	t
s	∞	∞	∞	1	∞	∞	∞	∞
1	-3	∞	∞	∞	∞	-3	∞	8
2	-2	∞	∞	∞	4	1	∞	∞
3	-1	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	-4	-1	∞	∞	3	∞
5	-1	3	-1	∞	∞	∞	2	∞
6	∞	∞	∞	∞	-3	-2	∞	∞
t	∞	-8	∞	∞	∞	-1	-2	∞

Step (3) We now use the shortest path algorithm discussed in Chapter I to find the minimum cost (shortest path) solution for the matrix of modified costs D^* to detect whether negative cycles exist or not. The arcs in the minimum cut are saturated which implies that their corresponding $d^*_{ij} = \infty$ and therefore these arcs cannot be members of any negative cycles. Therefore we can split the network into two separate networks and solve the minimum cost problem for each of them separately.

This is shown in Tables 3.15. and 3.16.

Table 3.15. Modified Costs for the First Subnetwork.

	s	3
s	∞	1
3	-1	∞

Table 3.16. Modified Costs for the Second Subnetwork.

	1	2	4	5	6	t
1	∞	∞	∞	-3	∞	8
2	∞	∞	4	1	∞	∞
4	∞	-4	∞	∞	3	∞
5	3	-1	∞	∞	2	∞
6	∞	∞	-3	-2	∞	∞
t	-8	∞	∞	-1	-2	∞

Applying the shortest paths algorithm to Tables 3.15. and 3.16. and following Step (3) of the algorithm 3.3.2. we find that for the matrix of Table 3.15. no negative cycles exist, and for the matrix of Table 3.16. we find that a negative cycle exists. The matrix of negative cycle corresponding to Table 3.16. is shown in Table 3.17. Table 3.18. gives the route matrix necessary to detect the negative cycle.

We see from Table 3.17. that $d_{66} = -4$ and so a negative cycle exists where N is a member and this negative cycle has been found at the fifth iteration stage of the shortest

path algorithm, so we stopped our iteration procedures and detect this negative cycle.

It follows from Table 3.18. that the negative cycle is A_{64} , A_{42} , A_{25} , A_{56} . Therefore we need to superimpose a certain flow over this negative cycle to eliminate it. We compute the amount of flow to be superimposed on the present flow by

$$\delta = \min (e_{64} , e_{42} , e_{25} , e_{56})$$

$$= \min (5 , 7 , 1 , 3)$$

$$= 1 .$$

By superposition of $\delta = 1$ unit of flow will result in the optimum flow in that part. If we apply again the shortest path algorithm for the new flow, we will find no more negative cycles in the network, therefore the optimum flow has been reached and this is shown in Figure 3.11. The cost for this flow is

$$Z = 132 \text{ at a maximum flow } = 16 \text{ units.}$$

Table 3.17. Negative Cycle Matrix

	1	2	4	5	6	t
1	0	-4	0	-3	-1	8
2	4	0	4	1	3	12
4	0	-4	0	-3	-1	8
5	3	-1	3	0	2	11
6	-3	-7	-3	-6	(-4)	5
t	-8	-12	-14	-11	-9	0

Table 3.18. Route Matrix

	1	2	4	5	6	t
1	5	5	5	5	5	t
2	5	4	4	5	5	5
4	5	2	2	2	5	5
5	1	2	2	1	6	1
6	5	2	4	4	5	5
t	1	5	5	1	5	1

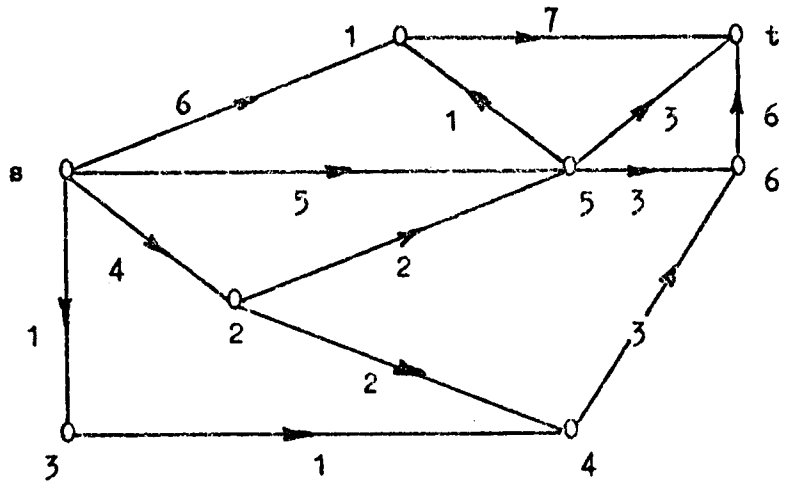


Figure 3.11. Minimal Cost Flow for Example 3.3.3.

Before we end this chapter let us give some comments:

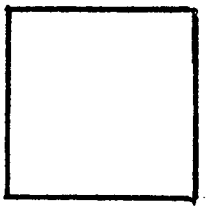
(1) The first algorithm is not recommended when v is large, since we have to find, for each feasible flow at each stage of the solution, a shortest path solution.

(2) In case of large v or for finding the maximal flow at minimal cost, the second algorithm is recommended as we only solve the problem of maximal flow once and negative cycles are detected easily using the shortest path algorithm.

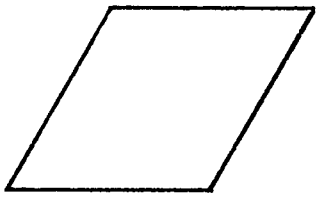
CHAPTER IV

FLOW CHARTS AND FORTRAN IV COMPUTER PROGRAMS

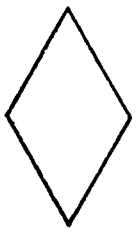
The flow charts for the main algorithms presented in Chapters I and III are shown on the following pages. We use the standard flow charts symbols shown in Figure 4.1.



(a) Process

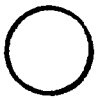


(b) Input / Output



(c) Decision

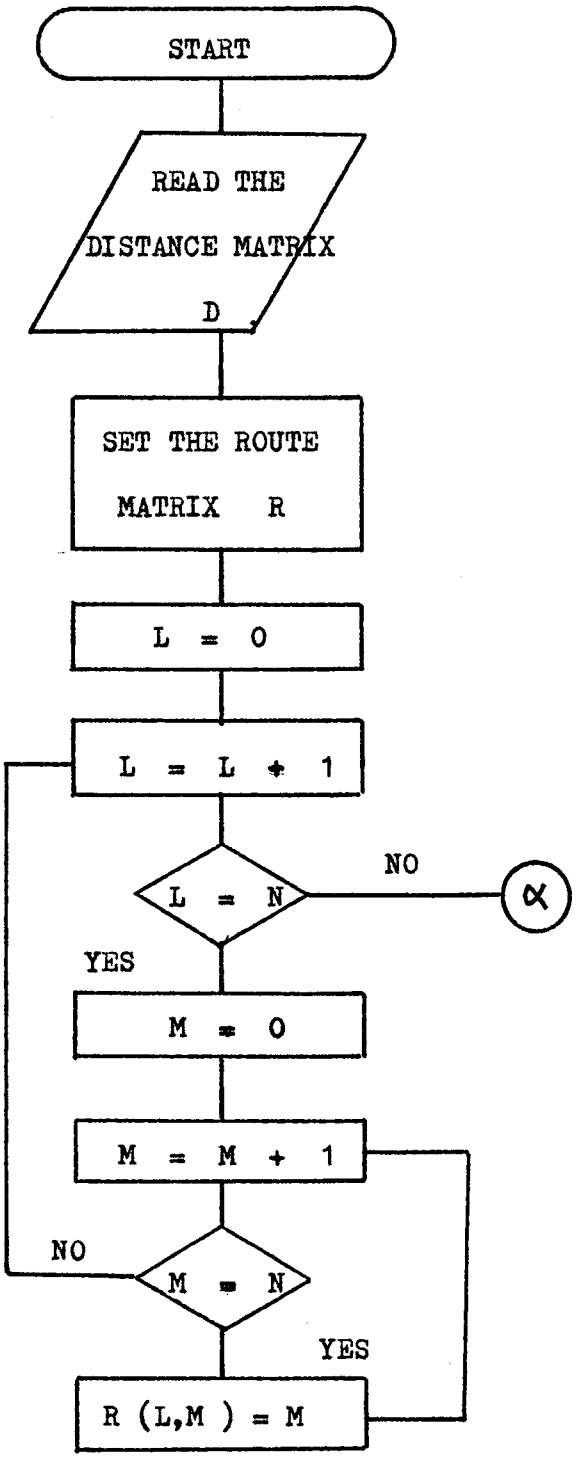
(d) Terminal

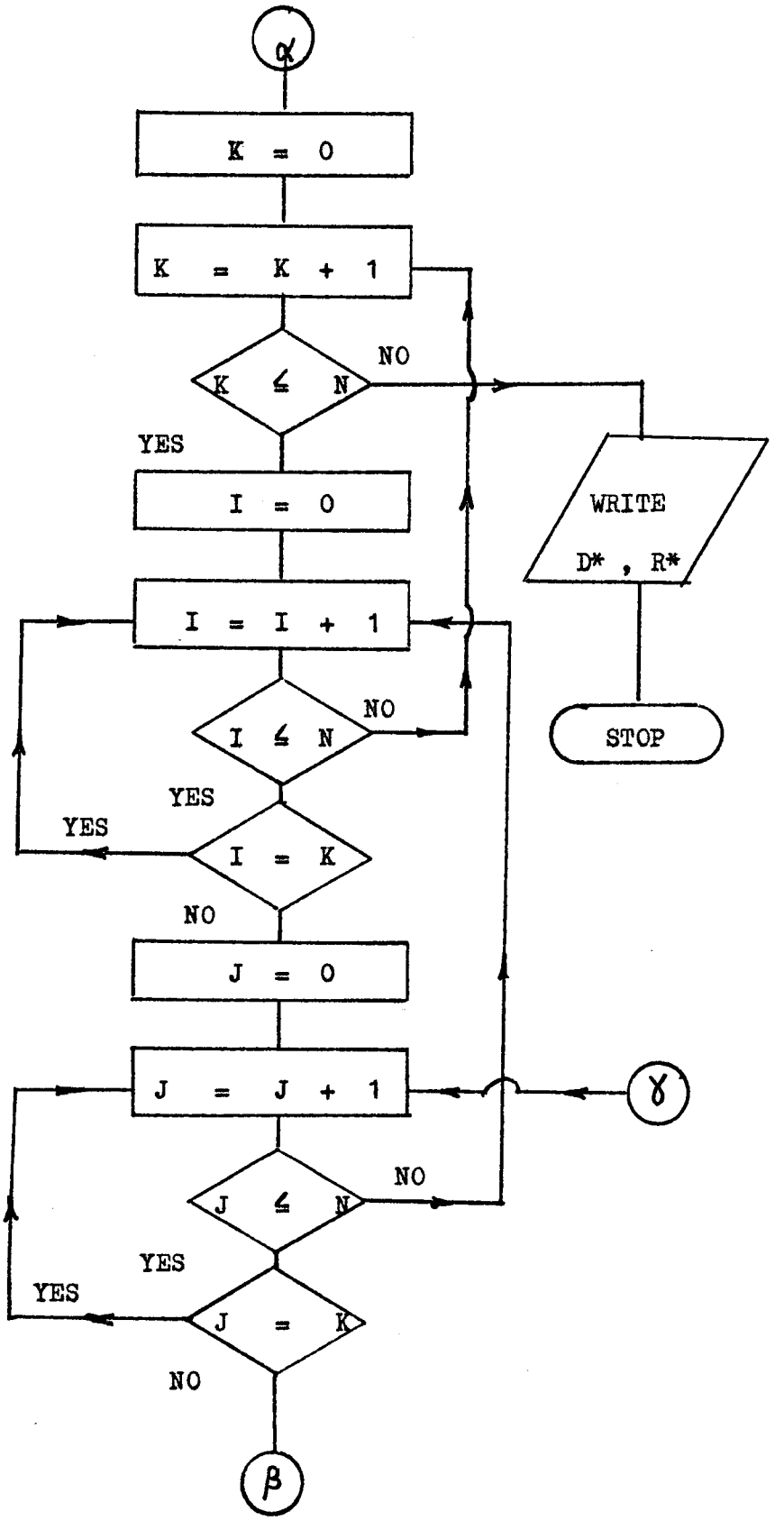


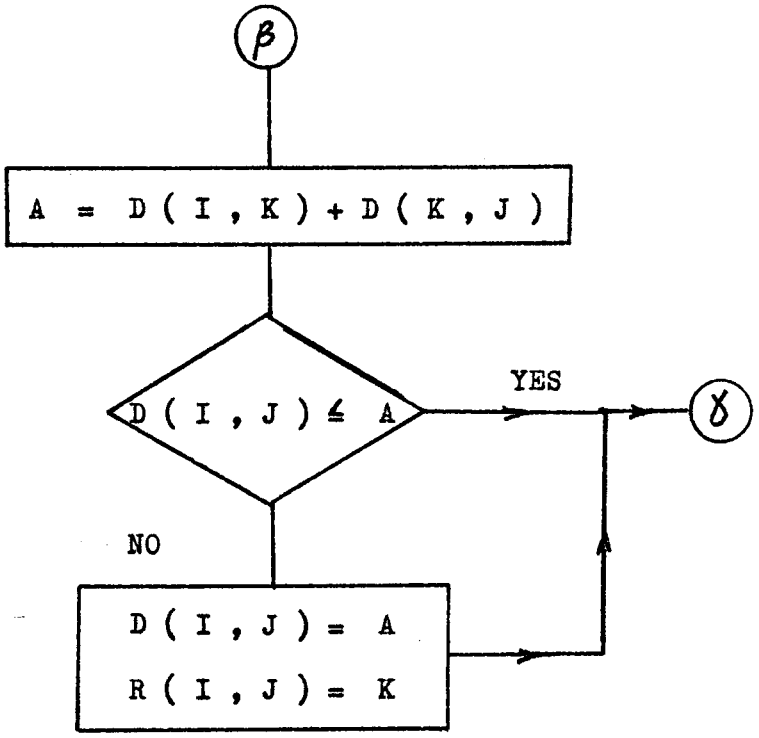
(e) Off Page Connector

Figure 4.1. Standard Flow Charting Symbols.

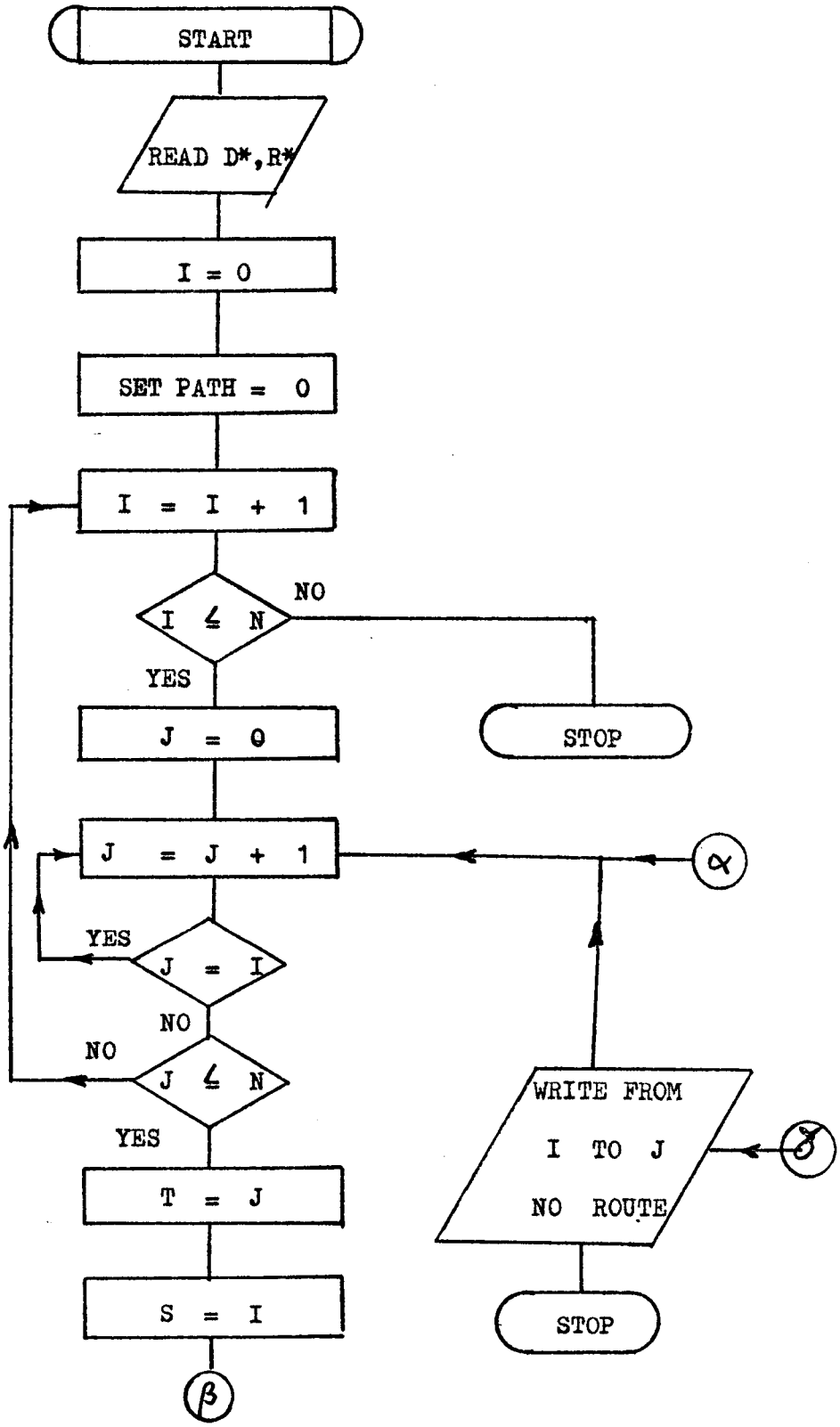
4.1. Flow Chart for the Shortest Path Algorithm

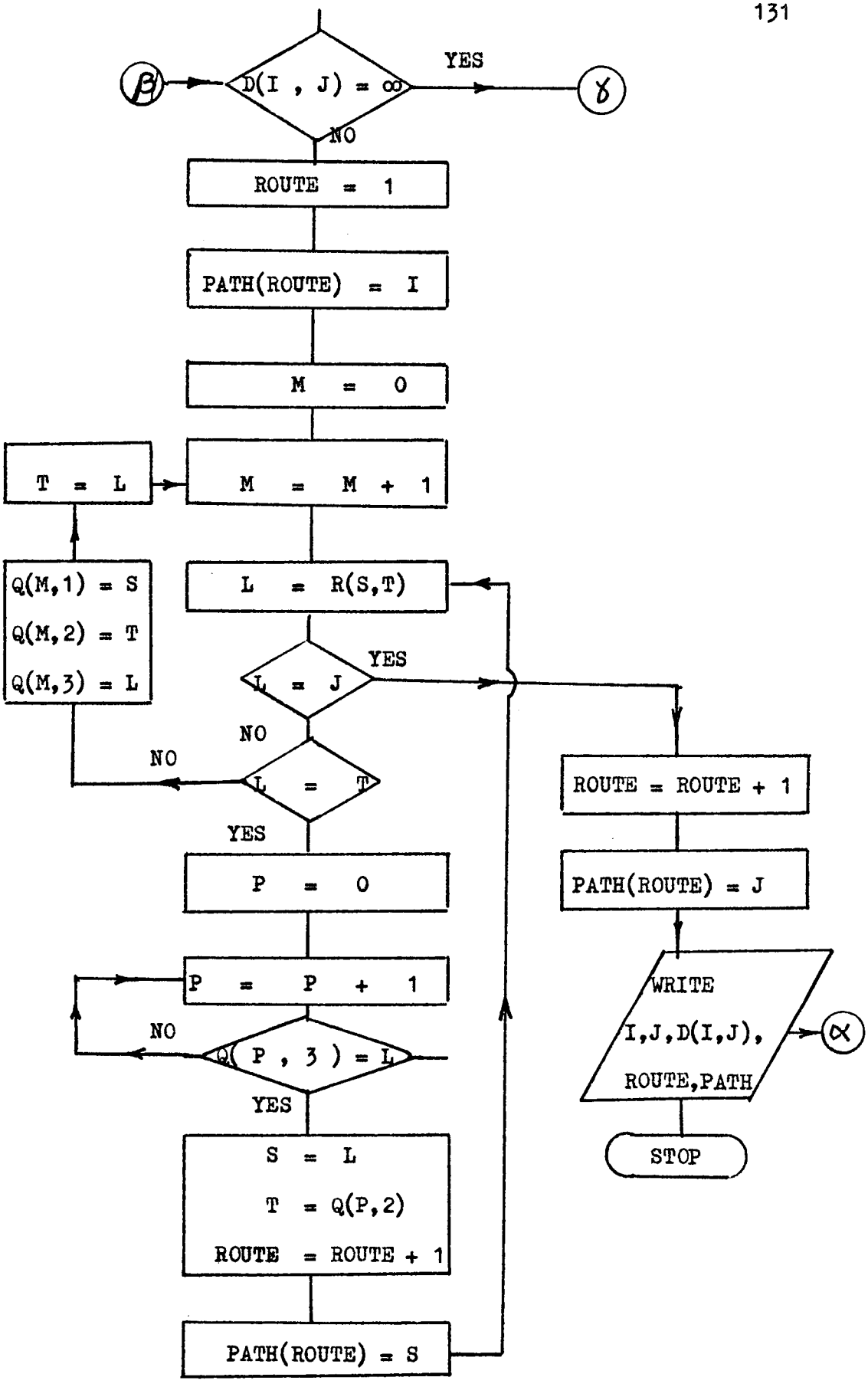



α
K = 0
K = K + 1
K ≤ N
NO
YES
I = 0
WRITE
D* , R*
I = I + 1
STOP
I ≤ N
NO
YES
YES
I = K
NO
J = 0
J = J + 1
γ
J ≤ N
NO
YES
YES
J = K
NO
β

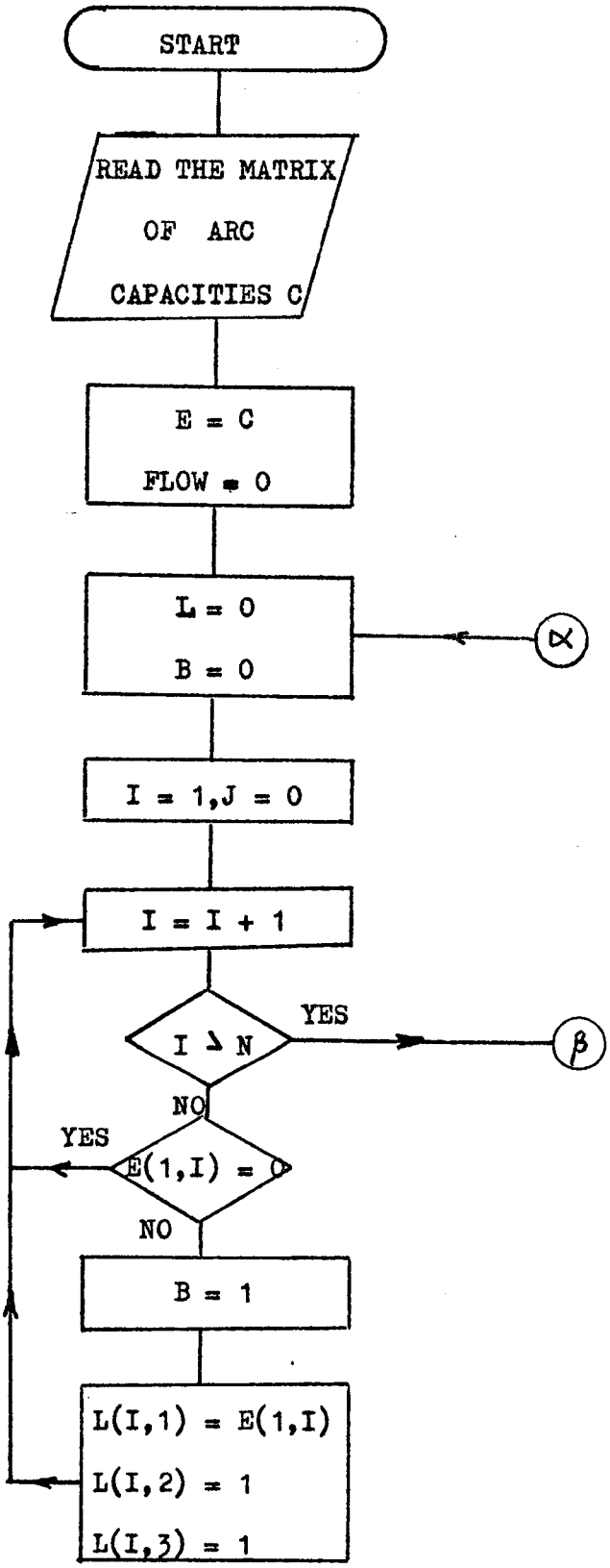

β
A = D (I , K) + D (K , J)
D (I , J) ≤ A
YES
γ
NO
D (I , J) = A
R (I , J) = K

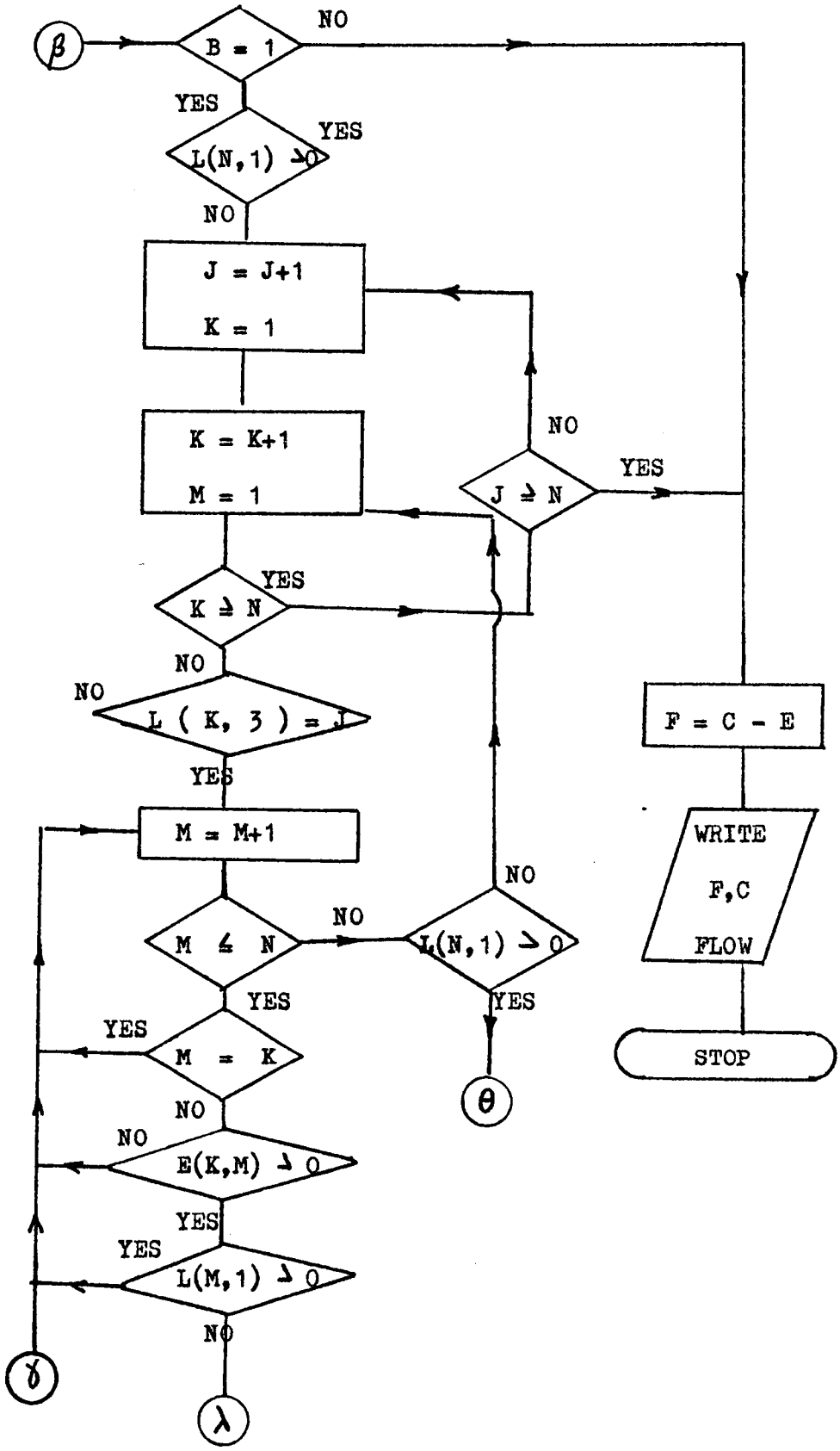
4.2. Flow Chart to Track All Shortest Paths in a Graph

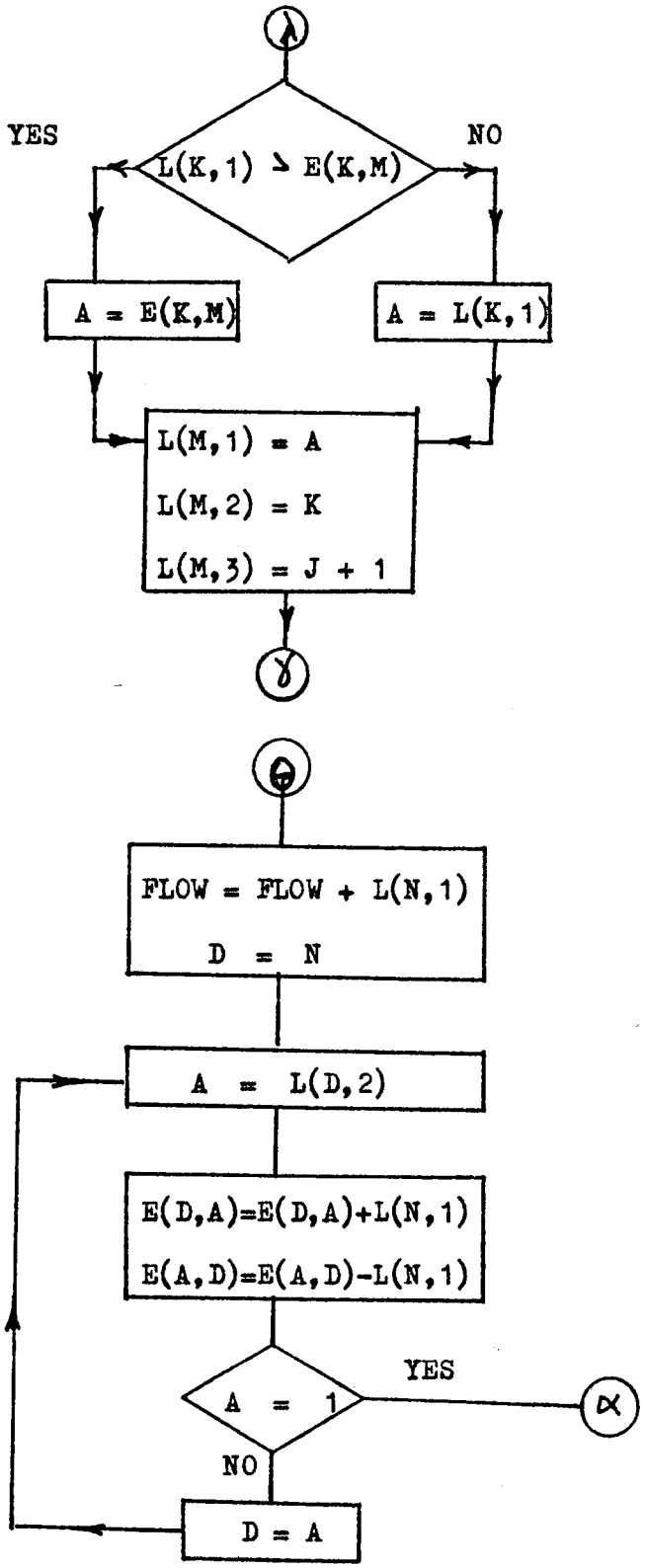



β
D(I , J) = ∞
YES
γ
NO
ROUTE = 1
PATH(ROUTE) = I
M = 0
T = L
M = M + 1
Q(M,1) = S
Q(M,2) = T
Q(M,3) = L
L = R(S,T)
L = J
YES
NO
NO
L = T
YES
P = 0
P = P + 1
NO
Q(P , 3) = L
YES
S = L
T = Q(P,2)
ROUTE = ROUTE + 1
PATH(ROUTE) = S
ROUTE = ROUTE + 1
PATH(ROUTE) = J
WRITE
I,J,D(I,J),
ROUTE,PATH
α
STOP

4.3. Flow Chart for the Maximal Flow Algorithm




β
B = 1
NO
YES
L(N,1) ≻ 0
YES
NO
J = J+1
K = 1
K = K+1
M = 1
NO
J ≥ N
YES
K ≥ N
YES
NO
NO
L (K, 3) = J
YES
M = M+1
NO
M ≤ N
NO
L(N,1) ≻ 0
YES
YES
M = K
YES
NO
NO
E(K,M) ≻ 0
YES
YES
L(M,1) ≻ 0
NO
γ
λ
θ
F = C - E
WRITE
F,C
FLOW
STOP


YES
L(K,1) > E(K,M)
NO
A = E(K,M)
A = L(K,1)
L(M,1) = A
L(M,2) = K
L(M,3) = J + 1
γ
θ
FLOW = FLOW + L(N,1)
D = N
A = L(D,2)
E(D,A)=E(D,A)+L(N,1)
E(A,D)=E(A,D)-L(N,1)
A = 1
YES
α
NO
D = A

4.4. Fortran IV Program for the Shortest Path Algorithm

```
C     THIS PROGRAM READS THE MATRIX OF ALL DISTANCES FOR A GIVEN
C     GRAPH, AND SETS UP THE ROUTE MATRIX.
C     THE PROGRAM RESULTS ARE AS FOLLOWS:
C     (1)  THE MATRIX OF SHORTEST DISTANCES BETWEEN ALL PAIRS OF
C          NODES IN THE GRAPH, DENOTED BY D*.
C     (2)  THE MATRIX NECESSARY FOR DETECTING ANY ROUTE BETWEEN
C          ANY PAIR OF NODES IN THE GRAPH, DENOTED BY R*.
C
C
      INTEGER D(20,20),R(20,20),Q(20,3),PATH(20),A,S,P,T,ROUTE,E
    1 READ(5,2,END = 99) N
    2 FORMAT(I5)
      DO 6 I=1,N
      DO 6 L=10,N,10
      K=L-9
    6 READ(5,7) (D(I,J),J=K,L)
    7 FORMAT(10I5)
C
C     THIS PORTION OF THE PROGRAM SETS UP THE ROUTE MATRIX R
C
      DO 10 I=1,N
      DO 10 J=1,N
      R(I,J)=J
   10 CONTINUE
```

```
C     THIS PORTION EXECUTES THE TRIPLE OPERATION, NAMELY THE
C     OPERATION DEFINED BY THE SYMBOL ":=".
C     THEN THE ROUTE MATRIX R IS UPDATED WHEN THE OPERATION  :=
C     IS EFFECTIVE.
C
      DO 50 K=1,N
      DO 40 I=1,N
      IF(I.EQ.K)GO TO 40
      DO 30 J=1,N
      IF(J.EQ.K)GO TO 30
      A=D(I,K)+D(K,J)
      IF(D(I,J).LE.A)GOTO 30
      D(I,J)=A
      R(I,J)=K
   30 CONTINUE
   40 CONTINUE
   50 CONTINUE
      WRITE(6,100)
      WRITE(6,70)N
   70 FORMAT(10X,'THE SOLUTION OF THE SHORTEST PATHS PROBLEM',
     */,/,6X,'THE NUMBER OF NODES IN THE NETWORK =  ',I4)
      WRITE(6,100)
      WRITE(6,60)
   60 FORMAT(10X,'THE MATRIX OF THE SHORTEST DISTANCES D*')
      DO 80 M=20,N,20
      L=M-19
      DO 80 I=1,N
```

```
   80 WRITE(6,90)(D(I,J),J=L,M)
      WRITE(6,100)
      WRITE(6,1000)
 1000 FORMAT(10X,'THE ROUTE MATRIX R* ')
      WRITE(6,100)
      DO 1100 K=20,N,20
      L=K-19
      DO 1100 I=1,N
 1100 WRITE(6,90)(R(I,J),J=L,K)
   90 FORMAT(/,3X,20I6,/)
  100 FORMAT(1H1)
      GO TO 1
   99 CONTINUE
      STOP
      END

$ENTRY
```

4.5. Fortran IV Program for Tracking All Shortest Paths

```
C     THIS PROGRAM DETECTS ALL POSSIBLE SHORTEST PATHS BETWEEN
C     ALL PAIRS OF NODES, AND WRITES THESE PATHS WITH THEIR TOTAL
C     DISTANCES.
C
      DO 600 I=1,N
      DO 500 J=1,N
      DO 155 E=1,N
      PATH(E)=0
  155 CONTINUE
      IF(J.EQ.I)GO TO 500
      S=I
      T=J
      IF(D(I,J).GE.10000)GO TO 5000
      ROUTE=1
      PATH(ROUTE)=I
      M=0
   60 M=M+1
   70 L=R(S,T)
      IF(L.EQ.J)GO TO 100
      IF(L.EQ.T)GO TO 80
      Q(M,1)=S
      Q(M,2)=T
      Q(M,3)=L
      T=L
      GO TO 60
```

```
   80 P=0
   90 P=P+1
      IF(P.GT.M)STOP
      IF(Q(P,3).NE.L)GO TO 90
      T=Q(P,2)
      S=L
      ROUTE=ROUTE+1
      PATH(ROUTE)=S
      GO TO 70
  100 ROUTE=ROUTE+1
      PATH(ROUTE)=J
      WRITE(6,200)I,J,D(I,J),(PATH(E),E=1,N)
  200 FORMAT(7X,'START. NODE  /   END NODE   /   TOTAL DIST.  /
     *  PATH',/,9X,I3,15X,I3,10X,I4,10X,20I3)
      GO TO 500
 5000 WRITE(6,700)I,J
  700 FORMAT(2X,'FROM ',I5,'   TO   ',I5,'  NO ROUTE')
  500 CONTINUE
  600 CONTINUE
      GO TO 1
   99 CONTINUE
      STOP
      END
$ENTRY
```

4.6. Fortran IV Program for the Maximal Flow Algorithm

```
C     THIS PROGRAM SOVES THE PROBLEM OF MAXIMAL FLOW IN A NETWORK.
C     THE RESULTS OF THIS PROGRAM ARE AS FOLLOWS:
C     (1)  THE VALUE OF THE MAXIMAL FLOW, DENOTED FLOW,
C     (2)  THE MATRIX OF THE ARC FLOWS, DENOTED F.
C     IN THE SEQUENCE OF THE PROGRAM WE WILL NEED THREE MATRICES,
C     NAMELY
C     (1)  THE MATRIX C(N*N),WHICH IS THE MATRIX OF ARC CAPACITIES
C     (2)  THE MATRIX E(N*N), WHICH IS THE MATRIX OF EXCESS CAPACI-
C          TIES.
C     (3)  THE MATRIX L (N*3),WHICH SERVES AS THE LABELING MATRIX.
C          ITS THREE COLUMNS ARE AS FOLLOWS:
C          L(J,1) = THE FLOW AUGMENTING VALUE
C          L(J,2) = NODE FROM WHICH THE VALUE L(J,1) ORIGINATES
C          L(J,3) = THE STEP NUMBER.
C          THE FOLLOWING INDICES SERVE IN THE SOLUTION AS FOLLOWS:
C     (1)  J REPRESENTS THE STEP NUMBER IN ONE ITERATION.
C     (2)  K THE NODE FROM WHICH AN AUGMENTING FLOW ORIGINATES.
C     (3)  M THE NODE AT WHICH THE FLOW ENTERS.
C
C
      INTEGER C(8,8),E(8,8),F(8,8),L(8,3),A,B,D,FLOW
    1 READ(5,2,END=99)N
    2 FORMAT(I4)
      DO 3 I=1,N
```

```
    3 READ(5,4)(C(I,J),J=1,N)
    4 FORMAT(7I5)
C
C     THE NEXT PORTION SETS UP THE MATRIX OF EXCESS CAPACITIES E,
C     WHERE INITIALLY THE FLOW = 0, AND  E = C.
C
      FLOW = 0
      DO 5 I=1,N
      DO 5 J=1,N
      E(I,J)=C(I,J)
    5 CONTINUE
C
C     THE NEXT PORTION SETS UP THE LABELING MATRIX L = 0
C
C
   10 DO 6 I=1,N
      DO 6 J=1,3
      L(I,J)=0
    6 CONTINUE
C
C     THE NEXT STATEMENTS LABEL THE NODES WHICH CAN BE REACHED FROM
C     THE SOURCE N1.
C
      B=0
      J=0
      DO 7 I=1,N
      IF(E(1,I).EQ.0)GO TO 7
```

```
      B=1
      L(I,1)=E(1,I)
      L(I,2)=1
      L(I,3)=1
    7 CONTINUE
      IF(B.EQ.0) GO TO 120
C
C     THE NEXT STATMENT CHECKS WHETHER THE SINK NODE HAS BEEN
C     LABELED OR NOT.
C
      IF (L(N,1).GT.0) GO TO 100
C
C     THE NEXT PART OF THE PROGRAM LABELS THE NODES IN THE NETWORK
C     UNTIL WE REACH THE SINK NODE.
C
   20 J=J+1
      K=1
   30 K=K+1
      M=1
      IF(K.LT.N) GO TO 40
      IF(J.EQ.N+1.AND.L(N,1).EQ.0) GO TO 120
      GO TO 20
   40 IF (L(K,3).EQ.J) GO TO 50
      GO TO 30
   50 M=M+1
      IF(M.LE.N) GO TO 60
      IF(L(N,1).GT.0) GO TO 100
```

```
      GO TO 30
   60 IF(M.EQ.K) GO TO 50
      IF(E(K,M).GT.0) GO TO 55
      GO TO 50
   55 IF(L(M,1).GT.0)GO TO 50
      IF(L(K,1).GT.E(K,M)) GO TO 70
      A=L(K,1)
      GO TO 80
   70 A=E(K,M)
   80 L(M,1)=A
      L(M,2)=K
      L(M,3)=J+1
      GO TO 50
C
C     THE NEXT PART REPRESENTS THE CHANGE OF THE FLOW, OR THE FLOW
C     INCREASING PROCEDURE TO BE MORE PRECISE.
C
  100 FLOW=FLOW+L(N,1)
  301 DO 300 I=1,N
  300 WRITE(6,180)(E(I,J),J=1,N)
      WRITE(6,140)
      WRITE(6,201)FLOW,L(N,1)
  201 FORMAT(/,15X,'PRESENT FLOW=',I4,3X,'ALPHA=',I4)
      D=N
  110 A=L(D,2)
      E(D,A)=E(D,A)+L(N,1)
      E(A,D)=E(A,D)-L(N,1)
```

```
      IF(A.EQ.1) GO TO 10
      D=A
      GO TO 110
  120 DO 130 I=1,N
      DO 130 J=1,N
      F(I,J)=C(I,J)-E(I,J)
  130 CONTINUE
      DO 400 I=1,N
  400 WRITE(6,180)(E(I,J),J=1,N)
      WRITE(6,140)
  140 FORMAT(1H1)
      WRITE (6,150) FLOW
  150 FORMAT(5X,'THE MAXIMAL FLOW IN THE NETWORK HAS A VALUE
     *MAX FLOW = ',I6)
      WRITE(6,140)
      WRITE(6,160)
  160 FORMAT(/,10X,'THE MATRIX OF THE CAPACITIES  C',/,/)
      DO 170 I=1,N
  170 WRITE(6,180)(C(I,J),J=1,N)
  180 FORMAT(/,15X,7I4,/)
      WRITE(6,140)
      WRITE(6,190)
  190 FORMAT(/,10X,'THE MATRIX OF ARC FLOWS  F',/,/)
      DO 200 I=1,N
  200 WRITE(6,180)(F(I,J),J=1,N)
      GO TO 1
      STOP
      END
```

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Busaker, R. G., and P. J. Gowen, "A Procedure for Determining a Family of Minimal-Cost Network Flow Patterns," ORO Technical Report 15, Operations Research Office, Johns Hopkins University, 1961.

2. Cooper & Steinberg, "Methods and Applications of Linear Programming," W. B. Saunder Company, Philadelphia, Pa., 1974.

3. Dantzig, George B., "Linear Programming and Extensions," Princeton University Press, Princeton, New Jersey, 1963.

4. Edward Menieka, "Optimization Algorithms for Networks and Graphs," Marcel Dekker Inc., New York and Basel, 1978.

5. Ford and Fulkerson, "Flows in Networks," Princeton University Press, Princeton, New Jersey, 1962.

6. Hadley, "Linear Programming," Addison-Wesley Pub. Co. Inc., Reading, Mass., 1962.

7. Hu T. C., "Integer Programming and Network Flows," Addison-Wesley Publishing Co. Inc., Reading, Mass., 1969.

8. Klein, M., "A Primal Method for Minimal Cost Flows," Management Science, 14(3) , pp. 205 - 220 (November, 1967).

9. Sivazlian & Stanfel, "Optimization Techniques in Operations Research," Prentice Hall, Englewood Cliffs, New Jersey, 1975.

VITA

Fathy Fouad Yassa-Greiss was born in Cairo, Egypt on August 15, 1950. He attended the primary, preparatory and secondary levels of the "Jesuites" school, College de la Sainte Famille, in Cairo and was graduated in July 1969.

The following October he entered the Faculty of Engineering of Cairo University in Egypt and in July 1974, he received the degree of Bachelor of Electrical Engineering (Electronics and Communications) with a rate of appreciation , Very Good with honors. The following September, he was appointed to the job of Teaching Demonstrator in the Department of Engineering Mathematics and Engineering Physics in the Faculty of Engineering, Cairo University.

In October 1974, he entered the Faculty of Science of Ain Shams University in Egypt, and in June 1976 he received the degree of Bachelor of Science in Mathematics with a rate of appreciation Very Good with honors.

In September 1978, he received a graduate teaching assistantship at The University of Tennessee, Knoxville and received the Master of Science Degree with a major in Mathematics in December 1979.