DEPARTMENT OF COMMUNICATIONS

CANADA

MINISTÈRE DES COMMUNICATIONS

THE SYNTHESIS OF COMPUTER-COMMUNICATION NETWORKS

by

John deMercado & Kalman Toth

Terrestrial Planning Branch
Department of Communications
Ottawa

May 1972

CANADA

THE SYNTHESIS of COMPUTER-COMMUNICATIONS NETWORKS

by

John deMercado & Kalman Toth

Terrestrial Planning Branch
Department of Communications
Ottawa

May 1972

ABSTRACT

In this study, techniques in the form of computationally feasible Algorithms are presented for the optimal synthesis of minimum cost simultaneous transmission networks and the near optimal synthesis of minimum cost time shared computer communication networks. Methods and Algorithms are also given for the synthesis of minimum cost non flow redundant networks.

Computer programs written in Fortran 4 and compiled on a Sigma 7 computer that implement these Algorithms are in the appendices.

FORWARD

This study was part of Kalman Toth's work for the Master of Engineering degree at Carleton University in 1972.

TABLE OF CONTENTS

INTRODUCTION

The computer industry reached maturity in the sixties and it now is certain that computer networks will be the wealth generators that propel Canada into the 21st century. An urgent need exists for computer systems to share each others software and hardware resources by coupling them together with communication links thereby creating what is called a computer-communication network.

Before costly and major network design commitments are made, it is clear that simulations must be undertaken to a priori ascertain the performance and cost of such large systems. This study provides analysis and synthesis techniques and algorithms for the topological design of large computer-communication networks.

An important problem treated in this study is the synthesis of a network providing the required channel capacities between various communication centres. That is, given the nodal configuration of the flow requirements between all pairs of nodes (terminals), the synthesis problem deals with determining the network(s) that satisfy the given flow requirements. A further constraint considers the determination of the network(s) that satisfy these terminal requirements at minimum total cost.

It is shown that although classical linear programming can, in theory, solve the above problems, in reality large scale network problems formulated using linear programming will lead to untractable computational requirements, especially for networks with large number of terminals. Thus, linear programming methods are usually impractical and other techniques to solve this problem are required.

Although a universal and alternate approach to linear programming has not yet been developed, several specialized methods have been given [8,11,13,14,15]. The most important recent contribution to the synthesis problem has been made by Mayeda [13] who using matrix representations gave a solution for the synthesis problem with uniform cost on all the communication channels. Mayeda also gave necessary conditions for the realizability of such networks. However, all of the methods given todate are special cases (uniform cost, unoriented networks, symetric considerations) and much work remains to be done to find a general solution to the synthesis problem. This study contributes further to the existing work in this area.

The study is divided into three chapters:-

The second and third sections of Chapter I present certain elementary concepts from set and graph theory that are used in section 1.4 to formally define a communication network. Section 1.5 differentiates between simultaneous transmission and time-shared communication networks; while section 1.6 introduces the analysis and synthesis problems. Section 1.7 offers a detailed presentation of the general analysis problem while section 1.8 introduces the constraints and variables of the synthesis problem. Sections 1.7 and 1.8 also give the details of linear programming formulations so that the reader can appreciate the computational difficulties inherent in such an approach. Chapter I, thus provides a general theoretical basis for the two chapters that follow, and also contains an introduction to network theory.

Many authors [3,4,5,6] have made contributions towards the design of networks by developing various simulation algorithms that are based on techniques for finding the shortest paths between pairs of terminals in a network. Chapter II of this study utilizes various shortest path techniques to develop some new synthesis algorithms. The "multiterminal shortest path" problem is solved in section 2.4 and the algorithm presented that implements the multiterminal problem is used as the basis for the synthesis algorithms that follow. Section 2.5 presents a new algorithm for the optimal synthesis of simultaneous transmission networks and section 2.6 presents a suboptimal synthesis algorithm for time-shared networks.

In this Chapter it is also shown that Floyd's Multiterminal Shortest Path Algorithm [5] is simply an extension of the elementary shortest path algorithm (from a single terminal to all others) and theorems 2.3.4 and 2.4.1 provide the basis for this result. The synthesis algorithms as well as the above mentioned theorems and results as presented in this study, are given for the first time, and are important and original contributions of this work.

In Chapter III, a procedure for synthesizing a time-shared communication network that exactly meets a priori given terminal requirements is presented. Section 3.2 contains some preliminaries. Section 3.3 contains both the algorithm that constructs such a network as well as the necessary conditions under which this can be accomplished. In section 3.4, some illustrative examples are presented and in section 3.5, the algorithm for constructing a communication network from some general (having negative capacities) network, that may arise in the synthesis procedure, is given.

The procedures described in Chapter III were originally suggested and sketched by Resh [14]. The proofs given by Resh were inadequate and unsatisfactory. In this work formal and new proofs of all these theorems and algorithms are given.

In addition to the refined theoretical development of Resh's work, the study also contains useful computer programs that implement these network synthesis procedures. These programs which were developed as part of this study are now resident on the Department of Communications computing installation at Shirley Bay. One of these programs has already been used by W.L. Hatton [17] for the analysis of a proposed satellite communication network.

It should be appreciated that some of the methods presented in this study have constraints, and therefore that the general solution relaxing these constraints still has not been found. It appears promising that more sophisticated shortest path techniques exist that could give better algorithms allowing the uniform cost restrictions in the procedures in Chapter III to be extended. These and other related problems should form the basis for further research in this area.

CHAPTER I

1.1 SUMMARY

This chapter presents the mathematical preliminaries that form the basis for the synthesis techniques given in Chapters II and III. Some set theory and graph theoretic concepts are reviewed and used to formally define a communication network. In this framework, definitions are given for both simultaneous transmission and time-shared communication networks. Finally, a brief discussion of some of the problems that arise in the analysis and synthesis of networks is presented. An explanation of the notation used in this study, is given in Appendix A.

1.2 MATHEMATICAL PRELIMINARIES

SET THEORY

Cartesian Product - If X and Y are two sets, then the set

$$X \times Y = \{ (x,y) \mid x \in X,\ y \in Y \} \tag{1.2.1}$$

is called the cartesian product of X and Y.

Relation - Any subset of X x Y is called a relation from X into Y and in particular, if X = Y = N, then any relation from N into N is called a relation on N.

Identity Relation - A relation in N, Δ_N, where

$$\Delta_N = \{ (i,i) \mid i \in N \} \tag{1.2.2}$$

is called the <u>identity relation</u> on N.

<u>Function</u> - A <u>function</u> h defined on some set X and taking on values in a set Y is denoted by:

$$h : X \to Y \tag{1.2.3}$$

<u>Restriction</u> - Given the function h and a set $D \subset X$, the function,

$$h^* : D \to Y \ni h^*(d) = h(d) \ \forall d \in D, \tag{1.2.4}$$

is called the <u>restriction</u> of h to D and it is denoted by: h/D.

1.3 <u>GRAPH THEORY</u>

In keeping with standard practice, an oriented <u>graph</u> G is defined as

$$G = (N,A) \tag{1.3.1}$$

where N is a non-empty point set (normally a finite set of points) and A is a relation on N. If $A = N \times N$, G is said to be <u>complete</u> while $A = N \times N - \Delta_N$ then G is said to be <u>quasicomplete</u>.

For a set of points $N_s \subset N$, a subgraph G_s is defined as

$$G_s = (N_s, A \cap (N_s \times N_s)). \tag{1.3.2}$$

If X is a non-empty proper subset of N, then the set S_X is called a <u>semicut</u> of G and $S_X \times S_X{}^c$ is called a cut of G. Then,

$$S_X = (X \times X^c) \bigcap A, \tag{1.3.3}$$

$$S_X \bigcup S_{X^c} = (X \times X^c) \bigcup (X^c \times X) \bigcap A. \tag{1.3.4}$$

Pictorally, the graph G is represented as follows. The elements of $N = \{1,2,\text{---},i,j,\text{---},n\}$ are points on the plane and are called nodes, centers or terminals; while the ordered pairs, $(i,j) \varepsilon A$ are called arcs, links or channels, and are directed line segments that originate at node i and terminate at node j. These line segments each bear an arrow head pointing from i to j; hence they are called directed arcs. An arc is called a loop if $i = j$.

From the representation of G, it is immediately apparent that a semicut S_X of a graph consists of all those arcs "emanating" from the set of nodes X and "entering" the set of nodes X^c. Removal of these arcs from the graph would destroy the connection from X to X^c. Since connections from X^c to X can exist, the set S_X is called a semicut as opposed to the set $S_X \bigcup S_{X^c}$ which is called a cut; that is, the removal from the graph of the arcs in the cut $S_X \bigcup S_{X^c}$ separates the graph into two disjoint subgraphs. Moreover, if the number of nodes in N is n, then the number of distinct semicuts in G is exactly the number of distinct proper non-empty subsets of N, that is,

$$\binom{n}{1} + \text{---} + \binom{n}{n-1} = 2^n - 2 = 2(2^{n-1} - 1) \tag{1.3.5}$$

Since there are exactly twice as many semicuts possible as there are cuts, the number of distinct cuts in G is $2^{n-1}-1$.

The notion of a quasicomplete graph is important in this study; and the following definitions are given to introduce this notion.

If $G = (N,A)$ is a quasicomplete graph, then for every non-empty proper subset X of N, it is obvious that $X \times X^c \subset A$, and thus

$$S_X = (X \times X^c) \cap A = (X \times X^c) \tag{1.3.6}$$

and

$$S_X \cup S_{X^c} = (X \times X^c) \cup (X^c \times X) \tag{1.3.7}$$

Each set of nodes, $N_k = \{n_1, n_2, ---, n_z\}$ where $p = n_1$, $q = n_z$ and where N_k is a non-empty subset of the nodes in the quasicomplete graph G, determines a set of arcs,

$$\Pi^k_{pq} = \{(n_i, n_{i+1}) \; / \; 1 < i < z-1\} \;, \; z \geq 2, \tag{1.3.8}$$

called the k^{th} <u>path</u> from p to q. Then a subpath Π^1_{uv} of Π^k_{pq} is,

$$\Pi^1_{uv} = \{(n_i, n_{i+1}) \; / \; u < i < y-1\} \subset \Pi^k_{pq} \tag{1.3.9}$$

where $1 \leq u < y \leq z$ and $v = n_y$.

To each path Π^k_{pq} there corresponds a set of arcs,

$$\Pi^k_{pp} = \Pi^k_{pq} \bigcup \{(q,p)\}, \tag{1.3.10}$$

called the <u>circuit</u> corresponding to the path Π^k_{pq}.

Observe that the set,

$$\begin{aligned}\Pi^k_{pq} &= \{(i,j) \in A \;/\; (j,i) \in \Pi^k_{pq}\} \\ &= \{(n_{i+1}, n_i) \;/\; 1 \leq i \leq z-1\}, \quad z \geq 2\end{aligned} \tag{1.3.11}$$

is the return path from q to p corresponding to the path Π^k_{pq}, while,

$$\begin{aligned}\overline{\Pi^k_{pq}} &= \{(p,q)\} \bigcup \Pi^k_{pq} \\ &= \{(i,j) \in A \;/\; (j,i) \in \Pi^k_{pp}\}\end{aligned} \tag{1.3.12}$$

is the circuit corresponding to $\overline{\Pi^k_{pq}}$ and that clearly,

$$\Pi^k_{pq} \bigcap \overline{\Pi^k_{pq}} = \emptyset, \tag{1.3.13}$$

$$\Pi^k_{pp} \bigcap \overline{\Pi^k_{pp}} = \emptyset. \tag{1.3.14}$$

Finally, if the quasicomplete graph G is also finite with n nodes, it is evident that the number of paths from any node p to any node q such that $p \neq q$ is,

$$\binom{n-2}{0}.0! + \binom{n-2}{1}.1! + \cdots + \binom{n-2}{n-3}\cdot(n-3)! + \binom{n-2}{n-2}\cdot(n-2)! \tag{1.3.15}$$

The following examples illustrate some of the graph theoretic concepts introduced above.

<u>Example 1.3.1</u> - The graph $G = (N,A)$ where $N = \{1,2,3,4,5\}$ and $A = \{(1,2), (1,4), (1,5), (2,1), (2,3), (2,4), (3,4), (4,5)\}$ is a finite oriented graph with the following representation.

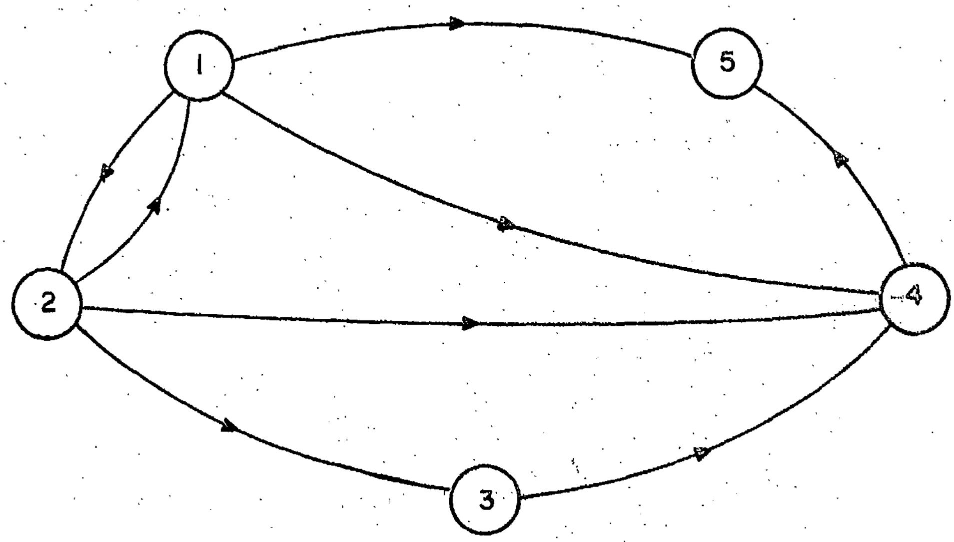

Figure 1.3.1

The semicut S_X corresponding to the non-empty proper subset $X = \{1,2,5\}$ of N can be found as follows,

$$X \times X^c = \{1,2,5\} \times \{3,4\} = \{(1,3),(1,4),(2,3),(2,4),(5,3),(5,4)\},$$

$$S_X = (X \times X^c) \bigcap A = \{(1,4),(2,3),(2,4)\}.$$

Now removing from G the arcs in S_{X^c} still leaves the arcs in S_X connecting X to X^c while removing the arcs in the cut,

$$S_X \bigcup S_{X^c} = \{(1,4),(2,3),(2,4),(4,5)\},$$

separates the graph G into two subgraphs, namely

$$G_1 = (\{1,3,5\}, \{(1,2), (1,5),(2,1)\}) = (N_1,A_1),$$

$$G_2 = (\{3,4\}, \{(3,4)\}) = (N_2,A_2).$$

From the original definition, it can be shown that both G_1 and G_2 are subgraphs of G. Putting $N_s = N_1$,

$$\begin{aligned} G_s &= (N_1, A \bigcap (N_1 \times N_1)) \\ &= (\{1,2,5\}, A \bigcap (\{1,2,5\} \times \{1,2,5\})) \\ &= (\{1,2,5\}, \{(1,2), (1,5), (2,1)\}) \\ &= G_1. \end{aligned}$$

Similarly it can be shown that G_2 is a subgraph of G.

Example 1.3.2 - The finite, quasicomplete, oriented graph G = (N,A) where N = {1,2,3,4} and A = {(1,2),(1,3),(1,4), (2,1),(2,3),(2,4),(3,1),(3,2),(3,4),(4,1),(4,2),(4,3)} has the following representation:

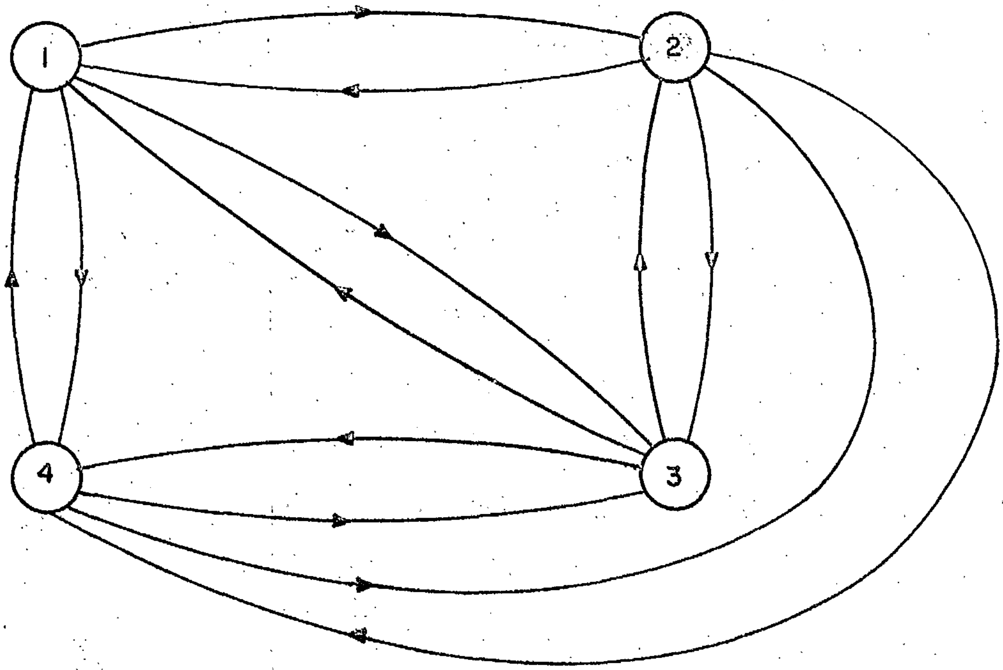

figure 1.3.2

The number of possible paths from a node p to another node q where $p \neq q$ and $p,q \in N$ is,

$$\binom{4-2}{0}\cdot 0! + \binom{4-2}{1}\cdot 1! + \binom{4-2}{2}\cdot 2! = 1+2+2 = 5.$$

For a given pair of nodes, $p = 1$ and $q = 2$, the five possible paths from 1 to 2 are

$$\Pi^{1}_{1,2} = \{(1,2)\},$$

$$\Pi^{2}_{1,2} = \{(1,3),\ (3,2)\},$$

$$\Pi^{3}_{1,2} = \{(1,4),\ (4,2)\},$$

$$\Pi^{4}_{1,2} = \{(1,3),\ (3,4),\ (4,2)\},$$

$$\Pi^{5}_{1,2} = \{(1,4),\ (4,3),\ (3,2)\}.$$

The circuit corresponding to $\Pi^{5}_{1,2}$ is,

$$\Pi^{5}_{1,1} = \Pi^{5}_{1,2} \bigcup \{(2,1)\} = \{(1,4),\ (4,3),\ (3,2),\ (2,1)\}$$

Observe that the "return" path corresponding to $\Pi^{5}_{1,2}$ is,

$$\overline{\Pi^{5}_{1,2}} = \{(2,3),\ (3,4),\ (4,1)\}$$

and that the circuit corresponding to $\overline{\Pi^{5}_{1,2}}$ is,

$$\overline{\Pi^{5}_{1,1}} = \{(1,2),\ (2,3),\ (3,4),\ (4,1)\}.$$

1.4 NETWORKS AND COMMUNICATION NETWORKS

In general a network $\mathbb{N}$ is defined as

$$\mathbb{N} = (G,\ w_1,\ w_2,\ \text{---}\ w_m), \tag{1.4.1}$$

where G is a finite, quasicomplete, oriented graph with n nodes. w_k is the k^{th} real-valued function defined on some set A, that is

$$w_k : A \rightarrow R; \quad k = 1,2,---,m. \tag{1.4.2}$$

In particular, a communication network is one in which the w_k's are non-negative real valued functions, namely,

$$w_k : A \rightarrow R^+ ; \quad k = 1,2,---,m. \tag{1.4.3}$$

The function w_k is usually called the "arc capacity" or "weighting function"; that is for each arc $(i,j) \varepsilon A$, $w_k(i,j)$ is referred to as the "weight" of arc (i,j), or the arc capacity.

The requirement that G be quasicomplete is by no means restrictive. That is, if the graph representing the actual network is not quasicomplete it is a simple matter to add to it the missing arcs, each with zero weight, so as to make it quasicomplete. In other words, it is always possible to represent any network by a quasicomplete graph.

If S_X is a semicut of G then for the network $\mathbb{N}$, the real number

$$\left| S_X \right|_{w_k} = \Sigma \quad w_k(i,j) \quad ; \quad \forall (i,j) \varepsilon S_X \tag{1.4.4}$$

is called the value of the semicut S_X with respect to the weighting function w_k. Whenever no confusion can arise, $|S_X|$ will be written instead of $|S_X|_{w_k}$.

The "sum" and the "difference" of two networks $\mathbb{N}_1 = (G,w_1)$ and $\mathbb{N}_2 = (G, w_2)$, say, is defined as:

$$(G,w_1) \pm (G,w_2) = (G, w_1 \pm w_2), \tag{1.4.5}$$

and, therefore, for any semicut S_X in G,

$$|S_X|_{w_1 \pm w_2} = |S_X|_{w_1} \pm |S_X|_{w_2}. \tag{1.4.6}$$

The following weighting functions will be used in this study:

$$w_1 = c: \ A \to R^+ \tag{1.4.7}$$

$$w_2 = f: \ A \to R^+ \tag{1.4.8}$$

$$w_3 = d: \ A \to R^+ \tag{1.4.9}$$

$$w_4 = t: \ A \to R^+ ; \ t(i,j) = \min\{|S_X|_c \ / \ (i,j) \varepsilon S_X\} \tag{1.4.10}$$

The function c, is the channel capacity function and c(i,j) represents the channel capacity of the channel (i,j). In a communication problem, c(i,j) is usually given in terms of bandwidth or as a bit rate. It represents the maximum speed at which a message may be transferred along the given channel (i,j).

The function f is the message rate or flow function and f(i,j) is the rate at which a message is actually being transmitted along arc (i,j). When subscripts are used $f_{pq}(i,j)$, means that a given amount of flow (see section 1.5) from p to q is taking place along the arc (i,j) and that its value is f_{pq} (i,j).

The function d is called the cost function. Usually d(i,j) is either the cost per unit capacity (for the synthesis problem) or it is the cost per unit flow (for the analysis problem).

The function t is called the terminal capacity function. Formally t(i,j) is the maximum flow that can be achieved between terminals i and j given that no other flows are introduced into the network. It is by the Ford Fulkerson [7] theorem equal to value of the minimum valued semicut separating i and j.

For computational purposes, the functions c,f,d and t will be represented by the n by n matrices C,F,D and T. Each matrix entry, c(i,j), f(i,j), d(i,j) and t(i,j) etc. represent the respective values of the functions as defined above. Since no loops are present at the nodes of a communication network, the diagonal elements of these matrices are not defined.

1.5 SCOPE OF THE STUDY

In this study, two large classes of communication networks will be discussed. These are,

1) Simultaneous transmission communication networks,
2) Time-shared communication networks.

In simultaneous transmission networks all communication centers are able to both transmit and receive messages at the same time. This, for example, is the mode of communications in the common telephone system. It is assumed that some form of channel selection or switching allows messages with different origins and destinations to pass along common links without interfering with each other. The flow of messages from terminal p to terminal q in a network is called the commodity with origin p and destination q. This commodity may be distributed among the paths that join p to q, and the value of the flow of this commodity on arc (i,j) is denoted by $f_{pq}(i,j)$. In addition, other independent message flows, say $f_{st}(i,j)$ could exist along the same arc at the same time, this leads then to what is called a multicommodity flow problem.

In general, for an n node communication network there are n(n-1) commodities, two for each node pair (one in each direction).

The total flow from p to q is given in such a network by

$$f^t_{pq} = \sum_{k \neq p} f_{pq}(p,k) = \sum_{k \neq q} f_{pq}(k,q); \quad \forall k \in N. \qquad (1.5.1)$$

and the total message flow for all commodities in arc (i,j) is,

$$f^t(i,j) = \sum_{p,q} f_{pq}(i,j); \quad \forall p \neq q \in N. \tag{1.5.2}$$

In contrast to the above, networks engaged in time-shared communications are networks, in which communication occurs between only a single pair of nodes in a particular interval of time. This system is used in time-shared computing installations. This is still a multicommodity situation and the commodities are not only distinct but also are transferred in non-overlapping intervals of time.

It follows that the total flow from p to q, f^t_{pq}, is again given by (1.5.1), and that this value is less than or equal to the terminal capacity, t(p,q). Furthermore the flow in a given arc (i,j) is simply $f_{pq}(i,j)$ and the capacity of arc (i,j) must be large enough to allow the maximum of the $f_{pq}(i,j)$'s to pass along that arc.

1.6 THE ANALYSIS AND SYNTHESIS OF COMMUNICATION NETWORKS

In the following two sections, the analysis of communication networks is discussed and the synthesis problem is introduced.

In the case where the capacity and cost functions c and d for the network $\mathbb{N} = (G,d,c,f,t)$ are known, $\mathbb{N}$ is obviously a physical entity. The analysis problem, then, consists of obtaining the message flow function f, and the terminal capacity function t.

The contrasting case is the one in which a network $\mathbb{N} = (G,d,c,t)$ is to be synthesized. That is, G, d and t are known and c is to be found. It is possible to have as a result of synthesis, networks having arc capacity functions that give terminal capacities that are all larger or at least as large as the apriori specified entries in t. These resulting networks are considered feasible (not necessarily optimal) solutions as the arcs have at least enough capacity to satisfy the terminal capacity requirements.

In all cases, t is called the terminal capacity requirement function and the entries in t will be called simply the terminal requirements.

1.7 THE ANALYSIS PROBLEM

In the analysis problem, a communication network $\mathbb{N} = (G,d,c,f,t)$ is given, and the network configuration G, the arc costs d (cost per unit flow) and the arc capacities c are known.

The obtaining of the message flow function f, constitutes the first analysis problem. It is obvious that for any type of communication network, several flow functions are feasible. Any flow function that assumes non-negative values that do not exceed the corresponding arc capacities, is said to be a feasible flow pattern. Among these flow patterns, there exists at least one that is called an optimal flow pattern.

The optimality may be based on maximum total network flow, on minimum network cost; on maximum flow at minimum cost. Thus several possibilities exist and for each one there is a set of constraints, that is for each case there is a function called the objective function that has to be optimized.

The following observations apply to all the cases considered in this study.

1) All values of flow in a communication network are non-negative (that is, greater than or equal to zero).

2) Flow is conserved at all nodes, hence, the sum total of message flow incident on a node is equal to the sum total flow emanating from that node.

3) The total flow in a given arc must not exceed the capacity of that arc at any time.

4) The objective function must be expressed in terms of the independent variables, that is, the entries in f.

5) Optimization implies that the flow function is to be evaluated so that the objective function takes either its maximum or minimum value. Since the

operations are addition, subtraction or multiplication, this optimum is well defined if the independent variables are bounded from above and below.

Case 1.7.1 - Single Commodity - Maximize Flow:

Given: A single commodity flow network where $N = \{s,1,2,---,t\}$ and $f_{st}(i,j)$, $(i,j) \in A$, is the flow from s to t of the single commodity on arc (i,j).

Required: To find the flow pattern that maximizes the total flow from s to t.

Solution: Knowing that $f_{st}(i,j)$ is the flow value on arc (i,j) and letting the maximum flow from s to t be v, the constraints may be stated as follows:

$$f_{st}(i,j) \geq 0 \quad ; \quad \forall\ (i,j) \in A, \tag{1.7.1}$$

$$\sum_{k \neq j} f_{st}(k,j) - \sum_{k \neq j} f_{st}(j,k) = \begin{cases} -v \text{ if } j=s, \\ 0 \text{ if } j \neq s,t, \\ v \text{ if } j=t; \ s,t \in N, s \neq t, \end{cases} \tag{1.7.2}$$

$$f_{st}(i,j) \leq c(i,j); \quad \forall (i,j) \in A. \tag{1.7.3}$$

Observe that (1.7.2) is simply the statement of the conservation of flows.

Since v is to be maximized, the objective function is z = v, and z is maximized subject to constraints

(1.7.1), (1.7.2) and (1.7.3), that is,

$$\text{Maximize } z = v \tag{1.7.4}$$

Note that all entries in f are bounded and that v is a function of some of the values in f.

<u>Case 1.7.2</u> - <u>Single Commodity - Minimize Cost:</u>

<u>Given</u>: A single commodity flow network where the flow from s to t is v.

<u>Required</u>: To find the flow pattern that minimizes the cost.

<u>Solution</u>: Remembering that the "throughput" v is no longer a variable but of fixed value and assuming that a feasible flow pattern corresponding to v exists then the constraints are given by equations (1.7.1); (1.7.2); (1.7.3) above.

The objective function z is now dependent upon the arc costs. To arrive at a solution for f it is necessary to minimize z,

$$\text{Minimize } z = \sum_{(i,j)} f_{st}(i,j).d(i,j);\ \forall(i,j)\varepsilon A. \tag{1.7.5}$$

Obviously, the maximum throughput at minimum cost can be obtained for a given network by adopting the above approach. That is, by first evaluating the maximum flow v using case 1.7.1 and then finding the pattern that minimizes the cost, given that the throughput is v as outlined in case 1.7.2.

Case 1.7.3 - Simultaneous Transmission - Multicommodity - Maximum Flow:

Given: An n node network with n(n-1) commodities.

Required: To find the flow pattern that maximizes the sum of all the commodity flows.

Solution: Letting v_{pq} be the undetermined commodity flow value for nodes p and q ($p \neq q$, $p,q \varepsilon N$), the constraints are,

$$f_{pq}(i,j) \geq 0 \; ; \quad \forall (i,j) \; \varepsilon \; A; \quad \forall \; p,q \varepsilon N, \; p \neq q, \tag{1.7.6}$$

$$\sum_{k \neq j} f_{pq}(k,j) - \sum_{k \neq j} f_{pq}(j,k) = \begin{cases} -v_{pq}, & \text{if } j=p, \\ 0, & \text{if } j \neq p,q, \\ v_{pq}, & \text{if } j=q, \end{cases} \tag{1.7.7}$$

$$\forall \; p,q \varepsilon N, \; p \neq q,$$

$$\sum_{p,q} f_{pq}(i,j) \leq c(i,j); \; \forall (i,j) \; \varepsilon \; A; \; \forall p,q \varepsilon N, \; p \neq q. \tag{1.7.8}$$

To arrive at a solution it is necessary to maximize z,

$$\text{Maximize } z = \sum_{p,q} v_{pq} \; ; \; \forall \; p,q \varepsilon N, \; p \neq q. \tag{1.7.9}$$

Case 1.7.4 - Simultaneous Transmission - Multicommodity - Minimum Cost :

Given: An n node network with n(n-1) commodities where the magnitude of each commodity of flow is a constant $v_{pq} \geq 0 \quad p \neq q \in N$.

Required: To find the flow pattern that minimizes the total network cost.

Solution: Equations (1.7.6), (1.7.7) and (1.7.8) are the system constraints for this case as well, where v_{pq} is no longer a variable quantity.

The objective function, however, changes to take the arc costs into account. Then, from (1.5.2),

$$\text{Minimize } z = \sum_{(i,j)} d(i,j).f^t(i,j); \; \forall (i,j) \varepsilon A$$

$$= \sum_{(i,j)} d(i,j). \sum_{p,q} f_{pq}(i,j) \,, \qquad (1.7.10)$$

$$\forall \; p,q \varepsilon N, \; p \neq q,$$

to find the minimum total cost of the network.

Case 1.7.5 - Time-shared Multicommodity - Maximum Flow:

Given: An n node network with n(n-1) commodities.

Required: To find the flow pattern that maximizes the sum of all the commodity flows.

Solution: Letting v_{pq} be the undetermined flow value for nodes p and q ($p,q\varepsilon N$, $p\neq q$), the constraints are

$$f_{pq}(i,j) \geq 0;\ \forall(i,j)\varepsilon A;\ \forall\ p,q\varepsilon N,\ p\neq q, \tag{1.7.11}$$

$$\sum_{k\neq j} f_{pq}(k,j) - \sum_{k\neq j} f_{pq}(j,k) = \begin{cases} -v_{pq}, & \text{if } j\neq p, \\ 0, & \text{if } j=p,q, \\ v_{pq}, & \text{if } j=q, \end{cases} \tag{1.7.12}$$

$$\forall\ p,q\varepsilon N,\ p\neq q,$$

$$f_{pq}(i,j) \leq c(i,j);\ \forall\ (i,j)\ \varepsilon\ A;\ \forall\ p,q\varepsilon N,\ p\neq q. \tag{1.7.13}$$

Observe that the constraints for this problem are identical to those of the Simultaneous Transmission Problem in Case 1.7.3 except for (1.7.13). In the time-shared case only one commodity of flow may appear in the arc (i,j) at any instant in time, hence $f_{pq}(i,j)$ could take on a value up to the value of c(i,j). In Case 1.7.3 all commodities appear simultaneously in (i,j) hence the summation in (1.7.8).

Subject to the constraints (1.7.11), (1.7.12) and (1.7.13) the solution is found by maximizing z, namely

$$\text{Maximize } z = \sum_{p,q} v_{pq}\ ;\ p,q\varepsilon N, p\neq q \tag{1.7.14}$$

Case 1.7.6 - Time-Shared - Multicommodity - Minimum Cost:

Given: An n node network with n(n-1) commodities, where the $v_{pq} \geq 0 \ni p \neq q \in N$ are constants.

Required: To find the flow pattern that minimizes the total network cost.

Solution: The constraints are identical to those of Case 1.7.5. The solution is found once again by minimizing z,

$$\text{Minimize} \quad z = \Sigma\, d(i,j).f^*(i,j); \; \forall(i,j)\varepsilon A, \tag{1.7.14}$$

where $f^*(i,j)$ is the average flow value expected in arc (i,j). If each of the commodities has an "equal chance" to utilize arc (i,j), then,

$$f^*(i,j) = \frac{\Sigma\, f_{pq}(i,j)}{n(n-1)} \; ; \; \forall(i,j)\varepsilon A; \; p,q\varepsilon N, \; p\neq q. \tag{1.7.15}$$

Comments

(1) Closer examination of the above cases shows that the complexity of these solutions increases very rapidly as the number of nodes in the network is increased. For example, for an n node network in Case 1.7.3, the number of unknown variables as well as the number of constraints is $n(n-1)(n^2-n+1)$. For case 1.7.5 the number of unknowns is $n(n-1)(n^2-n+1)$ and the number of constraints is $2n^2 \cdot (n-1)^2$. Since the computing time for solution using

a linear program is dependent upon the number of constraints as well as the number of unknowns, the size of the task grows almost without bound as the number of nodes increases. Hence the use of linear programs to solve network problems is limited by the computer time the analyst can afford to spend on a given problem. This shortcoming is what has prompted the "modern" network theorists to develop more efficient graph theoretic based algorithms to obtain solutions instead of using linear programming. Although only some of these problems have been solved this way, the ones that are give the network analyst some useful tools for modelling larger network problems. Some of these algorithms are presented in this study.

(2) The second, analysis problem requires that the terminal capacity function be evaluated. It has been shown that t is dependent upon another function, namely, the arc capacity function, that is,

$$t(p,q) = \min \{ |S_X|_c \ / \ (p,q) \in S_X \} \tag{1.4.10}$$

Thus the minimum valued semicut separating nodes p and q, has a value equal to the maximum possible flow from p to q. This is exactly case 1.7.5. Then $t(p,q) = v_{pq}$ and t is found using linear programming.

(3) The theorem that originally solved the analysis problem for finding t, was formulated by Ford and Fulkerson [7] and has subsequently become the central theorem in network theory. Since a formal definition (1.4.10) of the terminal capacity function is essential for the network synthesis problem, Ford and Fulkerson's "Max-Flow, Min-Cut" theorem is stated (without proof). In addition, some extensions are presented as these analytic tools are useful in analyzing networks once they have been synthesized.

Theorem 1.7.1 - Maximum-Flow, Minimum-Cut: [6,7,9]

For any network $\mathbb{N} = (G,c,t)$ where G is an n node, finite, quasicomplete, oriented graph and where c is the capacity function defined on the arcs of G, the maximum flow from some terminal p to another terminal q, called the terminal capacity t(p,q), is equal to the minimum valued semicut containing (p,q). (see equation (1.4.10)).

The direct result of this theorem is that algorithms can and have been formulated to solve the maximum flow problem using methods other than linear programming. The use of this theorem is illustrated in the following example.

Example 1.7.1 - Find $t(1,3)$.

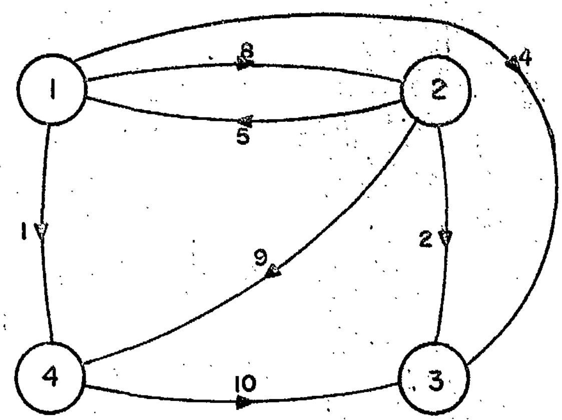

figure 1.7.1

Given the net, $\mathbb{N} = (G,c,t)$, where $G = (N,A)$ is quasicomplete, finite and oriented, $N = \{1,2,3,4\}$ and $A = N \times N - \Delta_N$; the capacity entries are noted beside the corresponding arcs in figure 1.7.1 and in the matrix C.

$$C = \begin{bmatrix} * & 8 & 4 & 1 \\ 5 & * & 2 & 4 \\ 0 & 0 & * & 0 \\ 0 & 0 & 10 & * \end{bmatrix}$$

In order to find $t(1,3)$, all semicuts S_X must be examined in which $1 \varepsilon X$ and $3 \varepsilon X^c$. Then the values of these semicuts are computed and $t(1,3)$ takes on the minimum of these values. Thus

$$X_1 = \{1\}, \quad |S_{X_1}|_c = |\{1\} \times \{2,3,4\}| = 8+4+1 = 13$$

$$X_2 = \{1,2\}, \quad |S_{X_2}|_c = |\{1,2\} \times \{3,4\}| = 4+1+2+9 = 16$$

$$X_3 = \{1,4\}, \quad |S_{X_3}|_c = |\{1,4\} \times \{2,3\}| = 8+4+0+10 = 22$$

$$X_4 = \{1,2,4\}, \quad |S_{X_4}|_c = |\{1,2,4\} \times \{3\}| = 4+2+10 = 16$$

Then $t(1,3) = 13$, the minimum valued semicut. Q.E.D.

Comments

The number of semicuts that must be tested for any terminal pair is,

$$\binom{n-2}{0} + \text{---} + \binom{n-2}{n-2} = 2^{n-2}, \tag{1.7.16}$$

where n is the total number of nodes in the network. Again it is evident that as n increases, the number of semicuts that must be examined increases rapidly. To overcome this inefficiency, many authors have developed labelling algorithms that locate these minimum cuts quickly. A familiar one is one that T.C. Hu [9] developed, which is based on the theorem of Ford-Fulkerson [7] . This algorithm is presented below and then an example given to illustrate its application.

In the Algorithm 1.7.1 presented below, for an n node network $\mathbb{N}$, where G is a finite, oriented quasicomplete graph, $N = \{p,1,2,---,n-2,q\}$ and the arc capacities $c(i,j)$ are given, the maximum flow from p to q is found. This corresponds by definition to the terminal capacity $t(p,q)$.

Algorithm 1.7.1:

A. Labelling Routine -

1. a) Initially, all nodes are unlabelled and unscanned and flows in the arcs are zero. Labels are of the form "$(j,\pm,\ \varepsilon(i))$" where this label corresponds to the i^{th} node.

   b) Label node p with $(p,+,\varepsilon(p) = \infty)$. Now p is labelled and unscanned while all other nodes are unlabelled and unscanned.

2. Choose any labelled, unscanned node i.

   a) If for some unlabelled neighbour of j where $c(j,i) > 0$, $j \neq i$, there is a flow $f(j,i) > 0$, then label j by $(i,-,\varepsilon(j))$ where $\varepsilon(j) = \min\{\varepsilon(i), f(j,i)\}$.

   b) If for some unlabelled neighbour of j where $c(i,j) > 0$ and $f(i,j) < c(i,j)$ then label j by $(i,+,\varepsilon(j))$ where $\varepsilon(j) = \min\{\varepsilon(i), c(i,j) - f(i,j)\}$.

   Now j is labelled but unscanned. Do this for all such neighbours of i.

   Now change the label on i by encircling the "+" or "-" sign. Node i is now labelled and scanned.

3. Repeat step 2 until either,

a) q is labelled then go to routine B or

b) no more nodes can be labelled and then stop.

B - Augmentation Routine -

1. let $z = q$.

2. a) If for node z the label is $(k,\pm,\varepsilon(z))$ then increase $f(k,z)$ by $\varepsilon(q)$.

b) If for node z the label is $(k,-,\varepsilon(z))$ then decrease $f(k,z)$ by $\varepsilon(z)$.

3. a) If $k = p$, erase all labels and go to step A (2).

b) If $k \neq p$, let $z = k$ and go to step 2.

Observe that upon termination at step A-3b), the terminal capacity, $t(p,q)$ is given by:

$$t(p,q) = \sum_j f(p,j) = \sum_j f(j,q) \; ; \; \forall \; j \neq p,q \in N \qquad (1.7.17)$$

and that the minimum semicut S_X is found by placing the labelled nodes into the set X.

Also note that by executing this algorithm $n(n-1)$ times for all terminal pairs, the terminal capacity function in the form of its matrix of values is found.

The following example demonstrates this algorithm.

Example 1.7.2

The network configuration and constants are indicated in figure 1.7.2. In the notation "x,y" beside each arc; the x represents the capacity of that arc and the y represents the flow in that arc. Then our initial configuration without labels (except for p) and without flows is:

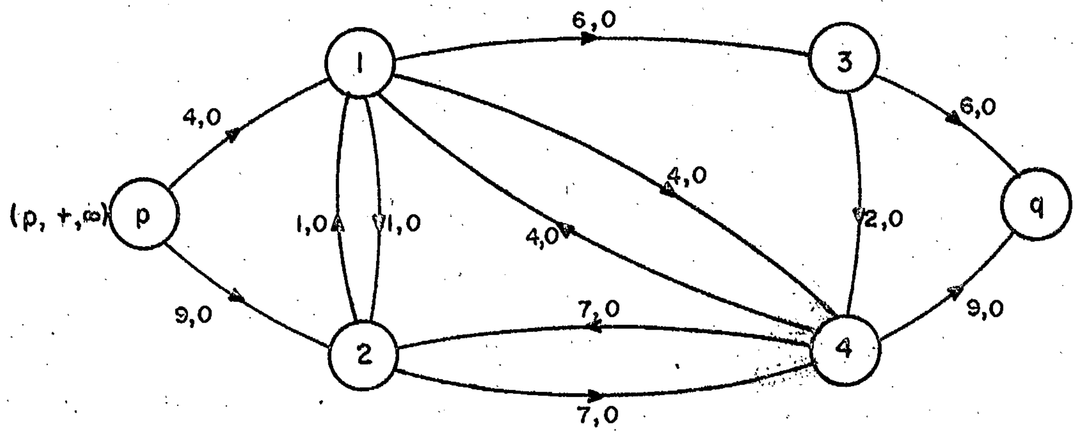

figure 1.7.2

Starting with routine A; nodes 1 and 2 are selected and labelled as neighbours of node p, the only labelled node at this step. Consequently, node p has been scanned.

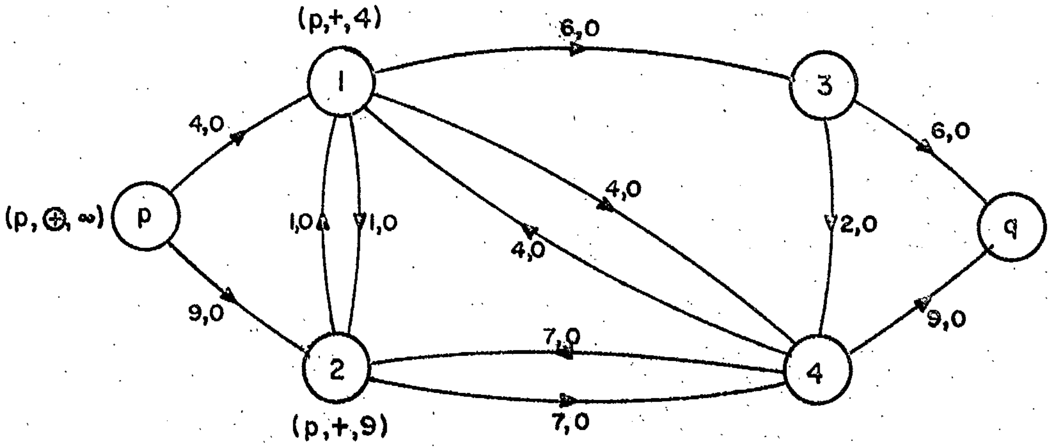

figure 1.7.3

This procedure is continued until "breakthrough" to q has been achieved.

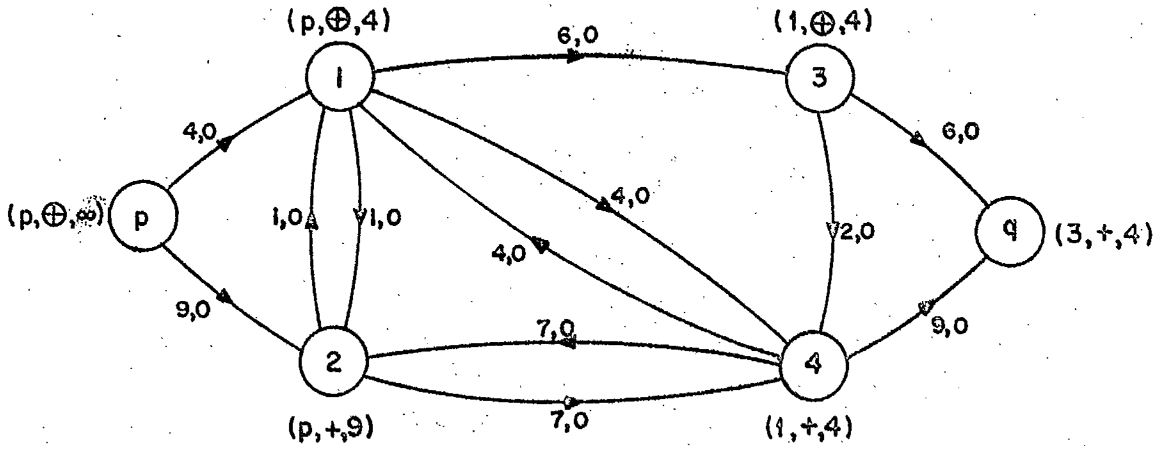

figure 1.7.4

Now that node q has been labelled, flows are changed along the path from p to q using Routine B and then the labels are all erased.

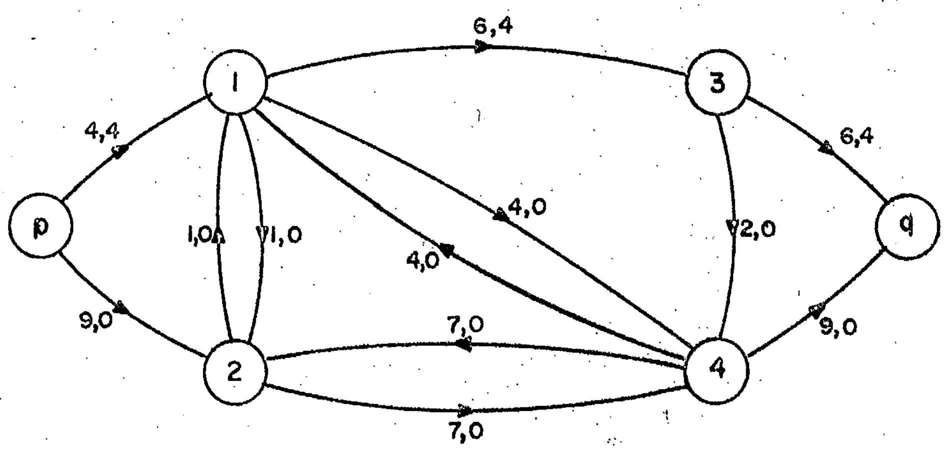

figure 1.7.5

Using Routine A again a "breakthrough" is found to node q and the following labelled configuration is obtained.

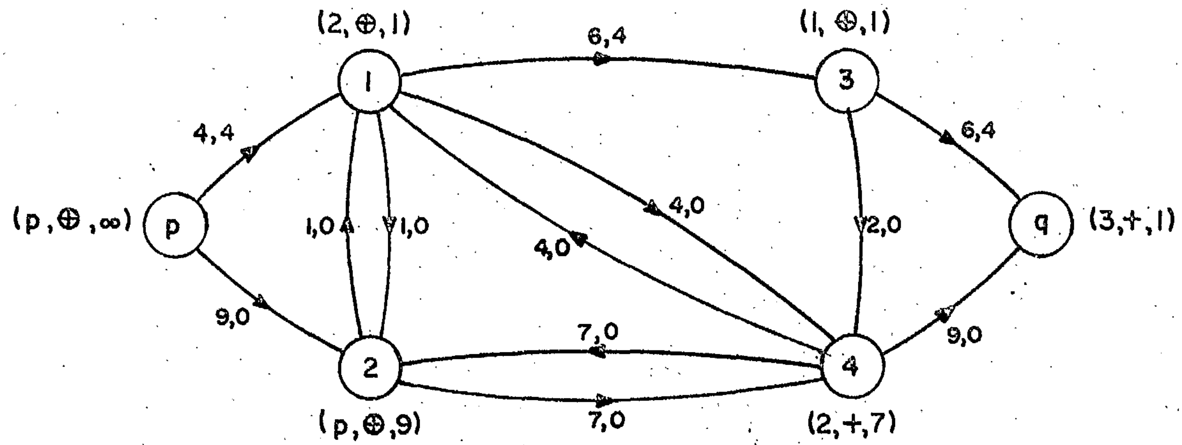

figure 1.7.6

Routine B gives the next flow pattern:

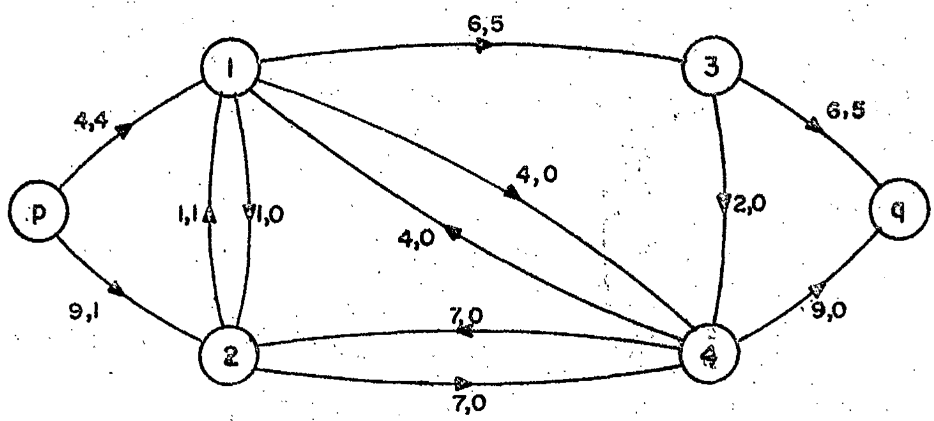

figure 1.7.7

Executing Routine A once more gives:

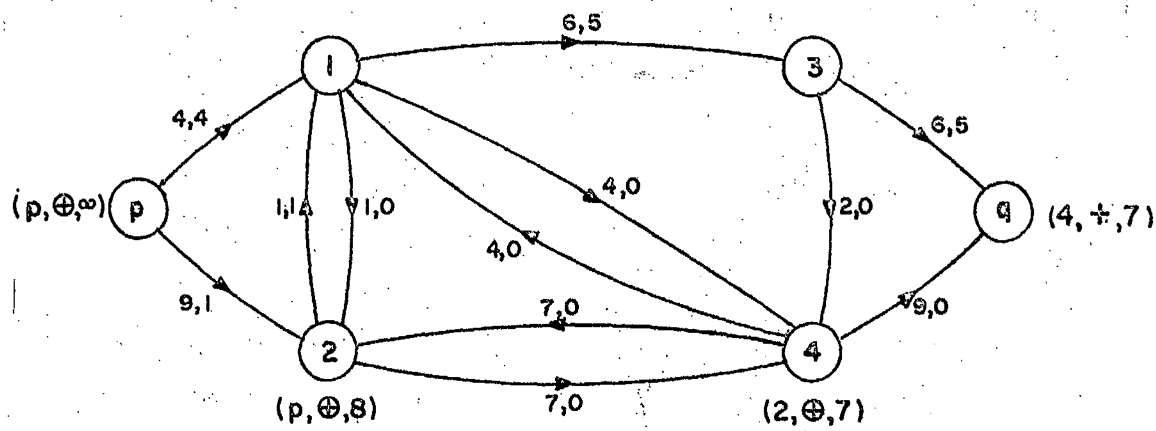

figure 1.7.8

The resultant flow pattern is:

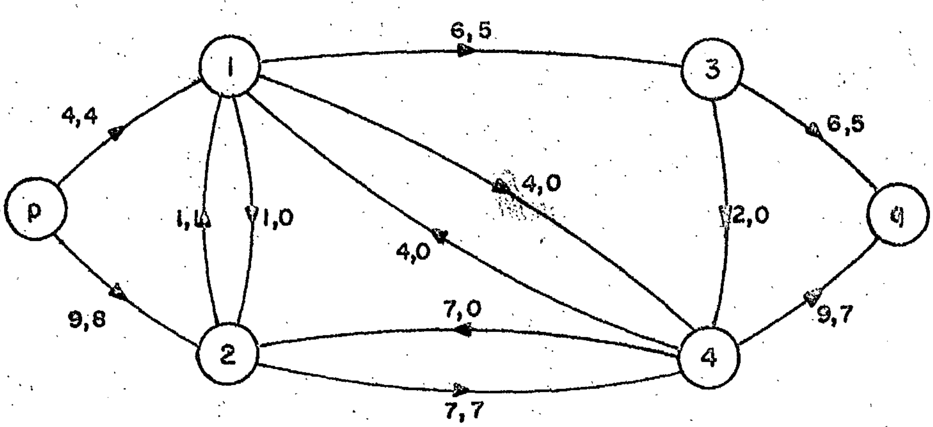

figure 1.7.9

Trying to label once more does not succeed, we have:

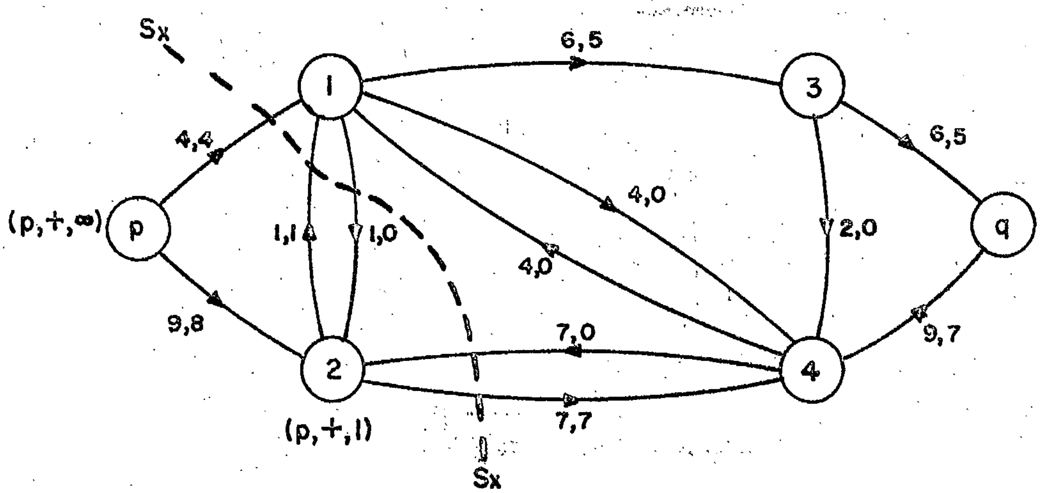

figure 1.7.10

Observe that the minimum valued semicut is S_X where X = {p,2} (elements of X are the labelled nodes that remain at termination). Then $\left|S_X\right|_c$ = c(p,1) + c(2,1) + c(2,4) = 4+1+7 = 12 and t(p,q) = 12. This can be varified by calculating that the sum of the flows leaving p equals the sum of the flows arriving at q which is again 12.

1.8 THE SYNTHESIS PROBLEM

In the synthesis problem, it is required to construct the communication network $\mathbb{N} = (G,d,c,t)$ where the network configuration G is an n node, finite, oriented, quasicomplete graph. The costs $d(i,j)$ and the terminal capacities $t(i,j)$ are also known. The solution must find the values of the capacity function on all arcs in G, so that t is satisfied; ∴ at the same time the network topology is found. Thus in the analysis problem, it is required to find t, knowing c, but in the synthesis problem c must be found given t.

A first attempt to solve this problem might be to construct an arc for each entry of the terminal capacity function, that is, set $c(i,j) = t(i,j); \forall(i,j) \in A$. The resultant network contains links between all possible pairs of nodes. This satisfies the simultaneous transmission problem as all nodes can communicate with all other nodes. at the same time, consequently, the time-shared case is certainly satisfied. However, is it an optimal cost solution? At a glance it is obvious that it isn't. If arc costs are not uniform then a given requirement, can certainly be satisfied by using a route that may be cheaper than the direct route. In the time-shared case, the quasicomplete configuration allocates a dedicated line to one terminal pair. This results in under-utilization as the line is active only a very small fraction of the time, remaining idle for the rest of the time. These observations suggest that optimal capacity assignment requires thoughtful formulation.

If the object is to satisfy the requirements at minimum total network cost, the most obvious method of solution is linear programming. As in the analysis problem, the set of constraints and the objective function to be optimized is given for the time-shared case.

Furthermore each t(p,q) in t, represents a commodity of flow that is required to flow from p to q. Then the flow pattern in the network for each commodity must satisfy these requirements without violating the arc capacities and at the same time must be such as to minimize the total network cost.

Only multicommodity problems are considered here.

Case 1.8.1 - Simultaneous Transmission - Minimum Cost - Synthesis:

Given: An n node network with known costs d and known requirements t.

Required: To find the capacity function c that satisfies the requirements at minimum total network cost for the simultaneous transmission problem.

Solution: Knowing that all the network flows are positive, that flow at the nodes obeys the conservation laws and that the flow in each arc must be less than the capacity of that arc, the constraints are,

$$f_{pq} \geq 0; \forall\ (i,j) \in A; \forall\ p,q\in N,\ p\neq q, \tag{1.8.1}$$

$$\sum_{k\neq j} f_{pq}(k,j) - \sum_{k\neq j} f_{pq}(j,k) = \begin{cases} -t(p,q), & \text{if } j=p \\ 0, & \text{if } j\neq p,q \\ t(p,q), & \text{if } j=q, \end{cases} \tag{1.8.2}$$

$$\forall\ p,q\in N,\ p\neq q,$$

$$c(i,j) - \sum_{p,q} f_{pq}(i,j) = 0; \quad \forall(i,j)\in A; \forall p,q\in N,\ p\neq q. \tag{1.8.3}$$

An optimal solution is found by minimizing z.

$$\text{Minimize} \quad z = \sum_{(i,j)} d(i,j).c(i,j) \quad \forall(i,j)\in A. \tag{1.8.4}$$

<u>Case 1.8.2</u> - <u>Time-Shared - Minimum Cost - Synthesis</u>:

<u>Given</u>: An n node network with known costs d and known requirements t.

<u>Required</u>: To find the capacity function that satisfies the time-shared requirements at minimum network cost.

<u>Solution</u>: The constraints for this problem are,

$$f_{pq}(i,j) \geq 0; \forall(i,j)\in A, \forall p,q\in N,\ p\neq q, \tag{1.8.5}$$

$$\sum_{k \neq j} f_{pq}(k,j) - \sum_{k \neq j} f_{pq}(j,k) = \begin{cases} -t(p,q), & \text{if } j=p, \\ 0, & \text{if } j \neq p,q, \\ t(p,q), & \text{if } j=q, \end{cases} \quad (1.8.6)$$

$$\forall\ p,q \varepsilon N,\ p \neq q,$$

$$c(i,j) - f_{pq}(i,j) = 0;\ \forall (i,j) \varepsilon A;\ \forall\ p,q \varepsilon N,\ p \neq q. \quad (1.8.7)$$

Subject to the above constraints

$$\text{Minimize } z = \sum_{(i,j)} d(i,j).c(i,j);\ \forall (i,j) \varepsilon A. \quad (1.8.8)$$

Comment

The values of capacity generated by these solutions are, in general, non-integers. Since capacity rental is usually quantized, implementation of simulation results requires that these non-integers be rounded off to their next largest integer. Thus, excess capacity is left in the system and the network cost is increased, that is, the final result is usually suboptimal even when linear programming is used.

1.9 CONCLUSIONS

In the cases presented in section 1.8, the commodity flows $f_{pq}(i,j)$, as well as the capacities $c(i,j)$, are obtained. That is, each linear program solves for a large number of unknowns such as flows, that are not required in the network synthesis cases. Moreover for large n, the number of unknowns and the number of constraints is unmanageably large as has already been pointed out in two of the analysis cases. This leads to the conclusion that methods more efficient than linear programming must be found to arrive at practical solutions quickly and efficiently.

It was pointed out that for the analysis problem certain labelling techniques can be applied. For the multicommodity synthesis, since no such methods exist to-date, it is concluded that one must settle for some sub-optimal synthesis procedures. In Chapter II of this study, some shortest path techniques are presented that tend to minimize total network cost. In Chapter III, a synthesis procedure is developed that exactly allows the determination of c, given the terminal requirements and the arc constraints.

CHAPTER II

2.1 SUMMARY

This chapter develops methods for the synthesis of simultaneous transmission and time-shared communication networks using various shortest path techniques. The need for computationally feasible methods was pointed out in Chapter I, where it was shown that due to the presence of huge numbers of constraints and unknowns, linear programming formulations for large network problems leads to difficult and often impossible computational problems.

First certain path definitions are given, then various shortest path algorithms are developed based on these shortest path notions. These algorithms permit readily computable solutions to be found for these synthesis problems. The simulation programs that implement these algorithms for the simultaneous transmission and the time-shared cases are given in appendices B and C.

2.2 MATHEMATICAL PRELIMINARIES

PATH DEFINITIONS

In Chapter I, the k^{th} path, Π_{pq}^{k}, was defined to be the sequence of arcs that join terminal p to terminal q. Here $d(\Pi_{pq}^{k})$ is defined to be the length of the path Π_{pq}^{k} and it is given as the sum of the "lengths" of the arcs that are in that path. In the synthesis procedures developed in this chapter, these lengths are the arc costs, namely, the d(i,j). Then,

$$d(\Pi^k_{pq}) = \sum_{(i,j)} d(i,j) \; ; \forall (i,j) \in \Pi^k_{pq} . \qquad (2.2.1)$$

Thus, the <u>shortest path</u> $\hat{\Pi}_{pq}$ is defined to be the one whose length is smallest among all m paths joining p to q. That is,

$$d(\hat{\Pi}_{pq}) = \underset{k}{\mathrm{MIN}} \{d(\Pi^k_{pq})\} , \; k = 1, 2, ---, m. \qquad (2.2.2)$$

These definitions are illustrated in the following example.

<u>Example 2.2.1</u> -

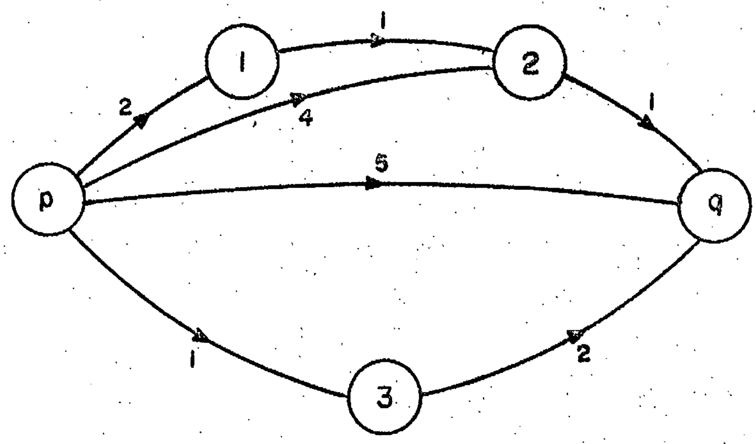

figure 2.2.1

Consider the network shown in figure 2.2.1 with arc costs (lengths) as indicated. By inspection, the possible paths and their lengths are:

$$\Pi^1_{pq} = \{(p,1),\ (1,2),\ (2,q)\}; \qquad d(\Pi^1_{pq}) = 4,$$

$$\Pi^2_{pq} = \{(p,2),\ (2,q)\}\ ; \qquad d(\Pi^2_{pq}) = 5,$$

$$\Pi^3_{pq} = \{(p,q)\}\ ; \qquad d(\Pi^3_{pq}) = 5,$$

$$\Pi^4_{pq} = \{(p,3),\ (3,q)\}\ ; \qquad d(\Pi^4_{pq}) = 3.$$

Therefore the shortest path is given by -

$$d(\hat{\Pi}_{pq}) = \underset{k}{\text{MIN}}\ \{d(\Pi^k_{pq})\},\ k = 1,\ 2,\ 3,\ 4$$

$$= 3$$

so that,

$$\hat{\Pi}_{pq} = \Pi^4_{pq} = \{(p,3),(3,q)\}\ .$$

In communication problems all arc costs are non-negative, and the minimum in (2.2.2) exists. However, in many general network problems where these lengths may be negative, further constraints must be placed on the system so that all shortest paths can be well defined. That is, the general problem for all circuits Π^k_{pp} where $p \in N$ and $k = 1, 2, ---, m$, must satisfy

$$d(\Pi^k_{pp}) \geq 0.$$

If for some circuit Π^k_{pp}, it should happen that $d(\Pi^k_{pp}) < 0$, then this circuit is called a <u>negative circuit</u>. This means that if $(i,j) \in \Pi^k_{pp}$ and if Π^h_{uv} $(u \neq v \neq p)$ is a path where $(i,j) \in \Pi^h_{uv}$, then $\hat{\Pi}_{uv}$ is not well defined, that is, the minimum is an undefined negative number.

The detection of negative circuits is important in network problems as their absence is one check of the validity of a shortest path computation.

2.3 THE SHORTEST PATH FROM A GIVEN NODE TO ALL OTHER NODES IN A NETWORK

Given the network $\mathbb{N} = (G,d)$ where d is the arc cost function, and it is required to find the shortest paths from a given node p to all other nodes, $1,2,---,n-1$ in this network; algorithm 2.3.1 solves this problem using a labelling procedure that was first formulated by Ford and Fulkerson [7].

<u>Algorithm 2.3.1 - (Shortest Path Algorithm)</u>

<u>Step 1)</u> - Assign to all nodes i labels of the form

$[\cdot, d(\Pi^*_{pi})]$ where $d(\Pi^*_{pp}) = 0$, and $d(\Pi^*_{pi}) = \infty$, $i \neq p$. $d(\Pi^*_{pi})$ is called an "intermediate shortest path length" and Π^*_{pi} is called an "intermediate shortest path". (Initially the shortest paths to all nodes $i \neq p$ are assigned infinite length).

Step 2) - Find an arc (i,j) for node $j, j \neq i$, such that,

$$d(\Pi^*_{pi}) + d(i,j) < d(\Pi^*_{pj}). \tag{2.3.1}$$

When such an arc is found, put $d(\Pi^*_{pj}) = d(\Pi^*_{pi}) + d(i,j)$ and rewrite the label for node j to read, $[i, d(\Pi^*_{pj})]$. Repeat this step until labels can no longer be changed; at this point terminate. The intermediate shortest paths have become the desired shortest paths and for all nodes $j \neq p$, $d(\hat{\Pi}_{pj}) = d(\Pi^*_{pj})$.

Step 3) - To identify the nodes in the shortest paths from node p to some node $j \neq p$:

a) Put $k = j$.

b) Identify i from the label $[i, d(\Pi_{pk})]$ on node k. Then i is a node in the shortest path $\hat{\Pi}_{pj}$. If i does not exist there is no shortest path from p to j.

c) Put $k = i$. If $k = p$ then terminate, otherwise return to 3-b).

The proof that algorithm 2.3.1 finds the shortest path from a given node p to all other nodes and that it terminates in a finite number of steps is established in the following lemmas and theorems [8]. The lemmas are stated while the theorems are proved.

THEOREM 2.3.1 - Step 2 of Algorithm 2.3.1 terminates after a finite number of labellings.

PROOF: For any node $j \neq p$, $d(\Pi^*_{pj})$ is either decreased in value or unchanged. Thus the magnitude of the intermediate shortest path for node j is bounded from above by the initial value.

Due to the nature of the algorithm, termination in step 2 occurs when some $d(\Pi^*_{pj})$ can no longer be reduced in value. It only remains then to show that for all nodes $j \neq p$; $d(\Pi^*_{pj})$ has a lower bound, that is, that the labels cannot be reduced indefinitely.

If there exists some semicut S_X whose value corresponds to infinity where $p \in X$, it is clear that for all $j \in X^c$, $d(\Pi^*_{pj}) = \infty$, that is, all shortest paths from p to j have infinite value. In such cases, the algorithm certainly is not able to find some arc that reduces $d(\Pi^*_{pj})$ to a finite value. These $d(\Pi^*_{pj})$ remain upper bounded, and are not relabelled, consequently, they cannot be reduced indefinitely and they do not affect termination.

Since all the arc lengths (costs) are positive, that is, $d(i,j) \geq 0$ and initially $d(\Pi^*_{pi}) = \infty$ then $d(\Pi^*_{pj})$ is always either a finite non-negative number or an arbitrarily large one. This places a lower bound on $d(\Pi^*_{pj})$ of zero and the theorem is proved.

Observe that the lower bound on $d(\Pi^*_{pj})$ implies that negative circuits cannot exist. This is true for the networks considered in this chapter, since negative arc costs are not considered.

<u>LEMMA 2.3.2</u> - If at termination, the label of node k, namely $d(\Pi^*_{pk})$ is finite, then a node i which is on the path Π^*_{pk} will be found at each iteration of step 3-b) of the algorithm.

From Theorem 2.3.1 and Lemma 2.3.1,

<u>LEMMA 2.3.2</u> - Algorithm 2.3.1 terminates in a finite number of steps.

Finally, it is required to show that the shortest paths are indeed found in step 3 of the algorithm.

<u>THEOREM 2.3.2</u> - At termination of Algorithm 2.3.1, the path Π^*_{pq} found in step 3 is the shortest path from p to q.

PROOF: On termination of the algorithm, some path Π^*_{pq}, $q \neq p$ is found using step 3 and its value, $d(\Pi^*_{pq})$ is found in step 2. Suppose that some other path Π^k_{pq}, $\Pi^k_{pq} \neq \Pi^*_{pq}$, exists that is "shorter" than Π^*_{pq} that is $d(\Pi^k_{pq}) < d(\Pi^*_{pq})$. Then

$$d(\Pi^k_{pq}) = \sum_{(i,j)} d(i,j) < d(\Pi^*_{pq});\forall\ (i,j) \in \Pi^k_{pq} \qquad (2.3.4)$$

But the algorithm has terminated and no more relabelling can occur; thus equation (2.3.1) cannot be satisfied and it follows that,

$$d(\Pi^*_{pi}) + d(i,j) \geq d(\Pi^*_{pj});\ \forall(i,j) \in \Pi^k_{pq}. \qquad (2.3.5)$$

An equation of the form (2.3.5), can be written for each $(i,j) \in \Pi^k_{pq}$, and substituting $d(\Pi^*_{pp}) = 0$ gives,

$$\sum_{(i,j)} d(i,j) \geq d(\Pi^*_{pq});\forall(i,j) \in \Pi^k_{pq} \qquad (2.3.6)$$

However, equations (2.3.4) and (2.3.6) contradict each other and therefore Π^*_{pq} must be the required shortest path, that is, $\hat{\Pi}_{pq} = \Pi^*_{pq}$ at termination and the theorem is proved.

THEOREM 2.3.3 - Any path that is a subpath of a shortest path is itself a shortest path.

PROOF: Let Π^k_{pi} and Π^k_{ij} be subpaths of $\hat{\Pi}_{pj}$. If Π^k_{pi} is not a shortest path from p to i then some other path Π^m_{pi}, $m \neq k$, must exist where $d(\Pi^m_{pi}) < d(\Pi^k_{pi})$. But Π^m_{pi} is a subpath of some other path $\Pi^h_{pj} \neq \hat{\Pi}_{pj}$, therefore $d(\Pi^h_{pj}) < d(\hat{\Pi}^h_{pj})$. But $\hat{\Pi}pj$ is the shortest path and this is a contradiction. Hence Π^k_{pi} must be the shortest path $\hat{\Pi}_{pi}$.

Similarly is can be shown that $\hat{\Pi}_{ij}=\Pi^k_{ij}$ and the theorem is proved.

COMMENTS

1. In step 2 of the shortest path algorithm where more than one shortest path is present, only the first one encountered is selected. This fact, together with Theorem 2.3.3 implies that upon termination of Algorithm 2.3.1, a tree is formed whose arcs are all members of the shortest paths.

2. Computational errors can be detected by checking the sign of $d(\Pi^*_{pp})$. If $d(\Pi^*_{pp})$ is negative then this implies that a negative circuit exists and the problem is not well defined.

3. Step 2 of Algorithm 2.3.1 does not provide a systematic way of either scanning each node j or searching for a node i that satisfies (2.3.1). In order to arrive

at a solution rapidly the following convention is adopted, namely that, for each node $j = 1, 2, ---, (n-1)$ inequality (2.3.1) is tested for all nodes $i = p, 1, ---, (n-1)$, $i \neq j$. Algorithm 2.3.1 can now be rewritten as,

<u>Algorithm 2.3.2</u> -

<u>Step 1)</u> - Same as for Algorithm 2.3.1

<u>Step 2)</u> - a) For $j = 1, 2, ---, (n-1)$,

For $i = p, 1, ---, (n-1), i \neq j$,

If $d(\Pi^*_{pi}) + d(i,j) < d(\Pi^*_{pj})$ then put $d(\Pi^*_{pj}) = d(\Pi^*_{pi}) + d(i,j)$, and the label for node j is rewritten to read: $[i,d(\Pi^*_{pj})]$.

b) If at least one label is changed in a) then repeat 2a). Otherwise terminate step 2.

<u>Step 3</u> - Same as for Algorithm 2.3.1.

Observe that for each application of step 2a), (2.3.1) is scanned $(n-1)^2$ times.

Algorithm 2.3.2 is now illustrated in the following example:

Example 2.3.1 -

The network $\mathbb{N}$ = (G,d) is shown in figure 2.3.1. N = {p,1,2,3,4}, A = {(p,1),(p,4),(4,1),(1,2),(4,2), (4,3), (3,2)} and the arc costs, d(i,j); $\forall$ (i,j) ε A, are indicated beside the corresponding arcs,

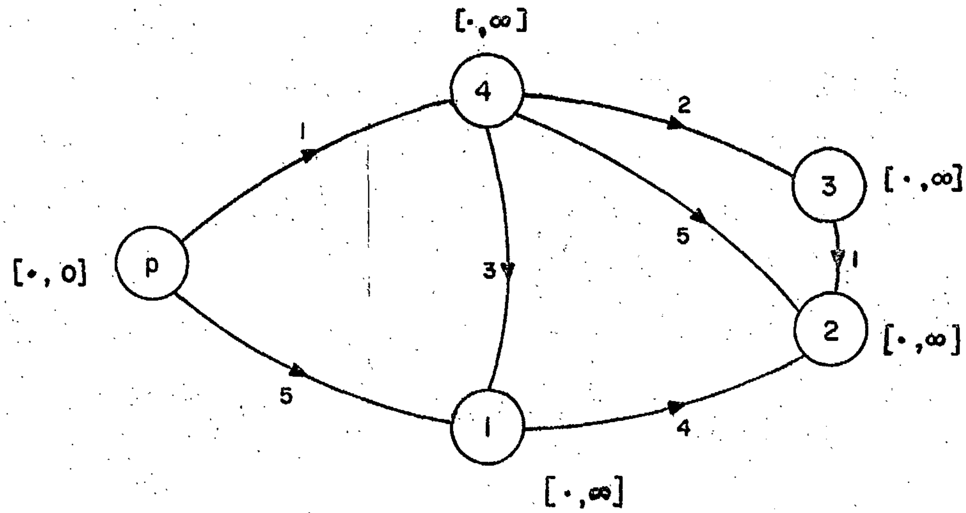

figure 2.3.1

Step 1 of Algorithm 2.3.1 assigns the labels shown in the figure above. Then, the first iteration of step 2a) gives:

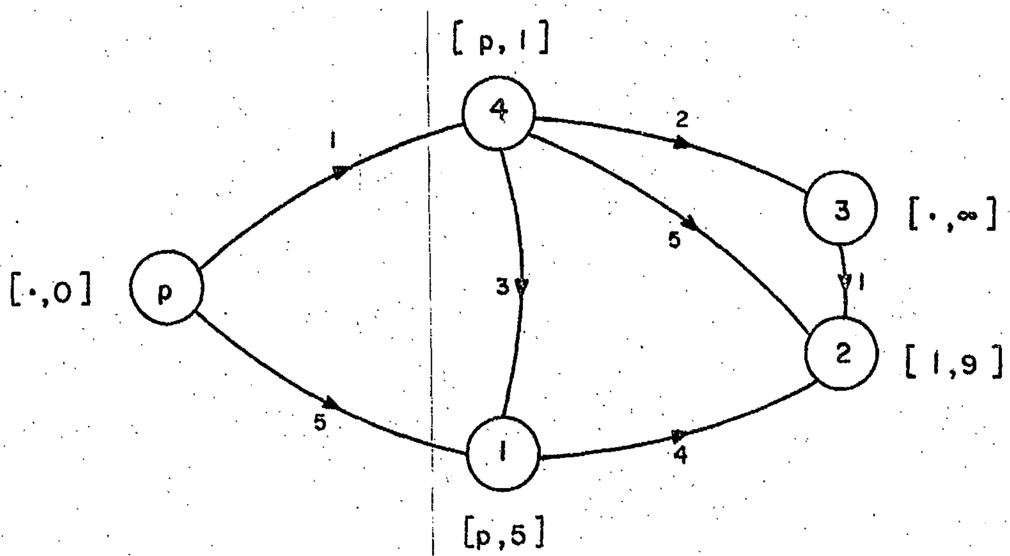

figure 2.3.2

(Note that three labels have changed.)

On scanning all four nodes once more it is found that the labels for nodes 1, 2 and 3 have changed and the resulting figure is shown below:

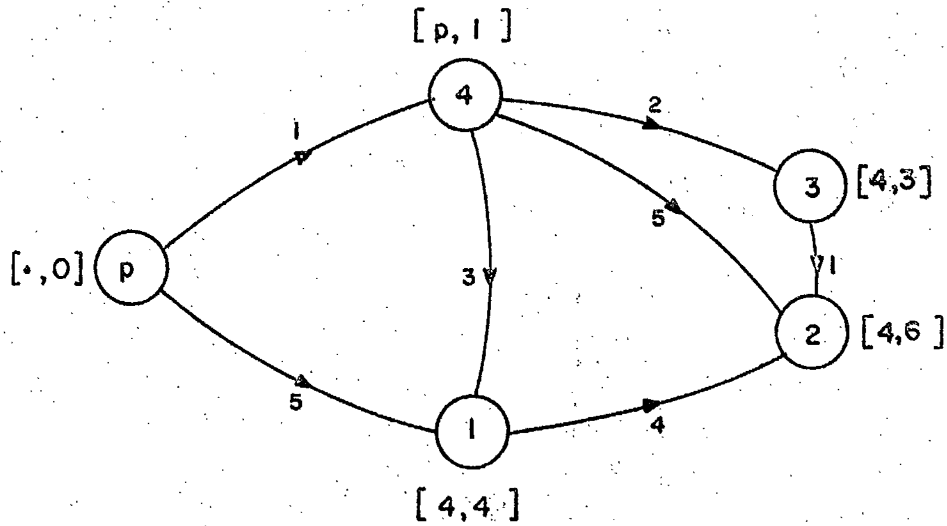

figure 2.3.3

Finally, on termination of step 2:

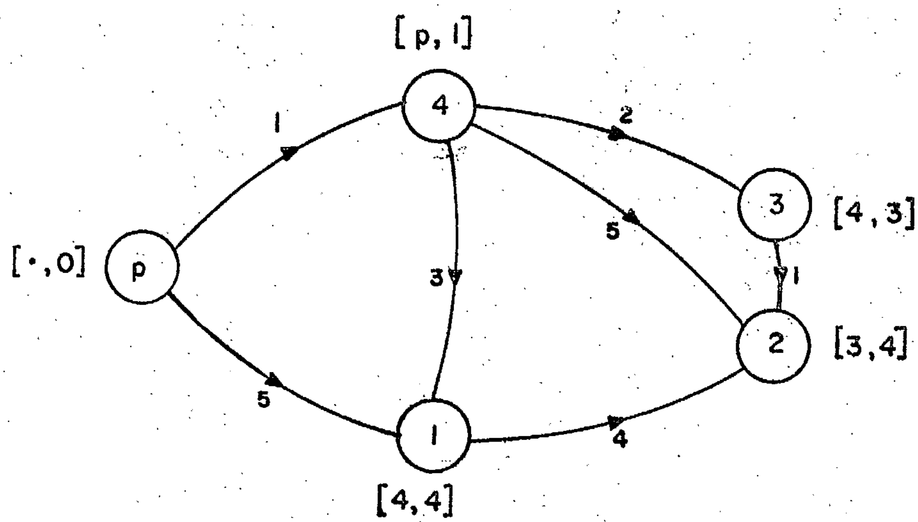

figure 2.3.4

Then, executing step 3,

$$d(\hat{\Pi}_{p,1}) = 4, \quad \hat{\Pi}_{p,1} = \{(p,4),(4,1)\},$$

$$d(\hat{\Pi}_{p,2}) = 4, \quad \hat{\Pi}_{p,2} = \{(p,4),(4,3),(3,2)\},$$

$$d(\hat{\Pi}_{p,3}) = 3, \quad \hat{\Pi}_{p,3} = \{(p,4),(4,3)\},$$

$$d(\hat{\Pi}_{p,4}) = 1, \quad \hat{\Pi}_{p,4} = \{(p,4)\}.$$

Note that the graph formed using only those arcs that are in these shortest paths gives the tree shown in figure 2.3.5. This also demonstrates the validity of Theorem 2.3.3.

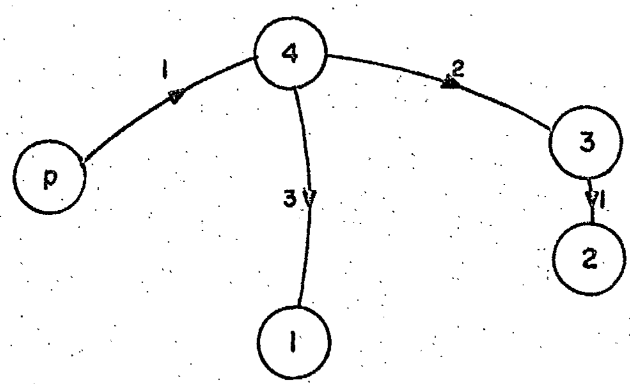

figure 2.3.5

Utilizing the following theorem, some further modifications on Algorithm 2.3.1 can be made.

THEOREM 2.3.4 - For termination to occur, step 2a) of Algorithm 2.3.2 is performed at least once and at the very most n-1 times.

PROOF: First of all, observe that step 2a) of Algorithm 2.3.2 investigates all possible ways of changing all labels, except for p, in the network. Hence, the first time that 2a) does not get a label change, termination occurs in 2b). Consequently, at the very least, step 2a) is performed once before termination. This special case occurs when for some network, $|S_X|_d = \infty$ for $X = \{p\}$, that is, node p cannot relabel any of its neighbouring nodes.

Now, the maximum number of application of 2a) occurs if after each application at least one node i is left which can change the label of some other nodes $j\varepsilon N$, $j \neq i \neq p$. Suppose that after the first iteration some nodes $j\varepsilon N$, $j \neq p$ are relabelled, then, since $d(\Pi^*_{pk}) > d(\Pi^*_{pp})$ $\forall\ k\varepsilon N$, $k \neq p$, (that is, all intermediate path lengths are greater than $d(\Pi^*_{pp}) = 0$.) any application of (2.3.1) cannot relabel p, hence p cannot relabel nodes after the first iteration. In the worst case, then, node $i \neq p$ is the only node that can relabel other nodes (excluding p of course), and after the second iteration, some other nodes $j \varepsilon N$, $j \neq i \neq p$ are relabelled. Since all intermediate path lengths $d(\Pi^*_{pk})$, $k \varepsilon N$, $k \neq p \neq i$, are greater than $d(\Pi^*_{pi})$, node i cannot be relabelled using (2.3.1) and hence node i cannot relabel any nodes. If at each application of 2a) the worst case results (that is, only one node remains that can relabel other nodes) then it follows that at application n-1, n-1 nodes are relabelled for the last time. Since the n^{th} node has no nodes left to relabel, it too is relabelled for the last time. Hence, at the very most, n-1 applications of step 2a) need be performed and the theorem is proved.

COROLLARY

From this theorem it follows that the first iteration of step 2a) puts $d(\Pi^*_{pj}) = d(p,j)$ for all j and $p \varepsilon N$, $j \neq p$. Performing this assignment first requires that only n-2 iterations of step 2a) be performed to guarantee a solution.

A new notation will now be utilized to rewrite the shortest path algorithm in more suitable for computer implementation.

A useful computer programmable representation of the variables in this algorithm is to arrange them in array form. Now it has been shown that arc costs d(i,j) are entries in the matrix D, and now the shortest path length array, $\bar{l}_p$ will be defined to be

$$\bar{l}_p = [d(\Pi^*_{p,1}), d(\Pi^*_{p,2}), \ldots, d(\Pi^*_{p,n-1})], \tag{2.3.7}$$

and the shortest path node array ϕ_p is defined to be

$$\bar{\phi}_p = [\phi_{p,1}, \phi_{p,2}, \ldots, \phi_{p,(n-1)}]. \tag{2.3.8}$$

Observe that (2.3.7) and (2.3.8) together present a convenient way of representing the labels and that an entry for p is unnecessary. Henceforth, the entries of $\bar{l}$ will be called labels.

Then the required representation of the shortest path algorithm is;

<u>Algorithm 2.3.3</u> -

<u>Step 1)</u> - a) For $j = 1, 2, ---, (n-1)$, do 1b).

b) $d(\Pi^*_{pj}) = d(p,j)$

If $d(p,j) < \infty$ then $\phi_j = p$, otherwise $\phi_j = j$.

<u>Step 2)</u> - a) For $k = 1, 2, ---, (n-2)$, do 2b) and 2c).

b) For $j = 1, 2, ---, (n-1)$,

For $i = 1, 2, ---, (n-1)$, $i \neq j$,

If $d(\Pi^*_{pi}) + d(i,j) < d(\Pi^*_{pj})$

then $d(\Pi^*_{pj}) = d(\Pi^*_{pi}) + d(i,j)$ and $\phi_j = i$.

c) If no labels have been changed in 2b) terminate step 2 and initiate step 3.

Step 3) - To identify the nodes on the shortest paths from p to $j \neq p$:

a) Put $k = j$.

b) Identify i from the value of ϕ_k. Then if $\phi_k = k$, no shortest path from p to j exists, otherwise node i is on path $\hat{\Pi}_{pj}$.

c) Put $k = i$. If $k = p$ then terminate, otherwise return to 3b).

Observe that on termination, the pertinent data is stored in the $\bar{l}$ and $\bar{\phi}$ arrays. To illustrate the use of Algorithm 2.3.3 Example 2.3.1 is again worked out.

Example 2.3.2 - (With reference to figure 2.3.1 Matrix D)

$$D = \begin{array}{c} \\ P \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} P & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} * & 5 & \infty & \infty & 1 \\ \infty & * & 4 & \infty & \infty \\ \infty & \infty & * & \infty & \infty \\ \infty & \infty & 1 & * & \infty \\ \infty & 3 & 5 & 2 & * \end{bmatrix} \end{array}$$

Applying the first step 1 of this algorithm gives

$$\bar{l}_p = [5,\infty,\infty,1] \ ,$$

and $\bar{\phi}_p = [p\ 2,3,p]$.

Then listing the values of k,j, $\bar{l}_p$ and $\bar{\phi}_p$ when label changes occut it follows that,

$$k = 1,\ j = 1, \quad \bar{l}_p = [4,\infty,\infty,1]\ ; \quad \bar{\phi}_p = [4,2,3,p]\ ,$$

$$k = 1,\ j = 2, \quad \bar{l}_p = [4,6,\infty,1]\ , \quad \bar{\phi}_p = [4,4,3,p]\ ,$$

$$k = 1,\ j = 3, \quad \bar{l}_p = [4,6,3,1]\ , \quad \bar{\phi}_p = [4,4,4,p]\ ,$$

$$k = 2,\ j = 2, \quad \bar{l}_p = [4,4,3,1]\ , \quad \bar{\phi}_p = [4,3,4,p]\ ,$$

$$k = 3,\ j = 4, \quad \bar{l}_p = [4,4,3,1]\ , \quad \bar{\phi}_p = [4,3,4,p]\ .$$

(Note that step 2a) is executed three times before termination occurs).

$\bar{l}_p$ gives the magnitude of the various shortest paths and using step 3 of the algorithm, the shortest paths are extracted from the contents of $\bar{\phi}_p$, that is, at termination:

$$d(\hat{\Pi}_{p,1}) = 4, \quad \hat{\Pi}_{p,1} = \{(p,4),(4,1)\};$$

$$d(\hat{\Pi}_{p,2}) = 4, \quad \hat{\Pi}_{p,2} = \{(p,4),(4,3),(3,2)\};$$

$$d(\hat{\Pi}_{p,3}) = 3, \quad \hat{\Pi}_{p,3} = \{(p,4),(4,3)\};$$

$$d(\hat{\Pi}_{p,4}) = 4, \quad \hat{\Pi}_{p,4} = \{(p,4)\}.$$

The results in this example are identical to those in Example 2.3.1.

2.4 ALL SHORTEST PATHS IN A MULTI-TERMINAL NETWORK

In this section, a method is presented for finding the shortest paths between all pairs of nodes in a given network. It is shown that the algorithm that Floyd [3,5] originally formulated to solve this problem is actually just an extension of Algorithm 2.3.3.

Consider, once again, the network $\mathbb{N} = (G,d)$ where G is an n node finite oriented quasicomplete graph, and d is the cost function. Now Algorithm 2.3.3 manipulates the arrays D, $\bar{I}_p$ and $\bar{\phi}_p$ in such a way as to leave in $\bar{\phi}_p$ and $\bar{I}_p$ the shortest paths and their values from p to all other nodes in the network. It follows then, that by executing the algorithm n times, once for each node $s = p,1,---,(n-1)$ in the network, that the shortest paths from s to all other nodes are found for all s, that is,

$\hat{\Pi}_{sq}$, for all $s,q\varepsilon N$, $s \neq q$ are found and the multi-terminal shortest path problem is solved.

After each execution of Algorithm 2.3.3 the row vectors $\bar{l}_s$ and $\bar{\phi}_s$ are generated. Renumbering the nodes, $s = 1,2,\text{---},n$ and performing this procedure n times for all $s\varepsilon N$ gives the following two matrices. These are formed by collecting the row vectors.

$$L = \begin{bmatrix} \bar{l}_1 \\ \bar{l}_2 \\ \cdot \\ \cdot \\ \cdot \\ \bar{l}_n \end{bmatrix} = \begin{bmatrix} * & d(\Pi^*_{1,2}) & \cdot & \cdot & \cdot & d(\Pi^*_{1,n}) \\ d(\Pi^*_{2,1}) & * & \cdot & \cdot & \cdot & d(\Pi^*_{2,n}) \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ d(\Pi^*_{n,1}) & d(\Pi^*_{n,2}) & & & & * \end{bmatrix};$$

$$\Phi = \begin{bmatrix} \bar{\phi}_1 \\ \bar{\phi}_2 \\ \cdot \\ \cdot \\ \cdot \\ \bar{\phi}_n \end{bmatrix} = \begin{bmatrix} * & \phi_{1,2} & \cdot & \cdot & \cdot & \phi_{1,n} \\ \phi_{2,1} & * & \cdot & \cdot & \cdot & \phi_{2,n} \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ \phi_{n,1} & \phi_{n,2} & & & & * \end{bmatrix}$$

Steps 1, 2 and 3 are independent operations within Algorithm 2.3.2, hence, each step may be executed n times to arrive at a solution for the multi-terminal problem. Then the following Algorithm can now be formulated

Algorithm 2.4.1 -

Step 1) - a) For $s = 1,2,---,n$, do 1b).

b) For $j = 1,2,---,n$, $j \neq s$,

$d(\Pi^*_{sj}) = d(s,j)$,

If $d(s,j) < \infty$ then $\phi_{sj} = s$, otherwise $\phi_{sj} = j$.

Step 2) - a) For $s = 1,2,---,n$, do 2b).

b) For $k = 1,2,---,(n-2)$, do 2c) and 2d).

c) For $j = 1,2,---,n$, $j \neq s$,

For $i = 1,2,---,n$, $i \neq j$, $i \neq s$,

If $d(\Pi^*_{si}) + d(i,j) < d(\Pi^*_{sj})$

then $d(\Pi^*_{sj}) = d(\Pi^*_{si}) + d(i,j)$ and $\phi_{sj} = i$.

d) If no labels have been changed in 2c) then terminate step 2 and initiate step 3.

Step 3) - To identify the nodes on the shortest paths $\hat{\Pi}_{sj}$, $s = 1,2,---,n$, $j \neq s$:

a) Put $k = j$.

b) Identify i from the value of ϕ_{sk}. Then if $\phi_{sk} = k$, no shortest path from s to j exists, otherwise i is on path $\hat{\Pi}_{sj}$.

c) Put $k = i$. If $k = s$ then terminate, otherwise return to 3b).

Steps 2a) and 2b) can be interchanged without affecting the algorithm. This is due to the independence of the operations on each row in L, that is, any iteration on some $\bar{1}_x$ in L does not affect the calculations on some other $\bar{1}_y$, $y \neq x$, hence, the shortest paths from node x to nodes $j = 1,2,---,n, j \neq x$ can be computed in common steps to those for node y. Consequently the modification is justified.

Furthermore, in step 1, $d(\Pi^*_{ij}) = d(i,j)\ \forall\ (i,j) \in A$ and at some iteration of 2c) some $d(\Pi^*_{ij})$, $i \neq s \neq j$, is decreased in value. Hence, $d(\Pi^*_{ij}) \leq d(i,j)$ throughout the algorithm. Then replacing $d(i,j)$ with $d(\Pi^*_{ij})$ in Step 2c) certainly causes the algorithm to converge no less rapidly than before. This leads to

<u>Algorithm 2.4.2</u> -

<u>Step 1)</u> - Same as for Algorithm 2.4.1

<u>Step 2)</u> - a) For $k = 1,2,---,(n-2)$, do 2b) and 2f).

b) For $s = 1,2,---,n$, do 2c).

c) For $j = 1,2,---,n$, $j \neq s$, do 2d).

d) For $i = 1,2,---,n$, $i \neq j$, $i \neq s$ do 2e).

e) If $d(\Pi^*_{si}) + d(\Pi^*_{ij}) < d(\Pi^*_{sj})$

then $d(\Pi^*_{sj}) = d(\Pi^*_{si}) + d(\Pi^*_{ij})$ and $\phi_{sj} = i$.

f) If no labels have been changed in 2c) then terminate step 2 and initiate step 3.

<u>Step 3</u> - To identify the nodes on the shortest paths $\hat{\Pi}_{sj}$, $s = 1,2,---,n, j \neq s$:

a) Put $h = s$ and $k = j$.

b) Identify i from the value of ϕ_{1k}. Then if $\phi_{1k} = k$, no shortest path from s to j exists, otherwise i is on the path $\hat{\Pi}_{sj}$.

c) If $i \neq h$ then $h = i$ and go to step 3b).

d) If $h = s$ then terminate, otherwise put $h = s$, $k = i$ and go to step 3b).

Note that step 3 of the algorithm has been altered by the substitution made for $d(i,j)$ in step 2e). In Algorithm 2.4.1, if the introduction of the node i decreases the value of $d(\Pi^*_{sj})$ then the assignment $\phi_{sj} = i$ was made, that is the second last node in the path Π^*_{sj} (the one before j) is recorded. In Algorithm 2.4.2, putting $\phi_{sj} = i$ when $d(\Pi^*_{sj})$ is reduced does not guarantee that i is the second last node in $\hat{\Pi}_{sj}$. This only occurs if $d(\Pi^*_{ij}) = d(i,j)$. In general, since $d(\Pi^*_{ij})$ may be changed at some point in the algorithm, (that is, intermediate nodes may be found on the path from i to j) the matrix Φ must be searched for all subpaths of Π_{si} to find all the intermediate nodes.

Let the basic operation in step 2e) be the triple (s,i,j) where i is the intermediate node for the path $\hat{\Pi}_{sj}$. By inserting statement 2d) before statement 2b) in the algorithm above, the same triples are performed only in a different order, hence the outcome is certainly not altered. This modified form of the Algorithm reduces all labels $d(\Pi^*_{sj})\ \forall\ (s,j) \in A$ for each introduction of node i, i = 1,2,---,n, and with this change, Algorithm 2.4.2 introduces each node n-2 times to guarantee solution. It is now proposed that each node need only be introduced once to arrive at the desired solution, that is the following Algorithm 2.4.3 leads to a program that solves the multi-terminal shortest path problem.

<u>Algorithm 2.4.3</u> -

<u>Step 1)</u> - Same as for Algorithm 2.4.1.

<u>Step 2)</u> - For i = 1,2,---,n,
For s = 1,2,---,n, $s \neq i$,
For j = 1,2,---,n, $j \neq s$, $j \neq i$,
If $d(\Pi^*_{si}) + d(\Pi^*_{ij}) < d(\Pi^*_{sj})$
then $d(\Pi^*_{sj}) = d(\Pi^*_{si}) + d(\Pi^*_{ij})$ and $\phi_{sj} = i$.

<u>Step 3)</u> - Same as for Algorithm 2.4.2.

This is the same algorithm that Floyd presented to solve the multi-terminal shortest path problem [3,5].

The following theorem shows that step 2 need <u>not</u> be performed n-2 times to guarantee solution.

<u>THEOREM 2.4.1</u> - In step 2 of Algorithm 2.4.3, each node i, i = 1,2,---,n, need only be introduced at most once to guarantee that all shortest paths are found.

<u>PROOF</u>: The proof is inductive, that is a basic assumption is made for the introduction of some node i=k-1; then it is proved that from this assumption it must also be true for i = k; then if it is true for i = 1, the assumption is true for all i and the theorem can be proved. But first the following comments are given.

Step 2 of the algorithm contains every possible triple precisely once and these triples are arranged in n groups, with the k^{th} group, that corresponds to the introduction of node k, consisting of every useful triple with intermediate node k. Furthermore if after the introduction of some node k, some $d(\Pi^*_{sj})$ reaches its minimum value, then none of the nodes i > k can possibly be introduced to that path since these nodes cannot decrease the minimum. Suppose that this algorithm finds the shortest paths between all pairs of nodes that have k-1 as the highest numbered intermediate node. Further, suppose this is true for all nodes $i < k-1$; then these shortest paths are found after node $i = k-1$ has been introduced to all $d(\Pi^*_{sj})$ and before the k^{th} group has been reached.

Let k be the highest numbered intermediate node on path $\hat{\Pi}_{uv}$, then the subpaths $\hat{\Pi}_{uk}$ and $\hat{\Pi}_{kv}$ must have intermediate nodes that are smaller than k. By assumption $\hat{\Pi}_{uk}$ and $\hat{\Pi}_{kv}$ are reached before node k is introduced, hence $d(\hat{\Pi}_{uv})$ reaches its lower bound at $i = k$ and the shortest path $\hat{\Pi}_{uv}$ persists to the end. This assumption then, is true for $i = k$ and for $i = k-1$.

The first group of triples with intermediate node $i = 1$ ensures that the shortest path produced by a two arc path using node 1 as an intermediate node is obtained correctly. Therefore, the assumption is also true for $i = 1$. By induction it also follows that the assumption is true for all $i \varepsilon N$.

Letting $k = n + 1$, it follows that all of the shortest paths with intermediate nodes $i \leq n$ are found before the $(n+1)^{st}$ group. But these are all of the required shortest paths, consequently none of the nodes $i = 1,2,---,n$ need be reintroduced to improve any of the path lengths, hence the theorem is proved.

An example that illustrates Algorithm 2.4.3 is now presented.

<u>Example 2.4.1</u> - For the network of figure 2.4.1 the values of the cost function d are given as matrix D.

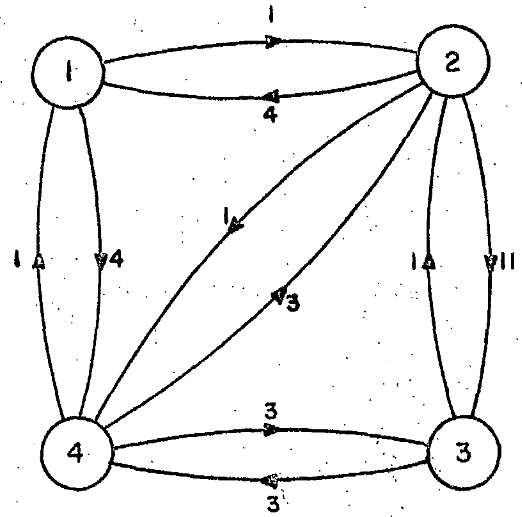

figure 2.4.1

$$D = \begin{bmatrix} * & 1 & \infty & 4 \\ 4 & * & 11 & 1 \\ \infty & 1 & * & 3 \\ 1 & 3 & 3 & * \end{bmatrix}$$

Then from step 1 of the algorithm L = D, and Φ is

$$\Phi = \begin{bmatrix} * & 1 & 3 & 1 \\ 2 & * & 2 & 2 \\ 1 & 3 & * & 3 \\ 4 & 4 & 4 & * \end{bmatrix}$$

Introducing node 1, that is, for iteration i = 1 it follows that

$$L = \begin{bmatrix} * & 1 & \infty & 4 \\ 4 & * & 11 & 1 \\ \infty & 1 & * & 3 \\ 1 & 2 & 3 & * \end{bmatrix} \qquad \Phi = \begin{bmatrix} * & 1 & 3 & 1 \\ 2 & * & 2 & 2 \\ 1 & 3 & * & 3 \\ 4 & 1 & 4 & * \end{bmatrix}$$

Now only $d(\Pi^*_{4,2})$ is improved using node 1, and iterating for i = 2,

$$L = \begin{bmatrix} * & 1 & 12 & 2 \\ 4 & * & 11 & 1 \\ 5 & 1 & * & 2 \\ 1 & 2 & 3 & * \end{bmatrix} \qquad \Phi = \begin{bmatrix} * & 1 & 2 & 2 \\ 2 & * & 2 & 2 \\ 2 & 3 & * & 2 \\ 4 & 1 & 4 & * \end{bmatrix}$$

For $i = 3$,

$$L = \begin{bmatrix} * & 1 & 12 & 2 \\ 4 & * & 11 & 1 \\ 5 & 1 & * & 2 \\ 1 & 2 & 3 & * \end{bmatrix} \qquad \phi = \begin{bmatrix} * & 1 & 2 & 2 \\ 2 & * & 2 & 2 \\ 2 & 3 & * & 2 \\ 4 & 1 & 4 & * \end{bmatrix}$$

Finally, for $i = 4$,

$$L = \begin{bmatrix} * & 1 & 5 & 2 \\ 2 & * & 4 & 1 \\ 3 & 1 & * & 2 \\ 1 & 2 & 3 & * \end{bmatrix} \qquad \phi = \begin{bmatrix} * & 1 & 4 & 2 \\ 4 & * & 4 & 2 \\ 4 & 3 & * & 2 \\ 4 & 1 & 4 & * \end{bmatrix}$$

From L and ϕ the shortest paths and their values are found on inspection of the entries of L directly and by applying step 3 of the algorithm to matrix ϕ.

$d(\hat{\Pi}_{12}) = 1, \qquad \hat{\Pi}_{12} = \{(1,2)\},$

$d(\hat{\Pi}_{13}) = 5, \qquad \hat{\Pi}_{13} = \{(1,2),(2,4),(4,3)\},$

$d(\hat{\Pi}_{14}) = 2, \qquad \hat{\Pi}_{14} = \{(1,2),(2,4)\},$

$d(\hat{\Pi}_{21}) = 2, \qquad \hat{\Pi}_{21} = \{(2,4),(4,1)\},$

$d(\hat{\Pi}_{23}) = 4, \qquad \hat{\Pi}_{23} = \{(2,4),(4,3)\},$

$d(\hat{\Pi}_{24}) = 1, \qquad \hat{\Pi}_{24} = \{(2,4)\},$

$d(\hat{\Pi}_{31}) = 3, \qquad \hat{\Pi}_{31} = \{(3,2),(2,4),(4,1)\},$

$d(\hat{\Pi}_{32}) = 1, \qquad \hat{\Pi}_{32} = \{(3,2)\},$

$d(\hat{\Pi}_{34}) = 2, \qquad \hat{\Pi}_{34} = \{(3,2),(2,4)\},$

$d(\hat{\Pi}_{41}) = 1, \qquad \hat{\Pi}_{41} = \{(4,1)\},$

$d(\hat{\Pi}_{42}) = 2, \qquad \hat{\Pi}_{42} = \{(4,1),(1,2)\},$

$d(\hat{\Pi}_{43}) = 3, \qquad \hat{\Pi}_{43} = \{(4,3)\}.$

2.5 THE SYNTHESIS OF SIMULTANEOUS TRANSMISSION NETWORKS

In this section, a procedure to synthesize a simultaneous transmission network $\mathbb{N} = (G,d,c,t)$ is presented. The terminal capacity function t represents the terminal requirements that must be simultaneously satisfied; the cost function d is specifiable in cost per unit of capacity; and the capacity function c is to be found.

The following program constructs a network that satisfies the requirements, moreover, no cheaper capacity configuration can be found to meet the given specifications.

Algorithm 2.5.1 -

Step 1) - For $i = 1,2,---,n$,

For $j = 1,2,---,n$, $j \neq i$,

$c(i,j) = 0$.

Step 2) - Using Algorithm 2.4.3 and the cost function d, find all the shortest paths $\hat{\Pi}_{pq}$ in the network.

Step 3) - For $p = 1,2,---,n$,

For $q = 1,2,---,n$, $q \neq p$,

$$c(i,j) = c(i,j) + t(p,q); \quad \forall (i,j) \in \hat{\Pi}_{pq}.$$

Observe that the algorithm is an accumulative process that begins with a network without any capacity on the arcs and then builds enough capacity in the network to satisfy each requirement, one at a time.

THEOREM 2.5.1: - Algorithm 2.5.1 constructs a network that exactly satisfies the terminal requirements and does so at minimum network cost.

PROOF: Because all terminal requirements must be satisfied simultaneously, dedicated lines must be constructed for each $t(p,q)$, $(p,q) \in A$ along specified routes to ensure that messages do not interfere with each other. Step 2 of the Algorithm selects such routes, that is, selects a path $\hat{\Pi}_{pq}$ for each $t(p,q)$.

Suppose that for some set of arcs X, $X \subset A$ there are m terminal requirements, such that for each $(u,v) \in X$, the corresponding path $\hat{\Pi}_{uv}$ contains some specific arc $(x,y) \in \hat{\Pi}_{uv}$ for all $(u,v) \in X$. Further, all other paths do not contain (x,y) as an element. Then at the termination of step 3, the capacity of arc (x,y) contains m terms, that is,

$$c(x,y) = \sum_{(u,v)} t(u,v); \; \forall (u,v) \in X. \tag{2.5.1}$$

No other requirements utilize arc (x,y) and $c(x,y)$ exactly satisfies all requirements as well as the $t(u,v)$, $(u,v) \in X$. Generalizing this arguement to any arc in the network it follows that all capacities exactly satisfy the requirements and it only now remains to establish that the minimum cost network is constructed.

Since the algorithm constructs a dedicated path for each $t(p,q)$ then there is certainly no cheaper way to build this path than along the shortest path $\hat{\Pi}_{pq}$. Thus, since this is done for all $(p,q) \in A$, certainly no lesser cost network that exactly satisfies all the requirements simultaneously can be synthesized.

Observe that the total network cost K can be expressed as follows,

$$K = \sum_{(i,j)} c(i,j).d(i,j) = \sum_{(i,j)} d(i,j) . \sum_{(p,q)} t(p,q)$$

$$\forall (i,j)\varepsilon A;\ \forall (p,q) \ni (i,j)\varepsilon \hat{\Pi}_{pq}. \qquad (2.5.2)$$

The following example illustrates the use of the algorithm.

Example 2.5.1 - A simultaneous transmission network is synthesized using Algorithm 2.5.1. Given the communication requirements t, the arc costs d (cost per unit capacity) and the configuration G, the capacity function c is to be evaluated for the network $\mathbb{N} = (G,d,c,t)$ of figure 2.5.1.

Then the matrix T contains the t(i,j).

$$T = \begin{bmatrix} * & 2 & 6 & 5 \\ 2 & * & 1 & 3 \\ 6 & 1 & * & 4 \\ 1 & 3 & 2 & * \end{bmatrix}$$

These requirements are indicated beside the corresponding arcs in figure 2.5.1.

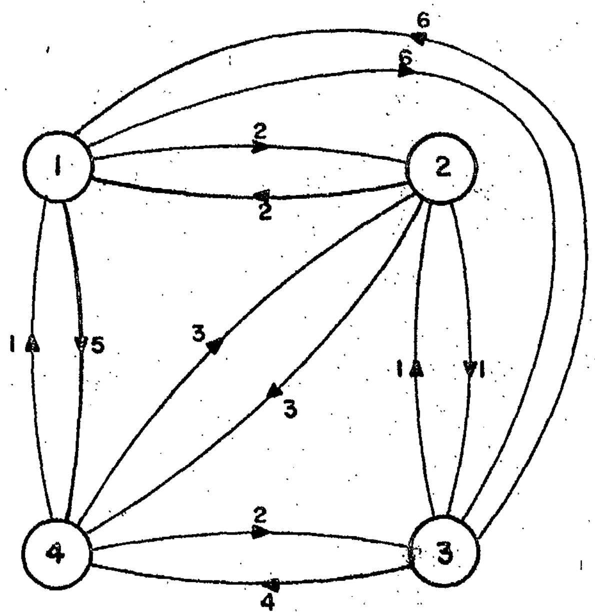

figure 2.5.1

The arc costs are found in the matrix D and the d(i,j) are shown in figure 2.5.2.

$$D = \begin{bmatrix} * & 1 & \infty & 4 \\ 4 & * & 11 & 1 \\ \infty & 1 & * & 3 \\ 1 & 3 & 3 & * \end{bmatrix}$$

(Note that these are the same costs as in Example 2.4.1.)

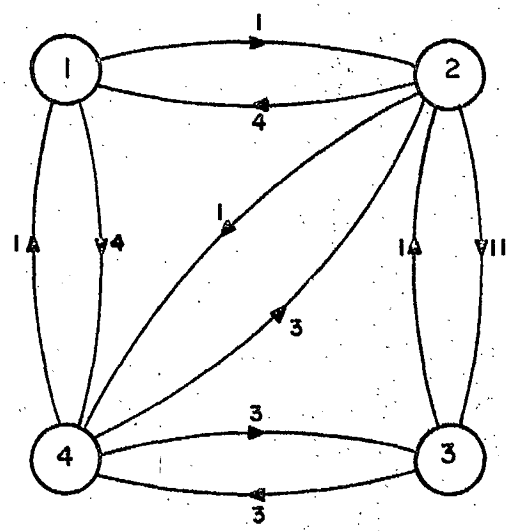

figure 2.5.2

Step 1 of Algorithm 2.5.1 initializes all the unknown capacities to zero.

As in Example 2.4.1 the magnitudes of all the shortest paths are found in L, That is,

$$L = \begin{bmatrix} * & 1 & 5 & 2 \\ 2 & * & 4 & 1 \\ 3 & 1 & * & 2 \\ 1 & 2 & 3 & * \end{bmatrix}$$

and the routes are found in Φ using step 3 of Algorithm 2.4.3.

$$\Phi = \begin{bmatrix} * & 1 & 4 & 2 \\ 4 & * & 4 & 2 \\ 4 & 3 & * & 2 \\ 4 & 1 & 4 & * \end{bmatrix}$$

Then from T, L and Φ it follows that

$t(1,2) = 2, \quad d(\hat{\Pi}_{1,2}) = 1, \quad \hat{\Pi}_{1,2} = \{(1,2)\},$

$t(1,3) = 6, \quad d(\hat{\Pi}_{1,3}) = 5, \quad \hat{\Pi}_{1,3} = \{(1,2),(2,4),(4,3)\},$

$t(1,4) = 5, \quad d(\hat{\Pi}_{1,4}) = 2, \quad \hat{\Pi}_{1,4} = \{(1,2),(2,4)\},$

$t(2,1) = 2, \quad d(\hat{\Pi}_{2,1}) = 2, \quad \hat{\Pi}_{2,1} = \{(2,4),(4,1)\},$

$t(2,3) = 1, \quad d(\hat{\Pi}_{2,3}) = 4, \quad \hat{\Pi}_{2,3} = \{(2,4),(4,3)\},$

$t(2,4) = 3, \quad d(\hat{\Pi}_{2,4}) = 1, \quad \hat{\Pi}_{2,4} = \{(2,4)\},$

$t(3,1) = 6, \quad d(\hat{\Pi}_{3,1}) = 3, \quad \hat{\Pi}_{3,1} = \{(3,2),(2,4),(4,1)\},$

$t(3,2) = 1, \quad d(\hat{\Pi}_{3,2}) = 1, \quad \hat{\Pi}_{3,2} = \{(3,2)\},$

$$t(3,4) = 4, \quad d(\hat{\Pi}_{3,4}) = 2, \quad \hat{\Pi}_{3,4} = \{(3,2),(2,4)\},$$

$$t(4,1) = 1, \quad d(\hat{\Pi}_{4,1}) = 1, \quad \hat{\Pi}_{4,1} = \{(4,1)\},$$

$$t(4,2) = 3, \quad d(\hat{\Pi}_{4,2}) = 2, \quad \hat{\Pi}_{4,2} = \{(4,1),(1,2)\},$$

$$t(4,3) = 2, \quad d(\hat{\Pi}_{4,3}) = 3, \quad \hat{\Pi}_{4,3} = \{(4,3)\}.$$

Step 3 of Algorithm 2.5.1 proceeds to add enough capacity to satisfy a requirement $t(i,j)$ along the shortest path $\hat{\Pi}_{ij}$.

Then for the first two requirements $t(1,2) = 2$ and $t(1,3) = 6$, 2 units of capacity are built along arc (i,j) to satisfy $t(1,2)$ and we add 6 units of capacity along the arcs in $\hat{\Pi}_{13}$ to satisfy $t(1,3)$ simultaneously, that is, $c(1,2) = 2 + 6 = 8$; $c(2,4) = 6$; $c(4,3) = 6$. Figure 2.5.3 shows this stage of synthesis.

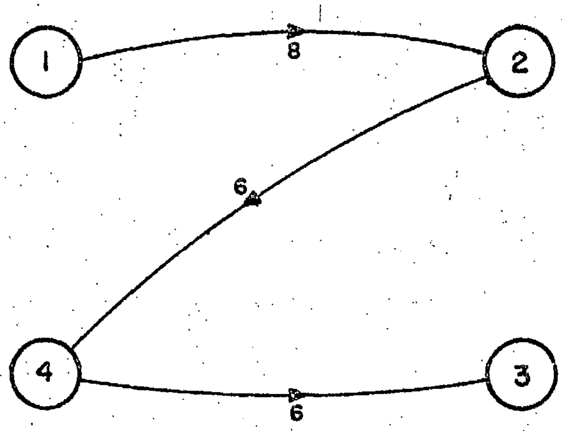

figure 2.5.3

Proceeding in this way for all requirements in the network, the synthesized network is:

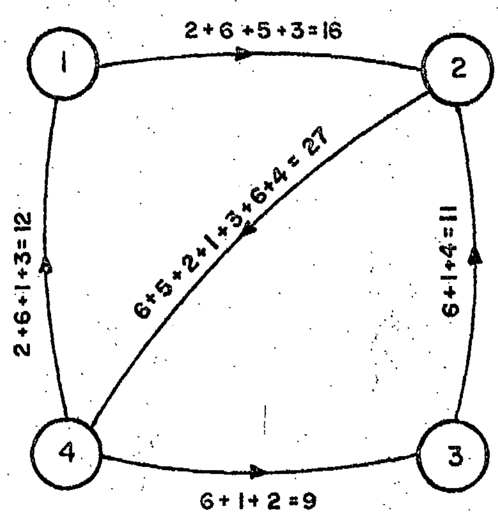

figure 2.5.4

and the arc capacity matrix is,

$$C = \begin{bmatrix} * & 16 & 0 & 0 \\ 0 & * & 0 & 27 \\ 0 & 11 & * & 0 \\ 12 & 0 & 9 & * \end{bmatrix}$$

<u>COMMENT</u>

Appendix B of this study contains the description of a computer program that implements Algorithm 2.5.1. A program listing and a programmed example are also given there. This conversational package offers a convenient way of solving the simultaneous transmission synthesis problem for networks containing up to fifteen nodes.

2.6 THE SYNTHESIS OF TIME-SHARED COMMUNICATION NETWORKS

An algorithm to solve the time-shared specifications for a communications network $\mathbb{N} = (G,t,d,c)$ is presented. The network configuration G, the terminal capacity function t and the cost function d are all known. In addition, it is assumed that some system of controls (multiplexing etc.) allows only one pair of nodes to communicate (in one direction) at one time. That is, in a given "time slice" only one terminal p may transmit information to some other terminal q. It is required to find a capacity function that satisfies the terminal capacity requirements t. At the same time it is desirable to keep the construction costs as low as possible.

An algorithm that solves this problem will be presented but first some essential notation is introduced.

If t_1,---,t_m are in decreasing order the distinct values of the terminal capacity function t, then the set of arcs in A on which t has the value t_μ is denoted by A_μ,

$$A_\mu = \{(i,j)\varepsilon A \mid t(i,j) = t_\mu\},\ 1 \leq \mu \leq m, \tag{2.6.1}$$

and $$A^*_\mu = A_1 \cup A_2 \cup \ldots \cup A_{\mu-1} \tag{2.6.2}$$

Algorithm 2.6.1 -

Step 1) - a) For each of the m distinct values t_μ in t, find the set of arcs $A_\mu \subset A$ on which t has the value t_μ.

b) Put $\mu = 1$.

c) For all $(i,j)\varepsilon A$ put $c(i,j) = 0$.

Step 2) - Using Algorithm 2.4.3 and the cost function d, find all the shortest paths $\hat{\Pi}_{pq}$ and their lengths $d(\hat{\Pi}_{pq})$ for all $(p,q)\varepsilon A$.

Step 3) - a) If A_μ is not empty then go to step 4.

b) If $\mu = m$ then terminate, otherwise put $\mu = \mu+1$.

Step 4) - Find the path $\hat{\Pi}_{uv}$ that satisfies :

$$d(\hat{\Pi}_{uv}) = \min\{d(\hat{\Pi}_{ij})/(i,j)\varepsilon A_\mu\} \tag{2.3.3}$$

Step 5) - a) If $d(\hat{\Pi}_{uv}) = 0$ then delete (u,v) from A_μ and go to step 3.

b) For all $(i,j)\varepsilon\hat{\Pi}_{uv}$ do 5c).

c) If $c(i,j) = 0$ then $c(i,j) = t_\mu$ and $d(i,j) = 0$.

d) Delete (u,v) from A_μ and go to step 2.

The algorithm, then, begins by arranging the requirements (the $t(i,j)$'s) in groups of distinct values and putting all the capacities initially to zero. It then processes the largest requirements through to the smallest in the following way:

The requirement with the correspondingly shortest path within a given group is satisfied by building capacity along the route where needed; the costs along that path are set to zero with the result that the cost function is modified from the previous step; using the "modified" cost function, the next terminal capacity requirement is satisfied.

In step 5a) if a given shortest path is zero, this implies that either a) all costs along the path have been modified to zero or b) that at least one arc along the path had zero cost at initiation and all other arcs in the path, if any, have been reduced to zero. If the second case should occur, that is some $d(i,j) = 0$ while $t(u,v) > 0$ where $(i,j)\varepsilon\bar{\Pi}_{uv}$, then the requirement from u to v may not be satisfied by the resultant network. However, since for case a) step 5a) gives a large saving in computation time by avoiding the recalculation of the shortest path in step 2, step 5a) should be included in the algorithm. To avoid error in cases of zero cost, it is necessary to assign a small but finite value to the otherwise zero cost arc or exclude step 5a) from the algorithm and suffer a loss in efficiency.

The following theorem shows that Algorithm 2.6.1 generates a network that satisfies all the given requirements.

THEOREM 2.6.1 - Algorithm 2.6.1 finds a network that satisfies all the given terminal capacity requirements of a time-shared system.

PROOF: Suppose that at some point, during the synthesis procedure $\mu = h$ and A_h^1 is the subset of arcs in A_μ that corresponds to the terminal requirements that have been considered up to that point. Then all the $t(i,j)$ that have been considered are those $(i,j) \in A_\mu^* \cup A_h^1$.

Assume that all requirements up to this point have been satisfied. Then at step 4 all the shortest paths are calculated and $\hat{\Pi}_{uv}$ is selected. Let $t(u,v)$ be the next requirement that is to be considered. Two cases can occur in step 5: a) If the cost of the path from u to v is zero, then no capacities are constructed at this step; b) otherwise; all arcs on the path from u to v are considered; and capacity equal to $t(u,v)$ is built on those arcs in $\hat{\Pi}_{uv}$ that have zero capacity.

In case a) the cost is zero since step 5c) has assigned zero costs and finite capacities to all arcs in $\hat{\Pi}_{uv}$. Since it has been assumed that all requirements considered in previous steps are satisfied and since all these requirements are not smaller than t(u,v) all of these non-zero capacities along $\hat{\Pi}_{uv}$ must be at least as large as t(u,v). Then t(u,v) is satisfied after step 5a) is completed.

Similarly, in case b), capacity need only be built on zero valued arcs since all others are at least as large as t(u,v); consequently, t(u,v) is satisfied after step 5c). Since t(u,v) is satisfied, and by the assumption that all requirements considered, previous to t(u,v) are satisfied, and also since the same assumption holds true for the very first terminal requirement considered, then by induction it follows that at termination all the requirements are satisfied and the theorem is proved.

COMMENTS

This algorithm constructs one capacity value per arc at the very most and once an individual c(i,j) is determined, it is not altered. The very worst case, as far as total network cost is concerned, is if the algorithm assigns values for all c(i,j), $(i,j) \varepsilon A$. This occurs when $c(i,j) = t(i,j)\ \forall (i,j) \varepsilon A$; and if for some t(u,v) an alternate route is selected as a path designated for construction of capacities, then the shortest path $\hat{\Pi}_{uv}$ at all subsequent steps is not the direct arc (u,v) and capacity is certainly not constructed on arc (u,v) in any of the steps that follow; that is, the worst case cannot occur.

Assume then, that this most expensive case is indeed constructed. Further, let the capacity function on each arc assume values that are distinct from all others. If c(u,v) is the last arc that is constructed, that is t(u,v) is the smallest entry in t; then removing this capacity by setting c(u,v) = 0 does not affect the requirement t(u,v) since all other c(i,j), $(i,j)\varepsilon A$, $i \neq u$, $j \neq v$, must have values that are greater than c(u,v) and if $n > 2$ then t(u,v) can use one of the remaining paths to satisfy the requirement. Recalling that (1.3.15) gives the maximum number of paths from one node to another and that this occurs in the quasicomplete graph, it follows that the worst case is not constructed and that Algorithm 2.6.1 yields a suboptimal synthesis.

By deleting in increasing order of t(i,j) the corresponding capacities c(i,j), a final capacity deletion reduces the network to one that no longer satisfies all the time-shared requirements. This occurs when for any node pair u and v some path $\Pi^k_{uv} \ni c(i,j) \geq t(u,v)\ \forall (i,j)$ $\varepsilon\ \Pi^k_{uv}$ cannot be found. The network that remains after the removal of the redundant arcs (all except the final deletion) is the one synthesized by the algorithm in the worst case. Thus if the optimal network cost is known from a linear programming formulation, a measure of the optimality can be made by subtracting this value from the calculated cost of the removed arc capacities.

The following example illustrates how the algorithm uses the "modified" cost function to find those arcs that certain terminal requirements may share.

Example 2.6.1 - A time-shared communications network is to be synthesized using Algorithm 2.6.1. Given the terminal requirements t, the arc costs d and the configuration G, the capacity function c is to be found for the network $\mathbb{N} = (G,t,d,c)$.

The matrix T contains the t(i,j) and these requirements are indicated beside the corresponding arcs in figure 2.6.1

$$T = \begin{bmatrix} * & 9 & 8 & 10 \\ 6 & * & 6 & 8 \\ 4 & 7 & * & 7 \\ 1 & 2 & 1 & * \end{bmatrix}$$

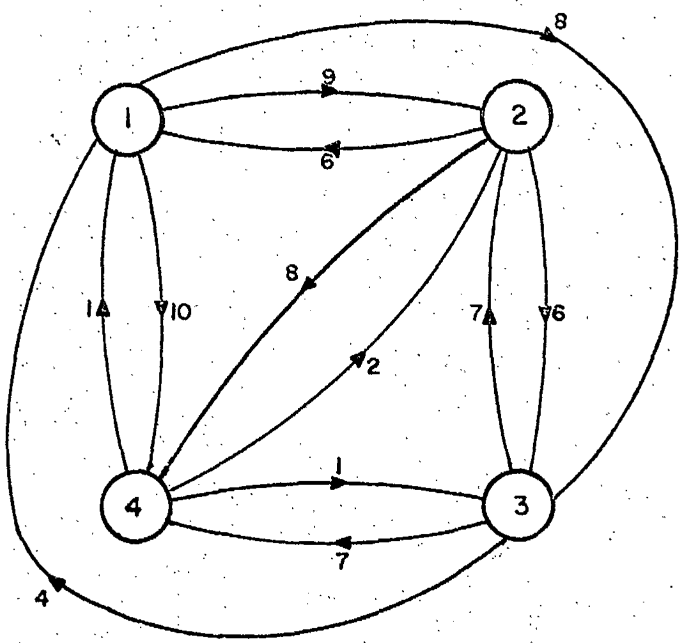

figure 2.6.1

The arc costs are found in the matrix D and the d(i,j) are shown in figure 2.6.2.

$$D = \begin{bmatrix} * & 1 & \infty & 4 \\ 7 & * & 6 & 2 \\ \infty & 1 & * & 5 \\ 2 & 8 & 2 & * \end{bmatrix}$$

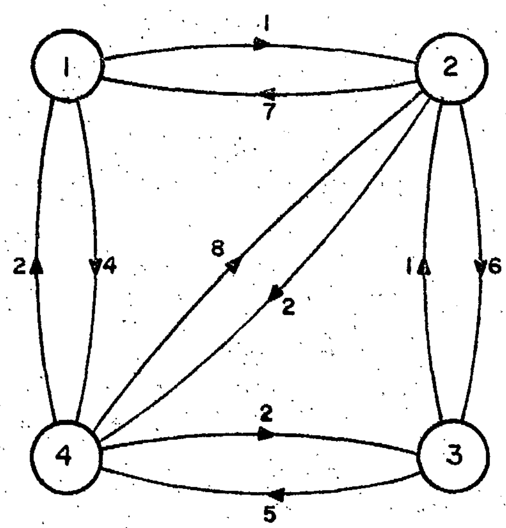

figure 2.6.2

Step 1a) of the algorithm gives:

$t_1 = 10,$ $\quad A_1 = \{(1,4)\},$

$t_2 = 9,$ $\quad A_2 = \{(1,2)\},$

$t_3 = 8,$ $\quad A_3 = \{(1,3),(2,4)\},$

$t_4 = 7,$ $\quad A_4 = \{(3,2),(3,4)\},$

$t_5 = 6, \qquad A_5 = \{(2,1),(2,3)\},$

$t_6 = 4, \qquad A_6 = \{(3,1)\},$

$t_7 = 2, \qquad A_7 = \{(4,2)\},$

$t_8 = 1, \qquad A_8 = \{(4,1),(4,3)\},$

For $\mu = 1$, arc (1,4) is selected as it is the only member in A_1. From the original D matrix $\hat{\Pi}_{14} = \{(1,2),(2,4)\}$ hence, $c(1,2) = c(2,4) = t(1,4) = t_1 = 10$ and the first two capacities are set to finite values. Then, the D matrix is modified to read:

$$D = \begin{bmatrix} * & 0 & \infty & 4 \\ 7 & * & 6 & 0 \\ \infty & 1 & * & 5 \\ 2 & 8 & 2 & * \end{bmatrix}$$

For $\mu = 2$ it follows that $d(\hat{\Pi}_{12}) = 0$, hence $t(1,2)$ must be satisfied and no construction is needed here.

For $\mu = 3$, $\hat{\Pi}_{2,4}$ is selected as the "shorter" path since $d(\hat{\Pi}_{2,4}) < d(\hat{\Pi}_{1,3})$. But $d(\hat{\Pi}_{2,4}) = 0$, therefore, no capacity assignments occur for $t(2,4)$. For $\hat{\Pi}_{1,3} = \{(1,2),(2,4),(4,3)\}$, arc (4,3) has unassigned capacity hence $c(4,3) = t(1,3) = t_3 = 8$. The arc costs are again modified.

$$D = \begin{bmatrix} * & 0 & \infty & 4 \\ 7 & * & 6 & 0 \\ \infty & 1 & * & 5 \\ 2 & 8 & 0 & * \end{bmatrix}$$

For $\mu = 4$, $d(\hat{\Pi}_{3,2}) = 1$ and $d(\hat{\Pi}_{3,4}) = 1$, either may be selected. Taking $\hat{\Pi}_{3,2} = \{(3,2)\}$, then $c(3,2) = t(3,2) = t_4 = 7$ and upon modifying D and recalculating the shortest paths gives $d(\hat{\Pi}_{3,4}) = 0$, that is, capacities need not be built to satisfy $t(3,4)$.

For $\mu = 5$ it is also found that $d(\hat{\Pi}_{2,3}) = 0$ so that $t(2,3)$ is already satisfied, however for $\hat{\Pi}_{2,1} = \{(2,4), (4,1)\}$ put $c(4,1) = t(2,1) = t_5 = 6$ to satisfy the requirement $t(2,1)$.

For $\mu = 6, 7, 8$, all shortest paths have zero length, hence further capacity is not added to the system.

The final network that satisfies the time-shared requirements is given in figure 2.6.3.

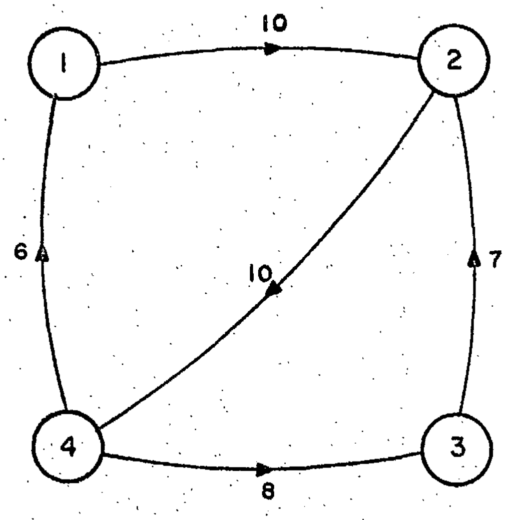

figure 2.6.3

COMMENTS

Construction stops once all nodes have paths to communicate with all others.

Appendix C of this study contains the description of a computer program that implements Algorithm 2.6.1. A program listing and a programmed example is also given in this Appendix.

2.7 CONCLUSIONS

In this chapter it was shown that shortest path techniques can be profitably used to find solutions for communication network problems; moreover, it was shown that in the case of the simultaneous transmission network a minimum cost configuration could be found along with the capacity function.

The time-shared synthesis was shown to be an easily programmed suboptimal procedure for obtaining a satisfactory solution. It is suggested that other algorithms employing the same shortest path techniques be developed to find other suboptimal procedures. Then by comparing the results from several such Algorithms for one set of requirements and costs, the minimum cost network obtained could be selected to give the best suboptimal solution for the time-shared problem. This is left for further study.

CHAPTER III

3.1 SUMMARY

In this chapter, a procedure is presented for synthesizing a time-shared communication network that exactly meets the terminal requirements; that is, the resultant network has no redundant capacity on any of its channels. The conditions under which such a network is realizable are given, and the methods presented permit constraints on the channels to be taken into account. A computer program that implements the synthesis procedure given in this Chapter is documented in Appendix D.

In any "time-slice", if there is no excess capacity in the network for a given terminal requirement t(p,q), that is, if the maximum flow from p to q is exactly t(p,q), then no network with less total capacity can be found that exactly satisfies the terminal requirements. This implies, that in the case where the cost per unit capacity is the same for all arcs, the minimum cost network will be found using the algorithms that follow.

In section 3.2, some necessary preliminaries are given. The algorithm for synthesizing a general network is presented and proved in section 3.3. In section 3.4 some illustrative examples are given, and section 3.5 presents an algorithm for finding the communication network with only non negative capacities should the general network contain negative capacities.

It should be noted that theorems 3.2.1, 3.3.1, 3.5.1 and 3.5.2 have been stated somewhat differently by Resh [14].

3.2 MATHEMATICAL PRELIMINARIES

In this section, some preliminary definitions and theorems are introduced. These results are essential to the development that follows.

Let $t_1, t_2, ---, t_m$, be in increasing order, the distinct values of the terminal capacity function t. Then the set of arcs in A on which t has the value t_μ, will be denoted by A_μ, thus

$$A_\mu = \{(i,j) \varepsilon A \;/\; t(i,j) = t_\mu\} \; ; \; 1 \leq \mu \leq m \tag{3.2.1}$$

and since the number of arcs in A is n(n-1), then $1 \leq m \leq n(n-1)$.

A^*_μ will denote the set of arcs in the union of the sets $A_1, A_2, ---, A_{\mu-1}$, that is,

$$A^*_\mu = A_1 \bigcup A_2 \bigcup --- \bigcup A_{\mu-1} \; ; \; 1 \leq \mu \leq m+1 \tag{3.2.2}$$

Clearly $A^*_1 = \emptyset$ and $A^*_{m+1} = A$.

Given a circuit Π^k_{pp} in G and a real number α, the network $\mathbb{N} = (G,g)$ where,

$$g(i,j) = \begin{cases} \alpha, & \text{if } (i,j) \in \Pi^k_{pp}, \\ -\alpha, & \text{if } (i,j) \in \overline{\Pi^k_{pp}}, \\ 0, & \text{otherwise,} \end{cases} \tag{3.2.3}$$

is called the bicircuit net in G, corresponding to the pair (Π^k_{pp},α). Further, two networks are called "circuit p equivalent" or c_p-equivalent if one of them is the result of the addition to the other of a finite number of bicircuit nets.

Theorem 3.2.1 - Given a network $\mathbf{N} = (G,c,t)$, for every arc (i,j) in A, there exists at least one semicut S_X containing (i,j) such that the restriction t/S_X of the function t attains its maximum on the arc (i,j).

Proof: There is always at least one semicut in G containing arc (i,j), namely, the semicut $S_{\{i\}} = \{i\} \times \{i\}^c$. Furthermore, since G has a finite number of nodes, the number of distinct semicuts containing (i,j) is finite, and for some semicut S_X among them (where S_X is the minimum valued semicut according to the definition of the function t in (1.4.10)), it follows that,

$$t(i,j) = \left|S_X\right|_c \tag{3.2.4}$$

Also from the definition of t, for any $(i',j') \neq (i,j)$,

$$t(i',j') \leq \left|S_X\right|_c \ ; \ \forall (i',j') \in S_X \tag{3.2.5}$$

and combining (3.2.4) and (3.2.5), gives

$$t(i',j') \leq \left|S_X\right|_c = t(i,j), \tag{3.2.6}$$

and therefore that $t(i',j') \ngtr t(i,j)$. Q.E.D.

<u>m - restriction</u>: for any network $\mathbb{N} = (G,c,t)$, by saying that a semicut S_X in G is an <u>m - restriction</u> for some arc (i,j) (with respect to the function t) it is meant that the semicut S_X contains (i,j) and that the restriction t/S_X of the function t attains its maximum at the arc (i,j). When no confusion should arise, it will be simply stated that S_X is an m - restriction for (i,j).

<u>essential equality</u>: for any two arcs (i,j) and (i',j') in the set A, the following equality is true, namely, $t(i,j) = t(i',j') = t_\mu$. If this equality implies that a semicut in G is an m - restriction for (i,j) if and only if S_X is an m - restriction for (i',j'), then it is called an <u>essential equality</u>.

Throughout this chapter, the symbol $\mathcal{S}_\mu$ will denote the set of semicuts in G, each of which is an m - restriction for some arc in A_μ.

THEOREM 3.2.2 Given a network $\mathbb{N} = (G,c,t)$ whose terminal capacity function t contains only essential equalities, if $S_X \in \mathcal{S}_\mu$ is a semicut in G then:

i) S_X is an m - restriction for every arc (i,j) in A_μ and, therefore $A_\mu \subset S_X$ and $A_\mu \subset \mathcal{S}_\mu$.

ii) If $\mu < \mu'$ then $S_X \cap A_{\mu'} = \emptyset$.

PROOF: (i) S_X being in $\mathcal{S}_\mu$ implies that there exists an arc (i,j) in A_μ such that S_X is an m - restriction for (i,j). By the definition of A_μ, $t(i,j) = t(i',j')$ for every arc (i',j') in A_μ and since each such equality is an essential equality, S_X is an m - restriction for every arc (i',j') in A_μ. Then every element in A_μ is an element in S_X and $A_\mu \subset S_X \subset \mathcal{S}_\mu$.

(ii) Assume that $S_X \cap A_{\mu'} \neq \emptyset$. Then there is at least one arc (i,j) such that $(i,j) \in A_{\mu'}$ and $(i,j) \in S_X$. If (i,j) is in $A_{\mu'}$, then $t(i,j) = t_{\mu'}$ and if (i,j) is in S_X then, since the maximum value of t on S_X is t_μ, $t(i,j) < t_\mu$. But $\mu<\mu'$ makes $t_\mu<t_{\mu'}$ and there is a contradiction, hence, $S_X \cap A_{\mu'} = \emptyset$.

LEMMA 3.2.1 $S_X = S_X \cap A^*_{\mu+1}$

PROOF: By THEOREM 3.2.2 (i) and (ii), $A_\mu = S_X \cap A_\mu$ and $(S_X \cap A_{\mu+1}) \cup (S_X \cap A_{\mu+2}) \cup \cdots \cup (S_X \cap A_m) = \emptyset$. Then,

$$
\begin{aligned}
S_X \cap A^*_{\mu+1} &= (S_X \cap A_1) \cup \cdots \cup (S_X \cap A_\mu) \cup \emptyset \\
&= (S_X \cap A_1) \cup \cdots \cup (S_X \cap A_m) \\
&= S_X \cap (A_1 \cup \cdots \cup A_m) \\
&= S_X \cap A \\
&= S_X .
\end{aligned}
$$

<u>LEMMA 3.2.2</u> (i) $S_X \cap A^*_{\mu+1} = A_\mu \cup (S_X \cap A^*_\mu)$

(ii) $A_\mu \cap (S_X \cap A^*_\mu) = \emptyset .$

<u>PROOF:</u>

(i) $$
\begin{aligned}
A_\mu \cup (S_X \cap A^*_\mu) &= (S_X \cap A_\mu) \cup (S_X \cap A^*_\mu) \\
&= S_X \cap A^*_{\mu+1} .
\end{aligned}
$$

(ii) $$
\begin{aligned}
A_\mu \cap (S_X \cap A^*_\mu) &= (A_\mu \cap S_X) \cap A^*_\mu \\
&= A_\mu \cap (A_1 \cup \cdots \cup A_{\mu-1}) \\
&= (A_\mu \cap A_1) \cup \cdots \cup (A_\mu \cap A_{\mu-1}) \\
&= \emptyset .
\end{aligned}
$$

Utilizing the notion of an m - restriction, the following theorem shows that not all of the semicuts indicated in (1.4.10) need be evaluated to obtain the terminal capacity function.

THEOREM 3.2.3 - Given a network $\mathbb{N} = (G,c,t)$ for every $(i,j) \in A$, the following expression is an equivalent definition of the terminal capacity function t.

$$t(i,j) = \min \{ |S_X|_c \ / \ S_X \text{ is an m - restriction for } (i,j) \} \qquad (3.2.7)$$

PROOF: Let S^*_X be any semicut containing (i,j) but without being an m - restriction for (i,j). Then there is some $(i',j') \in S^*_X$, $(i',j') \neq (i,j)$ for which $t(i',j')$ is the maximum value of t/S^*_X. Therefore,

$$t(i,j) < t(i',j') \qquad (3.2.8)$$

Then from (1.4.10), since S^*_X is some semicut containing (i',j')

$$t(i',j') \leq |S^*_X| \qquad (3.2.9)$$

also by (3.2.8), (3.2.9) and the definition (1.4.10), it follows that

$$t(i,j) = \min \{ |S_X|_c \ / \ (i,j) \in S_X \} < |S^*_X| , \qquad (3.2.10)$$

therefore, those semicuts that contain (i,j) and are not m - restrictions for (i,j) do not affect the minimum; hence, (3.2.7) is an equivalent definition for the terminal capacity function.

In the synthesis algorithms, the capacity function is subscripted, that is $c_\mu : A \to R, \mu = 0,1,---,m$. This is because the algorithm is an accumulative process that begins with a network with no capacity and then adds capacity on various arcs. Then each c_μ represents the value of the capacity function at iteration μ; moreover when $\mu = m$, the procedure terminates and then $c = c_m$, the required capacity function.

3.3 ALGORITHM FOR SYNTHESIZING A NETWORK

Given a finite, quasicomplete, oriented graph $G = (N,A)$; given also the terminal capacity function $t : A \to R^+$ such that t contains only essential inequalities and that for every arc (i,j) in A there exists a semicut S_X in G which is an m - restriction for (i,j); then a network $\mathcal{N} = (G,c,t)$ where the capacity function c "realizes" t, is obtained as follows:

Algorithm 3.3.1

1. a) For each of the m distinct values t_μ in t, find the set of arcs $A_\mu \subset A$ on which t has the value t_μ.

b) For each A_μ, find the set of semicuts $\mathcal{S}_\mu$ where each $S_X \varepsilon \mathcal{S}_\mu$ is an m - restriction for all $(i,j) \varepsilon A_\mu$.

c) For all $(i,j) \varepsilon A$ put $c_o(i,j) = 0$.

d) Put $\mu = 1$.

2. For all arcs $(i,j) \varepsilon A$,

if $(i,j) \notin A_\mu$ then $c_\mu(i,j) = c_{\mu-1}(i,j)$

otherwise $c_\mu(i,j) = x(i,j)$ where,

$$\sum_{(i,j)} \{x(i,j) \mathbin{/} (i,j) \varepsilon A_\mu\} = t_\mu - \min \{\left|S_X\right|_{c_{\mu-1}} \mathbin{/} S_X \varepsilon \mathcal{S}_\mu\} \tag{3.3.1}$$

3. If $\mu = m$ then $c(i,j) = c_m(i,j)$ and terminate, otherise put $\mu = \mu+1$ and go to step 2.

Observe that on termination, c_m is the required capacity function c.

The following theorem proves that Algorithm 3.3.1 obtains the capacity function c for the network $\mathcal{N}$.

<u>THEOREM 3.3.1</u> Given the network $\mathcal{N} = (G,c,t)$ as defined above, Algorithm 3.3.1 finds the capacity function c so that for all $(i,j) \varepsilon A$

$$t(i,j) = \min \{\left|S_X\right|_c \mathbin{/} (i,j) \varepsilon S_X\} \tag{1.4.10}$$

that is, the terminal capacity function obtained by evaluating the semicuts in the resultant network is exactly the same as the terminal capacity function given before the synthesis procedure begins.

PROOF: According to Theorem 3.2.3, it is enough to show that for all $(i,j) \in A$,

$$t(i,j) = \min \{ |S_X|_c / \; S_X \text{ is an } m \text{ - restriction for } (i,j) \} \qquad (3.3.2)$$

Now, from step 1-a), the values of t are the numbers $t_i, ---, t_m$. Since t contains only essential equalities, then S_X is an m - restriction for all $(i,j) \in A_\mu$ and the minimum valued semicut in (3.3.2) is the same for all arcs in A_μ, hence it is enough to show that for every $\mu=1,2,---,m$,

$$t_\mu = \min \{ |S_X|_c / \; S_X \text{ is an } m \text{ - restriction for any } (i,j) \in A_\mu \} \qquad (3.3.3)$$

Moreover, step 1-b) selects the semicuts that are m - restrictions for $(i,j) \in A_\mu$ and places them in $\int_\mu$; therefore instead of showing (3.3.3) it is only necessary to show that for every $\mu=1,2,---,m$,

$$t_\mu = \min \{ |S_X|_c / \; S_X \in \int_\mu \} \qquad (3.3.4)$$

For any semicut $S_X \in \int_\mu$ the value of S_X can be written as follows:

$$|S_X|_c = |S_X|_{c_m}$$

$$= \Sigma\{c_m(i,j) \; / \; (i,j) \; \varepsilon \; S_X\}$$

$$= \Sigma\{c_m(i,j) \; / \; (i,j) \; \varepsilon \; S_X \bigcap A^*_{\mu+1}\} \qquad \text{(by Lemma 3.2.1)}$$

$$= \Sigma\{c_m(i,j) \; / \; (i,j) \; \varepsilon \; A_\mu$$

$$+ \Sigma\{c_m(i,j) \; / \; (i,j) \; \varepsilon \; S_X \bigcap A^*_\mu\} \qquad \text{(by Lemma 3.2.2)}$$

Since $(i,j) \; \varepsilon \; A_\mu$ implies that $(i,j) \not\varepsilon \; A_{\mu+1} \bigcup --- \bigcup A_m$, by the definition of c_μ in step 2, for every $(i,j) \; \varepsilon \; A_\mu$,

$$c_m(i,j) = c_{m-1}(i,j) = --- = c_\mu(i,j) = x(i,j) \tag{3.3.5}$$

Since $(i,j) \; \varepsilon \; S_X \bigcap A^*_\mu$ implies that $(i,j) \not\varepsilon \; A_\mu \bigcup --- \bigcup A_m$, by the definition of c_μ once more, for every $(i,j) \; \varepsilon \; S_X \bigcap A^*_\mu$,

$$c_m(i,j) = c_{m-1}(i,j) = --- = c_{\mu-1}(i,j). \tag{3.3.6}$$

Utilizing (3.3.5) and (3.3.6) yields:

$$|S_X|_c = \Sigma\{x(i,j) \; / \; (i,j) \; \varepsilon \; A_\mu\}$$

$$+ \Sigma\{c_{\mu-1}(i,j) \; / \; (i,j) \; \varepsilon \; S_X \bigcap A^*_\mu\} \tag{3.3.7}$$

The expression for $|S_X|_{c_{\mu-1}}$ can be expanded as follows by using Lemma 3.2.1; Lemma 3.2.2 and by observing that since $(i,j) \in A_\mu$ then $(i,j) \notin A^*_\mu = A_1 \bigcup --- \bigcup A_{\mu-1}$ and by the definition of c_μ, $c_{\mu-1}(i,j) = \cdots = c_o(i,j) = 0$ for every $(i,j) \in A_\mu$.

$$|S_X|_{c_{\mu-1}} = \Sigma\{ c_{\mu-1}(i,j) \ / \ (i,j) \in S_X \}$$

$$= \Sigma\{ c_{\mu-1}(i,j) \ / \ (i,j) \in A_\mu \}$$

$$+ \Sigma\{ c_{\mu-1}(i,j) \ / \ (i,j) \in S_X \bigcap A^*_\mu \}$$

$$= \Sigma\{ c_{\mu-1}(i,j) \ / \ (i,j) \in S_X \bigcap A^*_\mu \} \qquad (3.3.8)$$

Substituting (3.3.8) into (3.3.7), for every semicut S_X in $\mathcal{S}_\mu$,

$$|S_X|_c = \Sigma\{x(i,j) \ / \ (i,j) \in A_\mu\} + |S_X|_{c_{\mu-1}} \qquad (3.3.9)$$

From (3.3.1) in Algorithm 3.3.1, t_μ is given by,

$$t_\mu = \Sigma\{x(i,j) \ / \ (i,j) \in A_\mu\} + \min \{ |S_X|_{c_{\mu-1}} \ / \ S_X \in \mathcal{S}_\mu \}$$

$$= \min [\Sigma\{x(i,j) \ / \ (i,j) \in A_\mu\}$$

$$+ \{ |S_X|_{c_{\mu-1}} \ / \ S_X \in \mathcal{S}_\mu\}] \qquad (3.3.10)$$

since the values in x are real constants (3.3.9) and (3.3.10) together yield (3.3.4). Q.E.D.

COMMENTS

(i) Theorem 3.2.1 establishes that in a network where c is known and t is obtained from the semicuts evaluated with respect to c, each terminal capacity in t has an m - restriction associated with it. Consequently, if a network is to be synthesized, then to each of the values in t it is necessary that there correspond at least one m - restricted semicut, otherwise the network is not realizable.

The synthesis procedure requires that only essential equalities be present in the terminal capacity function. This means that any semicut that is an m - restriction for some arc in A_μ must also be an m - restriction for all other arcs in A_μ. Observe however that if the m - restriction condition is relaxed, two terminal capacities without m - restrictions can still be essentially equal. In such a case, at least one among the sets $\int_\mu$ will be empty, the function x in (3.3.1) is not defined and therefore, the synthesis procedure cannot be applied.

If the function t contains non-essential equalities while the m - restriction condition is satisfied for all arcs in A; t can always be perturbed slightly to obtain a new function t^ε which satisfies the m - restriction condition and contains only essential inequalities. This is done by inspecting t and changing the required values in t by a small amount to obtain t^{ε} so that the essential equality condition is satisifed and the m - restriction condition is maintained. Then any realization of t^{ε} reduces to a realization of t as ε approaches zero. This approach is further discussed and illustrated in Example 3.4.2.

(ii). In the synthesis procedure of Algorithm 3.3.1, the function x used in the definition of c_{μ} is not uniquely defined (it is so only in the case where each A_{μ} has only one element, that is, only in the case where there are $m = n(n-1)$ distinct values of the function t). This suggests that possibly more than one realization of the given function t could exist. In such cases, it is natural to admit the presence of constraints on the arcs of the network. Then, among the possible realizations, the required ones are those satisfying the constraints on the arcs of the network. This is a bonafide optimum synthesis procedure.

Suppose that each of the arcs (i_1,j_1), (i_2,j_2),$\cdots$, (i_k,j_k) is an element of A_{μ} and $t(i_1,j_1)$, $t(i_2,j_2)$, ---, $t(i_k,j_k)$ are essentially equal subject to the m - restriction condition; then the most simple constraint is to select one of these arcs as one that is "preferred" to the others. Since the synthesis procedure obtains an optimal solution for the uniform cost case the function d can be redefined, that is, the constraints in the synthesis of a network can be conveniently represented by the values in the <u>constraint function</u> $d:A \rightarrow R^+$. Hence for the arcs A_μ, choosing the minimum of the values $d(i,j)$, $(i,j) \in A_\mu$, is the optimization criterion; that is, $x(i,j)$ takes on the assignment in (3.3.1) such that the arc (i,j) corresponds to the arc in A_μ with the minimum valued constraint. This is illustrated in Example 3.4.3.

3.4 SOME EXAMPLES USING ALGORITHM 3.3.1

In what follows, three examples illustrating the synthesis procedure are given. In the first example, the function t contains only essential equalities and there is no constraint function, that is, the assignment in (3.3.1) is arbitrary when there is more than one arc in A_μ. The second example is one in which t must be perturbed since some non-essential equalities are present. The third example considers the same function t as in the first example except with constraints.

Matrix notation is used for the functions, that is, d, c and t are represented by the matrices of values, D, C and T. A simpler notation for semicuts is also used; for example the semicut $\{(2,1), (2,3), (4,1), (4,3)\}$ is denoted by 24/13 (numbers greater than 9 are not used here to represent nodes).

<u>Example 3.4.1</u> - The capacity matrix C realizing the terminal capacity matrix T under no constraints is obtained by using Algorithm 3.3.1.

$$T = \begin{bmatrix} * & 2 & 2 & 2 \\ 3 & * & 4 & 6 \\ 3 & 7 & * & 8 \\ 3 & 5 & 4 & * \end{bmatrix}$$

From steps 1-a) and 1-b) of the algorithm, t_μ, A_μ and $\mathcal{S}_\mu$ are determined, and are

$$\begin{array}{lll}
t_1 = 2, & A_1 = \{(1,2),(1,3),(1,4)\}, & \mathcal{S}_1 = \{1/234\}, \\
t_2 = 3, & A_2 = \{(2.1),(3,1),(4,1)\}, & \mathcal{S}_2 = \{234/1\}, \\
t_3 = 4, & A_3 = \{(2,3),(4,3)\}, & \mathcal{S}_3 = \{124/3, 24/13\}, \\
t_4 = 5, & A_4 = \{(4,2)\}, & \mathcal{S}_4 = \{14/23, 4/123\}, \\
t_5 = 6, & A_5 = \{(2,4)\}, & \mathcal{S}_5 = \{12/34, 2/134\}, \\
t_6 = 7, & A_6 = \{(3,2)\}, & \mathcal{S}_6 = \{134/2, 34/12\}, \\
t_7 = 8, & A_7 = \{(3,4)\}, & \mathcal{S}_7 = \{123/4, 23/14, 13/24, \\
 & & \quad 3/124\}.
\end{array}$$

By simple inspection of each triple $(t_\mu, A_\mu, \mathcal{S}_\mu)$ for $\mu = 1,---,7$, t contains only essential equalities and the m - restriction condition is satisfied for all arcs.

Therefore, a capacity function realizing t can be obtained.

Using step 1-c) the initial capacity matrix is

$$C_o = \begin{bmatrix} * & 0 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}$$

From step 2 of the algorithm, for $\mu = 1$, $x(1,2) + x(1,3) + x(1,4) = 2 - \min\{0\} = 2$. Then taking $x(1,2) = 0$, $x(1,3) = 2$, $x(1,4) = 0$, as one possible definition of the function x on A_1 gives,

$$C_1 = \begin{bmatrix} * & 0 & 2 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}$$

For $\mu = 2$, $x(2,1) + x(3,1) + x(4,1) = 3 - \min\{0\} = 3$. Taking $x(2,1) = 3$, $x(3,1) = 0$, $x(4,1) = 0$, then,

$$C_2 = \begin{bmatrix} * & 0 & 2 & 0 \\ 3 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}$$

For $\mu = 3$, $x(2,3) + x(4,3) = 4 - \min\{2,3\} = 2$. Hence putting $x(2,3) = 0$, $x(4,3) = 2$ gives,

$$C_3 = \begin{bmatrix} * & 0 & 2 & 0 \\ 3 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 2 & * \end{bmatrix}$$

For $\mu = 4$, $x(4,2) = 5 - \min\{4,2\} = 3$ and then the only definition of the function x on A_4 is $x(4,2) = 3$ and,

$$C_4 = \begin{bmatrix} * & 0 & 2 & 0 \\ 3 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 3 & 2 & * \end{bmatrix}$$

For $\mu = 5$, $x(2,4) = 6 - \min\{2,3\} = 4$,

$$C_5 = \begin{bmatrix} * & 0 & 2 & 0 \\ 3 & * & 0 & 4 \\ 0 & 0 & * & 0 \\ 0 & 3 & 2 & * \end{bmatrix}$$

For $\mu = 6$, $x(3,2) = 7 - \min\{3,3\} = 4$,

$$C_6 = \begin{bmatrix} * & 0 & 2 & 0 \\ 3 & * & 0 & 4 \\ 0 & 4 & * & 0 \\ 0 & 3 & 2 & * \end{bmatrix}$$

Finally, for $\mu = 7$, $x(3,4) = 8 - \min\{4,7,4,4\} = 4$,

$$C_7 = \begin{bmatrix} * & 0 & 2 & 0 \\ 3 & * & 0 & 4 \\ 0 & 4 & * & 4 \\ 0 & 3 & 2 & * \end{bmatrix}$$

Then $C = C_7$ represents the capacity matrix that realizes the terminal capacity entries in T. From C the network representation is shown in figure 3.4.1.

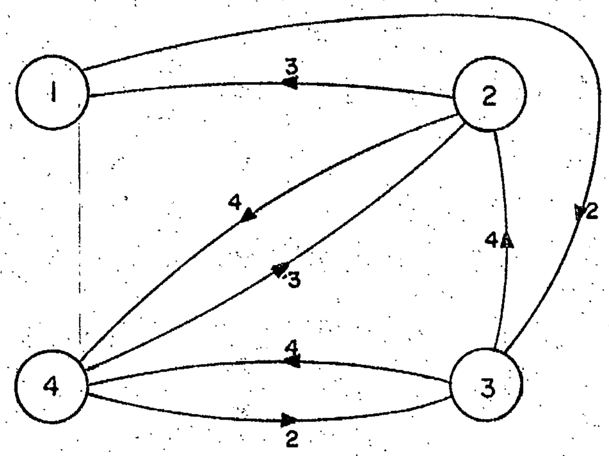

figure 3.4.1.

<u>Example 3.4.2</u> - The capacity matrix C realizing the terminal capacity matrix T is obtained using the synthesis procedure. Again no constraints are given.

$$T = \begin{bmatrix} * & 1 & 1 & 1 \\ 4 & * & 6 & 5 \\ 4 & 6 & * & 5 \\ 4 & 6 & 6 & * \end{bmatrix}$$

However in this example the matrix T contains non-essential equalities. Indeed, the equality $t(2,3) = t(3,2)$ implies that the arcs (2,3) and (3,2) should be put in the same A_μ while no semicut can contain both those arcs.

Then, consider instead the matrix T^ε corresponding to t^ε , the perturbed terminal capacity function,

$$T^\varepsilon = \begin{bmatrix} * & 1 & 1 & 1 \\ 4 & * & 6 & 5 \\ 4 & 6+\varepsilon_1 & * & 5 \\ 4 & 6+\varepsilon_2 & 6+\varepsilon_3 & * \end{bmatrix}$$

where $\varepsilon_1, \varepsilon_2, \varepsilon_3$ are arbitrarily small numbers such that $0<\varepsilon_1<\varepsilon_2<\varepsilon_3$. Now only essential equalities are present.

Following the procedure of Example 3.4.1, a capacity matrix C^{ε} realizing T^{ε} is constructed. Letting $\varepsilon_1, \varepsilon_2$ and ε_3 tend to zero in the matrix C^{ε}, the matrix C_X realizing T, is obtained.

From step 1 in Algorithm 3.3.1,

$$
\begin{array}{lll}
t_1 = 1, & A_1 = \{(1,2),(1,3),(1,4)\}, & \mathcal{J}_1 = \{1/234\}, \\
t_2 = 4, & A_2 = \{(2,1),(3,1),(4,1)\}, & \mathcal{J}_2 = \{234/1\}, \\
t_3 = 5, & A_3 = \{(2,4),(3,4)\}, & \mathcal{J}_3 = \{23/14, 123/4\}, \\
t_4 = 6, & A_4 = \{(2,3)\}, & \mathcal{J}_4 = \{2/134, 12/34\}, \\
t_5 = 6+\varepsilon_1, & A_5 = \{(3,2)\}, & \mathcal{J}_5 = \{3/124, 13/24\}, \\
t_6 = 6+\varepsilon_2, & A_6 = \{(4,2)\}, & \mathcal{J}_6 = \{34/12, 134/2\}, \\
t_7 = 6+\varepsilon_3, & A_7 = \{(4,3)\}, & \mathcal{J}_7 = \{4/123, 14/23, \\
 & & \qquad 24/13, 124/3\},
\end{array}
$$

$$
C_0^{\varepsilon} = \begin{bmatrix} * & 0 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}
$$

For $\mu = 1$, $x(1,2) + x(1,3) + x(1,4) = 1 - \min\{0\} = 1$; and putting $x(1,2) = 1$, $x(1,3) = 0$, $x(1,4) = 0$,

$$C_1^{\varepsilon} = \begin{bmatrix} * & 1 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}$$

For $\mu = 2$, $x(2,1) + x(3,1) + x(4,1) = 4 - \min\{0\} = 4$; and putting $x(2,1) = 4$, $x(3,1) = 0$, $x(4,1) = 0$

$$C_2^{\varepsilon} = \begin{bmatrix} * & 1 & 0 & 0 \\ 4 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}$$

For $\mu = 3$, $x(2,4) + x(3,4) = 5 - \min\{4,0\} = 5$, If $x(2,4) = 5$ then $x(3,4) = 0$,

$$C_3^{\varepsilon} = \begin{bmatrix} * & 1 & 0 & 0 \\ 4 & * & 0 & 5 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}$$

For $\mu = 4$, $x(2,3) = 6 - \min\{9,5\} = 1$,

$$C_4^{\varepsilon} = \begin{bmatrix} * & 1 & 0 & 0 \\ 4 & * & 1 & 5 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}$$

For $\mu = 5$, $x(3,2) = 6+\varepsilon_1 - \min\{0,1\} = 6+\varepsilon_1$,

$$C_5^{\varepsilon} = \begin{bmatrix} * & 1 & 0 & 0 \\ 4 & * & 1 & 5 \\ 0 & 6+\varepsilon_1 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}$$

For $\mu = 6$, $x(4,2) = 6+\varepsilon_2 - \min\{6 + \varepsilon_1, 7 + \varepsilon_1\} = \varepsilon_2 - \varepsilon_1$,

$$C_6^{\varepsilon} = \begin{bmatrix} * & 1 & 0 & 0 \\ 4 & * & 1 & 5 \\ 0 & 6+\varepsilon_1 & * & 0 \\ 0 & \varepsilon_2-\varepsilon_1 & 0 & * \end{bmatrix}$$

For $\mu = 7$, $x(4,3) = 6 + \varepsilon_3 - \min\{\varepsilon_2-\varepsilon_1, 1 + \varepsilon_2 - \varepsilon_1, 5, 1\}$

$$= 6 + \varepsilon_3 + \varepsilon_1 - \varepsilon_2,$$

$$C_7^{\varepsilon} = \begin{bmatrix} * & 1 & 0 & 0 \\ 4 & * & 1 & 5 \\ 0 & 6+\varepsilon_1 & * & 0 \\ 0 & \varepsilon_2-\varepsilon_1 & 6+\varepsilon_3+\varepsilon_1-\varepsilon_2 & * \end{bmatrix}$$

Then letting ε_1, ε_2 and ε_3 tend to zero, the desired capacity matrix is,

$$C = \begin{bmatrix} * & 1 & 0 & 0 \\ 4 & * & 1 & 5 \\ 0 & 6 & * & 0 \\ 0 & 0 & 6 & * \end{bmatrix}$$

and this network has the following representation:

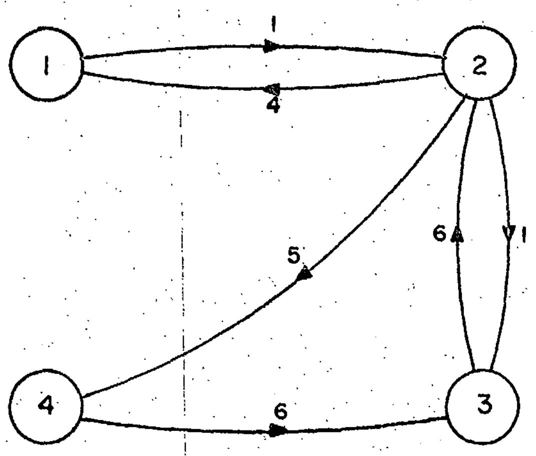

figure 3.4.2

Example 3.4.3 - The capacity matrix C, realizing the matrix T under the constraints in D is obtained as follows:

$$T = \begin{bmatrix} * & 2 & 2 & 2 \\ 3 & * & 4 & 6 \\ 3 & 7 & * & 8 \\ 3 & 5 & 4 & * \end{bmatrix} , \quad D = \begin{bmatrix} * & 2 & 10 & 100 \\ 3 & * & 3 & 8 \\ 100 & 5 & * & 2 \\ 1 & 5 & 2 & * \end{bmatrix}$$

The procedure to be followed is that of Example 3.4.1 with the only difference being, that for the definition of the function x on the set A_μ, (μ=1,---,7), the constraint matrix is taken into account.

Initially,

$$C_o = \begin{bmatrix} * & 0 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}$$

For $\mu = 1$, $x(1,2) + x(1,3) + x(1,4) = 2 - \min\{0\} = 2$ and since $d(1,2) < d(1,3) < d(1,4)$, putting $x(1,2) = 2$, $x(1,3) = 0$, $x(1,4) = 0$,

$$C_1 = \begin{bmatrix} * & 2 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix}$$

For $\mu = 2$, $x(2,1) + x(3,1) + x(4,1) = 3 - \min\{0\} = 3$ and according to the matrix D we must let $x(4,1) = 3$, $x(2,1) = 0$, $x(3,1) = 0$, and therefore

$$C_2 = \begin{bmatrix} * & 2 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 3 & 0 & 0 & * \end{bmatrix}$$

For $\mu = 3$, $x(2,3) + x(4,3) = 4 - \min\{0,3\} = 4$ and since $d(4,3) < d(2,3)$, $x(4,3) = 4$ while $x(2,3) = 0$, and,

$$C_3 = \begin{bmatrix} * & 2 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 3 & 0 & 4 & * \end{bmatrix}$$

For $\mu = 4$, $x(4,2) = 5 - \min\{6,7\} = -1$, that is, no choice can be made and,

$$C_4 = \begin{bmatrix} * & 2 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 3 & -1 & 4 & * \end{bmatrix}$$

For $\mu = 5$, $x(2,4) = 6 - \min\{0,0\} = 6$,

$$C_5 = \begin{bmatrix} * & 2 & 0 & 0 \\ 0 & * & 0 & 6 \\ 0 & 0 & * & 0 \\ 3 & -1 & 4 & * \end{bmatrix}$$

For $\mu = 6$, $x(3,2) = 7 - \min\{1,2\} = 6$,

$$C_6 = \begin{bmatrix} * & 2 & 0 & 0 \\ 0 & * & 0 & 6 \\ 0 & 6 & * & 0 \\ 3 & -1 & 4 & * \end{bmatrix}$$

For $\mu = 7$, $x(3,4) = 8 - \min\{6,6,8,6\} = 2$,

$$C_7 = \begin{bmatrix} * & 2 & 0 & 0 \\ 0 & * & 0 & 6 \\ 0 & 6 & * & 2 \\ 3 & -1 & 4 & * \end{bmatrix}$$

Therefore, the required matrix is $C = C_7$ and the corresponding network has the following representation.

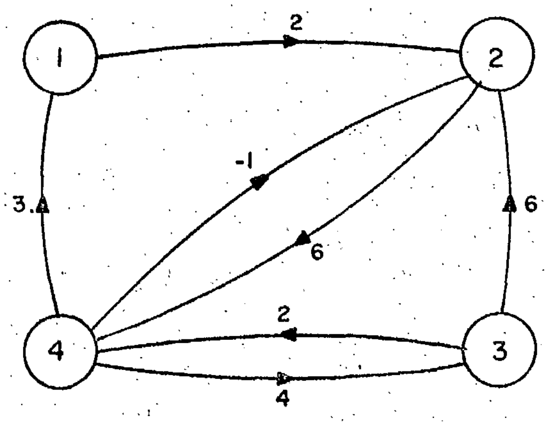

figure 3.4.3

3.5 SYNTHESIZING A COMMUNICATION NETWORK

The algorithm used for the synthesis of a network realizing the given terminal capacity function, can be used for the synthesis of communication networks. It has been assumed that the restriction for the communication network is that all values of t and c be non-negative.

As in Example 3.4.3, Algorithm 3.3.1 may construct a network in which some of the capacities are negative real numbers. Since, in such cases, the synthesized network is not a communication network, it is of interest to know if there is an equivalent communication network.

In what follows, the conditions under which such an equivalent communication network exists are given, an algorithm for obtaining such a network is presented and an example illustrating the procedure is given.

<u>THEOREM 3.5.1</u> - If two networks are C_p - equivalent, then they are equivalent (that is, they have the same terminal capacity function).

<u>PROOF</u>: Let N_1 and N_2 be the two C_p - equivalent networks where,

$$N_1 = (G, c_1) \; ; \quad N_2 = (G, c_2) \tag{3.5.1}$$

If $N_{g_1} = (G, g_1)$, ---, $N_{g_k} = (G, g_k)$ are the bicircuit nets that are applied to N_1 to give N_2, then,

$$c_2 = c_1 + g_1 + --- + g_k \tag{3.5.2}$$

To show that N_1 and N_2 are equivalent, it is enough to show that for each semicut S_X in G,

$$\left|S_X\right|_{c_1} = \left|S_X\right|_{c_2} \tag{3.5.3}$$

But (3.5.2) and (3.5.3) yields,

$$\left|S_X\right|_{c_2} = \left|S_X\right|_{c_1+g_1+---+g_k}$$

$$= \left|S_X\right|_{c_1} + \left|S_X\right|_{g_1} + --- + \left|S_X\right|_{g_k} \tag{3.5.4}$$

and it is only necessary to show that,

$$\left|S_X\right|_{g_\lambda} = 0, \text{ for every } \lambda = 1, 2, ---, k \tag{3.5.5}$$

to get (3.5.3).

It has been shown that $N_{g_\lambda} = (G, g_\lambda)$ is a bircuit net for each $\lambda = 1, 2, ---, k$, if

$$g_\lambda(i,j) = \begin{cases} \alpha_\lambda & , \text{ if } (i,j) \in \Pi^h_{pp}, \\ -\alpha_\lambda & , \text{ if } (i,j) \in \overline{\Pi^h_{pp}}, \\ 0 & , \text{ otherwise,} \end{cases} \tag{3.5.6}$$

where α_λ is some real number and Π^h_{pp} is some circuit in G corresponding to g_λ.

For each S_X in G since,

$$(S_X \cap \Pi^h_{pp}) \cap (S_X \cap \overline{\Pi^h_{pp}}) = S_X \cap (\Pi^h_{pp} \cap \overline{\Pi^h_{pp}}) = \emptyset, \tag{3.5.7}$$

then,

$$\begin{aligned}
|S_X|_{g_\lambda} &= \Sigma\{g_\lambda(i,j) \;/\; (i,j) \in S_X\} \\
&= \Sigma\{g_\lambda(i,j) \;/\; (i,j) \in S_X \cap (\Pi^h_{pp} \cup \overline{\Pi^h_{pp}})\} \\
&= \Sigma\{g_\lambda(i,j) \;/\; (i,j) \in (S_X \cap \Pi^h_{pp}) \cup (S_X \cap \overline{\Pi^h_{pp}})\} \\
&= \Sigma\{g_\lambda(i,j) \;/\; (i,j) \in (S_X \cap \Pi^h_{pp})\} + \Sigma\{g_\lambda(i,j)/(i,j) \in (S_X \cap \overline{\Pi^h_{pp}})\} \\
&= \alpha_\lambda \cdot \text{card } (S_X \cap \Pi^h_{pp}) + (-\alpha_\lambda) \text{ card } (S_X \cap \overline{\Pi^h_{pp}}) \\
&= \alpha_\lambda \cdot [\text{card } (S_X \cap \Pi^h_{pp}) - \text{card } (S_X \cap \overline{\Pi^h_{pp}})] \\
&= 0
\end{aligned}$$

Thus (3.5.5) is true and the theorem is proved.

The following algorithm takes a network with some negative capacities on the arcs and obtains a communication network that is equivalent to that network.

<u>Algorithm 3.5.1</u>

1-a) For all $(i,j) \in A$, let $c_o(i,j) = c(i,j)$.

b) Put $\mu = 0$.

2- Locate some arc (p,q) such that $c_\mu(p,q) < 0$. If no such arc exists then terminate since c_μ is the capacity function of the required communication network.

3- Locate some path Π^k_{pq} $(p,q) \notin \Pi^k_{pq}$, and for any (i,j) in the path, $c(i,j) \neq 0$ and at least one such arc has positive capacity.

4-a) Then $\Pi^k_{pp} = \Pi^k_{pq} \bigcup \{(q,p)\}$ and $\mathbb{N}_g = (G,g)$ is the bicircuit net that corresponds to the pair (Π^k_{pp},α) where

$$\alpha = -\min \{c_\mu(i,j) \ / \ (i,j) \in \Pi^k_{pp},\ c_\mu(i,j) > 0\}.$$

b) Put $c_{\mu+1}(i,j) = c_\mu(i,j) + g(i,j) \ \forall (i,j) \in A$.

c) Put $\mu = \mu+1$.

d) If $c_\mu(p,q) < 0$ then go to step 3, otherwise, $c_o(i,j) = c_\mu(i,j) \ \forall (i,j) \in A$, $\mu = 0$ and go to step 2.

During the course of the following theorem, Algorithm 3.5.1 is shown to be valid.

THEOREM 3.5.2 - A network $\mathbb{N} = (G,c,t)$ is C_p-equivalent to a communication network $\mathbb{N}_m = (G,\ c_m,\ t_m)$, if and only if, $t(i,j)$ and $c(i,j) + c(j,i)$ are both non-negative for each $(i,j) \in A$.

PROOF: Suppose the network $\mathbb{N}$ is C_p-equivalent to $\mathbb{N}_m$. Since $\mathbb{N}_m$ is a communication network, $t_m(i,j) \geq 0$ for all $(i,j) \in A$. By theorem 3.5.1, $\mathbb{N}$ and $\mathbb{N}_m$ are also equivalent, hence $t_m(i,j) = t(i,j) \geq 0$.

Now consider any arc (i,j) in A and apply to $\mathbb{N}_m$ some bicircuit net $\mathbb{N}_g = (G,g)$ corresponding to the pair (Π^k_{pp},α) where $(i,j) \in \Pi^k_{pp}$ and α is a constant. Then $c_\mu = c_m + g$ is the capacity function of the resultant network and since all values in c_m are positive, $c_\mu(i,j) = c_m(i,j) + g(i,j) = c_m(i,j) + \alpha$, $c_\mu(j,i) = c_m(j,i) + g(j,i) = c_m(j,i) - \alpha$, hence, $c_\mu(i,j) + c_\mu(j,i) = c_m(i,j) + c_m(j,i)$. Since a finite number of applications of bicircuit nets results in $\mathbb{N}$, then, $c(i,j) + c(j,i) > 0$.

Now suppose that $t(i,j)$ and $c(i,j) + c(j,i)$ are both non-negative for each $(i,j) \in A$. In order to show that $\mathbb{N}$ is C_p-equivalent and hence equivalent to $\mathbb{N}_m$, it is enough to show that if $\mathbb{N}$ contains arcs with negative capacities, then it is C_p-equivalent to a network in which the number of arcs with negative capacities is one less than the number of such arcs in $\mathbb{N}$. Continuing in this manner, a negative valued capacity is eliminated in each new C_p-equivalent network until all capacities are positive and the C_p-equivalent communication network is obtained. This procedure is just the one that is performed by Algorithm 3.5.1; hence, proving the above proposition, also proves the algorithm.

It is only necessary, then, to show that if $(p,q) \in A$ such that $c(p,q) < 0$, then it is possible to construct a finite sequence $\mathbb{N}_o, \mathbb{N}_1$, ---, $\mathbb{N}_s$ of nets where $\mathbb{N}_\mu = (G, c_\mu)$, $\mu = 0,1,---,s$, such that $\mathbb{N}_\mu$ is C_p-equivalent to $\mathbb{N}$ for each $=1,2,---,s$ and $c_s\ (p,q) > 0$. It is proposed, then, that Algoritbm 3.5.1 constructs such a sequence.

First of all note that the algorithm starts off with $c_o = c$ and then builds new capacity functions by applying selected bicircuit nets. Hence, in general, $\mathbb{N}_\mu = (G, c_\mu)$ and $\mathbb{N}_{\mu-1} = (G, c_{\mu-1})$ are C_p-equivalent and in particular, $\mathbb{N}$ and $\mathbb{N}_s$ are C_p-equivalent.

In step 4 of the algorithm, since α is always negative (assuming that an appropriate Π^k_{pq} can be found in step 3), the capacity on arc (p,q) is increased at each iteration of step 4-b), that is, $c_\mu(p,q)$, $\mu = 0,1,---,s$, increases monotonically towards zero. Note however, that if $c(p,q)$ + $c(p,q) < 0$, then α cannot be guaranteed to be strictly negative since $c(q,p)$ may be reduced to zero and, subsequently $c(p,q)$ cannot be increased further towards zero. Thus $c(i,j) + c(j,i) \geq 0$ for all $(i,j) \in A$ guarantees this monitonic increase towards zero.

At each iteration of step 4 either a) $c_{\mu}(q,p)$ is the minimum valued capacity on Π^k_{pp} and hence $c_{\mu+1}(p,q)$ is positive and $c_{\mu+1}(q,p) = 0$ or b) some arc in Π^k_{pp} other than (q,p) has its corresponding capacity reduced to zero. It follows then, that if the capacity of arc (p,q) is not increased to a positive value at iteration $\mu+1$, then there is one less appropriate path (a path that has no zero capacities and at least one position capacity) at iteration $\mu+1$.

It remains only to show that for $c_{\mu}(p,q) < 0$ an appropriate path can always be found. If this is true, since the number of paths and hence the number of appropriate paths, is finite, it follows from the monotonic property that $c_s(p,q) \geq 0$ for some finite $\mu = s$.

Suppose that for $c_{\mu}(p,q) < 0$ no appropriate path can be found. Then all paths joining p to q have either all the capacities with negative values or at least one arc with zero capacity. Therefore, since $c_{\mu}(p,q)$ is also negative, no path from p to q supports positive flow and $t(p,q) < 0$. But this is a contradiction and it is concluded that an appropriate path can always be found if $c_{\mu}(p,q) < 0$.

Q.E.D.

The following example uses the results from Example 3.4.3 to illustrate the use of Algorithm 3.5.1.

Example 3.5.1 - From Example 3.4.3 we have:

$$T = \begin{bmatrix} * & 2 & 2 & 2 \\ 3 & * & 4 & 6 \\ 3 & 7 & * & 8 \\ 3 & 5 & 4 & * \end{bmatrix} \qquad C = \begin{bmatrix} * & 2 & 0 & 0 \\ 0 & * & 0 & 6 \\ 0 & 6 & * & 2 \\ 3 & -1 & 4 & * \end{bmatrix}$$

Note that although the elements of T are non-negative, there is one entry in C, namely c(4,2), that is negative. Since c(4,2) + c(2,4) = 5 is a positive number, the conditions of Theorem 3.5.2 are satisfied and a communication network N_m equivalent to the network defined by T and C, can be found.

Step 1 of the algorithm identifies $c_o = c$ as the first capacity configuration in the sequence and step 2, locates arc (4,2) as the first (and only one) with negative capacity.

Step 3 locates path $\Pi^1_{42} = \{(4,1), (1,2)\}$ as an appropriate path, that is, one with at least one positive capacity and no zero valued capacities.

Step 4 evaluates α,

$$\alpha = -\min \{c_o(4,1), c_o(1,2), c_o(2,4)\}$$
$$= -2$$

and then adds to the net $N_0 = (G, c_0)$ the bicircuit net corresponding to the pair $(\Pi^1_{42}, -2)$ which gives the network $N_1 = (G, c_1)$ where,

$$C_1 = \begin{bmatrix} * & 0 & 0 & 2 \\ 2 & * & 0 & 4 \\ 0 & 6 & * & 2 \\ 1 & 1 & 4 & * \end{bmatrix}$$

Since there are no negative capacities in C_1, $C_m = C_1$ and the communications network $N_m = (G, c_m)$ has been found. This graph representation is shown in figure 3.5.1.

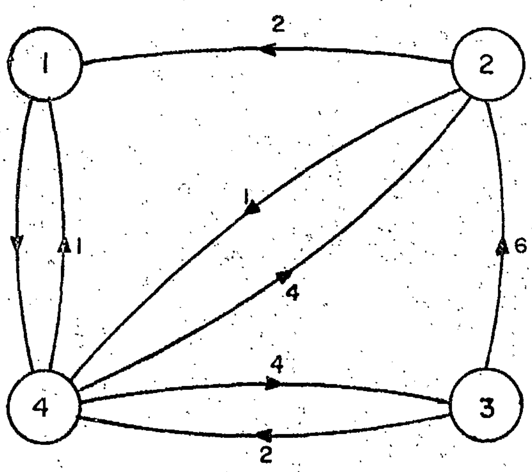

figure 3.5.1

Note however, that the communication network does not necessarily satisfy the constraint matrix D in the original problem.

3.6 CONCLUSIONS

In this chapter, an optimal procedure for synthesizing a network with uniform cost channels was formulated. It was shown that a network exactly satisfying the requirements is only realizable under certain conditions. Finally, it was demonstrated that a communication network can be constructed from a general network given that certain restrictions are not violated.

Although the emphasis was on communication networks, it should be apparent that the synthesis procedure offered in Algorithm 3.3.1 finds application in economics and operations-research problems where appropriate interpretation of negative capacities may exist.

It remains for further study, to find a method for obtaining a network that exactly satisfies the requirements, only with a non-uniform cost criterion.

APPENDIX A

NOTATION*

R	the set of real numbers
R^+	the set of non-negative real numbers
ε	"belongs to"
$\notin$	"does not belong to"
X^c	if X is a subset of some set N, then X^c will be used to denote all elements of N which do not belong to X; it is called the "complement" of X (with respect to N).
X-Y	the set of the elements in X which do not belong to Y.
card X	the number of elements in X

* Further details on the elementary set operations used in proving certain theorems, can be found in [12].

APPENDIX B

SHORT 1 (,K)

PROGRAM DESCRIPTION

Program SHORT1(,K) is a computer package that implements the simultaneous transmission synthesis presented in Algorithm 2.5.1. It is written in Extended FORTRAN IV H for a Sigma 7 time-shared computer installation.

The main line of the program is shown in the flow-chart in figure B with the associated subroutines appropriately indicated. Although the variables used in the chart are those used in Algorithm 2.5.1, this is an equivalent representation of SHORT1(,K). The transformation from the algorithm to the program is achieved by referring to table B.

Subroutine NETRED is a generalized routine used by all three synthesis packages presented in this study. The input program "asks" the user for the pertinent network data, that is,"asks" for the number of nodes N, the terminal requirement matrix T and the arc cost matrix D. Provision has also been made (in all three packages) to input this data from a file prepared on disc. The user in such cases, should assign this disc file to logical device #1 before execution.

Subroutines NETSHORT and NETROUTE realize Floyd's shortest path Algorithm 2.4.3. NETSHORT finds L, the matrix containing the lengths of all the shortest paths, and Φ, the corresponding node matrix while NETROUTE finds the shortest path (the sequence of nodes) between some specified pair of terminals. These two routines have also been incorporated in the main line of SHORT2(,K).

The main feature of this package (as well as the other two programs) is that it is completely conversational due to the time-shared environment in which it resides. Since no unusual programming techniques were used, the program listing that follows is straightforward. The user should note that the program, as it stands, can handle a network of at most 15 nodes and that due to page width limitations, each terminal requirement may range from 000.0 to 999.9 and each arc cost entry may take on integer values from 0 to 99999. It should be realized that the program may be readily changed to compute larger networks by allocating more storage for the matrices.

ALGORITHM VARIABLES	PROGRAM VARIABLES
N	N
C	CAP
T	TERM
D	COST
L	SHORT
Φ	NODE
$\hat{\Pi}$	R

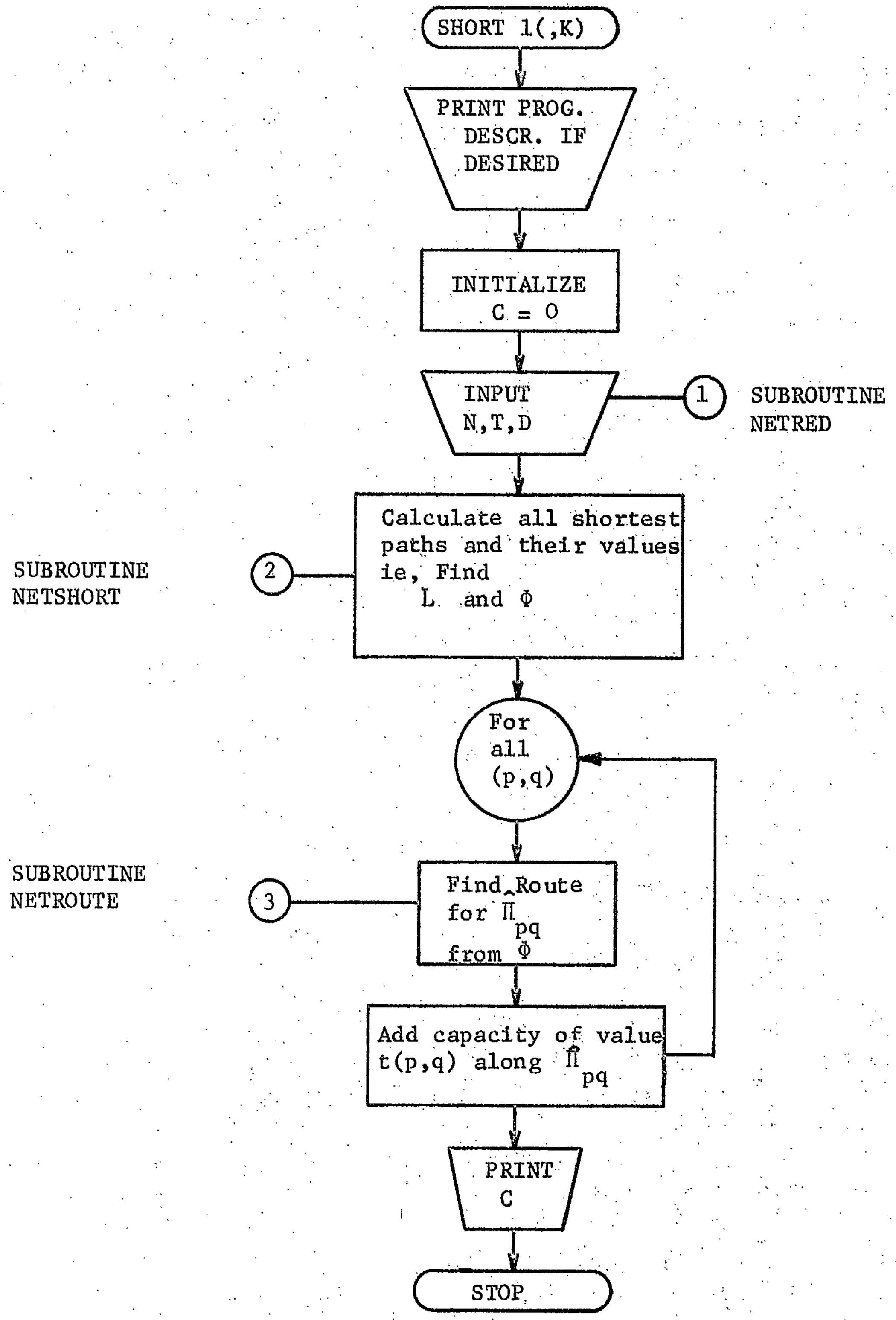
SHORT 1(,K)
PRINT PROG. DESCR. IF DESIRED
INITIALIZE C = 0
INPUT N,T,D
1
SUBROUTINE NETRED
Calculate all shortest paths and their values ie, Find L and Φ
2
SUBROUTINE NETSHORT
For all (p,q)
Find Route for Π pq from Φ
3
SUBROUTINE NETROUTE
Add capacity of value t(p,q) along Π pq
PRINT C
STOP

figure B

Example 2.5.1-

```
!LOAD
ELEMENT FILES: SHORTIB
OPTIONS:
F: 1
F:

SEV.LEV. = Ø
XEQ? Y


 THIS NETWORK SYNTHESIS PACKAGE IS AT YOUR COMMAND!
 PLEASE INPUT DATA AS REQUESTED BY THE PROGRAM!
 DO YOU WISH TO SEE THE PROGRAM DESCRIPTION?
 ANSWER YES OR NO
?YES


 THIS PROGRAM SYNTHESIZES A COMMUNICATION  NETWORK
 GIVEN THE COMMUNICATION  CENTRES, THE TERMINAL CHANNEL
 CAPACITY REQUIREMENTS AND THE ARC  COST CONSTRAINTS.
 THE REQUIREMENTS ARE MET SIMULTANEOUSLY AND THEY DO
 NOT VARY WITH TIME. SHORTEST PATH TECHNIQUES ARE USED
 TO ARRIVE AT THE SOLUTION.

 INPUT:
      N-(INTEGER)-THE NUMBER OF COMMUNICATION  CENTRES:
       N IS THE DIMENSIONALITY  OF THE MATRICES BELOW.
      T-(DECIMAL)-THE TERMINAL CAPACITY MATRIX:
       EACH ENTRY, T(I,J), CONTAINS THE VALUE OF
       REQUIRED CHANNEL CAPACITY FROM TERMINAL I TO TERM-
       INAL J.
      D-(INTEGER)-THE ARC COST MATRIX:
       EACH ENTRY, D(I,J), IS THE COST PER UNIT CAPACITY
       ON ARC (I,J).

 OUTPUT:
      C-(DECIMAL)-THE REQUIRED CAPACITY MATRIX:
       EACH ENTRY,   C(I,J), IS THE CHANNEL CAPACITY
       OF ARC (I,J) THAT IS REQUIRED FOR THE SOLUTION.
      TT-(DECIMAL)-TOTAL NETWORK COST:
       TT=SUM(C(I,J)*D(I,J))  FOR ALL I AND J.


 ARE THE REQUIREMENTS AND THE COSTS SYMETRICAL?
 ANSWER YES OR NO
?NO


 INPUT THE NUMBER OF NODES PLEASE!
?4


 PLEASE INPUT 16 FLOATING POINT VALUES
 4 PER LINE TO FILL THE TERMINAL CAPACITY MATRIX!
```

```
 1:
?Ø,2,6,5
 2:
?2,Ø,1,3
 3:
?6,1,Ø,4
 4:
?1,3,2,Ø


 PLEASE INPUT 16 INTEGER VALUES
 4 PER LINE TO FILL THE ARC COST MATRIX!


 1:
?Ø,1,999,4
 2:
?4,Ø,11,1
 3:
?999,1,Ø,3
 4:
?1,3,3,Ø


 DO YOU WISH TO REVIEW YOUR INPUT?
 ANSWER YES OR NO
?YES


 THE TERMINAL CAPACITY MATRIX:

     .Ø    2.Ø    6.Ø    5.Ø
    2.Ø     .Ø    1.Ø    3.Ø
    6.Ø    1.Ø     .Ø    4.Ø
    1.Ø    3.Ø    2.Ø     .Ø


 THE ARC COST MATRIX:

      Ø      1    999      4
      4      Ø     11      1
    999      1      Ø      3
      1      3      3      Ø


 DO YOU WISH TO RE-ENTER YOUR DATA?
 ANSWER YES OR NO
?NO


 THE REQUIRED CAPACITY MATRIX BELOW REPRESENTS
 THE NETWORK THAT SIMULTANEOUSLY SATISFIES THE REQUIREMENTS!

    .Ø   16.Ø     .Ø     .Ø
    .Ø     .Ø     .Ø   27.Ø
    .Ø   11.Ø     .Ø     .Ø
  12.Ø     .Ø    9.Ø     .Ø


 TOTAL NETWORK COST IS-      93.ØØ
```

```
!EDIT
*EDIT SHORTI(,K)
*TS1-999
C
      COMMON/C1/N,TERM(15,15),COST(15,15)
      COMMON/C2/IK,JK
      COMMON/C4/CAP(15,15),NODE(15,15),R(15),KR
      INTEGER COST,R
      DATA YES/'Y'/
      WRITE(108,115)
  115 FORMAT(//' THIS NETWORK SYNTHESIS PACKAGE IS AT YOUR COMMAND!',
     */' PLEASE INPUT DATA AS REQUESTED BY THE PROGRAM!',
     */' DO YOU WISH TO SEE THE PROGRAM DESCRIPTION?',
     */' ANSWER YES OR NO')
      READ(105,103) ANSWER
      IF(ANSWER.NE.YES) GOTO 1
      WRITE(108,116)
  116 FORMAT(//' THIS PROGRAM SYNTHESIZES A COMMUNICATION  NETWORK',
     */' GIVEN THE COMMUNICATION  CENTRES, THE TERMINAL CHANNEL',
     */' CAPACITY REQUIREMENTS AND THE ARC  COST CONSTRAINTS.',
     */' THE REQUIREMENTS ARE MET SIMULTANEOUSLY AND THEY DO',
     */' NOT VARY WITH TIME. SHORTEST PATH TECHNIQUES ARE USED',
     */' TO ARRIVE AT THE SOLUTION.')
      WRITE(108,117)
  117 FORMAT(/' INPUT:',
     */'        N-(INTEGER)-THE NUMBER OF COMMUNICATION  CENTRES:',
     */'         N IS THE DIMENSIONALITY  OF THE MATRICES BELOW.',
     */'        T-(DECIMAL)-THE TERMINAL CAPACITY MATRIX:',
     */'         EACH ENTRY, T(I,J), CONTAINS THE VALUE OF',
     */'         REQUIRED CHANNEL CAPACITY FROM TERMINAL I TO TERM-',
     */'         INAL J.',
     */'        D-(INTEGER)-THE ARC COST MATRIX:',
     */'         EACH ENTRY, D(I,J), IS THE COST PER UNIT CAPACITY',
     */'         ON ARC (I,J).')
      WRITE(108,118)
  118 FORMAT(/' OUTPUT:',
     */'        C-(DECIMAL)-THE REQUIRED CAPACITY MATRIX:',
     */'         EACH ENTRY,   C(I,J), IS THE CHANNEL CAPACITY',
     */'         OF ARC (I,J) THAT IS REQUIRED FOR THE SOLUTION.',
     */'        TT-(DECIMAL)-TOTAL NETWORK COST:',
     */'         TT=SUM(C(I,J)*D(I,J))  FOR ALL I AND J.')
    1 WRITE(108,901)
  901 FORMAT(//' ARE THE REQUIREMENTS AND THE COSTS SYMETRICAL?',
     */' ANSWER YES OR NO')
      ISW1=0
      READ(105,103) ANSWER
  103 FORMAT(A1)
      IF(ANSWER.NE.YES) GOTO 2
      ISW1=1
    2 CALL NETRED
      TOTCOST=0.0
      DO 3 IK=1,N
      DO 3 JK=1,N
      CAP(IK,JK)=0.0
      IF(ISW1.EQ.0) GOTO 3
      TERM(JK,IK)=TERM(IK,JK)
```

```
      COST(JK,IK)=COST(IK,JK)
    3 CONTINUE
      CALL NETSHORT
    7 NN=N-ISW1
      DO 5 IK=1,NN
      LL=IK*ISW1+1
      DO 5 JK=LL,N
      IF(IK.EQ.JK) GOTO 5
      CALL NETROUTE
      KR=KR-1
      DO 5 K=1,KR
      N1=R(K)
      N2=R(K+1)
      CAP(N1,N2)=CAP(N1,N2)+TERM(IK,JK)
      IF(ISW1.EQ.Ø) GOTO 5
      CAP(N2,N1)=CAP(N1,N2)
    5 CONTINUE
      WRITE(1Ø8,1Ø1)
  1Ø1 FORMAT(//' THE REQUIRED CAPACITY MATRIX BELOW REPRESENTS',
     */' THE NETWORK THAT SIMULTANEOUSLY SATISFIES THE REQUIREMENTS!'/)
      DO 6 IK=1,N
      WRITE(1Ø8,1Ø2)(CAP(IK,JK),JK=1,N)
      DO 6 K=1,N
      IF(IK.EQ.K) GOTO 6
      TOTCOST=TOTCOST+CAP(IK,K)*COST(IK,K)
    6 CONTINUE
      WRITE(1Ø8,1Ø4) TOTCOST
  1Ø4 FORMAT(//' TOTAL NETWORK COST IS-',F1Ø.2)
  1Ø2 FORMAT(15(F5.1,1X))
  99Ø WRITE(1Ø8,5Ø4)
  5Ø4 FORMAT(//' DO YOU WISH TO RESTART THIS PROGRAM?',
     */' ANSWER YES OR NO')
      READ(1Ø5,1Ø3) ANSWER
      IF(ANSWER.EQ.YES) GOTO 1
      END
      SUBROUTINE NETRED
C     THIS ROUTINE READS IN THE MATRIX SIZE, THE TERMINAL
C     CAPACITY MATRIX AND THE ARC COST MATRIX.
C
      COMMON/C1/N,TERM(15,15),COST(15,15)
      INTEGER COST
      DATA YES/'Y'/
  599 WRITE(1Ø8,6ØØ)
  6ØØ FORMAT(//' INPUT THE NUMBER OF NODES PLEASE!')
      READ(1Ø5,6Ø1)N
  6Ø1 FORMAT(I)
      MN=N*N
      WRITE(1Ø8,6Ø2)MN,N
  6Ø2 FORMAT(//' PLEASE INPUT ',I2,' FLOATING POINT VALUES'/
     * I2,' PER LINE TO FILL THE TERMINAL CAPACITY MATRIX!'//)
      DO 6Ø3 I=1,N
      WRITE(1Ø8,61Ø)I
  61Ø FORMAT(I2,':')
  6Ø3 READ (1,6Ø4)(TERM(I,J),J=1,N)
  6Ø4 FORMAT(15F)
      WRITE(1Ø8,6Ø5)MN,N
  6Ø5 FORMAT(//' PLEASE INPUT ',I2,' INTEGER VALUES'/
```

```
   * I2,' PER LINE TO FILL THE ARC COST MATRIX!'//)
     DO 606 I=1,N
     WRITE(108,610)I
 606 READ (1,607)(COST(I,J),J=1,N)
 607 FORMAT(15I)
     WRITE(108,620)
 620 FORMAT(//' DO YOU WISH TO REVIEW YOUR INPUT?',
    */' ANSWER YES OR NO')
     READ(105,621) ANSWER
 621 FORMAT(A1)
     IF(ANSWER.NE.YES) GOTO648
     WRITE(108,623)
 623 FORMAT(//' THE TERMINAL CAPACITY MATRIX:'/)
     DO 624 I=1,N
 624 WRITE(108,625)(TERM(I,J),J=1,N)
 625 FORMAT(15(1X,F5.1))
     WRITE(108,626)
 626 FORMAT(//' THE ARC COST MATRIX:'/)
     DO 627 I=1,N
 627 WRITE(108,628)(COST(I,J),J=1,N)
 628 FORMAT(15(I5,1X))
 648 WRITE(108,629)
 629 FORMAT(//' DO YOU WISH TO RE-ENTER YOUR DATA?',
    */' ANSWER YES OR NO')
     READ(105,621) ANSWER
     IF(ANSWER.NE.YES) GOTO 622
     GOTO 599
 622 RETURN
     END
     SUBROUTINE NETSHORT
     COMMON/C1/N,DUM1(225),SHORT(15,15)
     COMMON/C4/DUM2(225),NODE(15,15),DUM3(16)
     INTEGER SHORT
     DO 48 IK=1,N
     DO 48 JK=1,N
  48 NODE(IK,JK)=JK
     DO 50 K=1,N
     DO 50 IK=1,N
     DO 50 JK=1,N
     IF(JK.EQ.IK.OR.JK.EQ.K.OR.IK.EQ.K) GOTO 50
     NSH=SHORT(IK,K)+SHORT(K,JK)
     IF(SHORT(IK,JK).LE.NSH) GOTO 50
     NODE(IK,JK)=K
     SHORT(IK,JK)=NSH
  50 CONTINUE
     RETURN
     END
     SUBROUTINE NETROUTE
     COMMON/C2/IK,JK
     COMMON/C4/DUM2(225),NODE(15,15),P(15),KR
     INTEGER R
     K=2
     L=1
     M=2
     R(1)=IK
     R(2)=JK
  54 IF(NODE(R(L),R(M)).EQ.R(M)) GOTO 55
```

```
      DO 56 I=K,M,-1
   56 R(I+1)=R(I)
      R(M)=NODE(R(L),R(M))
      K=K+1
      GOTO 54
   55 IF(M.EQ.K) GOTO 57
      M=M+1
      L=L+1
      GOTO 54
   57 KR=K
      RETURN
      END
--EOF HIT AFTER 179.
*
```

APPENDIX C

SHORT2(,K)

PROGRAM DESCRIPTION

SHORT2(,K) is a package that uses many of the building blocks found in SHORT1(,K) to realize the synthesis of time-shared communication networks. It implements Algorithm 2.6.1 in FORTRAN for a Sigma 7 time-shared computer.

Figure C shows the flow-chart for Algorithm 2.6.1 and hence for SHORT2(,K). Table C provides the cross-reference from the variable names used in the algorithm to the variable names used in the FORTRAN program. Subroutines NETRED, NETSHORT and NETROUTE are described in Appendix B.

Subroutine NETSRT realizes the procedure in step 1a) of Algorithm 2.6.1.

ALGORITHM VARIABLES	PROGRAM VARIABLES
N	N
C	CAP
T	TERM
D	COST
L	SHORT
Φ	NODE
$\hat{\Pi}$	R
t	T
A	E
m	KT
μ	I

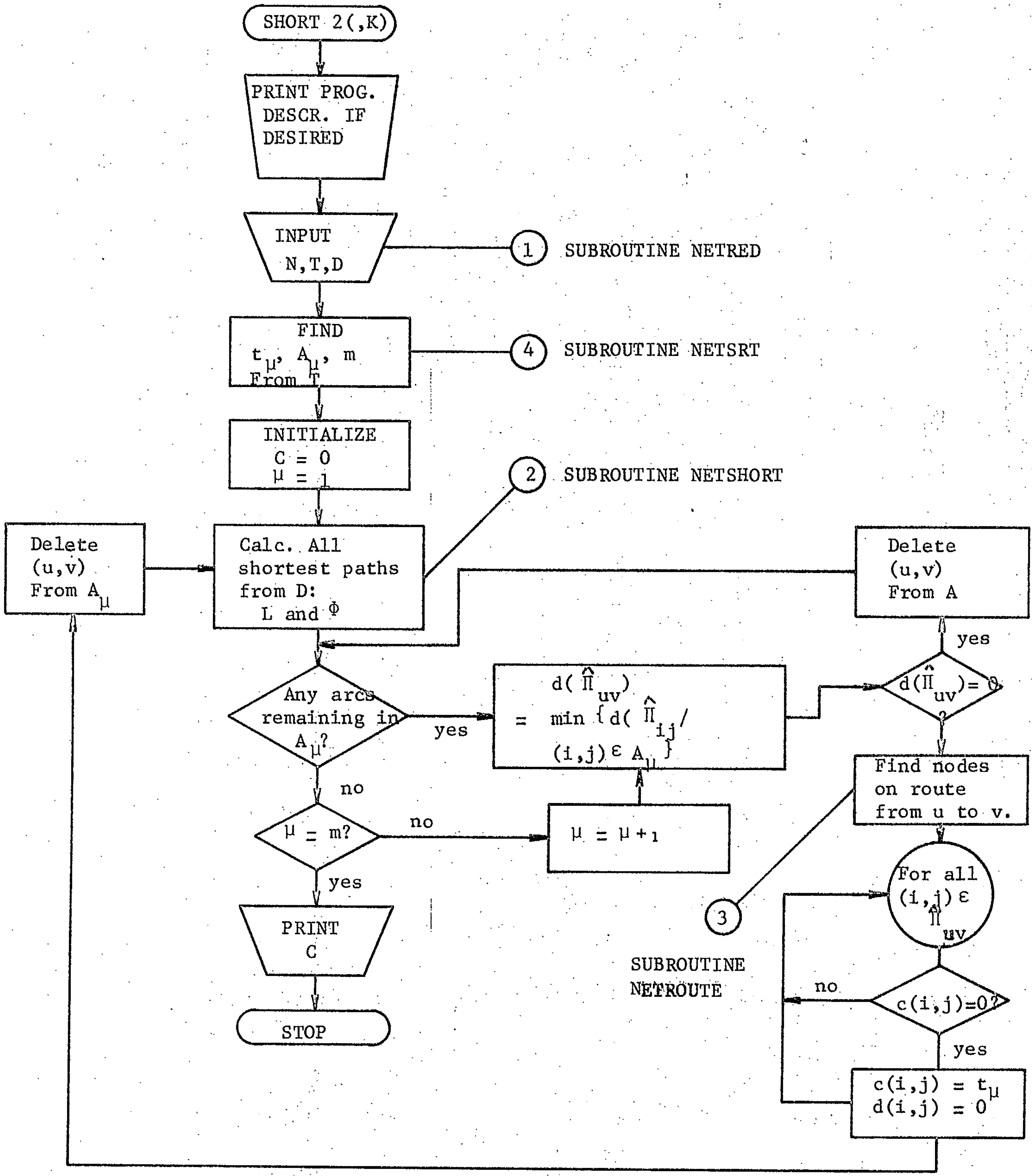
SHORT 2(,K)
PRINT PROG. DESCR. IF DESIRED
INPUT N,T,D
1
SUBROUTINE NETRED
FIND t_μ, A_μ, m From T
4
SUBROUTINE NETSRT
INITIALIZE C = 0 μ = 1
2
SUBROUTINE NETSHORT
Delete (u,v) From A_μ
Calc. All shortest paths from D: L and Φ
Delete (u,v) From A
yes
Any arcs remaining in A_μ?
yes
$d(\hat{\Pi}_{uv}) = \min\{d(\hat{\Pi}_{ij}) / (i,j) \in A_\mu\}$
$d(\hat{\Pi}_{uv}) = 0$
?
Find nodes on route from u to v.
no
μ = m?
no
$\mu = \mu + 1$
yes
PRINT C
STOP
3
SUBROUTINE NETROUTE
For all $(i,j) \in \hat{\Pi}_{uv}$
no
c(i,j)=0?
yes
$c(i,j) = t_\mu$
$d(i,j) = 0$

figure C

Example 2.6.1-

```
!LOAD
ELEMENT FILES: SHORT2B
OPTIONS:
F: 1
F:

SEV.LEV. = Ø
XEQ? Y


 THIS NETWORK SYNTHESIS PACKAGE IS AT YOUR COMMAND!
 PLEASE INPUT DATA AS REQUESTED BY THE PROGRAM!
 DO YOU WISH TO SEE THE PROGRAM DESCRIPTION?
 ANSWER YES OR NO
?YES


 THIS PROGRAM SYNTHESIZES A COMMUNICATION  NETWORK
 GIVEN THE COMMUNICATION  CENTRES, THE TERMINAL CHANNEL
 CAPACITY REQUIREMENTS AND THE  ARC COST CONSTRAINTS.
 THE REQUIREMENTS DO NOT  VARY WITH TIME AND THEY ARE
 TIME-SHARED IN SUCH A WAY THAT ONLY TWO TERMINALS MAY
 COMMUNICATE WITH ONE AND OTHER AT ONE TIME. SHORT-
 EST PATH TECHNIQUES ARE USED TO ARRIVE AT THE SOLUTION.

 INPUT:
      N-(INTEGER)-THE NUMBER OF COMMUNICATION  CENTRES:
       N IS THE DIMENSIONALITY  OF THE MATRICES BELOW.
      T-(DECIMAL)-THE TERMINAL CAPACITY MATRIX:
        EACH ENTRY,    T(I,J), CONTAINS THE VALUE OF
       REQUIRED CHANNEL CAPACITY FROM TERMINAL I TO TERM-
       INAL J.
      D-(INTEGER)-THE ARC COST MATRIX:
       EACH ENTRY,D(I,J), IS THE COST PER UNIT CAPACITY
      ON ARC (I,J).

 OUTPUT:
      C-(DECIMAL)-THE REQUIRED CAPACITY MATRIX:
       EACH ENTRY,   C(I,J), IS THE CHANNEL CAPACITY
       OF  ARC(I,J) THAT IS REQUIRED FOR THE SOLUTION.
      TT-(DECIMAL)-THE TOTAL NETWORK COST:
       TT=SUM(C(I,J)*D(I,J)) FOR ALL I AND J.


 INPUT THE NUMBER OF NODES PLEASE!
?4


 PLEASE INPUT 16 FLOATING POINT VALUES
 4 PER LINE TO FILL THE TERMINAL CAPACITY MATRIX!


 1:
?Ø,9,8,10
 2:
?6,Ø,6,8
```

```
 3:
?4,7,Ø,7
 4:
?1,2,1,Ø


 PLEASE INPUT 16 INTEGER VALUES
 4 PER LINE TO FILL THE ARC COST MATRIX!


 1:
?Ø,1,999,4
 2:
?7,Ø,6,2
 3:
?999,1,Ø,5
 4:
?2,8,2,Ø


 DO YOU WISH TO REVIEW YOUR INPUT?
 ANSWER YES OR NO
?YES


 THE TERMINAL CAPACITY MATRIX:

    .ØØ   9.ØØ   8.ØØ  1Ø.ØØ
   6.ØØ    .ØØ   6.ØØ   8.ØØ
   4.ØØ   7.ØØ    .ØØ   7.ØØ
   1.ØØ   2.ØØ   1.ØØ    .ØØ


 THE ARC COST MATRIX:

    Ø      1    999      4
    7      Ø      6      2
  999      1      Ø      5
    2      8      2      Ø


 DO YOU WISH TO RE-ENTER YOUR DATA?
 ANSWER YES OR NO
?NO


 THE REQUIRED CAPACITY MATRIX BELOW SATISFIES THE
 TIME-SHARED REQUIREMENTS AND REPRESENTS THE DESIRED NETWORK!

      .Ø      1Ø.Ø          .Ø          .Ø
      .Ø        .Ø          .Ø        1Ø.Ø
      .Ø       7.Ø          .Ø          .Ø
     6.Ø        .Ø         8.Ø          .Ø


 TOTAL NETWORK COST IS-       65.ØØ
```

```
!EDIT
*EDIT SHORT2(,K)
*TS1-999
C
C
C
      COMMON/C1/N,TERM(15,15),COST(15,15)
      COMMON/C2/T(210),LE(210),E(210,20),KT
      COMMON/C3/X(225),Y(225),XX,YY
      COMMON/C4/CAP(15,15),SHORT(15,15),NODE(15,15),R(15),KR
      COMMON/C5/CT(15,15)
      INTEGER COST,E,X,Y,XX,YY,SHORT,R,CT
      DATA YES/'Y'/
      WRITE(108,115)
  115 FORMAT(//' THIS NETWORK SYNTHESIS PACKAGE IS AT YOUR COMMAND!',
     */' PLEASE INPUT DATA AS REQUESTED BY THE PROGRAM!',
     */' DO YOU WISH TO SEE THE PROGRAM DESCRIPTION?',
     */' ANSWER YES OR NO')
      READ(105,103) ANSWER
      IF(ANSWER.NE.YES) GOTO 1
      WRITE(108,116)
  116 FORMAT(//' THIS PROGRAM SYNTHESIZES A COMMUNICATION  NETWORK',
     */' GIVEN THE COMMUNICATION  CENTRES, THE TERMINAL CHANNEL',
     */' CAPACITY REQUIREMENTS AND THE  ARC COST CONSTRAINTS.',
     */' THE REQUIREMENTS DO NOT  VARY WITH TIME AND THEY ARE ',
     */' TIME-SHARED IN SUCH A WAY THAT ONLY TWO TERMINALS MAY',
     */' COMMUNICATE WITH ONE AND OTHER AT ONE TIME. SHORT-',
     */' EST PATH TECHNIQUES ARE USED TO ARRIVE AT THE SOLUTION.')
      WRITE(108,117)
  117 FORMAT(/' INPUT:',
     */'       N-(INTEGER)-THE NUMBER OF COMMUNICATION  CENTRES:',
     */'        N IS THE DIMENSIONALITY  OF THE MATRICES BELOW.',
     */'       T-(DECIMAL)-THE TERMINAL CAPACITY MATRIX:',
     */'         EACH ENTRY,    T(I,J), CONTAINS THE VALUE OF',
     */'        REQUIRED CHANNEL CAPACITY FROM TERMINAL I TO TERM-',
     */'        INAL J.',
     */'       D-(INTEGER)-THE ARC COST MATRIX:',
     */'        EACH ENTRY,D(I,J), IS THE COST PER UNIT CAPACITY',
     */'       ON ARC (I,J).')
      WRITE(108,118)
  118 FORMAT(/' OUTPUT:',
     */'       C-(DECIMAL)-THE REQUIRED CAPACITY MATRIX:',
     */'        EACH ENTRY,   C(I,J), IS THE CHANNEL CAPACITY',
     */'        OF  ARC(I,J) THAT IS REQUIRED FOR THE SOLUTION.',
     */'       TT-(DECIMAL)-THE TOTAL NETWORK COST:',
     */'        TT=SUM(C(I,J)*D(I,J)) FOR ALL I AND J.')
    1 CALL NETRED
      TOTCOST=0.
      DO 2 JK=1,N
      DO 2 IK=1,N
      CT(IK,JK)=COST(IK,JK)
      K=N*(JK-1)+IK
      X(K)=IK
      CAP(IK,JK)=0.
    2 Y(K)=JK
      DO 3 IK=1,90
```

```
      LE(IK)=1
    3 T(IK)=0.0
      CALL NETSRT
      I=KT
   50 LP=LE(I)
      DO 5 JJ=1,LP
      DO 4 IK=1,N
      DO 4 JK=1,N
    4 SHORT(IK,JK)=COST(IK,JK)
      CALL NETSHORT
      MINCOST=10**6
      MINJ=0
      DO 10 J=1,LP
      LIP=E(I,J)
      IF(LIP.EQ.0) GOTO 10
      XX=X(LIP)
      YY=Y(LIP)
      IF(MINCOST.LT.SHORT(XX,YY)) GOTO 10
      MINCOST=SHORT(XX,YY)
      MINJ=J
   10 CONTINUE
      IF(MINJ.GT.0) GOTO 11
      WRITE(108,100)
  100 FORMAT(' ERROR EXISTS IN COST MATRIX')
      GOTO 990
   11 LIP=E(I,MINJ)
      XX=X(LIP)
      YY=Y(LIP)
      E(I,MINJ)=0
      CALL NETROUTE
      KR=KR-1
      DO 5 K=1,KR
      N1=R(K)
      N2=R(K+1)
      COST(N1,N2)=0
      IF(CAP(N1,N2).GT.0.001) GOTO 5
      CAP(N1,N2)=T(I)
    5 CONTINUE
      I=I-1
      IF(I.GT.0) GOTO 50
      WRITE(108,101)
  101 FORMAT(//' THE REQUIRED CAPACITY MATRIX BELOW SATISFIES THE',
     */' TIME-SHARED REQUIREMENTS AND REPRESENTS THE DESIRED NETWORK!'/)
      DO 6 IK=1,N
      WRITE(108,102)(CAP(IK,JK),JK=1,N)
      DO 6 K=1,N
      IF(IK.EQ.K) GOTO 6
      TOTCOST=TOTCOST+CAP(IK,K)*CT(IK,K)
    6 CONTINUE
      WRITE(108,104) TOTCOST
  104 FORMAT(//' TOTAL NETWORK COST IS-',F10.2)
  102 FORMAT(10(F6.1,2X))
  990 WRITE(108,504)
  504 FORMAT(//' DO YOU WISH TO RESTART THIS PROGRAM?',
     */' ANSWER YES OR NO')
      READ(105,103) ANSWER
  103 FORMAT(A1)
```

```
      IF(ANSWER.EQ.YES) GOTO 1
      END
      SUBROUTINE NETRED
C     THIS ROUTINE READS IN THE MATRIX SIZE, THE TERMINAL
C     CAPACITY MATRIX AND THE ARC COST MATRIX.
C
      COMMON/C1/N,TERM(15,15),COST(15,15)
      INTEGER COST
      DATA YES/'Y'/
  599 WRITE(108,600)
  600 FORMAT(//' INPUT THE NUMBER OF NODES PLEASE!')
      READ(105,601)N
  601 FORMAT(I)
      MN=N*N
      WRITE(108,602)MN,N
  602 FORMAT(//' PLEASE INPUT ',I2,' FLOATING POINT VALUES'/
     * I2,' PER LINE TO FILL THE TERMINAL CAPACITY MATRIX!'//)
      DO 603 I=1,N
      WRITE(108,610)I
  610 FORMAT(I2,':')
  603 READ (1,604)(TERM(I,J),J=1,N)
  604 FORMAT(10F)
      WRITE(108,605)MN,N
  605 FORMAT(//' PLEASE INPUT ',I2,' INTEGER VALUES'/
     * I2,' PER LINE TO FILL THE ARC COST MATRIX!'//)
      DO 606 I=1,N
      WRITE(108,610)I
  606 READ (1,607)(COST(I,J),J=1,N)
  607 FORMAT(10I)
      WRITE(108,620)
  620 FORMAT(//' DO YOU WISH TO REVIEW YOUR INPUT?',
     */' ANSWER YES OR NO')
      READ(105,621) ANSWER
  621 FORMAT(A1)
      IF(ANSWER.NE.YES) GOTO648
      WRITE(108,623)
  623 FORMAT(//' THE TERMINAL CAPACITY MATRIX:'/)
      DO 624 I=1,N
  624 WRITE(108,625)(TERM(I,J),J=1,N)
  625 FORMAT(10(1X,F5.2))
      WRITE(108,626)
  626 FORMAT(//' THE ARC COST MATRIX:'/)
      DO 627 I=1,N
  627 WRITE(108,628)(COST(I,J),J=1,N)
  628 FORMAT(10(I4,2X))
  648 WRITE(108,629)
  629 FORMAT(//' DO YOU WISH TO RE-ENTER YOUR DATA?',
     */' ANSWER YES OR NO')
      READ(105,621) ANSWER
      IF(ANSWER.NE.YES) GOTO 622
      GOTO 599
  622 RETURN
      END
```

```
      SUBROUTINE NETSRT
C
C     TERMINAL VALUES ARE SORTED INTO ASCENDING ORDER
C     IN T AND CORRESPONDINDING ARC NUMBERS IN E
C
      COMMON/C1/N,TERM(15,15),COST(15,15)
      COMMON/C2/T(210),LE(210),E(210,20),KT
      INTEGER E
      K=1
      DO 11 J=1,N
      DO 11 I=1,N
      IF(I.EQ.J) GOTO 11
      IF(TERM(I,J).NE.Ø.) GOTO 12
   11 CONTINUE
   12 T(1)=TERM(I,J)
      E(1,1)=N*(J-1)+I
      JM=J
      IMM=I
      JMM=1
      DO 5 J=JM,N
      IM=IMM*JMM+1
      JMM=Ø
      DO 5 I=IM,N
      IF(I.EQ.J) GO TO 5
      IF(ABS(TERM(I,J)).LE.Ø.ØØ1)  GOTO 5
      DO 4 L=1,K
      IF(ABS(T(L)-TERM(I,J))-.ØØ1)8,1,1
    1 IF(TERM(I,J).GT.T(L))GOTO4
    3 DO 1Ø KK=K,L,-1
      LP=LE(KK)
      DO  9 LL=1,LP
    9 E(KK+1,LL)=E(KK,LL)
      LE(KK+1)=LE(KK)
   1Ø T(KK+1)=T(KK)
      T(L)=TERM(I,J)
      E(L,1)=N*(J-1)+I
      LE(L)=1
      K=K+1
      GOTO 5
    8 LE(L)=LE(L)+1
      E(L,LE(L))=N*(J-1)+I
      GOTO5
    4 CONTINUE
      K=K+1
    7 T(K)=TERM(I,J)
      E(K,1)=N*(J-1)+I
    5 CONTINUE
      KT=K
      RETURN
      END
C
C
C
```

```
      SUBROUTINE NETSHORT
      COMMON/C1/N,DUM1(450)
      COMMON/C4/DUM2(225),SHORT(15,15),NODE(15,15),DUM3(16)
      INTEGER SHORT
      DO 48 IK=1,N
      DO 48 JK=1,N
   48 NODE(IK,JK)=JK
      DO 50 K=1,N
      DO 50 IK=1,N
      DO 50 JK=1,N
      IF(JK.EQ.IK.OR.JK.EQ.K.OR.IK.EQ.K) GOTO 50
      NSH=SHORT(IK,K)+SHORT(K,JK)
      IF(SHORT(IK,JK).LE.NSH) GOTO 50
      NODE(IK,JK)=K
      SHORT(IK,JK)=NSH
   50 CONTINUE
      RETURN
      END
      SUBROUTINE NETROUTE
      COMMON/C3/DUM4(450),XX,YY
      COMMON/C4/DUM5(450),NODE(15,15),R(15),KR
      INTEGER R,XX,YY
      K=2
      L=1
      M=2
      R(1)=XX
      R(2)=YY
   54 IF(NODE(R(L),R(M)).EQ.R(M)) GOTO 55
      DO 56 I=K,M,-1
   56 R(I+1)=R(I)
      R(M)=NODE(R(L),R(M))
      K=K+1
      GOTO 54
   55 IF(M.EQ.K) GOTO 57
      M=M+1
      L=L+1
      GOTO 54
   57 KR=K
      RETURN
      END
--EOF HIT AFTER 256.
```

APPENDIX D

NETSYM(,K)

PROGRAM DESCRIPTION

NETSYM(,K) is a program that implements the synthesis procedure developed in chapter III of this study. It too is written in FORTRAN for a Sigma 7 configuration.

From figure D.1 it is obvious that certain routines in the main line are common to those used in SHORT2(,K) and hence are not described here (refer to Appendices B and C for information on NETRED and NETSRT). Also, it should be noted that the flow-charts in figures D.1 and D.2 contain the variable names used in Chapter III. (Refer to table D for the cross-references required to interpret the FORTRAN variables used in NETSYM(,K).)

It is important to note that the main program sequentially executes a large subprogram, routine B. The main line implements Algorithm 3.3.1 while the subprogram implements Algorithm 3.5.1, that is, if the first part finds a general network with negative capacities, routine B constructs, if possible, a communication network.

Note that together, subroutines NETPACK and NETCOMB generate all possible semicuts containing the arcs in each arc set $A\mu$ and select those that are m-restrictions for the corresponding t_{μ}.

ALGORITHM VARIABLES	PROGRAM VARIABLES
N	N
C	CAP
T	TERM
D	COST
L	SHORT
Φ	NODE
Π	R
t	T
A	E
m	KT
μ	I
α	Z
$\int$	MINS

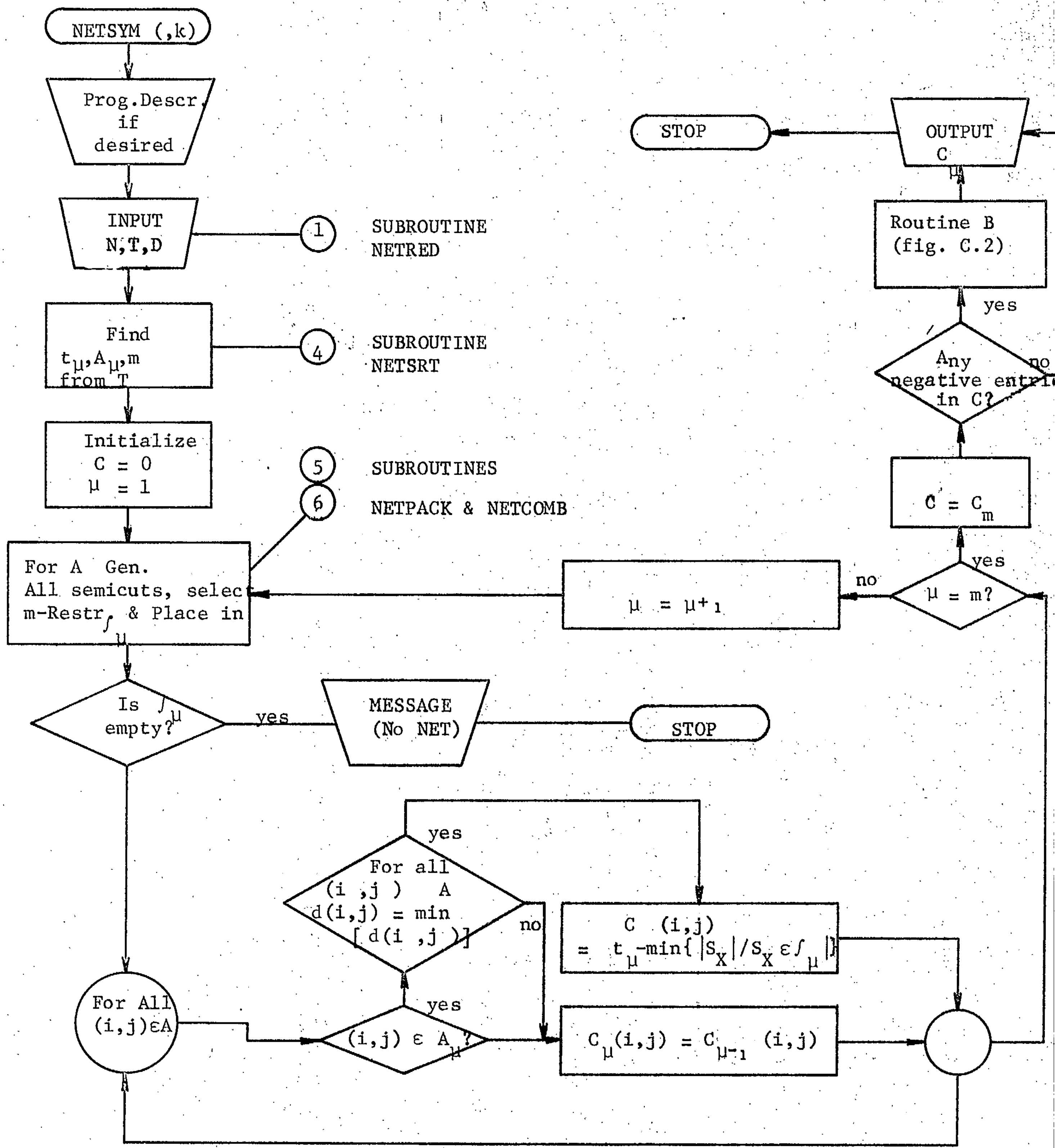
NETSYM (,k)
Prog.Descr. if desired
INPUT N,T,D
1
SUBROUTINE NETRED
Find t_μ,A_μ,m from T
4
SUBROUTINE NETSRT
Initialize C = 0 μ = 1
5
SUBROUTINES
6
NETPACK & NETCOMB
For A Gen. All semicuts, select m-Restr, & Place in $\mathcal{S}_\mu$
Is $\mathcal{S}_\mu$ empty?
yes
MESSAGE (No NET)
STOP
STOP
OUTPUT C_μ
Routine B (fig. C.2)
yes
Any negative entries in C?
no
C = C_m
yes
μ = m?
no
μ = μ+1
yes
For all (i,j) A d(i,j) = min [d(i,j)]
no
C (i,j) = t_μ-min{$|S_X|$/S_X ε $\mathcal{S}_\mu$}
For All (i,j)εA
yes
(i,j) ε A_μ?
C_μ(i,j) = $C_{\mu-1}$ (i,j)

figure D.1

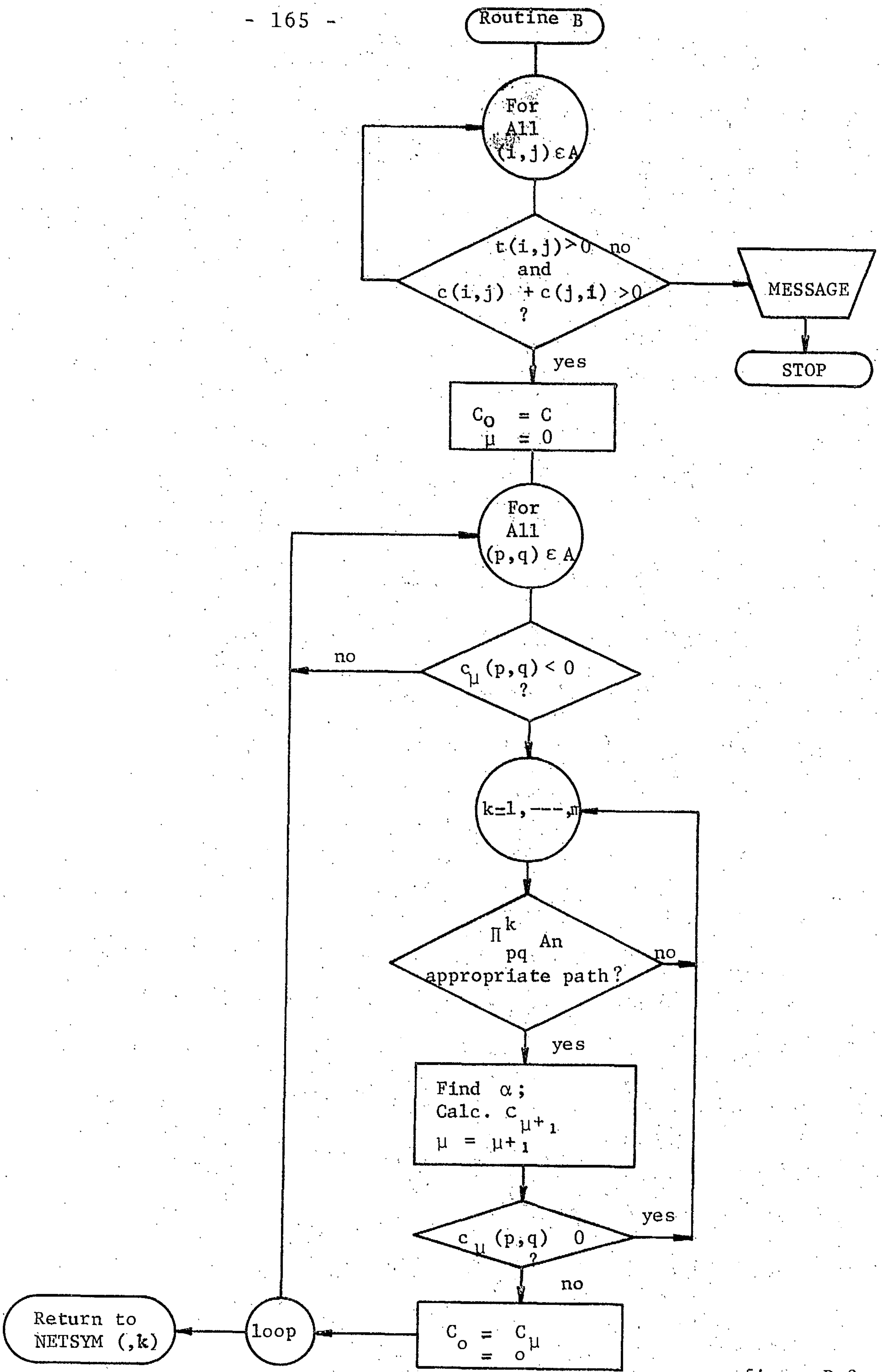
Routine B
For All (i,j) ε A
t(i,j) > 0 and c(i,j) + c(j,i) > 0 ?
no
MESSAGE
STOP
yes
C_o = C
μ = 0
For All (p,q) ε A
c_μ (p,q) < 0 ?
no
k=1, ---, m
Π^k_pq An appropriate path ?
no
yes
Find α;
Calc. c_μ+1
μ = μ+1
c_μ (p,q) 0 ?
yes
no
C_o = C_μ
= o
loop
Return to NETSYM (,k)

figure D.2

Examples 3.4.3 & 3.5.1-

```
!LOAD
ELEMENT FILES: NETSYMB
OPTIONS:
F: 1
F:

SEV.LEV. = Ø
XEQ? Y


 THIS NETWORK SYNTHESIS PACKAGE IS AT YOUR COMMAND!
 PLEASE INPUT DATA AS REQUESTED BY THE PROGRAM!

 DO YOU WISH TO SEE THE PROGRAM DESCRIPTION?
 ANSWER YES OR NO
?YES


 THIS PROGRAM SYNTHESIZES A COMMUNICATION
 NETWORK GIVEN THE COMMUNICATION  CENTERS, THE TERMINAL
 CHANNEL CAPACITY REQUIREMENTS AND THE ARC  CONSTRAINTS.
 THE REQUIREMENTS DO NOT VARY WITH TIME AND THEY ARE
 TIME-SHARED IN SUCH A WAY THAT ONLY TWO TERMINALS MAY
 COMMUNICATE WITH EACH OTHER AT ONE TIME. THE METHOD
 IS DEPENDENT ON THE PRESENCE OF REDUNDANT TERMINAL
 REQUIREMENTS. FURTHER PROGRAM DESCRIPTION IS AVAILABLE
 IN THE PROGRAM DOCUMENTATION.

 INPUT:
        N-(INTEGER)-THE NUMBER OF COMMUNICATION  CENTERS:
          N IS THE DIMENSIONALITY OF THE MATRICES BELOW.
        T-(DECIMAL)-THE TERMINAL CAPACITY MATRIX:
          EACH ENTRY,    T(I,J), REPRESENTS THE REQUIRED
          CHANNEL CAPACITY FROM TERMINAL(CENTER) I TO
          TERMINAL J.
        D-(INTEGER)-THE ARC CONSTRAINT MATRIX:
         EACH ENTRY, D(I,J), REPRESENTS THE RELATIVE VALUE
         OF CONSTRUCTING THE ARC (I,J). LOW VALUES OF D(I,J)
         GIVE THOSE ARCS HIGH CONSTRUCTION PRIORITIES.

 OUTPUT:
        C-(DECIMAL)-THE REQUIRED CAPACITY MATRIX:
           EACH ENTRY,   C(I,J), REPRESENTS THE CHANNEL CAPACITY
           THAT MUST BE CONSTRUCTED FROM I TO J IN ORDER TO
           ACHIEVE THE DESIRED SOLUTION


 INPUT THE NUMBER OF NODES PLEASE!
?4


 PLEASE INPUT  16 FLOATING POINT VALUES
  4 PER LINE TO FILL TERMINAL CAPACITY MATRIX
```

```
  1:
?Ø,2,2,2
  2:
?3,Ø,4,6
  3:
?3,7,Ø,8
  4:
?3,5,4,Ø

 PLEASE INPUT  16 INTEGER VALUES
  4 PER LINE TO FILL ARC CONSTRAINT MATRIX


  1:
?Ø,2,1Ø,1ØØ
  2:
?3,Ø,3,8
  3:
?1ØØ,5,Ø,2
  4:
?1,5,2,Ø


 DO YOU WISH TO REVIEW YOUR INPUT?
 ANSWER YES OR NO
?YES


 THE TERMINAL CAPACITY MATRIX:

   .ØØ  2.ØØ  2.ØØ  2.ØØ
  3.ØØ   .ØØ  4.ØØ  6.ØØ
  3.ØØ  7.ØØ   .ØØ  8.ØØ
  3.ØØ  5.ØØ  4.ØØ   .ØØ

 THE ARC CONSTRAINT MATRIX:

    Ø     2    1Ø   1ØØ
    3     Ø     3     8
  1ØØ     5     Ø     2
    1     5     2     Ø

 DO YOU WISH TO RE-ENTER YOUR DATA?
 ANSWER YES OR NO
?NO
```

```
THE CAPACITY MATRIX BELOW REPRESENTS THE NETWORK
THAT EXACTLY SATISFIES THE TERMINAL REQUIREMENTS!

   .Ø   2.Ø    .Ø    .Ø
   .Ø    .Ø    .Ø   6.Ø
   .Ø   6.Ø    .Ø   2.Ø
  3.Ø  -1.Ø   4.Ø    .Ø

NEGATIVE CAPACITIES ARE PRESENT ABOVE!
DO YOU WISH TO SEE THE COMMUNICATION NETWORK?
(THE NETWORK WITHOUT NEGATIVE CAPACITIES).
ANSWER YES OR NO!

?YES

THE COMMUNICATION NETWORK THAT EXACTLY SATISFIES THE
TERMINAL REQUIREMENTS IS REPRESENTED BY THE MATRIX BELOW!

   .Ø    .Ø    .Ø   2.Ø
  2.Ø    .Ø    .Ø   4.Ø
   .Ø   6.Ø    .Ø   2.Ø
  1.Ø   1.Ø   4.Ø    .Ø

DO YOU WISH TO SEE THE CALCULATED TERMINAL
CAPACITY MATRIX? ANSWER YES OR NO.
?YES

RESULTANT TERMINAL CAPACITY MATRIX:

   .Ø   2.Ø   2.Ø   2.Ø
  3.Ø    .Ø   4.Ø   6.Ø
  3.Ø   7.Ø    .Ø   8.Ø
  3.Ø   5.Ø   4.Ø    .Ø

DO YOU WISH TO RESTART THIS PROGRAM?
ANSWER YES OR NO
?NO
*STOP* Ø
```

```
!EDIT
*EDIT NETSYM(,K)
*TS1-999
      COMMON/C1/N,TERM(15,15),CONT(15,15)
      COMMON/C2/T(210),LE(210),E(210),KT
      COMMON/C3/X(225),Y(225),C(15),S(15)
      COMMON/C4/N2,L1,L2
      COMMON/C5/CAP(15,15),JB(15),IB(15)
      COMMON/C6/I
      COMMON/C7/MINS
      COMMON/C8/BUG
      INTEGER C,S,X,Y,XX,YY,CONT,E
      INTEGER P,Q
      REAL MINS
      DATA YES/'Y'/
      WRITE(108,800)
  800 FORMAT(//' THIS NETWORK SYNTHESIS PACKAGE IS AT YOUR COMMAND!',
     */' PLEASE INPUT DATA AS REQUESTED BY THE PROGRAM!',
     *//' DO YOU WISH TO SEE THE PROGRAM DESCRIPTION?',
     */' ANSWER YES OR NO')
      READ(105,982) ANSWER
      IF(ANSWER.NE.YES) GOTO 3
      WRITE(108,801)
  801 FORMAT(//' THIS PROGRAM SYNTHESIZES A COMMUNICATION',
     */' NETWORK GIVEN THE COMMUNICATION  CENTERS, THE TERMINAL',
     */' CHANNEL CAPACITY REQUIREMENTS AND THE ARC  CONSTRAINTS.',
     */' THE REQUIREMENTS DO NOT VARY WITH TIME AND THEY ARE',
     */' TIME-SHARED IN SUCH A WAY THAT ONLY TWO TERMINALS MAY',
     */' COMMUNICATE WITH EACH OTHER AT ONE TIME. THE METHOD',
     */' IS DEPENDENT ON THE PRESENCE OF REDUNDANT TERMINAL',
     */' REQUIREMENTS. FURTHER PROGRAM DESCRIPTION IS AVAILABLE',
     */' IN THE PROGRAM DOCUMENTATION.')
      WRITE(108,802)
  802 FORMAT(/' INPUT:',
     */'         N-(INTEGER)-THE NUMBER OF COMMUNICATION  CENTERS:',
     */'           N IS THE DIMENSIONALITY OF THE MATRICES BELOW.',
     */'         T-(DECIMAL)-THE TERMINAL CAPACITY MATRIX:',
     */'           EACH ENTRY,    T(I,J), REPRESENTS THE REQUIRED',
     */'           CHANNEL CAPACITY FROM TERMINAL(CENTER) I TO ',
     */'           TERMINAL J.')
      WRITE(108,804)
  804 FORMAT('         D-(INTEGER)-THE ARC CONSTRAINT MATRIX:',
     */'          EACH ENTRY, D(I,J), REPRESENTS THE RELATIVE VALUE',
     */'          OF CONSTRUCTING THE ARC (I,J). LOW VALUES OF D(I,J)',
     */'          GIVE THOSE ARCS HIGH CONSTRUCTION PRIORITIES.')
      WRITE(108,803)
  803 FORMAT(/' OUTPUT:',
     */'         C-(DECIMAL)-THE REQUIRED CAPACITY MATRIX:',
     */'            EACH ENTRY,   C(I,J), REPRESENTS THE CHANNEL CAPACITY'
     */'            THAT MUST BE CONSTRUCTED FROM I TO J IN ORDER TO',
     */'            ACHIEVE THE DESIRED SOLUTION')
    3 I=0
      DO 12 IK=1,90
      LE(IK)=1
   12 T(IK)=0.0
```

```
      KT=0
      CALL NETRED
      DO 2 J=1,N
      DO 2 IK=1,N
      K=N*(J-1)+IK
      X(K)=IK
      CAP(IK,J)=0.
    2 Y(K)=J
      DO 989 JEL=1,10
      ISW1=0
      DO 25 L=1,N
      DO 25 J=1,N
      IF(L.EQ.J)GOTO 25
      DO 24 K=1,N
      IF(L.EQ.K) GOTO 24
      IF(ABS(TERM(L,J)-TERM(K,L)).GE.0.005) GOTO 24
   26 TERM(K,L)=TERM(K,L)+0.01
      ISW1=1
   24 CONTINUE
      DO 25 K=1,N
      IF(J.EQ.K) GOTO 25
      IF(ABS(TERM(L,J)-TERM(J,K)).GE.0.005) GOTO 25
   23 TERM(J,K)=TERM(J,K)+0.01
      ISW1=1
   25 CONTINUE
      IF(ISW1.LE.0) GOTO 987
  989 CONTINUE
      WRITE(108,988)
  988 FORMAT(//' TOO MANY ESSENTIAL INEQUALITIES ARE PRESENT!'//)
      GOTO 990
  987 IF(JEL.LE.1)GOTO 985
      IF(BUG.NE.YES) GO TO 985
      WRITE(108,981)
  981 FORMAT(//' ESSENTIAL INEQUALITIES EXIST! DO YOU WISH',
     */' TO SEE THE PERTURBED TERMINAL CAPACITY MATRIX?',
     */' ANSWER YES OR NO')
      READ(105,982) ANSWER
  982 FORMAT(A1)
      IF(ANSWER.NE.YES)GOTO 985
      WRITE(108,986)
  986 FORMAT(//' THE PERTURBED TERMINAL CAPACITY MATRIX:'/)
      DO 983 L=1,N
  983 WRITE(108,984) (TERM(L,J),J=1,N)
  984 FORMAT(15(1X,F5.2))
  985 CALL NETSRT
   50 I=I+1
      IF(I.GT.KT)GOTO 999
      CALL NETPACK
      IF(N.EQ.(L1+L2+N2)) GOTO 992
      WRITE(108,991)
  991 FORMAT(' ERROR: ON RETURN FROM NETPACK')
      GOTO 990
  992 CALL NETCOMB
      MINCONT=10**6
      IF(MINS.LT.999.0) GOTO 27
      WRITE(108,540)
  540 FORMAT(//' A NETWORK THAT EXACTLY SATISFIES THE GIVEN TERMINAL.
```

```
    * CAPACITY REQUIREMENTS',/ 'DOES NOT EXIST!'//)
      GOTO 990
   27 MINJ=0
       LP=1
      JJ=I-1
      IF(I.LE.1) GOTO 37
      DO 201 J=1,JJ
  201 LP=LP+LE(J)
   37 LQ=LE(I)+LP-1
      DO 200 J=LP,LQ
      XX=X(E(J))
      YY=Y(E(J))
      IF(CONT(XX,YY).GE.MINCONT) GO TO 200
      MINCONT=CONT(XX,YY)
      MINJ=J
  200 CONTINUE
C     EVALUATE CORRESPONDING CAP  VALUE
      IF(MINJ.NE.0) GOTO199
      WRITE(108,993)
  993 FORMAT(//' ERROR: SOME ENTRIES IN THE RESTRAINT MATRIX ARE',
     */' GREATER THAN 10**6 AND THEY MUST BE REDUCED IN SIZE!'//)
      GOTO 990
  199 XX=X(E(MINJ))
      YY=Y (E(MINJ))
      CAP(XX,YY)=T(I)-MINS
      GO TO 50
  999 WRITE(108,501)
  501 FORMAT(//' THE CAPACITY MATRIX BELOW REPRESENTS THE NETWORK',
     */' THAT EXACTLY SATISFIES THE TERMINAL REQUIREMENTS!'//)
      DO 502 I=1,N
  502 WRITE(108,503)(CAP(I,J),J=1,N)
  503 FORMAT(15(1X,F5.1))
      ISW1=0
      DO 354 P=1,N
      DO 354 Q=1,N
      IF(P.EQ.Q) GOTO 354
  360 IF(CAP(P,Q).GE.0.) GOTO 354
      IF(ISW1.GE.1) GOTO 355
      ISW1=1
      WRITE(108,356)
  356 FORMAT(//' NEGATIVE CAPACITIES ARE PRESENT ABOVE!',
     */' DO YOU WISH TO SEE THE COMMUNICATIONS NETWORK?',
     */' (THE NETWORK WITHOUT NEGATIVE CAPACITIES).',
     */' ANSWER YES OR NO!'//)
      READ(105,512) ANSWER
  512 FORMAT(A1)
      IF(ANSWER.NE.YES) GOTO 990
  355 IF((CAP(P,Q)+CAP(Q,P)).GE.0.0) GOTO 357
      WRITE(108,358)
  358 FORMAT(//' A COMMUNICATIONS NETWORK DOES NOT EXIST!'//)
      GOTO 990
  357 DO 350 IK=1,N
      IF(IK.EQ.P.OR.IK.EQ.Q) GOTO 350
      IF(CAP(P,IK).LT.0.0.OR.CAP(IK,Q).LT.0.0) GOTO 350
      IF(CAP(P,IK).GE.0.1.OR.CAP(IK,Q).GE.0.1) GOTO 351
  350 CONTINUE
      DO 352 IK=1,N
```

```
      DO 352 JK=1,N
      IF(IK.EQ.JK) GOTO 352
      IF(IK.EQ.P.OR.JK.EQ.Q) GOTO 352
      IF(CAP(P,IK).LT.Ø..OR.CAP(IK,JK).LT.Ø..OR.CAP(JK,Q).LT.Ø.)GOTO352
      IF(CAP(P,IK).GE..1.OR.CAP(IK,JK).GE..1.OR.CAP(JK,Q).GE..1)GOTO359
  352 CONTINUE
      WRITE(1Ø8,353)
  353 FORMAT(//' TOO MANY NEGATIVE ENTRIES! A COMMUNICATIONS NETWORK',
     */' MAY NOT EXIST!'//)
      GOTO 99Ø
  351 Z=AMIN(CAP(P,IK),CAP(IK,Q),CAP(Q,P))
      CAP(P,IK)=CAP(P,IK)-Z
      CAP(IK,Q)=CAP(IK,Q)-Z
      CAP(Q,P)=CAP(Q,P)-Z
      CAP(P,Q)=CAP(P,Q)+Z
      CAP(Q,IK)=CAP(Q,IK)+Z
      CAP(IK,P)=CAP(IK,P)+Z
      GOTO  36Ø
  359 Z=AMIN(CAP(P,IK),CAP(IK,JK),CAP(JK,Q),CAP(Q,P))
      CAP(P,IK)=CAP(P,IK)-Z
      CAP(IK,JK)=CAP(IK,JK)-Z
      CAP(JK,Q)=CAP(JK,Q)-Z
      CAP(Q,P)=CAP(Q,P)-Z
      CAP(P,Q)=CAP(P,Q)+Z
      CAP(Q,JK)=CAP(Q,JK)+Z
      CAP(JK,IK)=CAP(JK,IK)+Z
      CAP(IK,P)=CAP(IK,P)+Z
      GOTO 36Ø
  354 CONTINUE
      IF(ISW1.LE.Ø) GOTO 333
      WRITE(1Ø8,361)
  361 FORMAT(//' THE COMMUNICATIONS NETWORK THAT EXACTLY SATISFIES THE',
     */' TERMINAL REQUIREMENTS IS REPRESENTED BY THE MATRIX BELOW!'//)
      DO 362 IK=1,N
  362 WRITE(1Ø8,363) (CAP(IK,JK),JK=1,N)
  363 FORMAT(15(1X,F5.1))
  333 WRITE(1Ø8,51Ø)
  51Ø FORMAT(//' DO YOU WISH TO SEE THE CALCULATED TERMINAL',
     */' CAPACITY MATRIX? ANSWER YES OR NO.')
      READ(1Ø5,5Ø5) ANSWER
      IF(ANSWER.NE.YES) GOTO 99Ø
      I=1
      T(I)=1Ø.Ø**6
      DO 1Ø IK=1,N
      DO 1Ø JK=1,N
      IF(IK.EQ.JK) GOTO 1Ø
      S(1)=IK
      S(N)=JK
      LL=Ø
      DO 11 KK=1,N
      IF(KK.EQ.IK.OR.KK.EQ.JK) GOTO 11
      LL=LL+1
      C(LL)=KK
   11 CONTINUE
      N2=N-2
      L1=1
      L2=1
```

```
      CALL NETCOMB
      TERM(IK,JK)=MINS
   10 CONTINUE
      WRITE(108,511)
  511 FORMAT(//' RESULTANT TERMINAL CAPACITY MATRIX:'/)
      DO 13 IK=1,N
   13 WRITE(108,503)(TERM(IK,JK),JK=1,N)
  990 WRITE(108,504)
  504 FORMAT(//' DO YOU WISH TO RESTART THIS PROGRAM?',
     */' ANSWER YES OR NO')
      READ(105,505) ANSWER
  505 FORMAT(A1)
      IF(ANSWER.EQ.YES)GOTO 3
      END
      SUBROUTINE NETRED
C     THIS ROUTINE READS IN THE MATRIX SIZE, THE TERMINAL
C     CAPACITY MATRIX AND THE ARC CONSTRAINT MATRIX.
C
      COMMON/C1/N,TERM(15,15),CONT(15,15)
      COMMON/C8/BUG
      INTEGER CONT
      DATA YES/'Y'/
  599 WRITE(108,600)
  600 FORMAT(//' INPUT THE NUMBER OF NODES PLEASE!')
      READ(105,601)N,BUG
  601 FORMAT(I,A1)
      MN=N*N
      WRITE(108,602)MN,N
  602 FORMAT(//' PLEASE INPUT ',I3,' FLOATING POINT VALUES'/
     * 1X,I2,' PER LINE TO FILL TERMINAL CAPACITY MATRIX'//)
      DO 603 I=1,N
      WRITE(108,610)I
  610 FORMAT(1X,I2,':')
  603 READ (1,604)(TERM(I,J),J=1,N)
  604 FORMAT(15F)
      WRITE(108,605)MN,N
  605 FORMAT(//' PLEASE INPUT ',I3,' INTEGER VALUES'/
     *1X,I2,' PER LINE TO FILL ARC CONSTRAINT MATRIX'//)
      DO 606 I=1,N
      WRITE(108,610)I
  606 READ (1,607)(CONT(I,J),J=1,N)
  607 FORMAT(15I)
      WRITE(108,620)
  620 FORMAT(//' DO YOU WISH TO REVIEW YOUR INPUT?',
     */' ANSWER YES OR NO')
      READ(105,621) ANSWER
  621 FORMAT(A1)
      IF(ANSWER.NE.YES) GOTO648
      WRITE(108,623)
  623 FORMAT(//' THE TERMINAL CAPACITY MATRIX:'/)
      DO 624 I=1,N
  624 WRITE(108,625)(TERM(I,J),J=1,N)
  625 FORMAT(15(1X,F5.2))
      WRITE(108,626)
  626 FORMAT(//' THE ARC CONSTRAINT MATRIX:'/)
      DO 627 I=1,N
```

```
  627 WRITE(108,628)(CONT(I,J),J=1,N)
  628 FORMAT(15(I5,1X))
  648 WRITE(108,629)
  629 FORMAT(//' DO YOU WISH TO RE-ENTER YOUR DATA?',
     */' ANSWER YES OR NO')
      READ(105,621) ANSWER
      IF(ANSWER.NE.YES) GOTO 622
      GOTO 599
  622 RETURN
      END
      SUBROUTINE NETSRT
C
C     TERMINAL VALUES ARE SORTED INTO ASCENDING ORDER
C     IN T AND CORRESPONDINDING ARC NUMBERS IN E
C
      COMMON/C1/N,TERM(15,15),CONT(15,15)
      COMMON/C2/T(210),LE(210),E(210),KT
      INTEGER E
      K=1
      LK=1
      DO 11 J=1,N
      DO 11 I=1,N
      IF(I.EQ.J) GOTO 11
      IF(TERM(I,J).NE.0.) GOTO 12
   11 CONTINUE
   12 T(1)=TERM(I,J)
      E(1)=N*(J-1)+I
      JM=J
      IMM=I
      JMM=1
      DO 5 J=JM,N
      IM=IMM*JMM+1
      JMM=0
      DO 5 I=IM,N
      IF(I.EQ.J) GO TO 5
      IF(ABS(TERM(I,J)).LE.0.001)  GOTO 5
      DO 4 L=1,LK
      IF(ABS(T(L)-TERM(I,J)).LE.0.001) GOTO 8
    1 IF(TERM(I,J).GT.T(L))GOTO4
      DO 10 KK=LK,L,-1
      T(KK+1)=T(KK)
   10 LE(KK+1)=LE(KK)
      T(L)=TERM(I,J)
      LE(L)=1
      LK=LK+1
    7 LP=0
      IF(L.LE.1) GOTO 2
      LQ=L-1
      DO 9 IK=1,LQ
    9 LP=LP+LE(IK)
    2 LQ=LP+1
      DO 3 IK=K,LQ,-1
    3 E(IK+1)=E(IK)
      E(LQ)=N*(J-1)+I
      K=K+1
      GOTO 5
```

```
    8 LE(L)=LE(L)+1
      GOTO 7
    4 CONTINUE
      LK=LK+1
      T(LK)=TERM(I,J)
      K=K+1
      E(K)=N*(J-1)+I
    5 CONTINUE
      KT=LK
      RETURN
      END
      SUBROUTINE NETPACK
      COMMON/C1/N,TERM(15,15),CONT(15,15)
      COMMON/C2/T(210),LE(210),E(210),KT
      COMMON/C3/X(225),Y(225),C(15),S(15)
      COMMON/C4/N2,L1,L2
      COMMON/C6/I
      INTEGER C,S,X,Y,XX,YY,E
      LY=Ø
      L1=Ø
      L2=N+1
      DO 51 J=1,N
   51 S(J)=Ø
      DO 52 J=1,N
   52 C(J)=J
      LP=1
      JJ=I-1
      IF(I.LE.1) GOTO 12
      DO 1Ø J=1,JJ
   1Ø  LP=LP+LE(J)
   12 LQ=LE(I)+LP-1
      DO 72 J=LP,LQ
      XX=X(E(J))
      YY=Y(E(J))
      IF(L1.EQ.Ø)GOTO73
      DO 74 LX=1,L1
      IF(XX.EQ.S(LX))GOTO7Ø
   74 CONTINUE
   73 L1=L1+1
      S(L1)=XX
      C(XX)=Ø
   7Ø IF(YY.EQ.LY) GOTO 72
      L2=L2-1
      S(L2)=YY
      C(YY)=Ø
      LY=YY
   72 CONTINUE
C     LEFT JUSTIFY NODE NUMBERS IN C
      J1=N-1
      DO 58 J=1,J1
      DO 58 J2=J,J1
   65 IF(C(J).NE.Ø) GOTO 58
      DO 59 IK=J,J1
   59 C(IK)=C(IK+1)
      C(N)=Ø
   58 CONTINUE
      DO 6Ø J=1.N
```

```
      IF(C(J).EQ.0) GOTO 61
   60 CONTINUE
   61 N2=J-1
      L2=N+1-L2
      RETURN
      END
      SUBROUTINE NETCOMB
      COMMON/C1/N,TERM(15,15),CONT(15,15)
      COMMON/C2/T(210),LE(210),E(210),KT
      COMMON/C3/X(225),Y(225),C(15),S(15)
      COMMON/C4/N2,L1,L2
      COMMON/C5/CAP(15,15),JB(15),IB(15)
      COMMON/C6/I
      COMMON/C7/MINS
      INTEGER C,S,X,Y,XX,YY,E,CONT
      REAL MINS
   62 MINS=10.0**6
      N3=N2+1
      DO 85 KK=1,N3
      LI=KK-1
      LJ=N3-KK
      IF(N2.EQ.0) GOTO 94
      IF(LJ.EQ.0) GOTO 43
      DO 80 KL=1,LJ
   80 JB(KL)=N2-LJ+KL
   43 MM=0
      M=1
      DO 35 L=1,N2
      IF(M.GT.LJ) GOTO 30
      IF(JB(M).GT.L) GOTO 30
      M=M+1
      GO TO 35
   30 MM=MM+1
      IB(MM)=L
   35 CONTINUE
C     FILL CENTRE PART OF S WITH C
    6 IF(LJ.EQ.0) GO TO 93
      DO 91 IK=1,LJ
   91 S(IK+L1)=C(JB(IK))
   93 IF(LI.EQ.0) GO TO 94
      DO 92 IK=1,LI
      II=L1+LJ+IK
   92 S(II)=C(IB(IK))
   94 FVAL=0.
      II=0
      K3=L1+LJ+1
      L3=L1+LJ
C     TEST FOR S-RESTRICTION
C     CALCULATE VALUE OF CUT AND COMPARE TO MINS
      L3=L1+LJ
      L4=L2+LI
      DO 100 K1=1,L3
      DO 100 K2=1,L4
      K3=L1+LJ+K2
      IF(TERM(S(K1),S(K3)).GT.T(I) )GOTO 108
  100 FVAL=FVAL+CAP(S(K1),S(K3))
```

```
      IF(FVAL.GE.MINS) GO TO 108
      MINS=FVAL
  108 KL=1
      IF(LJ.EQ.0) GO TO 85
   41 IF(JB(KL).GT.KL) GO TO 40
      IF(KL.EQ.LJ) GO TO 85
      KL=KL+1
      GOTO 41
   40 NA=JB(KL)-KL-1
      DO 42 L=1,KL
   42 JB(L)=L+NA
      GO TO 43
   85 CONTINUE
      RETURN
      END
--EOF HIT AFTER 463.
*
```

CURRICULUM VITAE

NAME:	Kalman Csaba Toth
ADDRESS:	96c Carling Avenue, Ottawa, Ontario.
TELEPHONE:	(613) 236-9236
DATE OF BIRTH:	6 October, 1946
PLACE OF BIRTH:	Hodmezovasarhely, Hungary
NATURALIZATION:	9 February, 1955 (Canadian Citizen)
MARITAL STATUS:	Single
HEALTH:	Excellent
LANGUAGES:	English (excellent); Hungarian (excellent); French (fair).
EDUCATION:	<u>1969</u> B.Eng. (Electrical) Carleton University

AWARDS:	1965-69	Kinsmen Club Fellowship
	1965	Carleton University Bursary
	1965	Ontario-Dominion Award
	1970	Carleton University Teaching Assistantship (Computing Science Workshop)

REFERENCES

[1] C. Berge and A. Ghouila-Houri — Programming, Games and Transportation Networks. Methuen and Co. Ltd. London. 1965.

[2] G.B. Dantzig — Linear Programs and Extensions. Princeton University Press, 1963.

[3] S.E. Dreyfus — An Appraisal of Some Shortest Path Algorithms. Rand Corporation, memorandum RM-5433-1-PR, September, 1968.

[4] E.W. Dijstra — A Note on Two Problems in Connection with Graphs. Numerische Mathematik, Vol. 1, 1959.

[5] R.W. Floyd — "Algorithm 97, Shortest Path", Comm. ACM, Vol. 5, No. 6, June 1962.

[6] L.R. Ford and D.R. Fulkerson — Flows in Networks. Princeton University Press. 1962.

[7] L.R. Ford and D.R. Fulkerson — Maximal Flow Through a Network. Canadian Journal of Mathematics, 8, pp 399-404, 1956.

[8] H. Frank and I.T. Frisch — Communication, Transmission and Transportation Networks. Addison-Wesley Publishing Company. 1971.

[9] T.C. Hu — Integer Programming and Network Flows. Addison-Wesley Publishing Company. 1969.

[10] A. Kaufmann — Graphs, Dynamic Programming and Finite Games. Academic Press, 1967.

[11] W.H. Kim and R.T.W. Chien — Topological Analysis and Synthesis of Communication Networks. Columbia University Press, 1962.

[12] S. Lipshutz — Theory and Problems of General Topology, Schamm Publishing Company, New York, 1965.

[13] W. Mayeda — On Oriented Communication Nets. I.R.E. Transactions on Circuit Theory, Vol. CT-9, pp. 261-267, September 1962.

[14] J.A. Resh — On the Synthesis of Oriented Communications Nets. I.E.E.E. Transactions on Circuit Theory, Vol. CT-12, No. 4, pp. 540-546, December 1965.

[15] J.A. Resh — On Networks and Bi-complete Graphs. Report R-174, Co-ordinated, Science Laboratory, University of Illinois, July 1962.

[16] J. de Mercado and N. Spyratos — On the Synthesis of Non Flow Redundant Networks. Proceedings 1972 MRI Symposium on Computer-Communications Networks and Tele-Traffic. Brooklyn Polytechnic, April, 1972.

[17] W.L. Hatton — A Simplified Analysis of Canadian Trunk Communications 1980. Communications Systems Technical Memorandum #16, Communications Research Centre, January 1972.

[18] J. deMercado, R. Roger, N. Spyratos, K.C. Toth. — Networks and Systems Studies. Terrestrial Planning Branch Summer Program 1971, Department of Communications.