ALGORITHMS FOR NETWORK EXPANSION

TIMOTHY ANDREW LANFEAR

A thesis submitted for the

degree of Doctor of Philosophy of the University of London

and the

Diploma of Membership of Imperial College

April 1985

Department of Electrical Engineering

Imperial College of Science and Technology

University of London

Abstract

In this thesis several new algorithms for the synthesis of communication networks are developed. More specifically, they are concerned with the expansion of networks, that is the addition of nodes and edges to a pre-existing network in some optimal fashion so as to satisfy increased terminal capacity matrix requirements.

The first problem to be considered is that of adding edges to a network so as to meet increased flow requirements between all pairs of nodes in the network, such that the total capacity of the resulting network is minimum. The algorithm proceeds by generating a sub-optimal network which meets the flow requirements and then modifies it by a sequence of branch exchanges which do not violate the flow requirements until a minimum capacity network is obtained. In the course of this work a new set of conditions under which it is easy to calculate the terminal capacity matrix of a network formed by the superposition of two other networks are given and the necessary conditions for an expansion to be realisable are developed.

Then an extension to this problem is considered, in which costs per unit capacity are attributed to each edge and the new objective function to be minimised is the total cost of the network. Again, the algorithm involves the development of a sub-optimal network which is modified by a series of branch exchanges, but in this instance, the ordering of the exchanges to produce the optimal network cannot be specified 'a priori' and so a tree search method to locate the optimal ordering is developed. A bounding function to speed the search is obtained through solving, by linear programming,

a relaxed version of the problem. The algorithm requires a large quantity of computing resources and so some locally optimal heuristics to solve the problem are suggested and evaluated.

Finally, these two algorithms are extended to the situation in which it is desired simultaneously to add nodes and edges to a network to synthesise a network satisfying a terminal capacity matrix of higher order than that of the original network.

CONTENTS

Statement of Originality

The following ideas reported in this thesis are believed to be original contributions to the study of networks of flow.

(1) The development of a set of conditions under which the terminal capacity matrix of a network formed by the superposition of two other networks may be easily calculated (Section 3.3.2).

(2) A branch exchange for reducing the capacity of a network whilst not changing the terminal capacity matrix of the network (Section 3.3.3).

(3) An application of (1) and (2) in an algorithm for expanding a network by the addition of branches such that the resulting network has minimum capacity and satisfies a requirement terminal capacity matrix (Sections 3.3.4 and 3.3.6).

(4) The development of the necessary conditions which an expansion must satisfy to be exactly realisable (Section 3.3.5).

(5) A branch exchange which is a generalisation of (2) for reducing the cost of a network in which each branch has a different cost per unit capacity (Section 4.2.2).

(6) An application of (1) and (5) in an algorithm and two sub-optimal heuristics for minimum cost expansion of networks (Sections 4.3 and 4.4).

(7) An extension of all the above to the simultaneous addition of nodes and edges to a network (Chapter 5).

Acknowledgements

I would like to thank my supervisor Dr. A. G. Constantinides for introducing me to the subject of graph theory and for his guidance and helpful comments during the preparation of this thesis.

I would like to thank my colleagues in the Signal Processing Section for providing a stimulating and enjoyable environment for engaging in research. My thanks are also due to both my family and friends for their support and forebearance.

Finally, I would like to thank the Science and Engineering Research Council who provided the finance for this research.

INTRODUCTION

1.1 Graphs as Structural Models

Many physical systems involve the communication, transmission or movement of commodities. The commodity transported may be such things as information, electricity or traffic, and so such systems as computer networks, radio and telephone networks and traffic systems all involve the flow of a commodity through a network. These networks may conveniently be modelled by the mathematical concept of a graph or network.

A graph may be considered as a set of points, termed nodes or vertices, which are connected together by lines known as edges, branches, arcs or links. The nodes represent points between which the commodity should be transported and the presence of an edge shows that there is a transportation path between the nodes at either end of the edge. Such a diagram represents the structure of the network.

Two networks may be structurally identical and yet have different characteristics because of the different properties of the elements of the network. This non-structural information concerning the network may be included in the model by attaching parameters or weights to the edges and vertices. Some typical parameters might be the maximum quantity of the commodity which may be transferred through a vertex, the cost of an edge or the reliability of an edge. Graph theory is the mathematical study of the objects resulting from this modelling process.

One aspect of graph theory is concerned with the development of algorithms both for the analysis of networks -that is the

determination of the properties of a network- and synthesis of networks -that is the generation of networks satisfying certain constraints or having certain properties. In many synthesis problems there may be a number of networks which might satisfy the constraints but there may be some criterion for choosing the best ie some cost function of the network should be maximised or minimised. Thus, the synthesis algorithms should not only generate a network which satisfies the constraints but also find the optimum network out of a number of candidate solutions.

Some typical problems will be described below.

One class of problems, which may be termed structural problems, is concerned with such questions as whether a path exists between a pair of vertices or, how many paths there are and if there are several paths, which is the shortest or longest. A typical synthesis problem of this class is concerned with the sythesis of networks with prescribed connectivity properties and with minimum total weight, where weights are ascribed to each candidate edge in the network.

The existence of a path between a pair of nodes implies nothing about the quantity of flow which can sent along the path. To include this information it is necessary to weight each edge (and possibly also each vertex) with a number representing the maximum quantity of flow it can accommodate. These numbers represent the capacities of channels, sources, terminals and relay points of the network. The Maximum Flow Problem is to find the maximum quantity of flow which can be transported between a pair of vertices. There are corresponding network synthesis problems where the aim is to

generate a network with the capacity to transport flow at prespecified levels between pairs of nodes. These again may be solved under some optimality constraint. It is problems of this nature which will be considered in detail in this thesis.

An example of a situation where graph theory has been used is the design of the Advanced Research Projects Agency Network (ARPANET). This is a computer communication network connecting computers across the USA. The design of this network brought together the disciplines of graph theory, computer science, communication theory, operations research and others. Many theoretical problems were encountered in the topological optimisation of this network to produce, at minimum cost, a network design which met the design constraints for reliability and traffic handling capacity [FR3], [FR4].

1.2 Definition of Terms

In this section some of the more common graph-theoretic terms used in this thesis will be defined. Any undefined terms will be used as in the books of Christofides [CH1] and Frank and Frisch [FR1].

A graph $G=(X,A)$ consists of two sets, a set of n elements $X=\{x_1,x_2,\ldots,x_n\}$ termed nodes or vertices and a set of m elements $A=\{a_{xi,xj} \mid x_i,x_j \in X\}$ termed edges, arcs, branches or links. It will often be convenient to refer to elements by their index only; thus, the node x_3 may be referred to as node 3 and edge $a_{x1,x2}$ as $a_{1,2}$ or even (1,2). The use and meaning of such symbols will be clear from the context.

Pictorially the set X may be represented by a set of points and the set A by the set of lines between x_i and x_j. If the order of i,j is important then the graph is directed and this is shown by an arrow on the line $a_{i,j}$; otherwise the graph is undirected. A path is any sequence of arcs where the final vertex of an arc is the initial vertex of the next. A graph is connected if there is a path between every pair of nodes, otherwise it is disconnected. For undirected graphs, the degree of a node is the number of edges incident upon the node; there are two corresponding terms for directed graphs, indegree and outdegree with obvious definitions.

Define two sets $\Gamma(x_i)$ and $\Gamma^{-1}(x_i)$

$$\Gamma(x_i)=\{x_j | a_{i,j} \in A\}$$

$$\Gamma^{-1}(x_i)=\{x_k | a_{k,i} \in A\}$$

The interpretation of these two sets is that $\Gamma(x_i)$ is the set of nodes which can be reached from x_i along a single edge and $\Gamma^{-1}x_i$ is the set of nodes which can reach to x_i along a single edge.

Given a graph G, a partial graph G_p of G is the graph (X,A') where $A' \subset A$, ie the partial graph has the same vertices but only some of the edges of the original graph. A subgraph of G is a graph $G_s=(X_s,A_s)$ such that $X_s \subset X$ and $\Gamma_s(x_i)= \Gamma(x_i) \cap X_s$ for every $x_i \in X_s$. Thus a subgraph consists of a subset of the vertices of G but contains all the edges in G between those vertices in the subset.

A tree is a special type of graph which is a connected graph with n vertices and (n-1) edges. A spanning tree of a graph G is a partial graph of G which is also a tree.

The most common branch weights are the flow along a branch $f_{i,j}=f(x_i,x_j)$, the capacity of a branch $b_{i,j}=b(x_i,x_j)$ which is the

upper bound on the flow along arc $a_{i,j}$ and the cost $c_{i,j}=c(x_i,x_j)$ which is the cost of the branch $a_{i,j}$. These parameters for the whole graph are given by the matrices F, the flow pattern, B, the branch capacity matrix, and C, the cost matrix. For undirected graphs the matrices B and C are symmetrical and the main diagonal elements are undefined.

A flow pattern is feasible if it satisfies the following two equations

$$\sum_{x_j \in \Gamma(x_i)} f_{i,j} - \sum_{x_k \in \Gamma^{-1}(x_i)} f_{k,i} = \begin{cases} v_{s,t} & \text{if } x_i = s \\ -v_{s,t} & \text{if } x_i = t \\ 0 & \text{otherwise} \end{cases}$$

$$f_{i,j} \leq b_{i,j} \quad \forall \ a_{i,j}$$

The first is an equation of conservation of flow stating that there are two vertices s and t which source and sink flow and at all other vertices all the flow entering the vertex also leaves. The constant v is the value of the flow. The second equation states that the flow in each arc is less than or equal to the capacity of the arc.

A branch cutset is a minimal set of branches, which when removed from the graph will disconnect the graph. (A set X is minimal with respect to some property P if no proper subset of X has property P.) Closely related to the concept of a branch cutset is the concept of a cut. Let X_1 and X_2 be two subsets of the set of vertices X. Define (X_1,X_2) to be the set of branches leading out of an element of X_1 and incident on an element of X_2 ie $(X_1,X_2) = \{a_{i,j} | x_i \in X_1, x_j \in X_2\}$. If X_1 is a set of nodes and $\overline{X}_1$ is the complementary set then $(X_1,\overline{X}_1)$ is a cut. If each arc in the cut has a capacity then the capacity of the cut $c(X_1,\overline{X}_1)$ is the sum of the

capacities of the arcs in the cut.

1.3 Description of Thesis

1.3.1 Subject Matter of Thesis

We examine in this thesis some new problems in the synthesis of networks of flow. These are concerned with the expansion of networks, viz. the situation where after an initial network has been constructed, the flow requirements change and it becomes necessary to modify the network by the addition of extra edges or by the addition of extra nodes. The additions should be made such that the new network is in some sense optimal.

This is a problem of combinatorial optimisation. The most simplistic approach to problems of this type is to try all possible expansions of the network and choose that which is best, but this approach fails because of the "combinatorial explosion". It is not uncommon in work of this nature for there to be 10^{20} or more candidates for the optimum solution and so it is clearly impractical to attempt to solve the problem by this approach. Therefore it is necessary to study the problem in great detail so as to develop an algorithm which is able to locate the optimum solution in some intelligent fashion and in a reasonable time.

A number of classical optimisation techniques such as linear programming, dynamic programming and branch and bound search methods together with some algorithms specific to graph theory will be used in the course of this thesis.

1.3.2 Contents of Individual Chapters

Outlines of the individual chapters of the thesis are given below.

Chapter two reviews the analysis and synthesis techniques for networks of flow. Single source/sink and multi-terminal network flow analysis methods are described. The synthesis algorithms described are for the generation of networks to satisfy flow requirements where any pair of vertices can be considered as source and sink. It is not possible to satisfy an arbitrary set of requirements and so the necessary conditions for a set of requirements to be realisable are given and a method for generating realisable requirements from an unrealisable set is described. Some algorithms for minimum cost synthesis are described and the previous work in the expansion of networks is presented.

Chapter three states a problem in the expansion of networks and provides the theoretical basis for the solution of the problem. The problem is concerned with the addition of branches to a network so as to satisfy a set of flow requirements greater than that which the network can handle, under the constraint that the total capacity added to the network should be minimum. An algorithm is proposed and discussed and the results of some computational experience are given.

Chapter four presents a generalisation of the problem of the previous chapter in which a cost is given to each candidate edge in the expansion and it is required that the total cost of the new network should be minimum. Some more theoretical results to aid in

solving it are derived and an algorithm to find the optimal solution is given. The algorithm is found to be computationally expensive and so a number of suboptimal heuristics are proposed and evaluated.

Chapter five presents an extension of the algorithms of the previous chapters in which nodes as well as branches may be added to the network so as to satisfy a set of flow requirements between the original nodes of the network and also some additional nodes. Thus we shall here present a general method for network expansion.

The final chapter presents the conclusions of the thesis and suggests some possible areas for further research.

SURVEY OF NETWORK ANALYSIS AND SYNTHESIS

2.1 Introduction

A new area in graph theory which opened up in the late 1950's was the study of the steady state flow of information through a network. This subject was first considered by Elias, Feinstein and Shannon [EL1], although Mayeda [MA1] was the first to formulate and solve significant new problems. The contributions made by these workers and others which followed them will be reviewed in this chapter. These network analysis and synthesis techniques will form the theoretical background to the main topic of this thesis ie the expansion of networks.

We shall begin by examining methods of analysis, that is methods of determining the maximum quantity of commodity which can be transported between a pair of nodes. This is then extended to multiterminal network analysis where all pairs of nodes may generate or absorb flow and it is shown that for a certain class of graphs termed pseudosymmetric the computation required to calculate the maximum flow between all pairs of nodes is much less than would be expected.

Following the above, the synthesis of networks will be examined. This section will be concerned with the generation of networks which are able to transport prespecified quantities of commodity between pairs of vertices. We shall examine the synthesis of both undirected and directed graphs and also methods for synthesising minimum cost graphs where the cost function for determining the optimum graph may be either the total capacity of the network or the total cost of the network where a cost per unit capacity is applied to each candidate branch for the solution

network.

The chapter will conclude with an examination of the work which has already been carried out in the field of the expansion of networks, which is concerned with the optimum addition of branches to a network so as to increase the source to sink flow under a number of different optimality criteria.

2.2 Network Flow Analysis

2.2.1 The Maximum-Flow Minimum-Cut Theorem

A problem which often arises in flow problems is to find a feasible flow pattern in a graph which maximises the value of the flow between source and sink ie to find the set $\{f_{i,j}\}$ which maximises $v_{s,t}$ subject to

$$\sum_{x_j \in \Gamma(x_i)} f_{i,j} - \sum_{x_k \in \Gamma^{-1}(x_i)} f_{k,i} = \begin{cases} v_{s,t} & \text{if } x_i = s \\ -v_{s,t} & \text{if } x_i = t \\ 0 & \text{otherwise} \end{cases}$$

$$f_{i,j} \leq b_{i,j} \quad \forall \ a_{i,j}$$

There are two questions to be answered here. (1) What is the value of the maximum flow? (2) What flow pattern gives the maximum flow?

The first question is answered by the Maximum-Flow Minimum-Cut Theorem which states that the maximum flow between a source x_s and sink x_t is equal to the capacity of the minimum cut between x_s and x_t ie

$$\max(v_{s,t}) = \min(c(X_1, \bar{X}_1)) \qquad X_1 \subset X,\ s \in X_1,\ t \in \bar{X}_1$$

where the maximisation of $v_{s,t}$ is over all flow patterns and the

minimisation of $c(X_1, \bar{X}_1)$ is over all sets $X_1 \subset X$. In 1956 three different proofs of this theorem were given. The proof of Ford and Fulkerson [FO1] is combinatoric in that it relies on the structure of the flow paths which maximises $v_{s,t}$; that of Elias, Feinstein and Shannon [EL1] uses a graph theoretic technique of decomposition into smaller graphs until such a point is reached that the solution is obvious for the simpler graphs and from which the solution of the original problem can be inferred; and the proof of Dantzig and Fulkerson [DA1] is based on the theory of linear programming.

A number of algorithms exist for generating the flow pattern which attains the upper bound on the flow set by the theorem [DI1], [ED1], [HU1], [JO1] which are all improvements on the well known labelling algorithm of Ford and Fulkerson [FO2].

2.2.2 Multiterminal Network Analysis.

We now consider the more general analysis problem of finding the maximum flows for all source-sink pairs when each of the flows is sent through the network separately. The information obtained about all the minimum cuts between all pairs of nodes is stored in a matrix T, the terminal capacity matrix (T-matrix) where the entry $t_{i,j}$ is the capacity of the minimum cut between x_i and x_j.

There is a number of physical situations in which the terminal capacity matrix is useful. From the point of view of reliability or vulnerability, the entry $t_{i,j}$ in the terminal capacity matrix represents the "weakest" section between x_i and x_j. Indeed, if the capacity of each edge is set to unity then the entry in the T-matrix is the minimum number of edges which must be removed to disconnect

x_i from x_j. From the point of view of communication networks, then if only one user is allowed on the system at one time (as in a teletype system where there is only one sender at a time) then the T-matrix gives the communication capacity between all pairs of nodes.

There are $n(n-1)$ entries in the matrix - (the main diagonal elements are meaningless)- and for general directed graphs this number of flow calculations must be performed to calculate the matrix. However, Gomory and Hu [GO1] showed that for undirected graphs the T-matrix can be calculated in $(n-1)$ flow calculations. This was later extended by Gupta [GU1] to include a certain class of directed graphs termed pseudosymmetric.

A graph is pseudosymmetric if

$$c(x_i,X)=c(X,x_i) \quad \forall \; x_i \in X$$

that is, if the sum of the capacities of the arcs directed towards a node is equal to the sum of the capacities of the arcs directed away from that node.

For a pseudosymmetric graph

$$\sum_{x_i \in X_i} c(x_i,X)= \sum_{x_i \in X_i} c(X,x_i) \quad \forall \; X_i \subset X$$

or

$$c(X_i,X)=c(X,X_i)$$

Since $X=X_i \cup \bar{X}_i$ and $X_i \cap \bar{X}_i=\emptyset$

the L.H.S. of this equation can be written

$$c(X_i,X_i \cup \bar{X}_i)=c(X_i,X_i)+c(X_i,\bar{X}_i)=c(X_i,\bar{X}_i)$$

and the R.H.S. as

$$c(X_i \cup \bar{X}_i, X_i) = c(X_i, X_i) + c(\bar{X}_i, X_i) = c(\bar{X}_i, X_i)$$

Thus for a pseudosymmetric graph

$$c(X_i, \bar{X}_i) = c(\bar{X}_i, X_i) \quad \forall \quad X_i \subset X$$

This shows that the T-matrix of a pseudosymmetric graph (which is a matrix of capacities of cuts) is symmetric.

An efficient algorithm for calculating the T-matrix of a pseudosymmetric graph relies upon the concept of flow-equivalence. Two graphs are flow-equivalent if the minimum cut between two nodes in one graph is equal to the minimum cut between the corresponding nodes in the other. The central idea of the algorithm is to take a graph and calculate flow equivalent tree from which the entry $t_{i,j}$ in the T-matrix is then easily determined by finding the capacity of the minimum capacity edge in the (unique) path between x_i and x_j.

The flow calculations required to calculate the flow equivalent tree may be simplified by using the idea of "vertex condensation". Suppose that an s to t maximum flow problem for a graph G=(X,A) has been solved for two vertices selected at random. The minimum cut partitions the vertices into two sets X_1, X_2. Consider two vertices x_i and x_j both in X_1 (or X_2) and we wish to calculate the value of the minimum cut between them. All the vertices in X_2 (or X_1) may be "condensed" into a single vertex for this flow calculation. The condensation is such that all nodes in X_2 are replaced by a single node, say x_k, and all edges $a_{i,j}$ with $x_i \in X_1$ and $x_j \in X_2$ are replaced by edges $a_{i,k}$ of the same capacity.

The process of condensation is shown in figs 2.1(a),(b),(c). The flow calculations to locate a minimum i-j cut may be performed on this condensed graph. The correctness of this procedure is shown in [GO1] where it is proved that all vertices in X_2 lie on the same side of the cut separating x_i and x_j, so that the internal properties of the subgraph X_2 do not enter into the minimum i-j cut calculation.

The method for generating the flow equivalent tree is as follows.

(1) For any x_i, x_j in a graph G, find the minimum i-j cut (X_1, X_2) and represent the cut by two generalised vertices X_1, X_2 with a branch of capacity $c(X_1, X_2)$ joining them.

(2) Choose two vertices in X_1, say x_k, x_l, and condense all the vertices in the remainder of G into a single vertex and find the minimum k-l cut (X_3, X_4). The resulting cut with value $c(X_3, X_4)$ is represented by a branch connecting X_3 and X_4 with X_3 or X_4 next to X_2 according to which of them is on the same side of the cut (X_3, X_4) as X_2.

(3) Repeat (2) until each generalised vertex contains only one node and the resulting graph is a tree. This point is reached after (n-1) flow calculations.

A formal proof of the correctness of the procedure is given in [GO1].

Example

Consider the pseudosymmetric graph shown in fig 2.2(a). Arbitrarily select x_2 and x_4 and locate the minimum 2-4 cut

$(\{x_1, x_4, x_5\}, \{x_2, x_3\})$ which has value 7. The graph is thus represented by the tree with two generalised vertices shown in fig 2.2(b). Next select vertices x_1 and x_4 and find the minimum 1-4 cut in the graph obtained by condensing x_2, x_3 fig 2.2(c). This cut has value 6 and the tree obtained is fig 2.2(d). Then find the minimum 2-3 cut with x_1, x_4 and x_5 condensed giving fig 2.2(e) and the minimum 4-5 cut with x_2 and x_3 condensed giving the final flow equivalent tree of fig 2.2(f).

The terminal capacity matrix of this tree is easily calculated and is equal to the T-matrix of the original graph.

$$T = \begin{bmatrix} - & 6 & 6 & 6 & 5 \\ 6 & - & 6 & 7 & 5 \\ o & 6 & - & 6 & 5 \\ 6 & 7 & 6 & - & 5 \\ 5 & 5 & 5 & 5 & - \end{bmatrix}$$

It should be noted that the flow equivalent tree is not unique; an alternative tree is shown in fig 2.2(g).

An alternative and useful view of the algorithm is to realise that it generates a set of (n-1) non-intersecting cuts which completely characterise the flow properties of the network. From this viewpoint it is intuitively clear that only (n-1) flow calculations need be performed to construct the flow equivalent tree. The set of non-intersecting cuts and their capacities for the given example are shown in fig 2.2(h).

2.3 Multiterminal Network Synthesis

In this section we shall examine the problem of synthesising a network where each entry in its terminal capacity matrix is equal to the corresponding entry in a requirement matrix. Since not all

requirement matrices are realisable as networks, as the various algorithms for realising requirement matrices are given, we shall build up the necessary and sufficient conditions for a matrix to be the terminal capacity matrix of a network. Finally we will give a method of generating a matrix that can be realised from an unrealisable matrix.

2.3.1 Properties of Terminal Capacity Matrices

In this section we will give some necessary conditions for a matrix to be realisable as a network and note the special cases for which the sufficiency conditions are particularly simple.

Two conditions which it is necessary for a matrix to satisfy for it to be a terminal capacity matrix are that it should be semiprincipally partitionable and that it should satisfy the triangle inequality. The necessity of these conditions was proved by Tang and Chein [TA1].

Definition

A matrix is said to be semiprincipally partitioned if, after possibly permuting rows and corresponding columns, the matrix satisfies the following:

(a) T is square

(b) T has only real non-negative entries

(c) T can be partitioned as below where T_{11} is square and all entries in T_{12} are equal to the smallest entry in T

$$T = \left[\begin{array}{c|c} T_{11} & T_{12} \\ \hline T_{21} & T_{22} \end{array}\right]$$

(d) Every submatrix on the main diagonal resulting from a

partitioning can again be partitioned until all submatrices on the main diagonal are of order one.

Theorem 2.1 [TA1]

If T is the terminal capacity matrix of a graph then T is semiprincipally partitionable.

Theorem 2.2 (The Triangle Inequality) [TA1]

If T is the terminal capacity matrix of a graph then

$$t_{i,j} \geq \min(t_{i,k}, t_{k,j})$$

Tang and Chein have shown [TA1] that for a graph with three nodes or less then the necessary and sufficient conditions for a matrix to be realisable as a network are that it should satisfy the conditions of theorems 2.1 or 2.2, but that for graphs with four or more nodes these conditions are necessary but not sufficient. It is not guaranteed in these cases that the branch capacities will be positive.

Mayeda [MA2] considered the realisation of a certain class of semiprincipally partitioned matrices called completely partitioned, for which a simple realisation scheme can be given.

Definition

A matrix R is completely partitioned if

(1) R is semiprincipally partitioned.

(2) When in semiprincipally partitioned form R can also be partitioned (without rearranging rows and columns) as

$$R=\left[\begin{array}{c|c} R_{11} & R_{12} \\ \hline R_{21} & R_{22} \end{array}\right]$$

where

(a) all elements in R_{21} are equal to the smallest element in R below the diagonal and

(b) all resulting submatrices on the main diagonal can be partitioned in the same way until the resulting diagonal matrices are of order one.

eg consider the semiprincipally partitioned matrix R

$$R=\left[\begin{array}{ccc|cc} - & 4 & 3 & 1 & 1 \\ 4 & - & 3 & 1 & 1 \\ \hline 4 & 8 & - & 1 & 1 \\ \hline 4 & 6 & 6 & - & 2 \\ 4 & 6 & 6 & 7 & - \end{array}\right]$$

which can also be partitioned without rearranging as

$$R=\left[\begin{array}{c|ccc|cc} - & 4 & 3 & 1 & 1 \\ \hline 4 & - & 3 & 1 & 1 \\ 4 & 8 & - & 1 & 1 \\ \hline 4 & 6 & 6 & - & 2 \\ 4 & 6 & 6 & 7 & - \end{array}\right]$$

Thus, R is completely partitioned and can be realised as in fig 2.3. All completely partitioned matrices can be similarly realised as networks as proved by Mayeda [MA2].

If the requirement matrix is symmetric ie the network realising the matrix is undirected, then again a simple method of network synthesis can be given.

Definition

If a matrix can be semiprincipally partitioned and is symmetric it is said to be principally partitioned.

Theorem 2.3 [MA1]

A symmetric matrix T is a terminal capacity matrix of a graph if and

only if, after possibly permuting rows and corresponding columns, it can be principally partitioned.

Since a complete partitioning of a symmetric matrix is also a principal partitioning, the realisation method for a principally partitioned matrix is the same as that for a completely partitioned matrix but with undirected arcs.

eg the matrix T below is realised as in fig 2.4.

$$T=\begin{bmatrix} - & 5 & 3 & 1 & 1 \\ 5 & - & 3 & 1 & 1 \\ 3 & 3 & - & 1 & 1 \\ 1 & 1 & 1 & - & 2 \\ 1 & 1 & 1 & 2 & - \end{bmatrix}$$

Thus, so far we have the following.

(a) The necessary and sufficient condition for a symmetric matrix to be realisable as an undirected graph is that it should be principally partitionable.

(b) A necessary condition for an unsymmetric matrix to be realisable as a directed graph is that it should be semiprincipally partitionable, but this is not a sufficient condition.

(c) A sufficient condition for an unsymmetric matrix to be realisable is that it can be completely partitioned, but this condition is stronger than is necessary.

2.3.2 Semigraphs

We shall now introduce some additional concepts which will enable us to give general necessary and sufficient conditions for realisability. The first of these is a semigraph.

We shall give methods for transforming a graph to and from a graph with the same terminal capacity matrix but a different branch

capacity matrix so that our synthesis procedures can be limited to the synthesis of this special type of graph known as a semigraph. The transformation technique is known as shifting. The concepts of semigraphs and shifting were introduced independently by Resh [RE1] and Sen and Frisch [SE1].

Definition [SE1]

A semigraph is a graph with n vertices $x_1, x_2, \ldots, x_n$ such that

$b_{i,j}=0$ $j>i+1$

$b_{i+1,i}$ is unrestricted in sign for $i=1,2,\ldots,(n-1)$

$b_{i,j} \geq 0$ otherwise.

Any graph with positive branch capacities can be converted to a semigraph by a process called shifting, which is best described using the ideas of forward, backward and double circuits, where a circuit is a closed path.

Definitions [SE1]

A forward circuit $L_f(i,j)$ is a directed circuit with two or more branches containing (i,j).

A backward circuit $L_b(i,j)$ is a directed circuit with two or more branches containing (j,i).

A double circuit L_d is a backward circuit and a forward circuit such that $(k,l) \in L_f(i,j)$ iff $(l,k) \in L_b(i,j)$.

For a given (i,j) and a given double circuit the procedure for shifting is:

(1) For all $(l,k) \in L_f(i,j)$ increase $b_{l,k}$ by a real number $s_{i,j}$.

(2) For all $(l,k) \in L_b(i,j)$ decrease $b_{l,k}$ by $s_{i,j}$.

To convert a graph to a semigraph shifting should be applied to all forward circuits of three branches such that

$$L_f(j,i)=\{(j,i),(i,i+1),(i+1,j)\}$$

and $s_{i,j}=b_{i,j}$

where i ranges from 1 to (n-2) in increasing order and j ranges from (i+2) to n in increasing order.

Example

Consider the graph in fig 2.5(a), which is to be converted to a semigraph. First select the circuit $L_f(3,1)=\{(3,1),(1,2),(2,3)\}$, $s_{1,3}=11$. The shifting procedure gives fig 2.5(b). Note that $b_{2,1}$ has become negative which is acceptable. Selecting the circuits $\{(4,1),(1,2),(2,4)\}$, $s_{1,4}=12$ and $\{(4,2),(2,3),(3,4)\}$, $s_{2,4}=24$ gives the semigraph of fig 2.5(c).

Since we will not allow negative flow, the maximum s-t flow is now no longer equal to the value of the smallest s-t cut, and further the value of the minimum s-t cut is not equal to the smallest s-t cutset. Despite the fact that many of the intuitive properties of graphs concerned with flows are modified we have the following important result.

Theorem 2.4 [SE1]

The terminal capacity matrix of a graph is invariant under shifting.

From these results we see that any terminal capacity matrix realisable as a graph with nonnegative branch capacities is realisable as a semigraph. We could then limit our synthesis procedure to the realisation of semigraphs if we could find a method for converting a semigraph to a graph with nonnegative branch capacities.

The method for converting a semigraph to a graph is as follows [SE1].

For each branch (i,j) with negative capacity $b_{i,j}$ apply shifting to all possible double circuits containing (i,j) in the forward circuit. For the double circuit $L_d(i,j)$ let

$$s_{i,j}=\text{Max}(0,m(i,j))$$

$$\text{where } m(i,j)= \underset{(l,k)\in L_b(i,j)}{\text{Min}} (b_{l,k}, -b_{i,j})$$

Theorem 2.5 [SE1] [RE1]

A semigraph can be converted to a graph with nonnegative branch capacities if and only if in the semigraph $b_{l,k}+b_{k,l}\geq 0$ $\forall$ (l,k) and all entries in the T-matrix are nonnegative.

From this theorem we have the corollary that a requirement matrix is realisable as a graph with nonnegative branch capacities if and only if it is realisable as a semigraph in which $b_{l,k}+b_{k,l}\geq 0$ $\forall$ (l,k). We may therefore restrict our synthesis procedures to the synthesis of semigraphs which may be then shifted to obtain graphs.

2.3.3 Cut Matrices

The other concept that should be introduced is that of a cut-matrix. The idea was introduced by Mayeda [MA2].

Definitions

Consider an arbitrary matrix M whose rows and columns correspond to the vertices of a graph. For an arbitrary subset of vertices $X_i \subset X$, the cut-matrix M_i is the submatrix of M formed by deleting the set of columns corresponding to X_i and the complementary set of rows.

If $m_{k,j}$ is an element of M_i then M_i is said to be a cut-matrix of $m_{k,j}$ and may be written explicitly as $M_i(k,j)$. If $m_{k,j}$ is a largest entry in M_i then M_i is said to be a min-cut matrix of $m_{k,j}$ and is written as $\mu_i(k,j)$ or μ_i.

Theorem 2.6 [MA2]

If T is the terminal capacity matrix of a graph, then every element of T has at least one min-cut matrix.

It can be shown that the three necessary conditions for a T-matrix to be realisable (1) it is semiprincipally partitionable (2) the triangle inequality holds and (3) every element has a min-cut matrix, are equivalent [MA2],[TA1].

Definitions

Consider a min-cut matrix $\mu_i(k,j)$. If $m_{k,j}$ is greater than or equal to all other elements in μ_i then μ_i is a semidistinct min-cut matrix of $m_{k,j}$ denoted by $S(k,j)$. If $m_{k,j}$ is the unique largest element then it is a distinct cut-matrix of $m_{k,j}$.

Two elements $m_{a,b}$, $m_{c,d}$ are said to be coupled if they are both equal to the largest element in the same min-cut matrix. Otherwise they are uncoupled. If $m_{a,b}$ and $m_{c,d}$ are coupled they may be realised by a single cut $(X_1,\bar{X}_1)$ ie $t_{a,b}=t_{c,d}=c(X_1,\bar{X}_1)$ with x_a, $x_c \in X_1$, x_b, $x_d \in \bar{X}_1$. Next, $m_{a,b}$, $m_{c,d}$ are completely coupled if every min-cut matrix of $m_{a,b}$ is a min-cut matrix of $m_{c,d}$ and vice versa. Finally $m_{a,b}$ and $m_{c,d}$ are min-coupled if the set of minimum a-b cuts is identical to the set of c-d cuts.

The next theorem shows that given a matrix R in semiprincipally partitioned form, if a semigraph realises the

elements $r_{k,k+1}$ $k=1,\ldots,(n-1)$ then all other elements above the diagonal are automatically realised if the semigraph has certain properties. This implies that, if R^0 is the matrix R but with $r_{i,j}=0$ for $j>i+1$, then our realisation methods need only consider the elements of R^0, provided that the semigraph has certain properties.

Theorem 2.7 [FR2]

Given a requirement matrix R in semiprincipally partitioned form and a semigraph such that $t_{k,k+1}=r_{k,k+1}$ for $k=1,\ldots,(n-1)$ then $t_{i,j}=r_{i,j}$ for $j>i+1$ if one of the following is true.

(1) Each entry $r_{i,j}$ for $j>i+1$ is coupled to some element $r_{a,b}=r_{i,j}$ by cut-matrix R^0_q such that in the semigraph $c(X_q,\bar{X}_q)=r_{a,b}$ when $(X_q,\bar{X}_q)$ is an i-j cut.

(2) $r_{k,k+1}=b_{k,k+1}$ for $k=1,\ldots,(n-1)$.

(3) The largest entry in R^0 above the main diagonal is smaller than all the entries below the diagonal.

(4) All branch weights in the semigraph are non-negative.

2.3.4 Realisation Algorithms for Directed Graphs

We are now in a position to consider algorithms for the realisation of directed graphs. Four algorithms, each more general than the previous, will be presented. They are the Substitution algorithm, the Perturbation algorithm, the Replacement algorithm and the Terminal Capacity Realisation algorithm.

The Substitution Algorithm [SE1] [RE1]

The Substitution algorithm is used to realise maximally distinct requirement matrices. A matrix M is maximally distinct if all off diagonal entries in M^0 are numerically distinct. A simple and direct algorithm can be given for the realisation of this type of matrix.

The algorithm will first be illustrated with an example and then described formally.

Consider the requirement matrix R and the matrix R^0 formed from it.

$$R=\begin{bmatrix} - & 3 & 3 & 3 \\ 10 & - & 4 & 4 \\ 11 & 6 & - & 5 \\ 12 & 8 & 7 & - \end{bmatrix} \quad \text{which gives} \quad R^0=\begin{bmatrix} - & 3 & 0 & 0 \\ 10 & - & 4 & 0 \\ 11 & 6 & - & 5 \\ 12 & 8 & 7 & - \end{bmatrix}$$

The element $r_{1,2}$ has only one min-cut matrix formed by $X_i=\{x_1\}$, $\bar{X}_i=\{x_2,x_3,x_4\}$. Hence we must have $c(\{x_1\},\{x_2,x_3,x_4\})=3$. The only edge in the semigraph crossing this cut is $a_{1,2}$. Hence $b_{1,2}=3$. Similarly we find that $b_{2,3}=4$ and $b_{3,4}=5$. Considering the entry $r_{3,2}=6$, we find that $c(\{x_1,x_3\},\{x_2,x_4\})=6$. But $b_{1,2}=3$ and $b_{3,4}=5$ and so $b_{3,2}=-2$. Similarly $c_{4,3}=3$. The element $r_{4,2}=8$ has two min-cut matrices defined by $(\{x_1,x_3,x_4\},\{x_2\})$ and $(\{x_1,x_4\},\{x_2,x_3\})$. Using the first matrix gives $b_{4,2}=7$ and the second gives $b_{4,2}=2$. Choosing $b_{4,2}=7$ will lead to a realisation of R but $b_{4,2}=2$ will not. In general we should select the min-cut matrix which gives the largest value of branch capacity. Continuing similarly we obtain the semigraph of fig 2.6(a) which can be shifted to the graph of fig 2.6(b).

The formal statement of the algorithm to realise a maximally

distinct matrix R is as follows.

(1) Find a semiprincipal partitioning of R and form R^0.

(2) Relabel the elements of R with a single subscript such that a<b implies that $r_a<r_b$. The subscripts range from 1 to $(n-1)(n-2)/2$. The mapping of $r_{a,b}$ onto r_c will be notated as $r_{a,b}\rightarrow r_c$ or $r_c\rightarrow r_{a,b}$.

(3) For $d=1,2,\ldots,(n-1)(n+2)/2$ form R^d from R^{d-1} by letting
$r_d^d=2r_d^{d-1}-\min_i \|\mu_i^{d-1}\|$. ($\|\mu_i^{d-1}\|$ = Σ elements in μ_i).
The minimisation is over the submatrices defined by X_i such that R_i^0 is a min-cut matrix of r_d.

(4) $R^{(n-1)(n+2)/2}$ is the branch capacity matrix.

The algorithm works because at each stage there is only one unknown capacity in each cut. This will always be the case if we consider the cuts in increasing order. Each min-cut matrix will only contain entries smaller than the one being considered and the branch capacities in the corresponding cut will all have been found except one.

It is known [FR2] that a maximally distinct matrix can be realised as a graph if and only if every element has at least one min-cut matrix and the Substitution Algorithm yields a semigraph for which

$$b_{i,j}+b_{j,i}\geq 0 \quad \forall\ (i,j)$$

and $b_{k,k+1}=r_{k,k+1}$ for $k=1,\ldots,(n-1)$

For non-maximally distinct matrices there may be elements which do not have distinct min-cut matrices and so it will be impossible to sequentially order the calculations so that there is

only one unknown element at each cut. More sophisticated algorithms are necessary in these situations.

The Perturbation Algorithm [FR2]

To realise a non-maximally distinct matrix a first approach is to convert the matrix to one that is maximally distinct and then try to realise it using the Substitution Algorithm. The matrix is converted to one that is maximally distinct by perturbing the equal elements in the matrix by small amounts, then the Substitution Algorithm is applied and then the perturbations are set to zero.

Formally the algorithm may be stated as follows.

(1) From R^0 form $R^0(\underline{e})$ where $\underline{e}$ is a vector which perturbs the equal elements of R^0. The set of l_1 identical elements are perturbed by quantities $0, e_1, 2e_1, \ldots, l_1 e_1$. Another set of l_2 equal elements are perturbed by $0, e_2, 2e_2, \ldots, l_2 e_2$ etc.

(2) Apply the Substitution Algorithm.

(3) Let $\underline{e}=\underline{0}$.

With the Perturbation Algorithm, we can realise some matrices which cannot be realised with the Substitution Algorithm but there are some realisable matrices which cannot be perturbed so that each element has a distinct min-cut matrix ie some elements are completely coupled so the Substitution Algorithm can not be applied. Also the algorithm may produce are semigraph for which $b_{i,j}+b_{j,i}<0$. The next most general algorithm which can be applied in such a situation is the Replacement Algorithm.

The Replacement Algorithm [FR2]

The Replacement Algorithm is a variation on the Substitution Algorithm in which there are cases in which a branch capacity may be larger than that which would be given by the Substitution Algorithm.

The realisation method handles the problem of an element not having a distinct min-cut matrix by insisting, in certain cases, that the element should be coupled to another element of the same value so that the two requirements can be realised by the same cut, ie the two elements should be in the same semidistinct cut matrix.

The algorithm is

(1) Given R find any R^0.

(2) Relabel the entries in R^0 as before.

(3) For $d=1,2,\ldots,(n-1)(n+2)/2$ find R^d from R^{d-1} by letting

$$r_d = \begin{cases} \text{Max}(-r_g^g, 2r_d^{d-1}) - \text{Min}_i \left\| S_i^{d-1}(d) \right\| & \text{if } r_d \to r_{j+1,j},\ r_g \to r_{j,j+1} \\ \text{Max}(0, 2r_d^{d-1}) - \text{Min}_i \left\| S_i^{d-1}(d) \right\| & \text{otherwise} \end{cases}$$

(4) $R^{(n-1)(n+2)/2}$ is the branch capacity matrix.

The modification to the first part of the equation in (3) is to constrain the branch capacities such that $b_{j+1,j}+b_{j,j+1} \geq 0$ for $j=1,2,\ldots(n-1)$ so that the resulting semigraph may be shifted to give a graph (Theorem 2.5).

For the second equation there are two possibilities.

If $r_d^d = 2r_d^{d-1} - \text{Min}_i \left\| S_i^{d-1}(d) \right\|$

then the value of r_d^d is the same as would be calculated in the

Substitution Algorithm where the branch capacity is being calculated so as to set the capacity of the cut $(X_i, \bar{X}_i)$ to that required by the matrix R.

If $r_d^d = \min_i \|S_i^{d-1}(d)\|$

then the capacity of the cut $(X_i, \bar{X}_i)$ is too large and the terminal capacity matrix can only be realised if r_d is coupled to another r_q such that

$$r_q^q = 2r_q^{q-1} - \min_j \|S_j^{q-1}(q)\|$$

so that the cut $(X_j, \bar{X}_j)$ realises the entry r_d in the requirement matrix.

There are still some matrices which cannot be realised by this scheme. The Terminal Capacity Realisation Algorithm is an extension of the Replacement Algorithm, and in terms of this algorithm the necessary and sufficient conditions for realisability can be given.

The Terminal Capacity Realisation Algorithm [FR2]

In this algorithm all steps except the third are as before but the calculation of the elements of the branch capacity matrix is

(3)

$$r_d = \begin{cases} \max(-r_g^g, 2r_d^{d-1}) - \min_i \|S_i^{d-1}(d)\| + a_d & \text{if } r_d \to r_{j+1,j},\ r_g \to r_{j,j+1} \\ \max(0, 2r_d^{d-1}) - \min_i \|S_i^{d-1}(d)\| + a_d & \text{otherwise} \end{cases}$$

$a_d \geq 0$

The difference between this algorithm and the previous is the presence of the non-negative constants a_d. The a_d's are chosen so that the following necessary conditions are satisfied.

(1) $b_{j+1,j}+b_{j,j+1} \geq 0$ for $j=1,2,..,(n-1)$

ie the semigraph can be shifted to obtain a graph.

(2)

$$r_d \leq \min_i \sum_{r_{a,b} \in \mu_i(d)} b_{a,b} \quad \forall d$$

ie the sum of the capacities of the arcs across the cuts defining the min-cut matrices of the network are greater than or equal to the requirement.

These constraints give a set of inequalities in terms of the a_d's and if these inequalities can be solved to give a set of non-negative a_d's then the matrix is realisable; otherwise it is not. The resulting graph is not unique since the set of a_d's is not unique and neither is the ordering of the elements of the requirement matrix by the mapping $r_{i,j} \rightarrow r_d$.

Generation of All Solutions [AG1]

Agrawal and Arora [AG1] have given a method for systematically enumerating all feasible realisations of a requirement matrix. Consider an element $r_{i,j}$ of a requirement matrix; this element will have a number of min-cut matrices, say n. Consider the set the set of n cuts $(X_k, \bar{X}_k)$, $k=1,2,\ldots,n$ corresponding to these n min-cut matrices. The arc capacities of the graph should be chosen such that

$$c(X_k, \bar{X}_k) \geq r_{i,j} \quad k=1,2,\ldots,n$$

with the constraint being an equality for at least one cut. Adding slack variables S_k this can be written as

$$c(X_k, \bar{X}_k)+S_k=r_{i,j} \qquad k=1,2,\ldots,n$$
$$\prod S_k=0$$

There is a similar set of constraints for each element $r_{i,j}$. The mathematical programming method to calculate the branch capacities is a variation of the Simplex Algorithm in which at least one of the slack variables is constrained to be non-basic ie equal to zero. The authors report the algorithm to be efficient for small problems.

2.3.5 The Minimum Excess Terminal Capacity Algorithm [GO1]

Given an arbitrary symmetric requirement matrix R, it may be desired to generate from it a realisable matrix R′ such that the elements of R′ are increased by the minimum value required to make the matrix realisable. The algorithm is due to Gomory and Hu [GO1] and is as follows.

(1) Consider the matrix R as the distance matrix of a graph.

(2) Find the longest spanning tree of the graph.

(3) Considering the branch weights now as capacities, R′ is the terminal capacity of this tree.

2.4 Minimum Cost Network Synthesis

We have seen that in general it is possible to synthesise a number of networks, each of which satisfies a requirement matrix. If a cost function were to be applied to the branches of the network from which a cost for the whole network could be calculated, then one of the realisations could be selected as the optimum. If the costs per unit capacity of each arc differ, then a linear programming formulation is the only possible approach [WI1]. The

linear program contains a very large number of constraints but Gomory and Hu [GO2] have given a method for simplifying the linear program to give a tractable problem.

If the unit costs are identical, and so it is the total capacity of the graph that is to be minimised, then a simple solution is available for symmetric matrices. Several different algorithms have been proposed by Wing and Chein [WI1], Chein [CH2] and Gomory and Hu [GO1]. The methods of Wing & Chien and Gomory & Hu will be described.

The algorithm of Wing and Chien is as follows. Let an 'elementary terminal capacity matrix' be defined as a matrix which can be put in the following form

$$T=\begin{bmatrix} - & t_1 & t_2 & t_3 & \cdots & t_{n-1} \\ t_1 & - & t_2 & t_3 & \cdots & t_{n-1} \\ t_2 & t_2 & - & t_3 & \cdots & t_{n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ t_{n-1} & t_{n-1} & t_{n-1} & t_{n-1} & \cdots & t_{n-1} \end{bmatrix}$$

where $t_1 \geq t_2 \geq t_3 \geq \ldots \geq t_{n-1}$. A minimum capacity realisation of this T-matrix is shown in fig 2.7(a). Furthermore, if a T-matrix is partitionable as follows

$$T=\left[\begin{array}{c|c} T_1 & T_0 \\ \hline T_0 & T_2 \end{array}\right]$$

where T_1, T_2 are elementary terminal capacity matrices and all elements of the matrices T_0 are equal to t_0 then the complete T-matrix can be realised by linking two networks realising T_1 and T_2 as shown in fig 2.7(b). The two linking branches can be placed

between any two pairs of nodes. The generalisation of this method to the case in which the matrix is partitionable into T_1, T_2,...T_p is obvious and every realisable T-matrix is so partitionable [MA1]. Other minimum capacity structures are discussed in [WI1].

The algorithm of Gomory and Hu begins with a linear tree (ie a chain) and transforms it into a graph with the same terminal capacity matrix but with minimum total capacity by (1) splitting the tree into a number of sections called uniform trees, (2) operating on each section with a process called circuit formation and then (3) recombining the resulting graphs.

A uniform tree is a tree in which all branches have the same capacity. The linear tree is split into a number of uniform trees as follows.

Let G^0, G^1, G^2 be three trees with branch capacity matrices B^0, B^1, B^2.

(1) Let the capacity of the minimum capacity branch in the linear tree G^0 be b_0.

(2) Form two linear trees G^1, G^2 where G^1 is a uniform tree with $b^1_{i,j} = b_0$ for all $a_{i,j}$ in G^0, and G^2 is a linear tree such that $b^2_{i,j} = b^0_{i,j} - b_0$ for all $a_{i,j}$ in G^0.

(3) Repeat this procedure on the resulting non-uniform trees until the graph is decomposed into a set of uniform trees.

Circuit formation is then applied to each uniform tree generated by the decomposition. Given a uniform tree capacity b_0 the operation of circuit formation is

(1) Reduce the capacity of each branch to $b_0/2$.

(2) Add a branch capacity of $b_0/2$ between the two end nodes of the tree.

The minimum capacity graph is the union of the circuits so formed. Figs 2.8(a)-(c) show an example. Gomory and Hu [GO1] prove that the terminal capacity matrix is invariant under this procedure and that the graph has minimum capacity.

2.5 Expansion of Networks

Only one problem in the field of expansion of networks has so far been satisfactorily covered. Suppose that there exists a network with two nodes identified as source and sink, the costs of increasing the capacity of each arc are known and there is a budgetary constraint. How should the money be spent on increasing the capacities of the various arcs so as to maximise the source to sink flow of the network? This problem has been examined for the case in which the cost of increasing the arc capacities is a linear function of the capacity by Fulkerson [FU1] and Hu [HU2] and for the discrete case in which the arc costs are non-linear functions of the arc capacity by Christofides and Brooker [CH3].

The algorithm of Fulkerson is a parametric variant of the labelling algorithm used to solve maximum flow problems. The algorithm of Hu is based on the concept of modified costs and will be described.

The algorithm can be used to solve both the problem of obtaining maximum flow v at fixed cost C at the problem of minimising the cost to obtain a flow v. These problems are

min $\sum c_{i,j} b'_{i,j}$

such that

$$\sum_i f_{i,j} - \sum_k f_{j,k} = \begin{cases} -v & j=s \\ 0 & j \neq s,t \\ v & j=t \end{cases}$$

$$0 \leq f_{i,j} \leq b'_{i,j} + b_{i,j}$$

max v

such that

$$\sum_i f_{i,j} - \sum_k f_{j,k} = \begin{cases} -v & j=s \\ 0 & j \neq s,t \\ v & j=t \end{cases}$$

$$0 \leq f_{i,j} \leq b'_{i,j} + b_{i,j}$$

$$\sum c_{i,j} b'_{i,j} = C$$

where $b'_{i,j}$ is the increment in the branch capacity.

The algorithm is

(1) Begin with the flow pattern F=0.

(2) Define modified costs $c'_{i,j}$ based on the current arc flows

$c'_{i,j}=0$ if $f_{i,j} < b_{i,j}$

$c'_{i,j}=c_{i,j}$ if $f_{i,j} \geq b_{i,j}$

$c'_{i,j}=-c_{i,j}$ if $f_{j,i} > b_{i,j} > 0$

(3) Ship the flow along the minimal cost path based on $c'_{i,j}$. The amount of flow is limited so that the $c'_{i,j}$ remain as defined in (2).

(4) If the total flow is v or the total cost is C, stop. Otherwise go to (2).

Consider the example in fig 2.9. Fig 2.9(a) shows the original arc capacities and fig 2.9(b) the unit costs. We shall increase the flow from 1 to 4 at minimum cost. The modified costs are zero until the flow reaches the maximum for the unmodified graph of 6. The arc flow with the modified costs in brackets for this value of flow are shown in fig 2.9(c). The minimum cost path is the arcs $a_{1,3}$, $a_{3,2}$, $a_{2,4}$. Thus the capacity of the arc $a_{3,2}$ should be increased. Note that $c'_{2,3}=-2$ since $f_{3,2} > b_{3,2}$. The flow can be

increased until v=10 without changing the modified costs (fig 2.9(d)) and so the solution up to this value of flow is to continue to increase $b_{3,2}$. Beyond this the least cost path is $a_{1,2}$, $a_{2,3}$, $a_{3,4}$ so to increase the flow beyond v=10, $b_{1,2}$, $b_{3,4}$ should be increased but $b_{3,2}$ reduced. At v=14, $b_{2,3}$ has returned to its original value (fig 2.9(e)).

Christofides and Brooker [CH3] have given an algorithm for expansion of networks in which new arcs may be added at discrete levels of capacity and the cost of adding an arc is not necessarily linearly related to the capacity. In contrast with the problem of the previous section this is a problem of combinatorial optimisation and the method proposed is a branch and bound tree search in which dynamic programming techniques are used to calculate the bounds. As candidate arcs are added in and excluded from the network, the algorithm notes the set of cuts in the graph which it is necessary to span to increase the flow of the graph and calculates an upper bound on the flow which can be obtained with the remaining budget. If the upper bound is less than the current best solution then backtracking can occur. The set of cuts is generated as the algorithm progresses and for computational reasons is restricted to cuts which are "disjoint" in the sense that the sets of arcs spanning the cuts should be disjoint. This bounding procedure greatly reduces the amount of the tree that need be explicitly searched to locate the optimum solution, and the authors report the technique to be very successful.

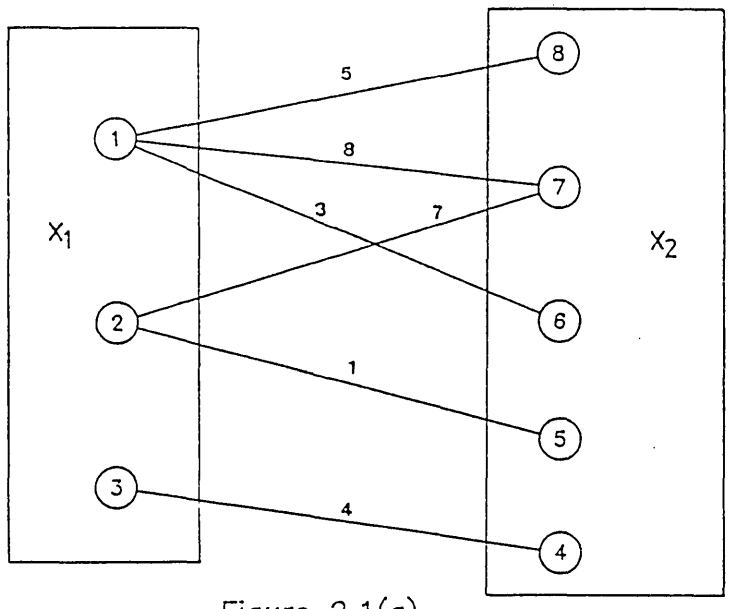

Figure 2.1(a)

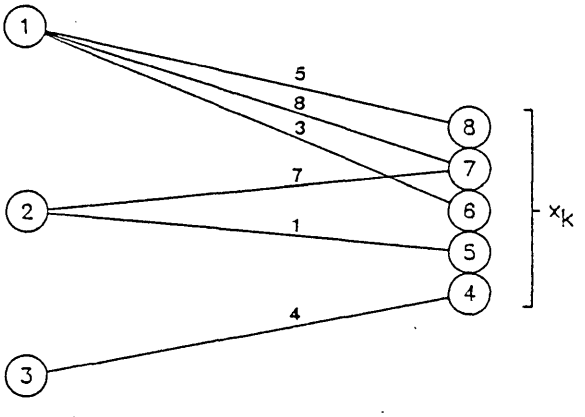

Figure 2.1(b)

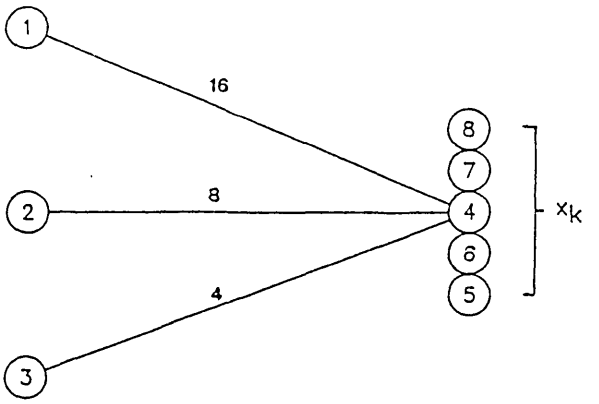

Figure 2.1(c)

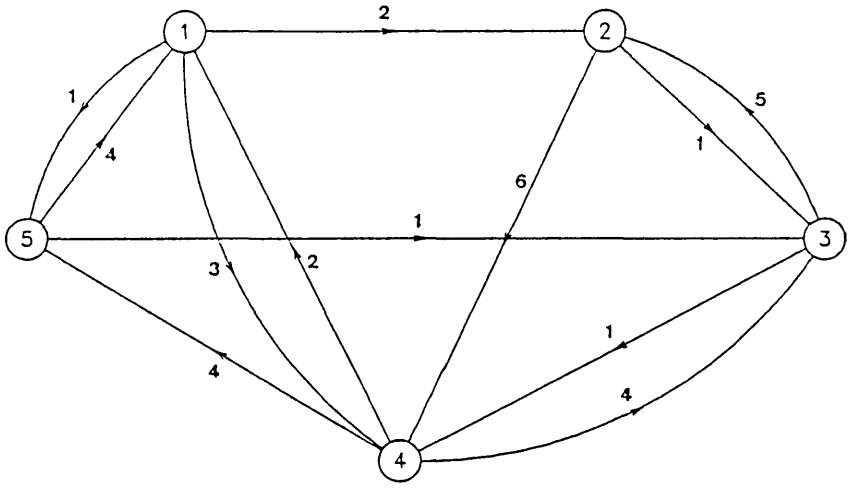

Figure 2.2(a)

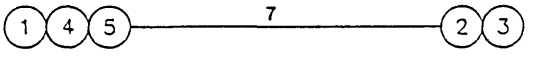

Figure 2.2(b)

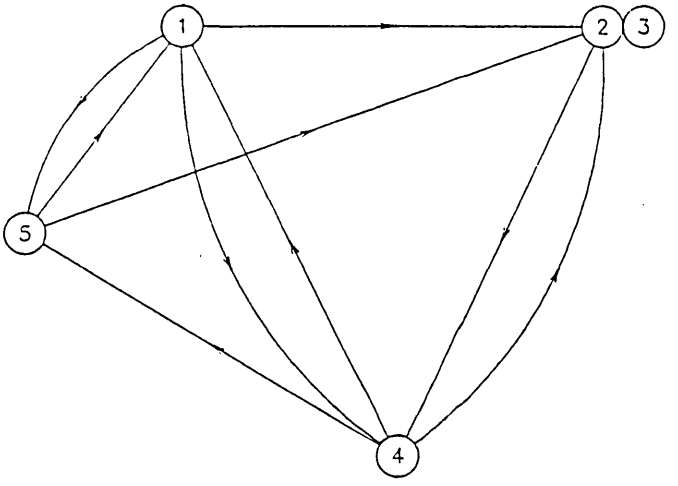

Figure 2.2(c)

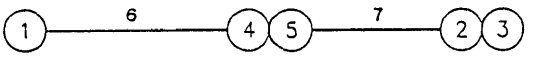

Figure 2.2(d)

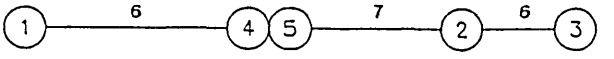

Figure 2.2(e)

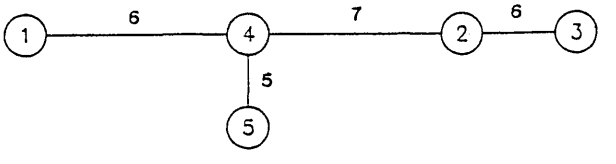

Figure 2.2(f)

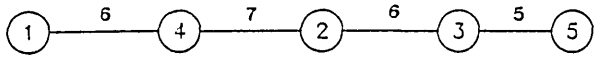

Figure 2.2(g)

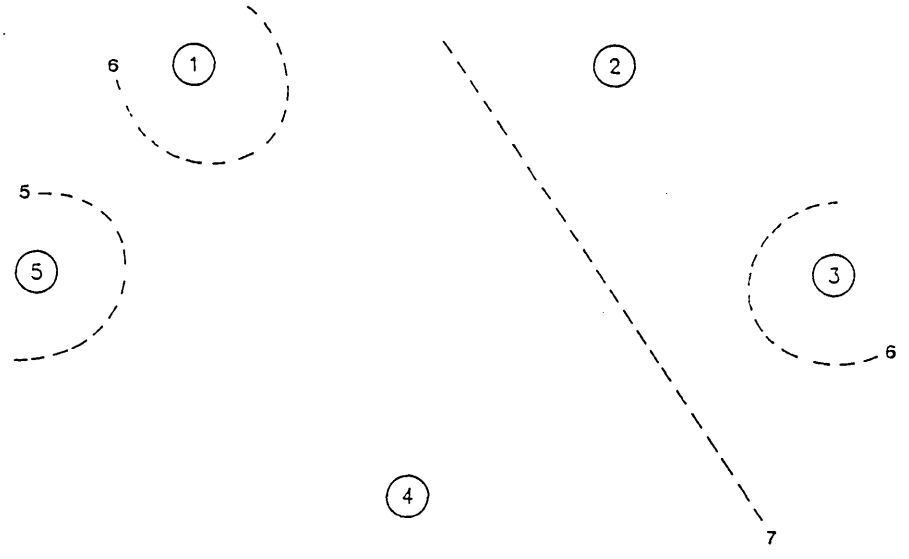

Figure 2.2(h)

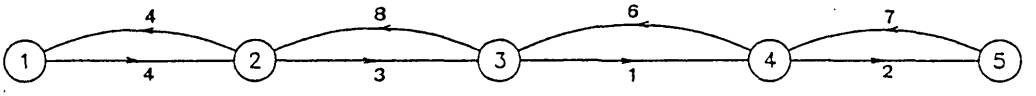

Figure 2.3

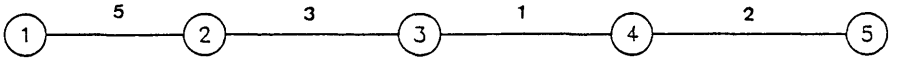

Figure 2.4

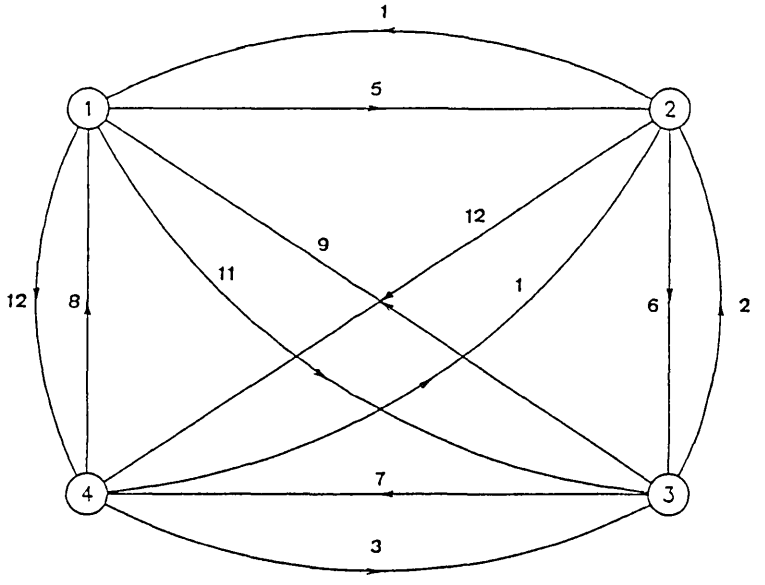

Figure 2.5(a)

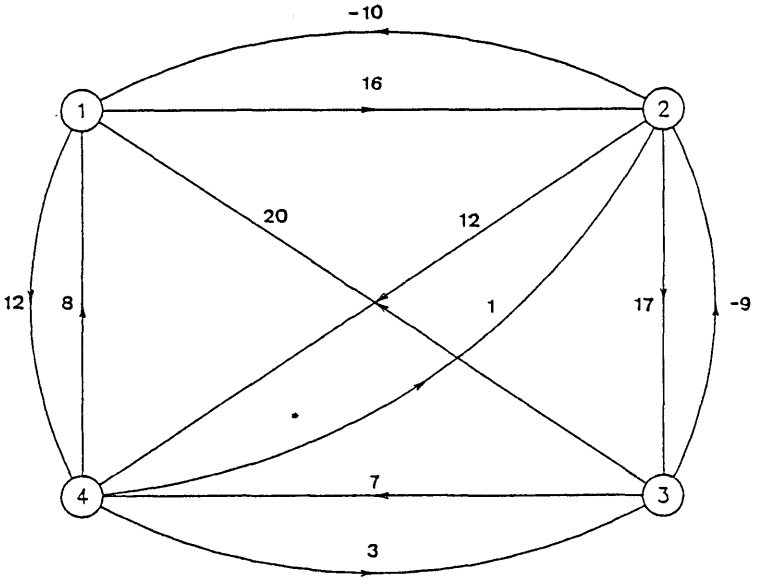

Figure 2.5(b)

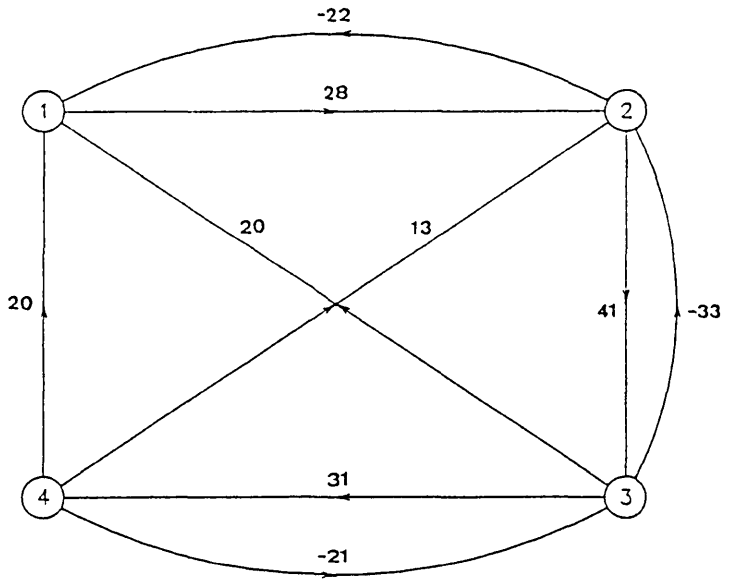

Figure 2.5(c)

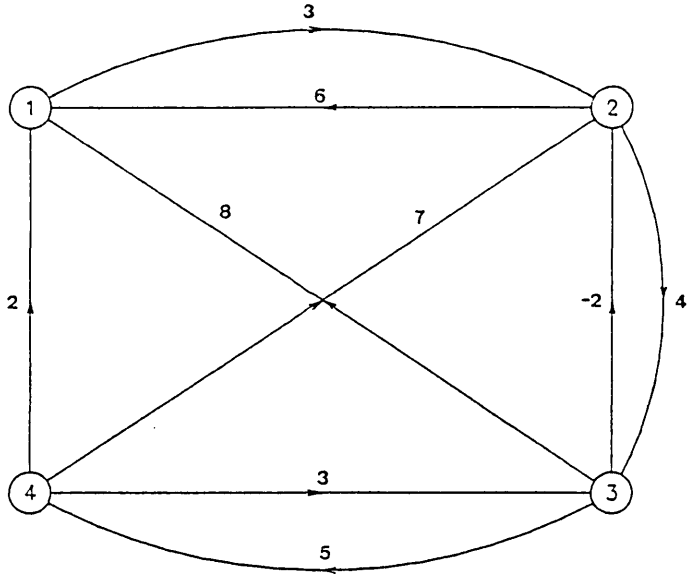

Figure 2.6(a)

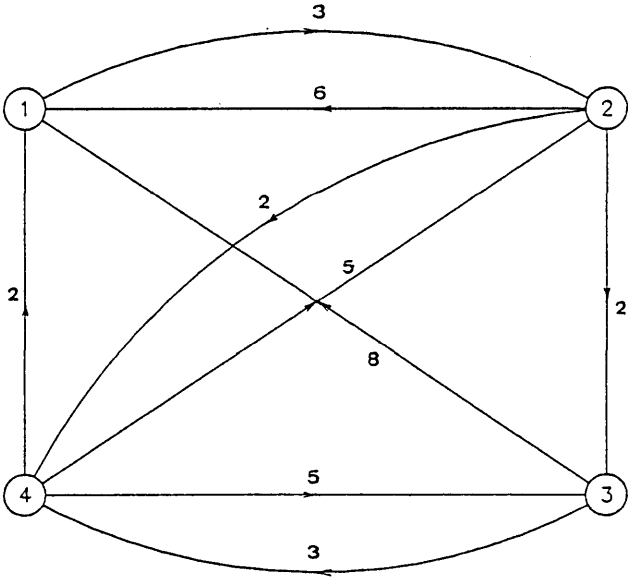

Figure 2.6(b)

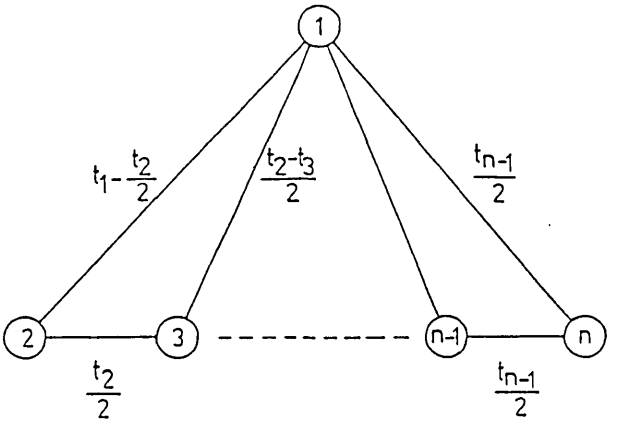

Figure 2.7(a)

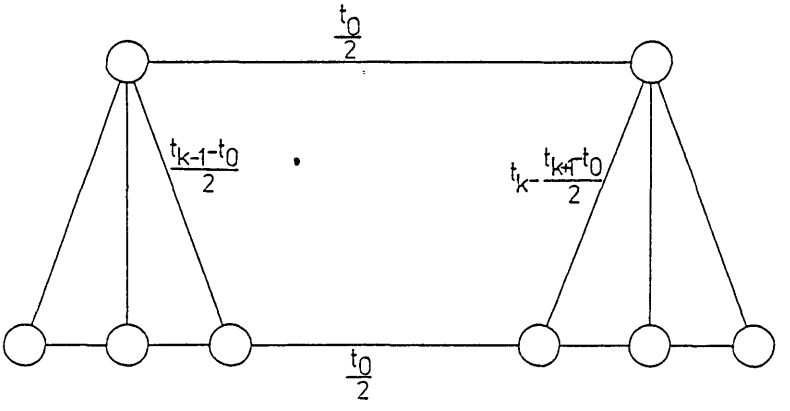

Figure 2.7(b)

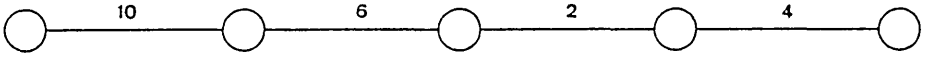

Figure 2.8(a)

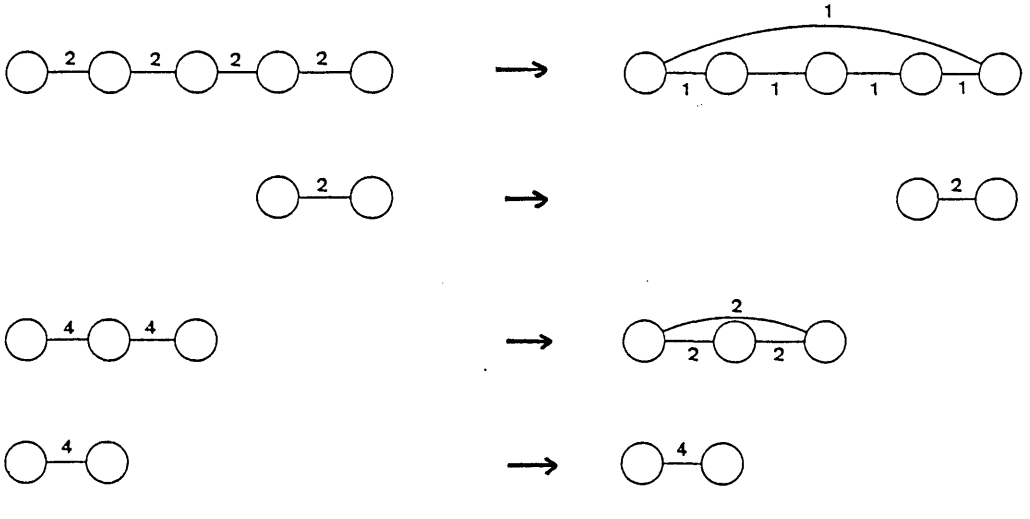

Figure 2.8(b)

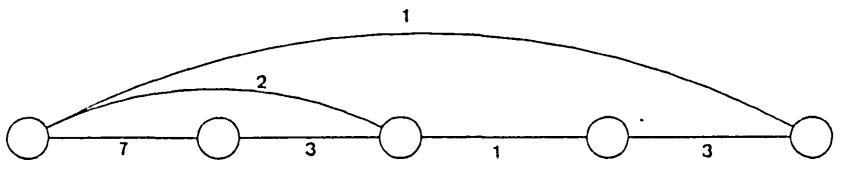

Figure 2.8(c)

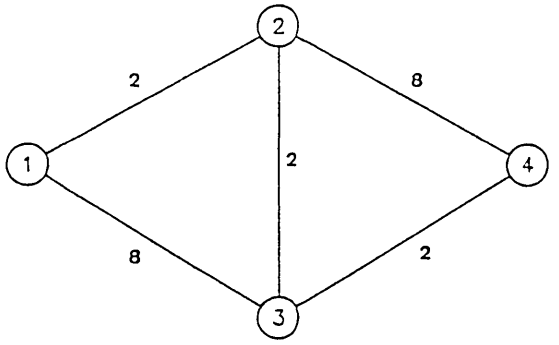

Figure 2.9(a)

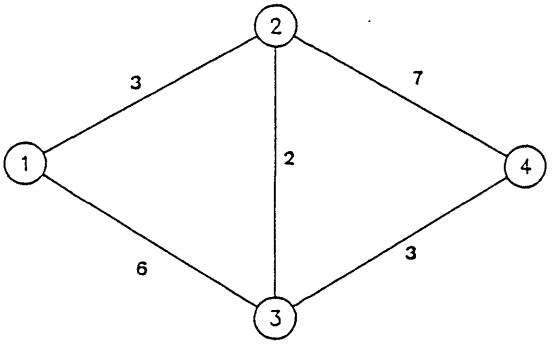

Figure 2.9(b)

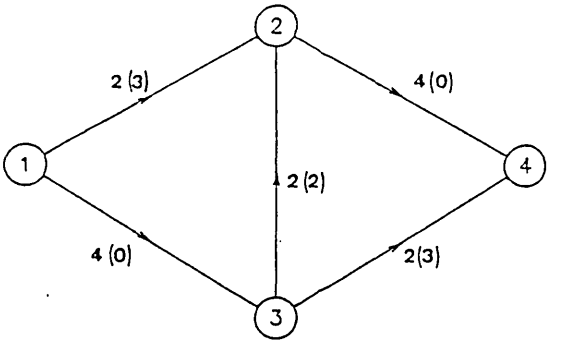

Figure 2.9(c)

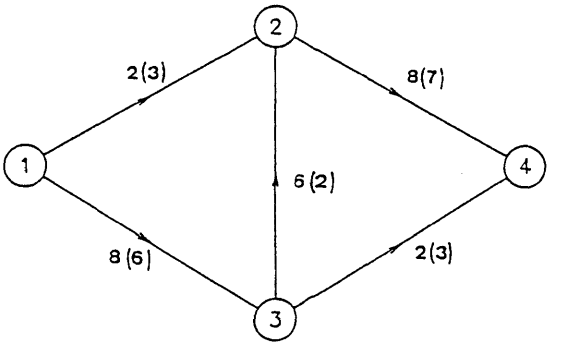

Figure 2.9(d)

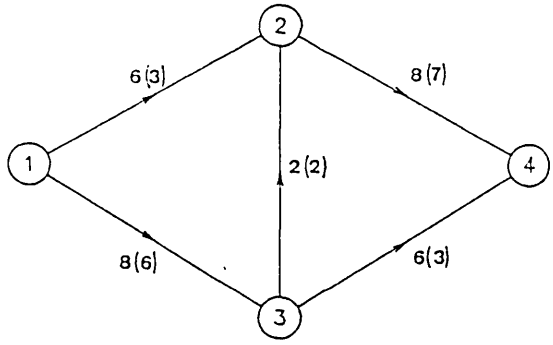

Figure 2.9(e)

OPTIMAL EXPANSION OF NETWORKS SUBJECT TO TERMINAL CAPACITY CONSTRAINTS

3.1 Introduction

In this chapter we shall pose a problem in the expansion of networks and give an algorithm for its solution. The problems examined in this chapter and the succeeding will be generalisations of those reviewed in Section 2.5 in the sense that we consider not the synthesis of networks which satisfy the flow requirements between one source and one sink but the multiterminal network expansion problem in which all nodes may source or sink flow. We restrict ourselves to the consideration of undirected graphs.

Suppose that there exists a network of flow such as a communication network and, as time passes, the communication requirements of the users increase beyond the capacity of the network. The problem that arises is how to add additional branches to the network, in some optimal fashion, such that the expanded network once again satisfies the requirements of the users. The simplest optimality criterion, which will be studied in this chapter, is that the added capacity should be minimum.

This problem may be stated in graph theoretic terms as follows. Let $G=(X,A)$ be an undirected graph with branch capacity matrix B and terminal capacity matrix T and let R be a realisable requirement terminal capacity matrix such that $r_{i,j} \geq t_{i,j}$ $\forall$ (i,j). How may a graph $G'=(X,A')$ with branch capacity matrix B', terminal capacity matrix T' be synthesised such that $t'_{i,j} \geq r_{i,j}$ $\forall$ (i,j), $b'_{i,j} \geq b_{i,j}$ $\forall$ (i,j) and $\sum b'_{i,j}$ is minimum?

3.2 Theoretical Preliminaries

Before proceeding any further there are some preliminary ideas which must be discussed. Suppose that we write a T-matrix as the sum of two or more matrices: the question now is; can we realise the individual matrices and add the corresponding branch capacity matrices to obtain a realisation of the original T-matrix? The difficulty with problems in the expansion of networks arises from the fact that, in general, the answer to this question is no. However, there are certain special cases in which T-matrices can be added as is shown by the following theorem due to Tang and Chein [TA1].

Theorem 3.1 [TA1]

The realisation G of a terminal capacity matrix T is obtainable by superimposing two graphs G^1 and G^2 (by the addition of their branch capacity matrices B^1, B^2) whose terminal capacity matrices are T^1 and T^2 such that $T=T^1+T^2$, if and only if for each node pair (i,j) there exists a cut which is minimum throughout G, G^1 and G^2.

Proof

If, for each node pair (i,j), there exists a cut which is minimum in G, G^1 and G^2, then the terminal capacities in G, G^1, G^2 are determined by this cut. However, since any branch capacity in G is just the sum of the corresponding branch capacities in G^1 and G^2, we thus have $t_{i,j}=t^1_{i,j}+t^2_{i,j}$. On the other hand, if $t_{i,j}=t^1_{i,j}+t^2_{i,j}$ and if there exists no cut which is minimum in G, G^1 and G^2 then any cut in G giving $t_{i,j}$ cannot be a minimum cut for both G^1 and G^2.

That is, the cut will assume values $c^1_{i,j}$ in G^1 and $c^2_{i,j}$ in G^2, where either $c^1_{i,j} > t^1_{i,j}$ or $c^2_{i,j} > t^2_{i,j}$ or both. Therefore

$$t_{i,j} = c^1_{i,j} + c^2_{i,j} > t^1_{i,j} + t^2_{i,j}$$

which is a contradiction.

This result shows that when making additions to a network it is necessary to consider not just the properties of the added branches in isolation but also the way in which they interact with the existing network.

It is also necessary to consider further the idea of the flow equivalent tree discussed in Section 2.2.2. It was pointed out that in general the flow equivalent tree of a graph is not unique. We shall now restrict the definition of the flow equivalent tree a little further so that the tree is unique by identifying each edge in the tree with one of the set of non-crossing cuts generated by the Gomory-Hu algorithm for multi-terminal network analysis. The following theorem due to Hu [HU3] shows that this is possible. The theorem is more general than we require in that it shows a spanning tree can be identified with any set of non-crossing cuts but we require the case where the cuts are those generated by the Gomory-Hu algorithm.

Theorem 3.2 [HU3]

A spanning tree with (n-1) nodes corresponds to a set of (n-1) non-crossing cuts uniquely.

Proof

The proof is by construction. Remove any link in the spanning tree; this will disconnect the tree into two components say T_1 and T_2.

Then let this link correspond to the cut (T_1, T_2). Continue to repeat this process for each sub-tree generated until all links have been removed. Thus from any spanning tree we get a set of $(n-1)$ non-crossing cuts. Conversely, from a set of $(n-1)$ non-crossing cuts we can construct a spanning tree as follows. Take a cut $(X, \bar{X})$; we can draw two supernodes connected by a link where each supernode symbolically represents a set of ordinary nodes. In one supernode we list the nodes in X and in the other we list the nodes in $\bar{X}$. This creates one link in the spanning tree. Now consider another cut $(Y, \bar{Y})$. Since $(Y, \bar{Y})$ does not cross $(X, \bar{X})$ we have $Y \subset X$ and $\bar{Y} \supset \bar{X}$ (or $Y \supset X$ and $\bar{Y} \subset \bar{X}$); then we can create a tree with three supernodes Y, $X-Y$, $\bar{X}$ (or X, $\bar{X}-\bar{Y}$, $\bar{Y}$). Continuing to repeat this for $(n-1)$ steps creates a spanning tree with $(n-1)$ links.

Henceforward, it is this restricted idea of a flow equivalent tree which will be used and we shall use the terminology of Hu [HU3] and refer to it as a cut-tree. We shall make use of some further terminology introduced in this paper. Outer nodes and inner nodes are nodes of a tree which have degree one and greater than one respectively and a star-tree is a tree with only one inner node.

3.3 The Algorithm

In this section we give an algorithm to solve the problem presented in the Introduction and prove that it converges correctly to the optimum solution. The algorithm is analysed in terms of its computational complexity and the analysis compared with the results of computational experience. It is illustrated with examples.

Broadly speaking, the algorithm operates as follows. Firstly,

the existing network is analysed. As mentioned above, when adding to a network, it is necessary to know the characteristics of the initial network so as to be able to predict the effect of the expansion on the terminal capacity matrix. Next, additions are made to the network so as to generate a new network which satisfies the requirement terminal capacity matrix constraints. This network is suboptimal because effort is directed towards satisfying the constraints rather than producing a minimum cost network. Finally, a series of transformations are applied to the network which progressively reduce the capacity of the network while still meeting the terminal capacity constraints.

These steps will now be described in detail.

3.3.1 Initial Network Analysis

The analysis performed on the initial network is to apply the Gomory-Hu algorithm so as to locate the set of non-crossing minimum cuts in the network and generate the cut-tree.

There is an alternative way of viewing the analysis which will be useful later. It will be explained by way of an example. Consider the network of fig 3.1(a), the set of cuts from this network in fig 3.1(b) and the cut-tree fig 3.1(c). The cuts partition the graph into subgraphs (which correspond to the supernodes of theorem 2.2) although some of the sub-graphs are degenerate cases consisting of a single node. This partitioning will be viewed in a hierarchical manner. If some of the nodes are gathered together into clusters of nodes or supernodes then the graph can be seen as a star-tree with some of the outer nodes being

supernodes. In the example if $\{x_7, x_8\}$ and $\{x_4, x_6\}$ are taken together as supernodes then a star-tree is formed of $\{x_1\}$, $\{x_2\}$, $\{x_5\}$, $\{x_7, x_8\}$ and $\{x_4, x_6\}$ as outer nodes and x_3 as the inner node (fig 3.1(d)). Each supernode may again be seen as a star-tree with some of the outer nodes being supernodes. In this example the two supernodes are star-trees of two nodes; x_4 and x_7 are the inner nodes and x_6 and x_8 are the outer nodes. There are no further supernodes consisting of star-trees of supernodes.

This hierarchical clustering of nodes into groups of supernodes will be used in the description of the method for progressively reducing the added capacity of the graph to a minimum.

3.3.2 Generation of Suboptimal Solution

The next stage of the algorithm is to add to the existing network so as to synthesise a suboptimal network which meets the requirement matrix constraints. We shall separate this problem into two questions:

(1) What is the structure of the added network?

(2) What are the capacities of the added branches to be?

The structure chosen for the added graph is that it be the same as the structure of the cut-tree of the original network. This structure is chosen because the effect upon the T-matrix of the original graph caused by adding this network is easily determined as will shortly be demonstrated. We shall then propose a method of calculating the branch capacities of the added network, illustrate it with an example and finally give a proof that the method achieves what we require of it.

The following theorem relates the cut-tree of the old network, the cut-tree of the new network and the capacities of the added branches.

Theorem 3.3

If B^1 is the B-matrix of the cut-tree of the original network

B^2 is the B-matrix of the cut-tree of the expanded network

B^3 is the B-matrix of the added network

then if $b^3_{i,j}>0$ for $(i,j) \in A_t$

and $b^3_{i,j}=0$ for $(i,j) \notin A_t$

then $b^2_{i,j}=b^1_{i,j}+b^3_{i,j}$ $\forall$ (i,j)

where A_t is the set of edges in the cut tree.

Proof

First consider a pair of nodes which are adjacent in the cut-tree. All i-j cuts are spanned by the edge $a_{i,j}$ and some are spanned by other edges also so the minimum increment to an i,j cut is $b^3_{i,j}$. The expansion network is so constructed that the minimum i-j cut in the original network is spanned only by $a_{i,j}$ and so

$$t'_{i,j}=b^1_{i,j}+b^3_{i,j} \quad (i,j) \text{ adjacent in cut-tree}$$

where $t'_{i,j}$ is the value of the minimum i-j cut in the expanded network.

Now consider any node pair (i,j) and the path $P_{i,j}$ in the cut-tree which joins them. Let $a_{k,l}$ be an edge in this path which will represent a particular i-j cut say $(X_i,\bar{X}_i)$. From the above argument the new capacity of this cut $c'(X_i,\bar{X}_i)$ is

$$c'(X_i,\bar{X}_i)=c(X_i,\bar{X}_i)+b^3_{k,l}=b^1_{k,l}+b^3_{k,l}$$

This is not necessarily the minimum i-j cut. Therefore

$$t'_{i,j} \leq b^1_{k,l} + b^3_{k,l}$$

where $t'_{i,j}$ is the capacity of the minimum i-j cut of the expanded network. This argument holds for all $a_{k,l} \in P_{i,j}$. Therefore

$$t'_{i,j} \leq \min(b^1_{k,l} + b^3_{k,l}) \quad a_{k,l} \in P_{i,j}$$

Now suppose there is an i-j cut $(X_j, \bar{X}_j)$ strictly less than any of these cuts

$$c'(X_j, \bar{X}_j) < \min(b^1_{k,l} + b^3_{k,l})$$

This cut must pass through one of the edges in the path $P_{i,j}$, say $a_{m,n}$ and is thus an m-n cut. Since it is not necessarily the minimum m-n cut

$$c'(X_j, \bar{X}_j) \geq b^1_{m,n} + b^3_{m,n} \quad a_{m,n} \in P_{i,j}$$

Thus we have found an i-j cut which does not satisfy the condition of the above argument and hence the assumption of the existence of a cut strictly less than the cuts represented by the edges in the path in the cut tree joining i and j is false. Hence,

$$t'_{i,j} = \min(b^1_{k,l} + b^3_{k,l}) \quad a_{k,l} \in P_{i,j}$$

which is the property required of a cut-tree. Therefore $B^2 = B^1 + B^3$ is the branch capacity matrix of the cut-tree of the new network and from B^2 it is easy calculate the T-matrix of the expanded network.

The remaining problem is that of choosing the capacities of the added branches in such a way that the constraints imposed by the requirement matrix are met. Clearly, it must be that the sum of the capacities of the branches incident upon a node is greater than or equal to the largest entry in the row or column of the requirement matrix corresponding to that node

ie $\sum_i b'_{n,i} \geq \max_j (r_{n,j})$ for all nodes n

where B′ is the branch capacity matrix of the expanded network

R is the requirement terminal capacity matrix.

Let the added branch capacities be chosen such that for the outer nodes of the cut-tree of the network this inequality becomes an equality

ie $\sum_i b'_{n,j} = \max_j(r_{n,j})$ for n an outer node of the cut-tree.

Clearly, adding any further capacity to these branches is superfluous provided that all other constraints are met.

It remains to choose the capacities of the branches joining a pair of inner nodes (j,k). Let the edge of the cut tree in question correspond to the cut $(X_i, \bar{X}_i)$. Form the cut matrix R_i from the requirement matrix R. The maximum entry in R_i indicates the maximum quantity of flow which must cross the cut $(X_i, \bar{X}_i)$ and so the added capacity should be chosen such that the capacity of the cut is increased to this amount.

ie $c'(X_i, \bar{X}_i) = \max(R_i)$

$= c(X_i, \bar{X}_i) + b_{j,k}$

where $c(X_i, \bar{X}_i)$= capacity of cut in original graph

$c'(X_i, \bar{X}_i)$= capacity of cut in expanded graph

$b_{j,k}$= capacity of branch added across $(X_i, \bar{X}_i)$ between nodes j and k.

Note that the method used to calculate the capacities of branches ending on an outer node is a degenerate form of this procedure in which the matrix R_i is the single row corresponding to the node.

This procedure will now be illustrated with an example which will bring out some further salient points.

Consider again the network of fig 3.1(a) which has B and T matrices

$$B=\begin{bmatrix} - & 5 & 11 & 1 & 0 & 0 & 3 & 4 \\ 5 & - & 14 & 0 & 0 & 0 & 5 & 0 \\ 11 & 14 & - & 5 & 5 & 0 & 3 & 0 \\ 1 & 0 & 5 & - & 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 & - & 0 & 3 & 0 \\ 0 & 0 & 0 & 5 & 0 & - & 0 & 0 \\ 3 & 5 & 3 & 0 & 3 & 0 & - & 9 \\ 4 & 0 & 0 & 0 & 0 & 0 & 9 & - \end{bmatrix} \qquad T=\begin{bmatrix} - & 24 & 24 & 6 & 8 & 5 & 18 & 13 \\ 24 & - & 24 & 6 & 8 & 5 & 18 & 13 \\ 24 & 24 & - & 6 & 8 & 5 & 18 & 13 \\ 6 & 6 & 6 & - & 6 & 5 & 6 & 6 \\ 8 & 8 & 8 & 6 & - & 5 & 8 & 8 \\ 5 & 5 & 5 & 5 & 5 & - & 5 & 5 \\ 18 & 18 & 18 & 6 & 8 & 5 & - & 13 \\ 13 & 13 & 13 & 6 & 8 & 5 & 13 & - \end{bmatrix}$$

and a requirement matrix

$$R=\begin{bmatrix} - & 26 & 40 & 40 & 22 & 6 & 22 & 20 \\ 26 & - & 26 & 26 & 22 & 6 & 22 & 20 \\ 40 & 26 & - & 42 & 22 & 6 & 22 & 20 \\ 40 & 26 & 42 & - & 22 & 6 & 22 & 20 \\ 22 & 22 & 22 & 22 & - & 6 & 25 & 20 \\ 6 & 6 & 6 & 6 & 6 & - & 6 & 6 \\ 22 & 22 & 22 & 22 & 25 & 6 & - & 20 \\ 20 & 20 & 20 & 20 & 20 & 6 & 20 & - \end{bmatrix}$$

The cut-tree of the network is shown in fig 3.1(c). The outer nodes are x_1, x_2, x_5, x_6 and x_8. The total capacity incident upon x_1 is 24; the maximum value of the first row of R is 40; therefore the capacity of the added branch $a_{1,3}$ must be $b_{1,3}=40-24=16$. Similarly $b_{2,3}=2$, $b_{3,5}=17$, $b_{7,8}=7$ and $b_{4,6}=1$. The remaining branches to be considered are $a_{3,4}$ and $a_{3,7}$. The cut-set for $a_{3,4}$ is $(\{x_4,x_6\},\{x_1,x_2,x_3,x_5,x_7,x_8\})$, the value of this cut in the original graph is 6 and the cut-matrix of R is

$$\begin{bmatrix} 40 & 26 & 42 & - & 22 & 6 & 22 & 20 \\ 22 & 22 & 22 & 22 & - & 6 & 25 & 20 \end{bmatrix}$$

whose largest entry is 42. Therefore, $b_{3,6}=42-6=36$. For $a_{3,7}$ the cut-set is $(\{x_7,x_8\},\{x_1,x_2,x_3,x_4,x_5,x_6\})$ whose value is 18. The cut-matrix is

$$\begin{bmatrix} 22 & 22 & 22 & 22 & 25 & 6 & - & 20 \\ 20 & 20 & 20 & 20 & 20 & 6 & 20 & - \end{bmatrix}$$

So $b_{3,7}=25-18=7$. The added network is shown in fig 3.1(e) and the new network in fig 3.1(f). Summing the capacities of the added network and the original cut-tree gives the cut-tree of the new network from which the T-matrix

of the expanded network can easily be calculated as

$$T' = \begin{bmatrix} - & 26 & 40 & 40 & 25 & 6 & 25 & 20 \\ 26 & - & 26 & 26 & 25 & 6 & 25 & 20 \\ 40 & 26 & - & 42 & 25 & 6 & 25 & 20 \\ 40 & 26 & 42 & - & 25 & 6 & 25 & 20 \\ 25 & 25 & 25 & 25 & - & 6 & 25 & 20 \\ 6 & 6 & 6 & 6 & 6 & - & 6 & 6 \\ 25 & 25 & 25 & 25 & 25 & 6 & - & 20 \\ 20 & 20 & 20 & 20 & 20 & 6 & 20 & - \end{bmatrix}$$

Notice that the entries $t'_{5,1}$, $t'_{5,2}$, $t'_{5,3}$, $t'_{5,4}$, $t'_{7,2}$, $t'_{7,3}$ and $t'_{7,4}$ are larger than the corresponding entries in R. Examining the cut-matrix of R corresponding to the cut $(\{x_1,x_2,x_3,x_4,x_6\},\{x_5,x_7,x_8\})$ indicates that we should have

$$c'(X,\bar{X})=22$$

but we have in our construction set this constraint to be the inequality

$$c'(X,\bar{X})=25>22$$

by spanning the cut with more capacity than is necessary to meet the requirements. Later, in the optimisation process, we shall permit the capacity of this and similar cuts to be reduced, if possible, to the amount specified by the requirement matrix. The reason for this discrepancy is considered in Section 3.3.5

To prove that a network constructed in the manner proposed will always be a satisfactory expansion network, we need to prove that every element in the T-matrix of the expanded network is greater than or equal to the corresponding element in the requirement matrix

$$\text{ie } t'_{i,j} \geq r_{i,j} \quad \forall\ (i,j)$$

Consider any element of T', say $t'_{i,j}$. The value of $t'_{i,j}$ is equal to the minimum capacity of the minimum capacity edge in the path from i to j in the cut-tree of the expanded network. Suppose this is the

edge (k,l) which corresponds to a cut $(X_m, \bar{X}_m)$

therefore $t'_{i,j}=t'_{k,l}$

Now consider the cut-matrix corresponding to the cut $(X_m, \bar{X}_m)$; the elements (i,j) and (k,l) are both in this matrix. But by construction we have

$$t'_{k,l}=\max(R_m)$$

therefore $t'_{i,j}=t'_{k,l} \geq$ any element in R_m

therefore $t'_{i,j} \geq r_{i,j}$ since $r_{i,j}$ is an element of R_m

3.3.3 Network Transformations which Preserve T-Matrix

Having generated a suboptimal network which satisfies the requirements, the final stage of the algorithm is to modify the network in such a way that the capacity of the network is reduced to a minimum without violating any of the constraints imposed by the requirement matrix.

The technique used to optimise the network is that of local search in which the solution neighbouring to the current solution is generated by an elementary transformation which exchanges one set of branches for another set with lower cost whilst still meeting the requirements. The idea of exchanging sets of branches is a powerful technique which has been successfully applied to a number of difficult graph-theoretic problems such as the Travelling Salesman Problem [LI1], the synthesis of minimum cost survivable networks [ST1] and the synthesis of small diameter networks [TS1]. Unlike these examples which are heuristic methods for locating good but sub-optimal solutions, the transformations discussed here can be so ordered that an optimal solution is obtained.

Consider fig 3.2(a) which shows three nodes x_1, x_2, x_3 and the added branches between them $a_{1,3}$ and $a_{2,3}$. Nodes x_1 and x_2 are of the type where the capacities of the added branches have been chosen so that

$$\sum_n b'_{1,n} = \max_i(r_{1,i})$$
$$\sum_n b'_{2,n} = \max_i(r_{2,i})$$

whereas this is not the case for node x_3. Therefore we may not reduce the total capacity of the branches incident upon x_1, x_2 but we may reduce the capacity incident upon x_3.

Consider now the network of fig 3.2(b). This has been derived from the network of fig 3.2(a) by reducing the capacities of $a_{1,3}$ $a_{2,3}$ by an amount b_0 and adding an arc $a_{1,2}$ of capacity b_0. This satisfies our requirements concerning the capacity of the branches incident on each node but has reduced the capacity of the network by b_0.

It now remains to investigate what values b_0 can take. There are three constraints upon b_0.

(1) Clearly we must have

$$b'_{1,3} - b_0 \geq 0$$
$$b'_{2,3} - b_0 \geq 0$$

because the added branches cannot have negative capacity.

(2) We must also maintain the constraint at node x_3

$$\sum_n b'_{3,n} \geq \max_i(r_{3,i})$$

(3) The third constraint is less obvious. Consider the minimum cut $(X_i, \bar{X}_i)$ with $x_1, x_2 \in X_i$, $x_3 \in \bar{X}_i$ shown in fig 3.2(c). The capacity of this cut is reduced by $2b_0$ and in the modified graph it should have sufficient capacity to transport the flow from

all nodes $x_i \in X_i$ to node x_3.

$$c'(X_i,\bar{X}_i)=c(X_i,\bar{X}_i)-2b_0 \geq \max_i(r_{3,i}) \quad x_i \in X_i$$

The value of b_0 may be maximised provided that it satisfies all these constraints. Notice that it is necessary to perform only one flow calculation for the location of the cut $(X_i,\bar{X}_i)$ and so the method is computationally efficient.

This argument may be significantly generalised so that the nodes x_1 and x_2 become supernodes X_1 and X_2. Instead of requiring of the transformation that

$$b_{1,n}=\max_i(r_{1,i}) \qquad b_{2,n}=\max_i(r_{2,i})$$

we require that

$$c(X_1,\bar{X}_1)=\max_i(R_1) \qquad c(X_2,\bar{X}_2)=\max_i(R_2)$$

All the constraints upon b_0 remain unchanged by this generalisation.

It was noted in Section 3.3.2 that the suboptimal network generated may be such that $t'_{i,j}>r_{i,j}$. The constraints on b_0 have been expressed in terms of the requirement matrix so that, where possible the capacities of the various cuts in the network will be reduced by the transformations so that $t'_{i,j}=r_{i,j}$.

3.3.4 Minimisation of Network Capacity

A technique for applying these transformations so as to minimise the capacity of any graph is developed in this section.

Consider fig 3.3(a) which shows a simple situation where the added branches form a star tree such that

$$\sum_j b'_{i,j} = \max_k(r_{i,k}) \quad i=1,2,3$$

$$\sum_j b'_{i,j} > \max_k(r_{i,k}) \quad i=4$$

The capacities of the branches are shown with each branch. The possible paths of length two to which the transformation may be applied are $\{a_{1,4},a_{4,2}\}$, $\{a_{1,4},a_{4,3}\}$ and $\{a_{2,4},a_{4,3}\}$. We need to decide on the best order of applying the transformations so as to minimise the capacity of the graph. Choosing to apply the transformation first to the path $\{a_{1,4},a_{4,2}\}$ results in the graph of fig 3.3(b) which has capacity 5 whereas if the transformation is applied to $\{a_{2,4},a_{4,3}\}$ and then to $\{a_{1,4},a_{4,3}\}$ the graph of fig 3.3(c) is obtained with capacity 4. Clearly, the rule that can be deduced from this example is that paths containing arcs of large capacity should be transformed first followed by those of smaller capacity.

This technique is generalised so that it can be applied to any graph. In section 3.3.1 the clustering of nodes into supernodes was described in an hierarchical manner. This viewpoint will be used below. At the highest level, a network is a star-tree with the outer nodes clusters of nodes X_i and the inner node being an ordinary node (fig 3.4(a)). The transformation technique for star-trees may now be applied to this tree giving a result such as that shown in fig 3.4(b). The internal properties of the sets X_i are now considered as these clusters are expanded one at a time. Each 'node' within X_i may again be a cluster of nodes. Consider the graph of fig 3.4(c) where one of the clusters has been expanded. A new star-tree, consisting of nodes $\{x_2,x_3,x_5,x_6,x_7,x_8\}$ has been generated to which the minimisation technique can be applied. The method continues by expanding clusters one at a time. On each expansion a new star-tree is formed, the capacity of which can be minimised by transformations.

An example of the application of the algorithm will now be given. The capacity of the network of fig 3.1 will be minimised. The star-tree at the highest level is shown in fig 3.5(a). Transforming this tree by the rules given leads to the expansion network shown in fig 3.5(b). The next stage is to expand the supernode {4,6} into its component nodes to create a star-tree with {4} as its inner node and {1,2,3,5,6,{7,8}} as its outer nodes. This gives fig 3.5(c). Transforming this star-tree gives fig 3.5(d). Finally supernode {7,8} is split into its components and transformations applied to the star-tree with {7} as its inner node and {1,2,3,4,5,6,8} as its outer nodes. This gives the final optimum expansion of fig 3.5(e).

It should be noted that the expansion may not be unique eg the network of fig 3.5(f) also has the same total capacity but a different structure.

There is a small problem which remains to be considered. Under some circumstances the branch capacities may become non-integral. From the constraints upon the transformation factor b_0 we have

$$2b_0 \leq c(X_i, \bar{X}_i) - \max(R_i)$$

If this is an equality and RHS is odd then the value of b_0 is not an integer. If this is undesirable then it can be avoided by multiplying all branch and terminal capacities in the network by a suitable power of 2. Alternatively, the value of the branch capacities can be truncated; this may lead to a solution which is not optimal but experience has shown that the error is always small.

That the capacity of the expanded graph is minimum may be seen

by considering the following points.

(1) The total capacity incident upon the outer nodes of the cut-tree was initially set at the minimum value which would satisfy the constraints and has not been changed at any time.

(2) The capacity of the arcs crossing minimum cuts in the graph have been reduced to the minimum possible without violating any constraints.

(3) The transformations have been performed in such an order as to minimise the capacity of the network.

3.3.5 Exactly Realisable Network Expansions

It was noted earlier in the example that some of the entries in the terminal capacity matrix of the sub-optimal expansion were greater than the corresponding entries in the requirement matrix and although the algorithm was constructed so that during the minimisation process the capacities of the appropriate cuts could be reduced, examination of the results shows that this did not occur. This specific point is investigated below and the necessary conditions are given so that a graph may be synthesed in such a way that it meets the requirement constraints exactly.

Theorem 3.4

A network expansion is exactly realisable if

$$r_{i,j}-t_{i,j}\geq\min(r_{i,k}-t_{i,k},r_{k,j}-t_{k,j})\ \forall\ (i,j,k)$$

Proof

Form the matrix D such that $d_{i,j}=r_{i,j}-t_{i,j}$. This matrix gives the increment in capacity that must be added to each cut in the

network to meet the new requirements. Now consider a pair of nodes x_i, x_j and the minimum cut between them in the original graph $(X,\bar{X})$ $x_i \in X$, $x_j \in \bar{X}$. Any third node x_k belongs either to the set X or to $\bar{X}$. If $x_k \in X$ then the increment in capacity $d_{j,k}$ can be no greater than the increment in capacity $d_{i,j}$ ie $d_{j,k} \leq d_{i,j}$. If $x_k \in \bar{X}$ then similarly $d_{i,k} \leq d_{i,j}$. Combining these results gives the inequality

$$d_{i,j} \geq \min (d_{i,k}, d_{k,j})$$

and if this is not satisfied for all i,j,k then the expansion is not exactly realisable.

In the example $d_{3,7} < \min (d_{3,8}, d_{8,7})$ and hence the expansion is not exactly realisable.

The algorithm as presented generates a 'dominating' requirement matrix whose entries are larger than the original requirements if necessary.

3.3.6 Computational Complexity

By making a number of reasonable assumptions we can make an estimate of the computational complexity of the algorithm.

The initial analysis procedure makes use of the Gomory-Hu algorithm so a graph of N nodes requires (N-1) applications of the Ford-Fulkerson algorithm. The other computationally intensive section of the algorithm is the calculation of the changes in branch capacity in the transformations, each of which requires a flow calculation. We therefore need to estimate the number of transformations that need to be applied.

To obtain some estimate of this number let us assume that for

an N node graph, at each expansion of a supernode, the number of nodes in the star-tree being transformed is increased over the number of nodes in the previous star-tree by a certain fraction of the total number of nodes (N/M). At the I'th step in the optimisation there are (IN/M) nodes in the star-tree so the number of transformations in this stage is

$$\frac{N}{M}\ \frac{(I-1)\ N}{M} \quad + \quad \frac{1}{2}\ \frac{N}{M}\left(\frac{N}{M}-1\right)$$

The total number of transformations is therefore

$$\sum_{I=1}^{M}\left[\frac{N}{M}\ \frac{(I-1)\ N}{M} \quad + \quad \frac{1}{2}\ \frac{N}{M}\left(\frac{N}{M}-1\right)\right]$$

$$=\frac{N^2}{2}+\frac{N^2}{2M}-\frac{N}{2}$$

$$=O(N^2)$$

Therefore in the whole algorithm we would expect to have to perform $O(N^2)$ flow calculations. Each flow calculation requires at most $O(N^3)$ applications of the Ford-Fulkerson labelling procedure [DI1], [ED1] and so the algorithm may be regarded as (at worst) $O(N^5)$. These are worst case figures and generally graphs encountered in practice will require fewer transformations on account of some branches having zero capacity and fewer passes of the labelling routine and so, as the practical results below show, the complexity will be less.

A computer program to implement the algorithm was written in FORTRAN 77 and run on a CDC Cyber 170/855. Graphs to be used as

input data for the algorithm were generated as follows. Beginning with a set of N vertices, a spanning arborescence was randomly generated and arc capacities were allocated from a uniform distribution. Each additional arc was then randomly generated so as to span the minimum cut in the network so far generated until the required number of arcs was added. The capacities of these arcs were also randomly selected from the same uniform distribution. It was found possible to operate the algorithm on graphs of medium size (up to about 40 nodes). The results from a number of these tests are summarised in table 3.1 and figs 3.6 and 3.7. Fig 3.6 plots the number of flow calculations against the size of the graph for a number of examples. The upper and lower ends of the lines mark the maximum and minimum number of calculations encountered. Shown on the graph is the line $N^2/2$ and it can be seen, as predicted, that this is an upper bound for the number of flow calculations which was reached in a few instances. Also shown is a regression line giving the average performance of the algorithm which is $N^{1.7}$.

The total running times of the algorithm are given in table 3.1 and fig 3.7. It can be seen that it is better than an $O(N^5)$ in practice; it is somewhere between $O(N^3)$ and $O(N^4)$.

3.4 Discussion and Conclusion

There have been essentially three ideas presented in this chapter.

(1) A new set of conditions under which the T-matrix of a graph formed by the addition of two B-matrices can easily be calculated have been given and it has been shown that these can be used for the generation of a sub-optimal network which

satisfies the flow constraints of a requirement matrix.

(2) An elementary network transformation for the reduction of the capacity of a graph while not violating any requirement matrix constraints has been given. To effect this transformation it is necessary to perform a single flow calculation. Since there is a strong interaction between the original network and the expansion network it is clearly necessary to perform some investigation of the properties of the graph whilst performing the optimisation and a single flow calculation is the minimum possible

(3) We have shown how a sequence of these elementary transformations should be performed so as to reduce to a minimum the capacity of an expanded network. A worst case analysis of the algorithms computational complexity was given. Generally the optimisation ran in less time than this calculation would suggest but occasionally a graph would attain to this worst case.

The algorithm can be viewed as a generalisation of algorithms presented in the past for network synthesis and optimal network synthesis where the extensions are to cope with the extra difficulties which accrue from having to consider the properties of the original network and its interaction with the expansion network.

Close connections with the algorithms of Wing and Chien [WI1] and Gomory and Hu [GO1] for minimum cost network synthesis can be seen. Firstly considering the method of Wing and Chien it can be seen that what are termed elementary nets in that paper are similar to the results of applying the transformation technique for the minimisation of a star-tree. These elementary nets are constructed

such that each vertex has just sufficient incident capacity to satisfy the flow requirements from that vertex. This can be compared with the algorithm for network expansion in which initially it is only at the outer nodes of the star-trees that the capacity requirements are satisfied whereas the inner nodes have excess capacity, but the algorithm attempts to reduce the excess capacity on the inner nodes towards zero. If a T-matrix were to satisfy the conditions specified by Wing and Chien and the properties of the external network did not interfere with the optimisation process then the final expansion network structure would be an elementary net.

There is also a similarity between the optimisation and the minimum cost synthesis procedure given by Gomory and Hu. In both instances there is excess capacity at interior nodes of a path (which are of length two in the algorithm for network expansion but of any length in that of Gomory and Hu) but not at the end nodes. The excess capacity is reduced by reducing the capacity of the edges within the path while adding an extra edge between the end nodes of the path to maintain the capacity constraints at these nodes.

Number of	Flow Calculations		Computing Time	
Nodes	Min	Max	Min	Max
5	5	13	0.28	0.38
8	5	20	0.6	1.35
10	13	51	1.35	7.1
15	22	61	3.1	8.0
20	48	152	18	50
25	64	145	92	120

Table 3.1

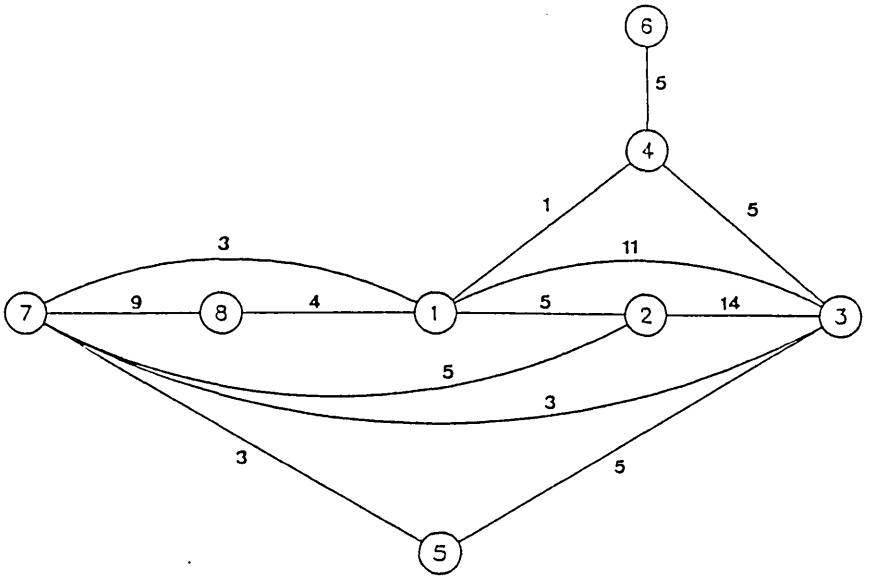

Figure 3.1(a)

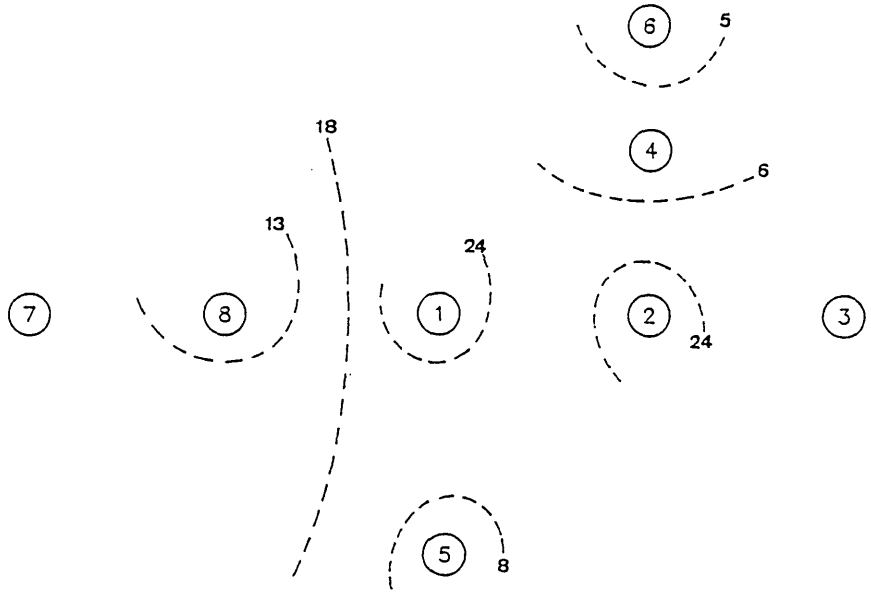

Figure 3.1(b)

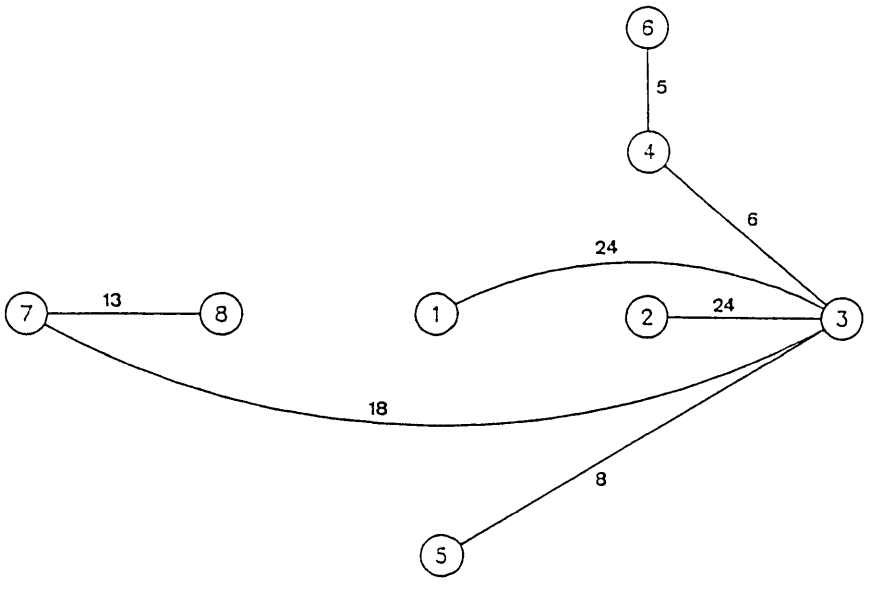

Figure 3.1(c)

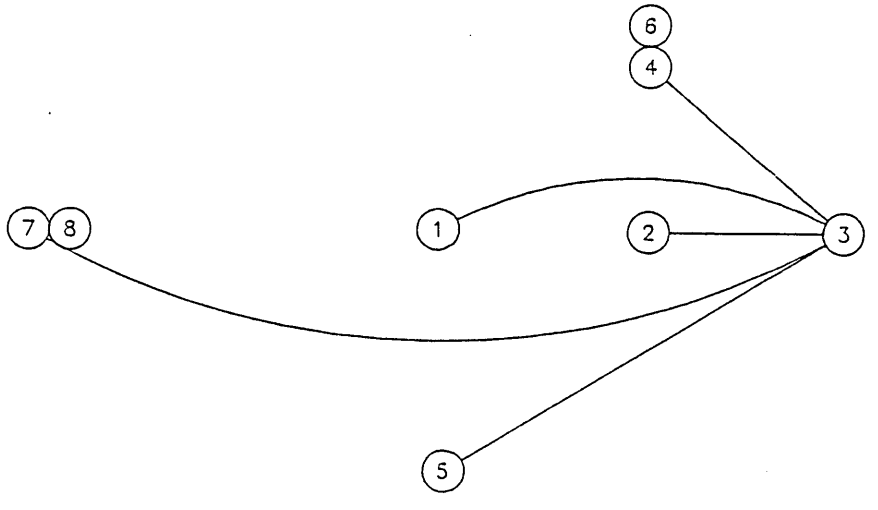

Figure 3.1(d)

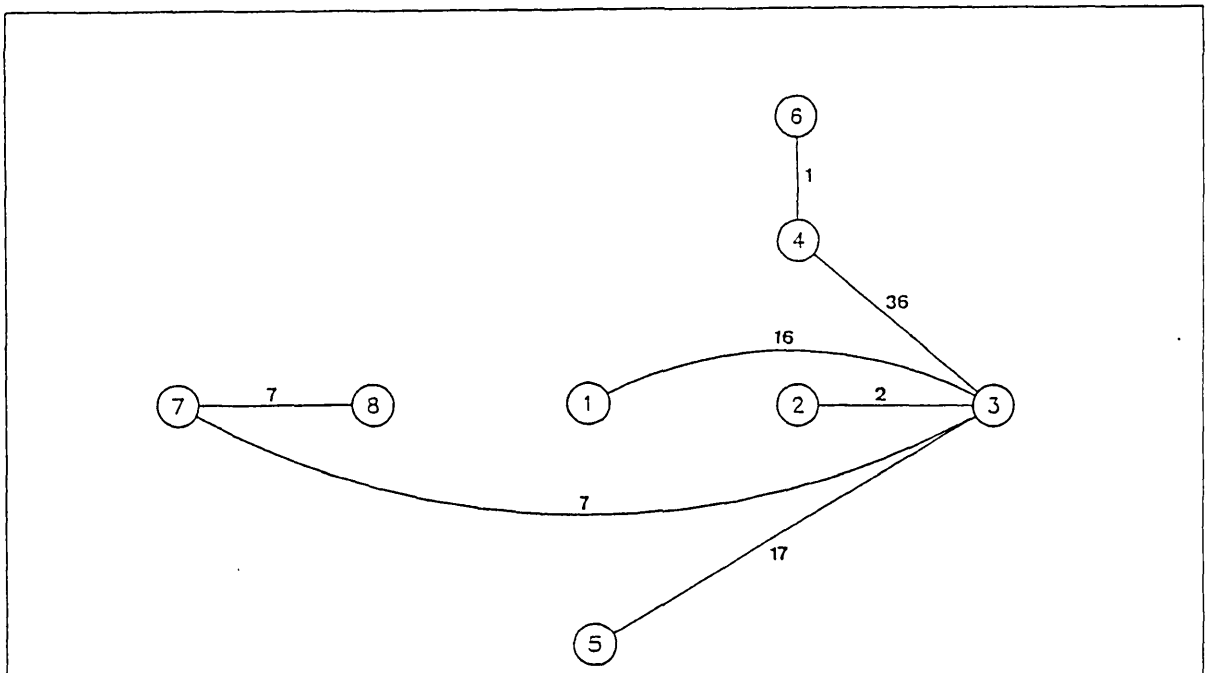

Figure 3.1(e)

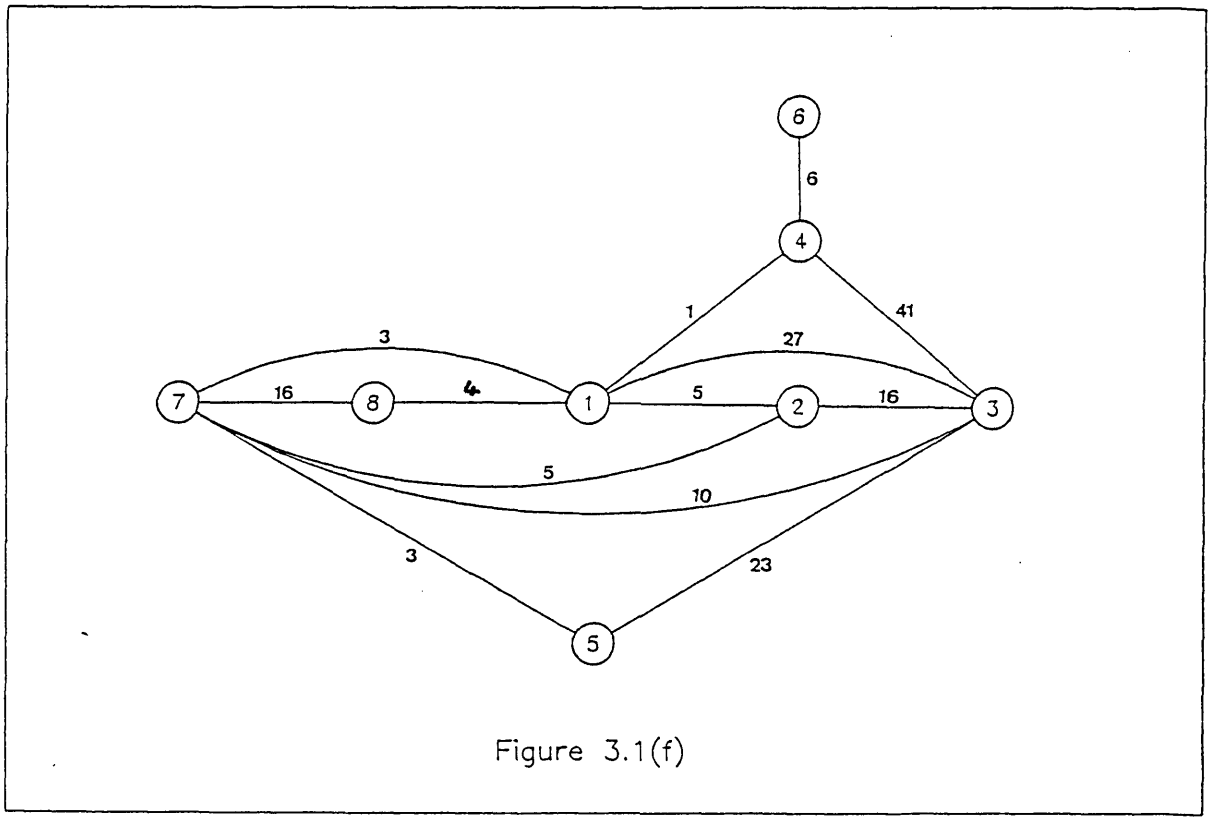

Figure 3.1(f)

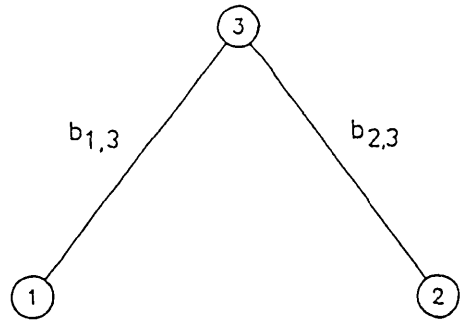

Figure 3.2(a)

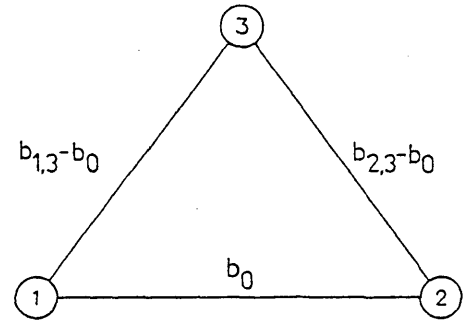

Figure 3.2(b)

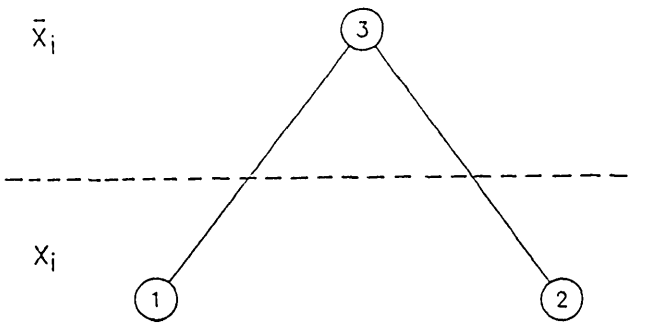

Figure 3.2(c)

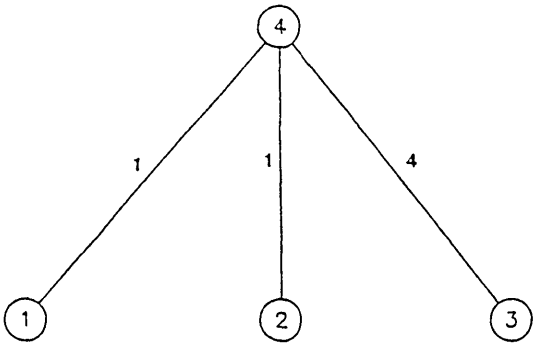

Figure 3.3(a)

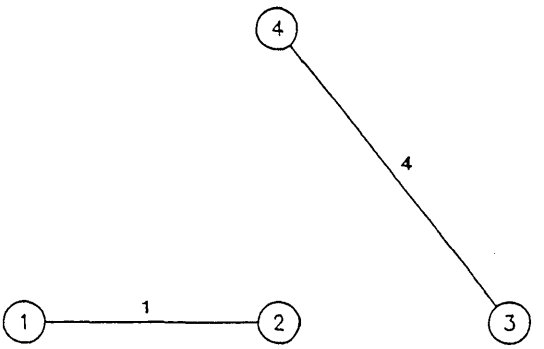

Figure 3.3(b)

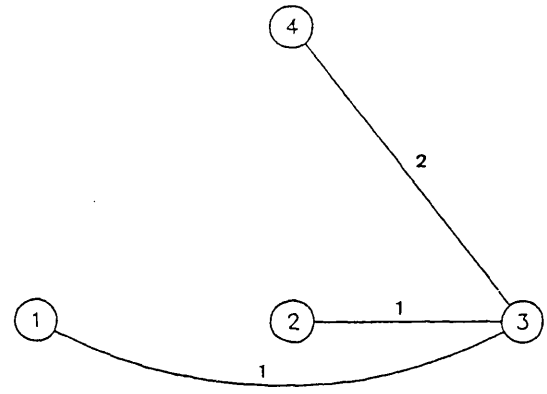

Figure 3.3(c)

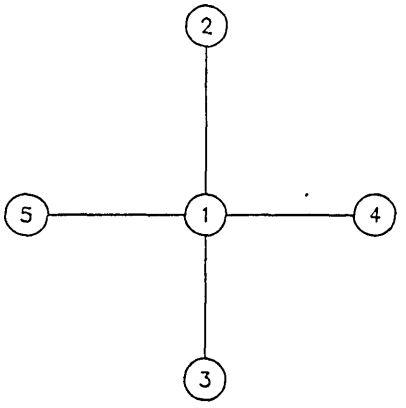

Figure 3.4(a)

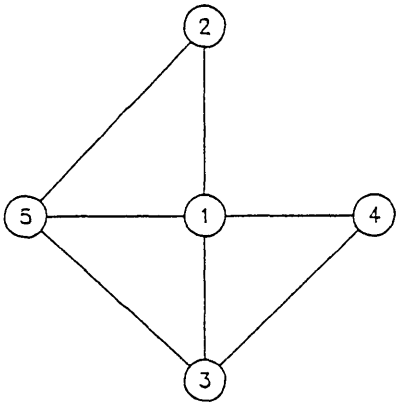

Figure 3.4(b)

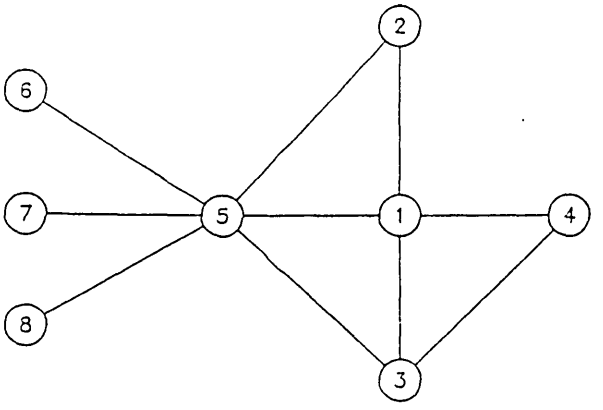

Figure 3.4(c)

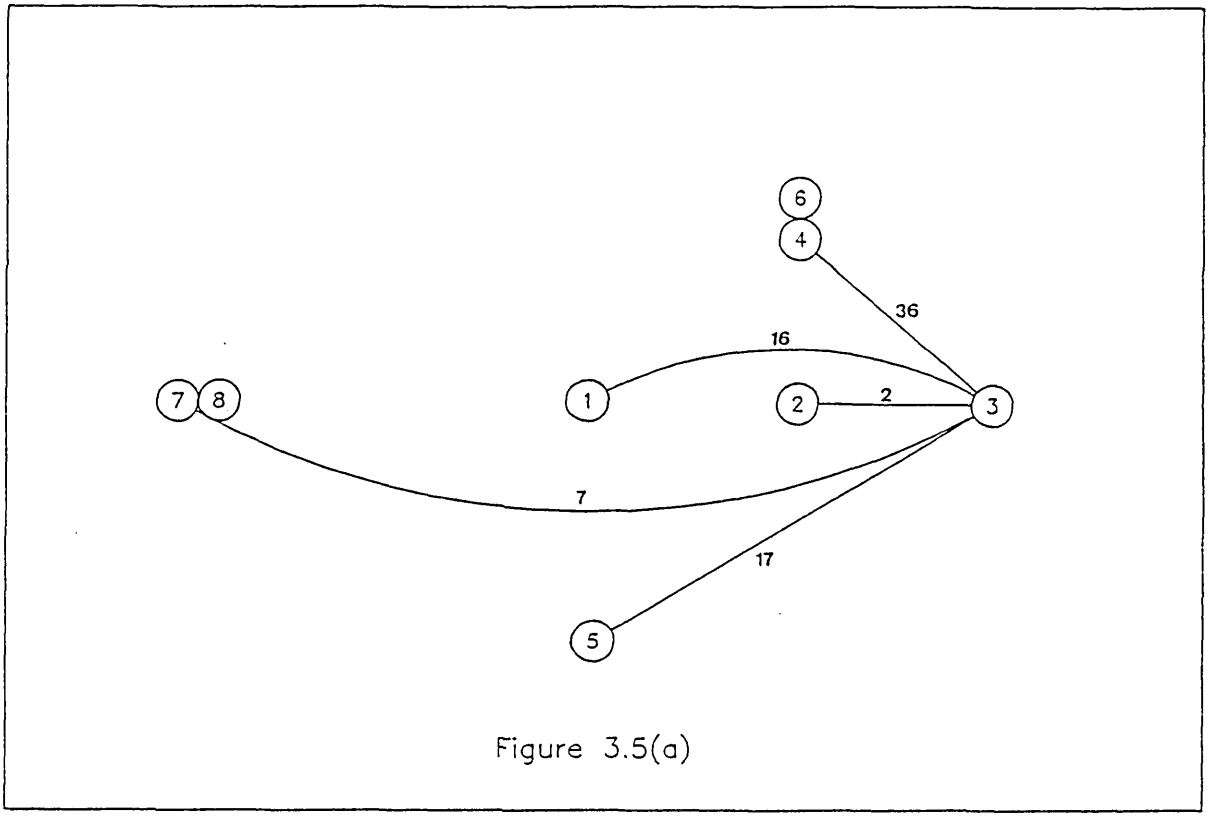

Figure 3.5(a)

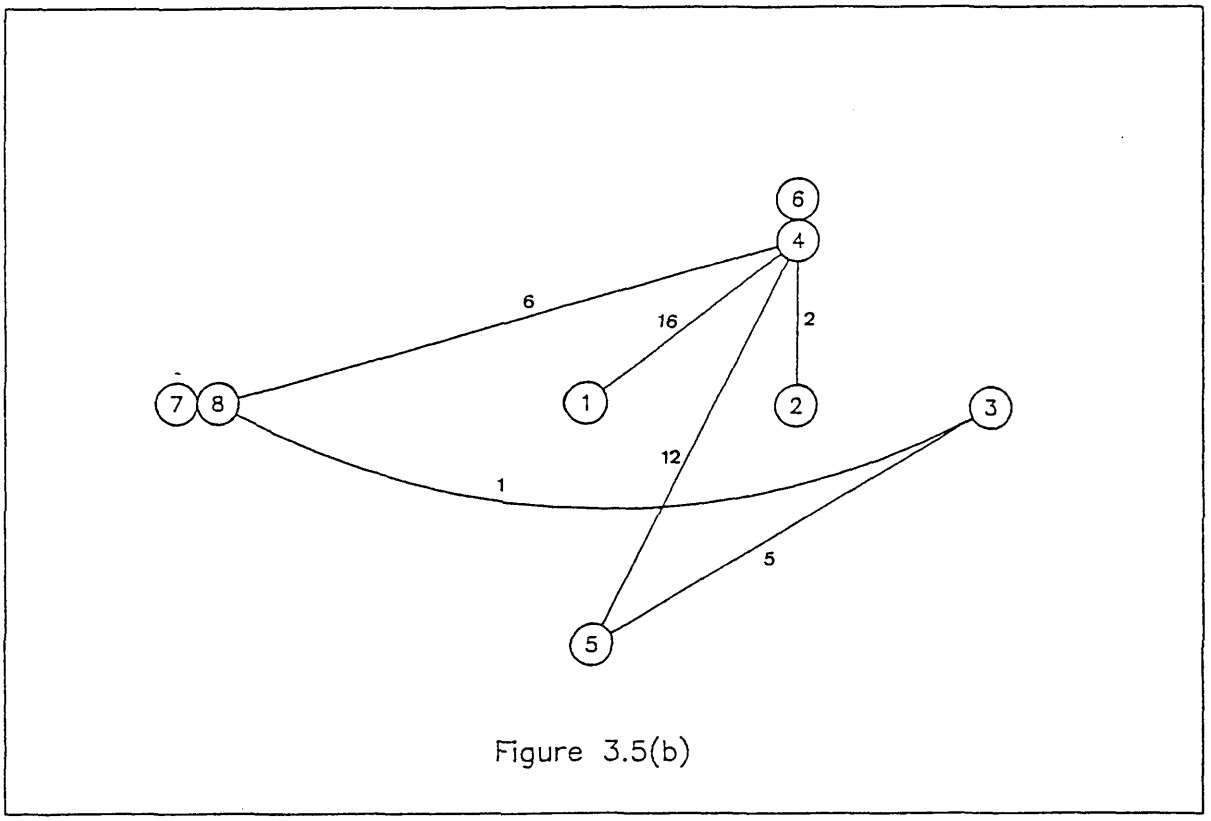

Figure 3.5(b)

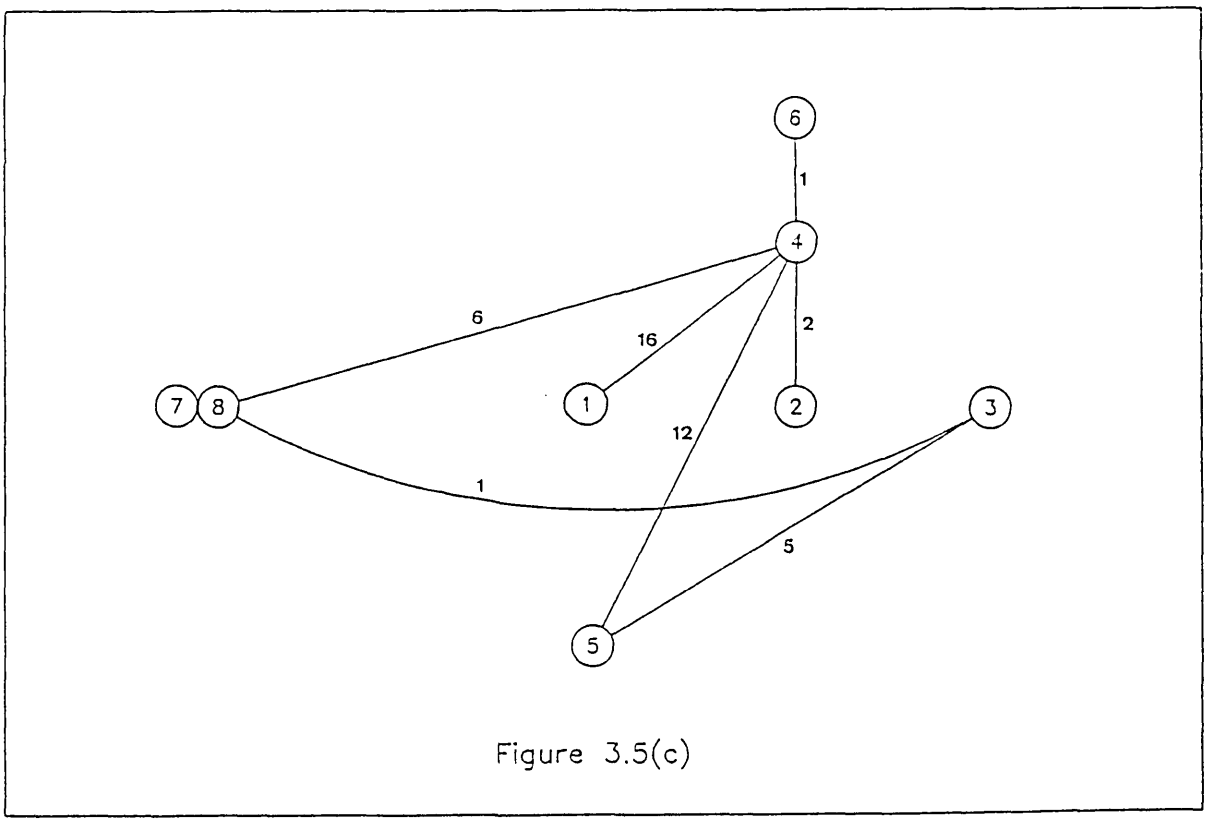

Figure 3.5(c)

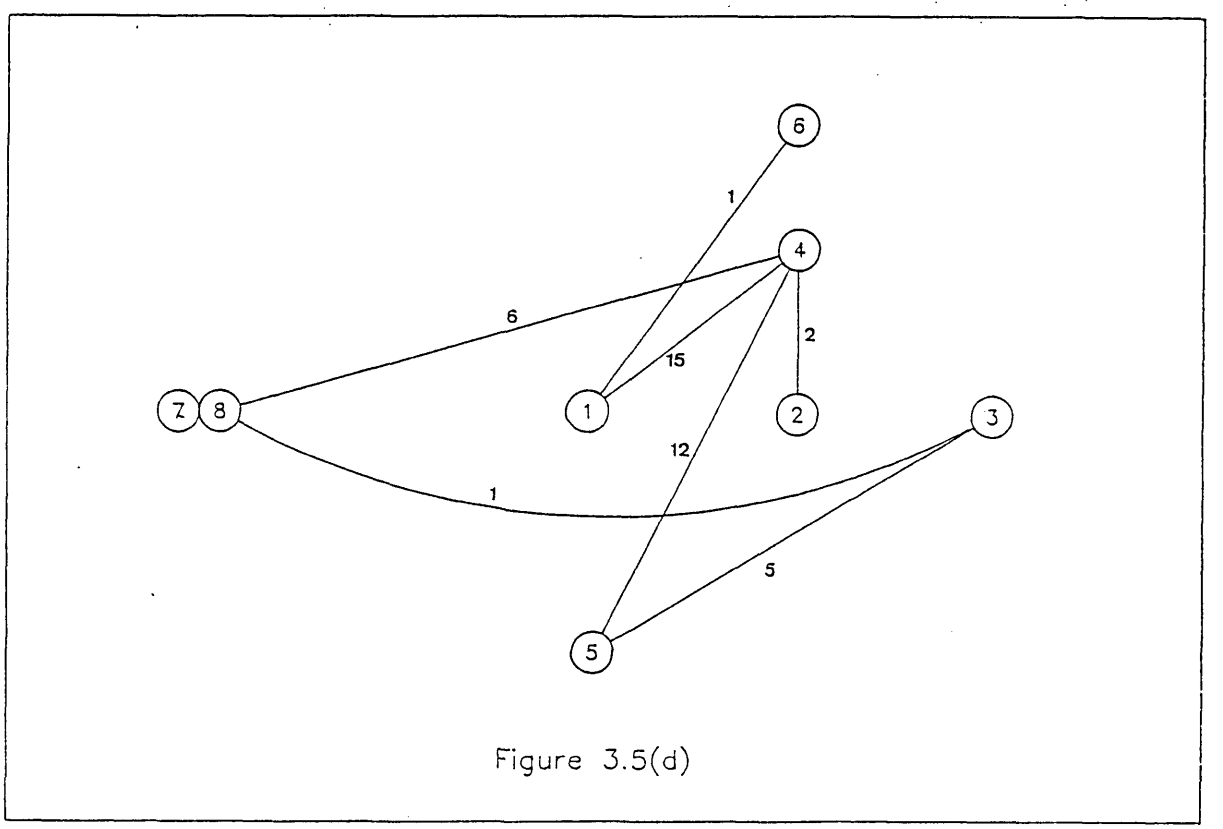

Figure 3.5(d)

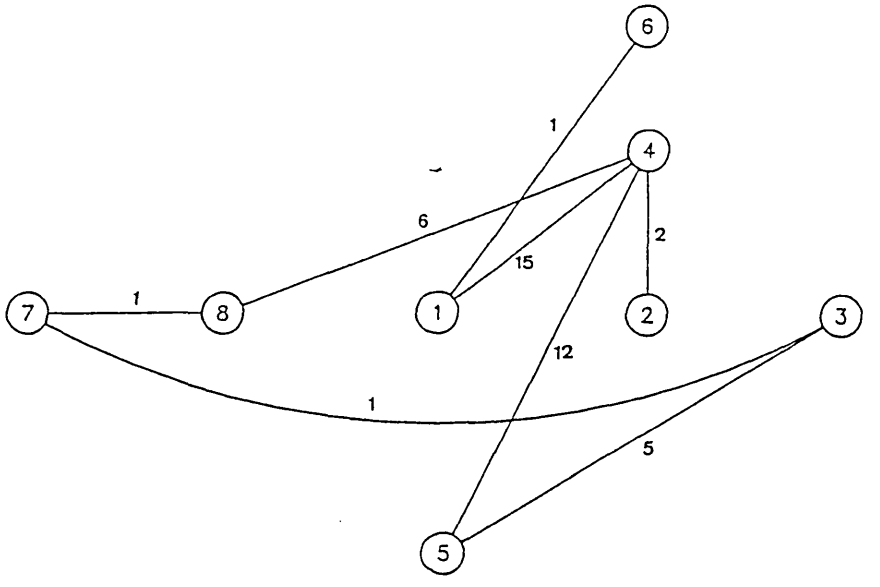

Figure 3.5(e)

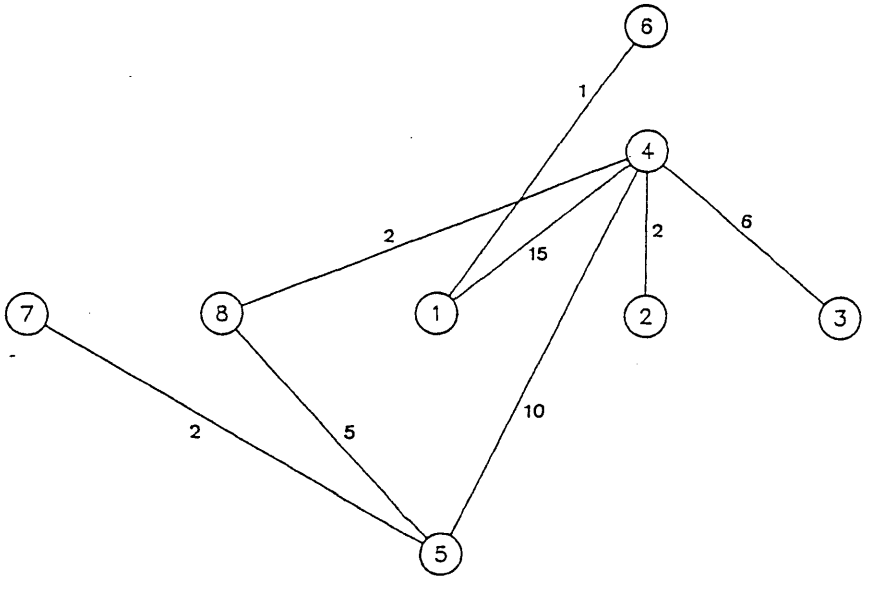

Figure 3.5(f)

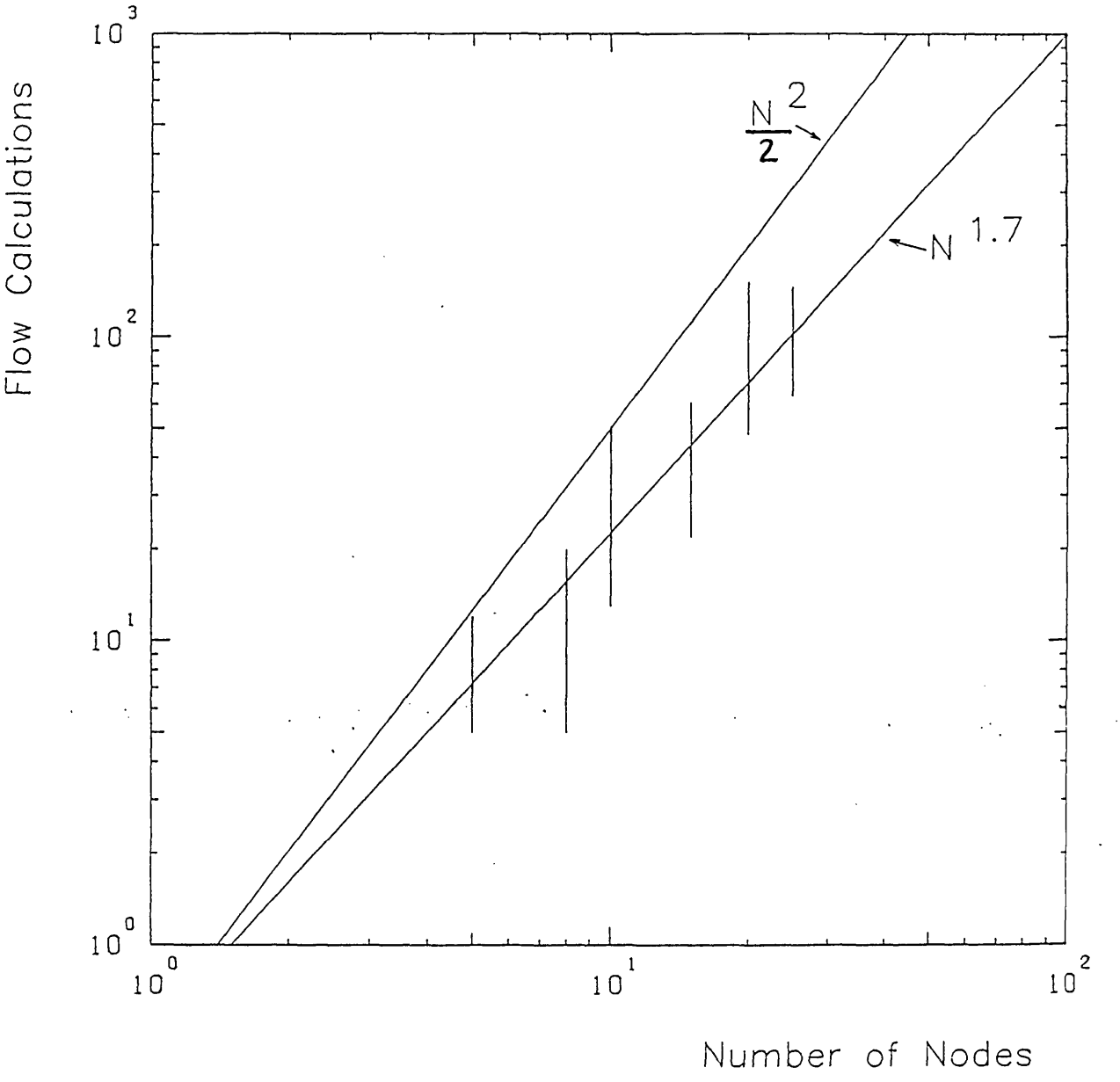
Flow Calculations
10^3
10^2
10^1
10^0
10^0
10^1
10^2
$\frac{N^2}{2}$
$N^{1.7}$
Number of Nodes

Figure 3.6

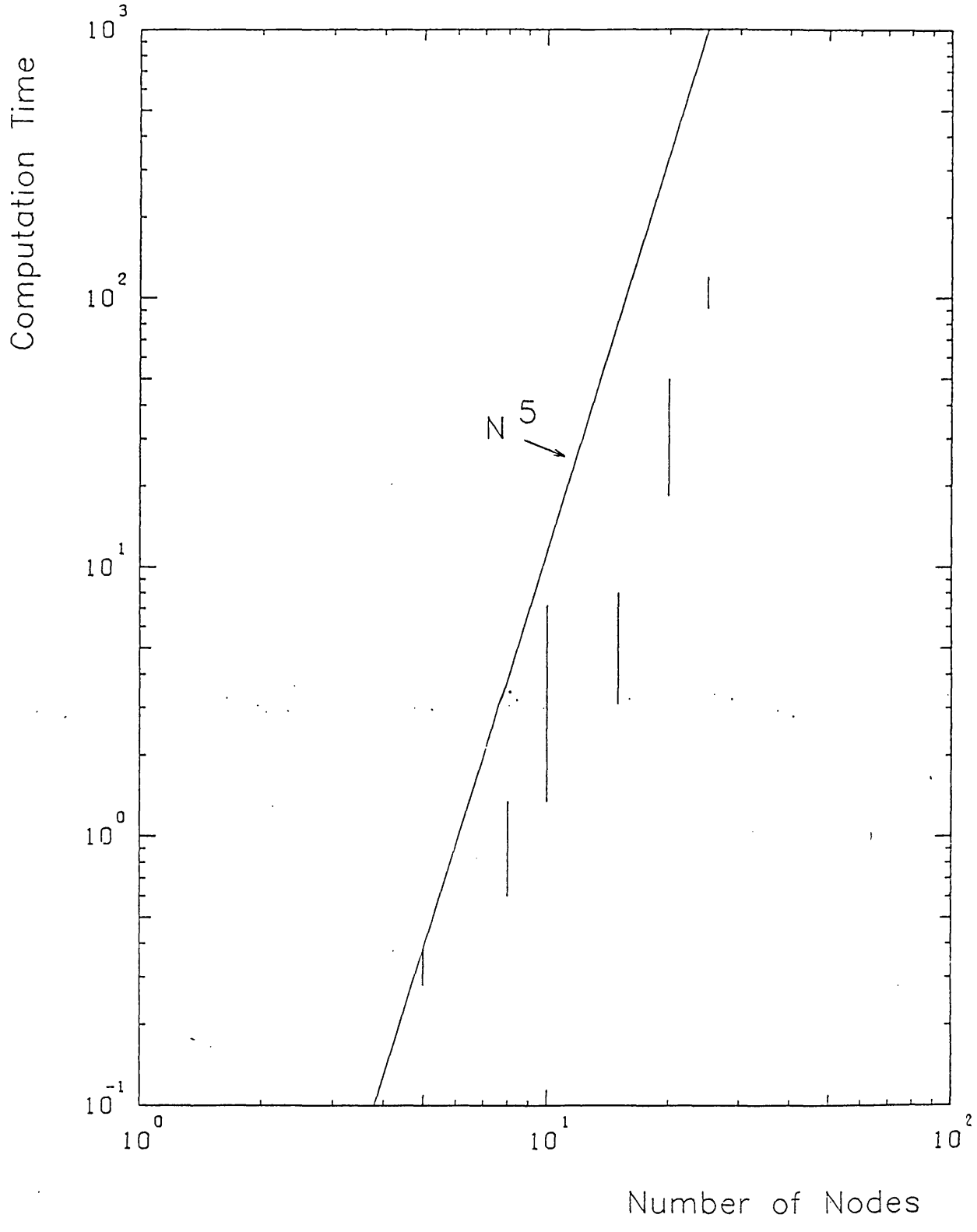
Computation Time
10^3
10^2
10^1
10^0
10^{-1}
N 5
10^0
10^1
10^2
Number of Nodes

Figure 3.7

NON-UNIFORM COST MATRIX

NETWORK EXPANSION

4.1 Introduction

In this chapter we consider a problem which is a generalisation of that considered in the previous chapter and suggest an algorithm for its solution. Since the problem is more general, the solution technique developed is much more complex than that developed in the previous chapter and takes considerably more computation, so some alternative heuristics which lead to sub-optimal solutions but with little computational effort are also suggested and evaluated.

As before, we postulate a situation in which it has become necessary to increase the terminal capacity matrix of a network in some optimal fashion. But now we attribute to each edge a cost proportional to the capacity of the edge - different for each edge - and require that the expansion should be such that the total cost of the added edges is minimum.

Formally the problem may be stated as follows. Given a graph $G=(X,A)$ of branch capacity matrix B, terminal capacity matrix T, a realisable requirement matrix R such that $r_{i,j} \geq t_{i,j}$ $\forall$ (i,j) and a cost matrix C, how may a graph $G'=(X,A')$ with branch capacity matrix B', terminal capacity matrix T' be synthesised such that $t'_{i,j} \geq r_{i,j}$ $\forall$ (i,j), $b'_{i,j} \geq b_{i,j}$ $\forall$ (i,j) and $\sum c_{i,j} b'_{i,j}$ is minimum?

As in the previous chapter the algorithm proceeds by first synthesising a sub-optimal network which meets the requirement matrix constraints and then modifies the resulting network in such a way that the total cost is minimised without violating any of the constraints. As a preliminary to presenting an algorithm for solving the problem, we examine some appropriate concepts in Section 4.2.

4.2 Theoretical Preliminaries

4.2.1 Edge Set in Expansion

At first sight, it might appear that for an N node graph all possible branches in the graph (ie N(N-1)/2 branches) would have to be examined as possible candidates for inclusion in the expansion, but in fact only a subset of the branches need be considered and the remainder can be eliminated 'a priori'.

Consider the cost matrix of the network to be a distance matrix and find the shortest paths between all pairs of nodes using some such technique as Floyds algorithm [FL1]. It is argued below that the only edges $a_{i,j}$ in the network which need be considered are those whose length ($c_{i,j}$) is the same as the length of the shortest path between x_i and x_j ie those edges $a_{i,j}$ which are themselves the shortest paths.

Consider a node pair (x_i,x_j) where the shortest path between them is of length $l_{i,j}$ and passes through nodes $(x_i,x_1,x_2,\ldots,x_k,x_j)$. Assume that a minimum cost network expansion exists which contains edge $a_{i,j}$ at capacity $b_{i,j}$ and remove arc $a_{i,j}$ and insert instead the edges $a_{xi,x1},a_{x1,x2},\ldots,a_{xk,xj}$ at capacity $b_{i,j}$. The capacity of the minimum cut between x_i and x_j has not been altered, and no other cuts have been reduced; in fact some minimum cuts may have been increased in capacity. But the cost of the network has been reduced since $l_{i,j}<c_{i,j}$ which contradicts the original assumption that the expansion was of minimum cost. Therefore, the only arcs which can be included in the solution are those which are the shortest path between pairs of edges. Let these edges be known as admissible edges.

4.2.2 Generalised Network Transformations

A more general network transformation is required to reduce the cost of a network than was presented in the previous chapter. We attempt to insert in the network arcs from the set of admissible arcs and remove other edges. The approach we take is to identify paths in the expansion whose end points are also end points of an admissible edge and replace the path with the single edge at an appropriate capacity level so as not to violate any constraints.

Consider the part of a network expansion shown in fig 4.1 in which $a_{i,j}$ is an admissible edge and $P_{i,j}=(x_i,\ldots x_m,x_n,\ldots x_j)$ is a path in the expansion network. The transformation we propose is to increase the capacity of the edge $a_{i,j}$ by an amount b_0 and reduce the capacity of the edges in the path, each by a different amount, where the reductions in capacities are chosen so as not to violate any flow constraints. To maintain the incident capacity constraint at x_i and x_j the first and last edges in the transformation should be decreased by b_0.

For convenience now and later on this transformation is decomposed into a set of simpler transformations. For ease of notation, let the edges in the path be labelled a_1, $a_2,\ldots,a_n$ and consider the set of 'partial paths' constructed as follows. Each partial path should contain edges a_1 and a_n; simplest path is that consisting solely of these two edges. The other paths are those consisting of a_1 and a_n together with all combinations of the other edges taken singly, in pairs, in threes, etc., the final path being all the edges of the original path. The transformation that can be applied to these simpler paths is also simpler in that the change of

capacity to each to each element is the same. So the transformation to the path P_m is to increment $a_{i,j}$ by a capacity b_m and decrement $a_{k,l}$ by a capacity b_0 if $a_{k,l} \in P_m$ where i, j are the end points of the path and k, l are interior points. Any general transformation of the type described above can be constructed out of a combination of these simpler transformations. It is necessary to include edges a_1, a_n in every path so that a transformation does not increase the capacity incident upon any node which is never required.

Let us examine the constraints upon the transformation factor b_m.

(1) Clearly a transformation should only be performed if it reduces the cost of the network ie we must have

$$b_m(\sum c_{k,l} - c_{i,j}) > 0$$

so $$b_m = 0 \text{ if } \sum c_{k,l} - c_{i,j} \leq 0$$

(2) The added branches must have positive capacity

ie $$b'_{k,l} - \sum b_m > 0$$

(3) The total capacity of the arcs incident upon every interior node of the path should be sufficient to satisfy the requirement matrix. These capacities are reduced by b_m or $2*b_m$ depending on whether one or two edges in the path are incident upon a particular node.

$$\text{ie } \sum_n b'_{k,n} - N*b_m > \max_l(r_{k,l}) \quad \{k \mid x_k \in P_m, k \neq i,j\}, \; n \in \{1,2\}$$

(4) It must be that reducing the capacity of the arcs in the path does not reduce any minimum cut in the network so much that the network no longer satisfies the requirement matrix constraints. Consider the networks of fig 4.2(a), (b), (c). For each node $k \neq i,j$ in the path, the minimum cut $(X_k, \bar{X}_k)$ with x_i, $x_j \in \bar{X}_k$,

$x_k \in X_k$ is reduced by an amount $2b_m$, b_m or 0 depending on how the path crosses the cut. The capacity of these cuts should remain large enough to transport the required flow for all nodes $x_i \in \bar{X}_k$ to x_k.

$$\text{ie } c'(X_k, \bar{X}_k) = c(X_k, \bar{X}_k) - N*b_m \geq \max(r_{k,1}) \quad x_k \in X_k \ , \ N \in \{0,1,2\}$$

It should be borne in mind that the capacities of the cuts are not constant but depend upon any previous transformations that may have occurred and it is from this fact that the difficulty of the problem arises.

(5) Finally, if the path only crosses the minimum cut in one arc only then there is a further constraint which must be satisfied, which is that the minimum cut which crosses the path in two places must also not be reduced excessively. Referring to fig 4.2(d) which illustrates such a situation, it can be seen that we must have

$$c'(X_m, \bar{X}_m) = c(X_m, \bar{X}_m) - 2*b_m \geq \max_1(r_{k,1})$$

The calculation of the maximum allowable value of b_m requires 2(N-2) flow calculations for an N node path.

4.2.3 Problem Relaxation

A useful technique is the consideration of a relaxed version of the problem. A relaxation is a version of an optimisation problem in which the objective function is unchanged but some of the constraints are changed or relaxed so as to make the optimisation more tractable. The solution of this simpler problem can often be used as an aid in the solution of the complete problem.

The relaxation we make is to ignore the constraints (4) and

(5) of Section 4.4.2, ie to ignore the possibility that the results of a transformation could violate any requirement matrix constraints. This relaxed problem can be solved by linear programming [DA2].

The linear programming formulation of the problem is as follows. Let $c_{i,j}$ be the reduction in network cost that arises from a transformation which adds unit capacity to the admissible edge $a_{i,j}$ and removes unit capacity from the path joining x_i and x_j; let $b_{i,j}$ be the total capacity added to edge $a_{i,j}$. Then the total reduction in network cost arising from all network transformations is

$$C = \sum b_{i,j} c_{i,j}$$

where the summation is over all pairs (i,j) such that $a_{i,j}$ is an admissible arc. This is the linear objective function to be maximised.

The constraints on the optimisation space to be searched are:

(1) The added arcs in the expansion should be all positive. This can be rewritten in terms of our new variables as follows. Each edge in the initial expansion tree is in a number of paths whose capacity is to be reduced. The total reduction in capacity should be less than the original capacity of the edge

$$\text{ie} \quad \sum b_{i,j} \leq b_{k,l}$$

where the edge set $\{a_{k,l}\}$ is the set of edges in the initial network expansion and the summation is over all paths $P_{i,j}$ which contain $a_{k,l}$.

(2) The total capacity incident on all nodes should be sufficient not to violate any constraints. Only the capacities incident upon inner nodes of the flow equivalent tree are changed so only these nodes need be considered. Let the initial added capacity at inner node x_k be q_k and the required capacity to meet the constraints be q'_k. The sum of the reduction in capacity of the paths passing through x_i must not reduce q_k to less than q'_k

$$\text{ie} \quad \Sigma b_{i,j} \leq (q_k - q'_k)/2$$

where the summation is over all paths $P_{i,j}$ passing through inner node x_k.

The constraints which are ignored are that the $b_{i,j}$'s should be less than some value so that the flow constraints are satisfied.

This problem is a linear objective function with linear constraints and so can be solved by linear programming methods.

4.2.4 Branch and Bound Optimisation

The optimisation technique used in the algorithm proposed to solve the problem described in the Introduction is the branch and bound or decision-tree search method [LA1]. The basic principle involved in decision-tree search methods is the partitioning of an initial problem P_0 into a number of sub-problems $P_1, P_2, \ldots, P_k$ (whose totality represents problem P_0) followed by an attempt to to resolve each one of these sub-problems. By resolve we mean

either (1) find an optimal solution

or (2) show that the value of the optimal solution to the sub-problem is worse than the best solution obtained so far.

This partitioning is represented by a tree (fig 4.3) in which each node of the tree represents a sub-problem. The reason for the partitioning of a problem P_0 into a number of sub-problems is usually that the sub-problems, because of their smaller size, are easier to resolve. However, in general, it may still be impossible to resolve a sub-problem P_i and so this problem is partitioned further. This partitioning, known as branching, is repeated for every sub-problem which cannot be resolved. Once a complete tree has been generated, then locating the optimum solution is a matter of resolving all the problems at terminal vertices of the tree. Obviously the problems at the leaves should represent fully the original problem

$$\text{ie } P_0 = \bigcup \{P_j | P_j : \text{leaf of tree}\}$$

For computational efficiency, it is also desirable (but not essential) that there should be no duplications in the generated sub-problems

$$\text{ie } P_i \cap P_j = \emptyset$$

The number of sub-problems generated increases exponentially with the depth of the tree and so generating and storing all the sub-problems before examination requires an impractically large amount of memory, and so a technique is required to generate and examine sub-problems sequentially. A commonly used strategy is known as depth-first search [TA2]. In this type of search, branching is continued from the last generated sub-problem until finally a sub-problem is generated which can be resolved. At that point, a backtracking step is taken ie the last-but-one sub-problem

generated is selected and branching continues from that vertex of the tree. The shape of the decision-tree when the first sub-problem is resolved is shown in fig 4.4 where the order of priority for investigation amongst existing sub-problems at this stage is indicated by the numbering.

The method as described so far requires that all sub-problems be generated and resolved directly, and as mentioned previously, the number of sub-problems may be very large. The quantity of computation required to fully investigate the tree may be reduced by the use of bounding. If at any point during a minimisation (maximisation) search a lower (upper) bound on the minimum (maximum) value of the solution of the sub-problem at this vertex can be calculated, and if this bound is greater (less) than the currently best known solution then it is unnecessary to resolve any further sub-problems emanating from this vertex and a backtracking step can be taken. If tight bounds can be obtained, then it is possible to exclude large parts of the tree from the search.

The order in which vertices should be examined when branching forward has not been fully specified. The branching function is the rule which determines this choice. The best function is that which as early as possible locates the optimum solution and for each optimisation problem a heuristic should be developed. A common strategy is to branch onto that vertex which has the lowest (highest) lower (upper) bound.

This optimisation technique is used in the algorithm to solve our problem with the bounds being calculated by solving, by linear programming, the relaxed version of the problem.

4.3 The Algorithm

We are now in a position to describe an algorithm to solve the problem presented in the Introduction.

The first section of the method is identical with that used in the algorithm of Chapter 3. The initial network is analysed as described in Section 3.3.1 and a sub-optimal solution which satisfies the constraints of the requirement matrix is generated as in Section 3.3.2. The remainder of the algorithm is concerned with the minimisation of the network cost by the application of network transformations.

Those transformations which are valid are identified first by finding those edges which are admissible (Section 4.2.1) and then examining the paths which join the ends of the edges in the set of admissible edges.

We assert that if a sequence of transformations of the type described in Section 4.2.2 is performed on the sub-optimally expanded network then a minimum cost network can be obtained. A proof of this is given in Section 4.3.4. A crucial and difficult question in the optimisation process is concerned with choosing the order in which the transformations should be performed should be rerouted so as to obtain a minimum cost network. Unlike the problem of the previous chapter, a set of rules for ordering the transformations cannot be given and it is the location of the optimal ordering which is the subject of the remainder of this section.

As was suggested above, we partition the problem into smaller

and simpler problems by considering the constraints one at a time until a tractable problem is obtained and search the tree thus obtained using bounds generated by solving a relaxed version of the problem to reduce the amount of the tree which need be explicitly searched.

The partitioning of the problem to generate the tree to be searched is as follows. The branching at the top level of the tree is to take each of the paths which are able to be transformed and apply the transformation on each of them separately so as to maximise the reduction of cost for the single transformation by maximising the capacity being rerouted. Thus, if there are N paths eligible for transformation then there is an N-fold branching from the top of the tree. From each resulting pendant vertex, branching can occur in (N-1) directions as each of the remaining paths are taken and transformed. The branching continues until eventually there are N! terminal nodes to the tree. One of these terminal nodes is the optimum solution to the expansion problem (see Section 4.3.4 below) and our task is to search the tree so as to locate the optimum solution.

Two things are required to enable us to perform this search efficiently

(1) A branching rule

(2) A bounding function

which are now be described.

4.3.1 A Branching Rule

The branching rule determines in which order the nodes of the tree should be visited; a good branching rule should guide the search so as to visit the node representing the optimum solution earlier rather than later. The determination of the rule is 'ad hoc'. In this instance the rule used is to branch first onto those transformations which give maximum decrease on network capacity, thus making the search akin to a steepest descent search; the global optimum is not necessarily located at the first attempt and so further searching is necessary. As the computational experience with the algorithm presented later shows, this heuristic has proved to be successful at rapidly locating the optimum.

4.3.2 A Bounding Function

To avoid searching the whole of the tree (which has $\Sigma_1^N i!$ nodes, where N is the depth of the tree) it is necessary to eliminate parts of the tree from the search using a bounding function. Suppose we are at a certain point in the tree which is not a terminal vertex. Solving the relaxed version of the problem as described in Section 4.2.4 gives an upper bound on the reduction in network cost available at any daughter node in the tree. This is because solving the relaxed version of the problem ignores some of the constraints and so the fully constrained optimum cannot be any greater. If the upper bound on the reduction in cost is less than the current best known reduction then a backtracking step can be taken and the part of the tree emanating from this node need not be explicitly searched.

The search terminates when all nodes have been searched either explicitly or by elimination by bounding.

4.3.3 Reduction of Tree Size

In Section 4.2.5 it was noted that it is desirable for computational efficiency that there should be no duplication of problems in the tree. The algorithm as presented so far, does generate a tree with duplications and in this section we suggest a method for eliminating this overlap of sub-problems.

Consider the situation in which two network transformations T_1, T_2 are completely independent of each other so that whether they are performed in the order T_1T_2 or T_2T_1 there is no difference in outcome. Thus, whether in the decision tree we branch first on T_1 and then on T_2 or vice versa, the sub-problems generated after these two transformations are identical and so the sub-trees are identical (fig 4.5). The tree search is much more rapid if such duplication can be avoided.

The way to store the vast amount of information concerning the transformations in a compact form is to record the various amounts of capacity rerouted each transformation and from these values a decision can be taken on whether a problem exists in another part of the tree.

Consider a part of a search tree depicted in fig 4.6. The root of this sub-tree may be any node in the decision tree. Suppose that the sub-problems emanating from node 2 have all been resolved. Examination of the problem at node 3 can lead to two outcomes. Either

(1) the value of the transformation at node 3 has already been generated in one of the problems emanating from node 2. In this case sub-problems emanating from node 3 have been previously generated in the part of the tree extending below node 2 and so there is no need to consider sub-problems 5, 6 etc.

Or

(2) the value of the transformation at node 3 is novel so the graphs corresponding to daughter nodes of node 3 have not been previously generated. Hence it is necessary to examine sub-problems 5, 6 etc. In this search node 3 can become the root of the tree corresponding to node 1 and a similar elimination process undertaken.

If the value of the transformation at node 3 is new then sub-problems 5 and 6 must be fully investigated. Sub-problem 5 must be investigated by searching alone but sub-problem 6 may be compared with 5 for elimination. Once sub-problem 3 and its successors have been examined then all the values of the transformations which occurred in the sub-tree may be collected together with those from sub-problem 2 and then sub-problem 4 resolved by comparing the value of the transformation with this increased set of values of transformations.

4.3.4 Convergence of Algorithm to Optimum Solution

In this section we prove that the algorithm as presented sythesises a minimum cost network. The argument used to prove that the algorithm converges to the optimum solution is similar to that used to prove that the Simplex Algorithm [DA2] solves the linear programming problem.

As was pointed out in Section 4.2.3, the objective function and the constraints of the problem are linear. A simple example of a constraint space with two variables and three linear constraints is illustrated in fig 4.7. The theory of linear programming says that the optimum value of the objective function is at one of the vertices of the space where several constraints meet so to locate the optimum one need only examine the value of the objective function at these points and determine which is largest (or smallest). The Simplex method of Dantzig is an algorithm for performing this operation in a systematic way. However, this algorithm is not applicable to the problem at hand because it is not possible to give explicit expressions for the constraints on the values of the transformations in terms of the flow requirements. Nevertheless, it is still so that the solution lies at a vertex of the notional polytope formed by all the constraints of the problem.

The fact that the global optimum lies at one of the vertices searched by the branch and bound algorithm follows from the argument below which is to the effect that for any vertex not searched by the algorithm, there is a vertex that is searched where the corresponding network has less cost.

Suppose there exists a vertex at which not all the transformations are at the maximum value possible without violating any constraints. Now the objective function to be maximised (to give a minimum cost network) is

$$C = \sum b_{i,j} c_{i,j}$$

where the coefficients $b_{i,j}$ are the values of the network transformations. Differentiating this expression wrt $b_{i,j}$ gives

$$\frac{dC}{db_{i,j}} = c_{i,j}$$

so increasing a transformation factor $b_{i,j}$ will increase the value of the objective function. The coefficient $b_{i,j}$ can be increased until another vertex of the polytope is reached at which it is no longer possible to increase the value of $b_{i,j}$ without violating any constraints. The new vertex corresponds to a network with less cost than that corresponding to the original vertex. This argument can be repeated for all transformations not at the maximum value until a vertex is reached at which all transformations are maximal. This vertex has less cost than the original vertex and will be examined by the algorithm.

4.3.5 An Example

An example showing the operation of the algorithm is given in this section.

Figure 4.8(a) shows a graph whose branch capacity matrix is

$$B = \begin{bmatrix} - & 8 & 10 & 19 & 0 \\ 8 & - & 0 & 7 & 2 \\ 10 & 0 & - & 0 & 0 \\ 19 & 7 & 0 & - & 2 \\ 0 & 2 & 0 & 2 & - \end{bmatrix}$$

and whose terminal capacity matrix is

$$T = \begin{bmatrix} - & 17 & 10 & 27 & 4 \\ 17 & - & 10 & 17 & 4 \\ 10 & 10 & - & 10 & 4 \\ 27 & 17 & 10 & - & 4 \\ 4 & 4 & 4 & 4 & - \end{bmatrix}$$

We wish to expand this graph with minimum cost to a network whose terminal capacity matrix is

$$R=\begin{bmatrix} - & 22 & 17 & 32 & 11 \\ 22 & - & 17 & 22 & 11 \\ 17 & 17 & - & 17 & 11 \\ 32 & 22 & 17 & - & 11 \\ 11 & 11 & 11 & 11 & - \end{bmatrix}$$

where the cost matrix for the expansion is

$$C=\begin{bmatrix} - & 10 & 8 & 10 & 5 \\ 10 & - & 6 & 7 & 6 \\ 8 & 6 & - & 9 & 1 \\ 10 & 7 & 9 & - & 8 \\ 5 & 6 & 1 & 8 & - \end{bmatrix}$$

The first stage of the algorithm is to generate the suboptimal network whose T-matrix meets the requirements. The expansion network to achieve this is shown in figure 4.8(b). Next we locate the edges in the network which are candidates for the expansion as described in Section 4.2.1. In this instance it turns out that all edges except (1,3) may be in the solution, which are nine in number. There are eight transformations which reduce the network cost which are

$P_1=\{(5,4),(4,1),(1,3)\}$

$P_2=\{(3,1),(1,4),(4,2)\}$

$P_3=\{(5,4),(4,1)\}$

$P_4=\{(5,4),(4,2)\}$

$P_5=\{(4,1),(1,3)\}$

$P_6=\{(2,4),(4,1)\}$

and the partial paths of P_1 and P_2

$P_7=\{(5,4),(1,3)\}$

$P_8=\{(3,1),(4,2)\}$

The complete search tree is shown in fig 4.9. The labels on the branches of the tree indicate the values of the transformations

applied to the paths to change the graph of the preceeding node to the graph of the succeeding node. The first solution generated has cost 57 and is obtained with the set of transformations (4,0,0,0,0,1,1,2) ie edges in the path P_1 are reduced in capacity by 4 and the edge (5,3) increased by 4 and the edges in paths 6,7,8 are changed in capacity by 1, 1 and 2 respectively fig 4.8(c). The second solution to be located is the global optimum with cost 51 obtained by the set of transformations (4,0,1,0,0,0,1,2) fig 4.8(d).

4.3.6 Computational Results

Since a branch and bound algorithm is essentially an 'ad hoc' technique for solving a problem, the only way to assess the effectiveness of a technique is to write a computer program for the algorithm and run it with a wide variety of input data. Such a program was written in FORTRAN 77 and run on a CDC Cyber 170/855 computer. The graphs used for testing the algorithm were generated as described in Chapter 3 and the cost matrix was generated by randomly selecting branch costs/unit capacity from a uniform distribution. Table 4.1 details the results of some typical runs of the computer program. (Many other graphs were synthesised during the course of the research and these results may be considered typical.) The information contained in the table is: the number of nodes in the graph, the number of nodes in the tree which were explicitly searched, an estimate of the total number of nodes in the tree, the central processor time required to execute the complete algorithm and the time taken to locate the optimum. Several points may be noted in this table

(1) For graphs of the same size, the time taken to reach the optimum

can vary considerably from problem to problem. Such behaviour is quite common in branch and bound algorithms since the effectiveness of the bounding function depends very much on the way in which the cost function varies when some of the constraints are relaxed.

(2) The complete search tree has an enormous number of vertices but the bounding function is effective in reducing the space which need be searched to a reasonable size.

(3) The size of graph which may be synthesised is quite small. This is because both the computer memory and computer processing time required for larger graphs would be excessive.

(4) There are several graphs (numbers 10,13,15) for which it was impossible to resolve all the sub-problems in a reasonable time. As is discussed below, these particular problems also gave rise to unusual performance by the sub-optimal heuristics.

4.4 Suboptimal Heuristics

Particularly for large problems, the branch and bound algorithm for the solution of the problem discussed in this chapter takes a large amount of computing time and memory and so there may be instances when it is desirable to obtain a good solution (not necessarily the best) with little effort. Some heuristics are suggested in this section which locate a 'good' solution and the solutions obtained are compared with the true optimum solutions.

Two heuristics are suggested for determining the order in which the transformations should be applied and some arguments as to why they might locate good solutions put forward.

(1) Sort the transformations according to the decrease in network cost obtainable for rerouting unit capacity such that those which give largest decrease are applied first. This heuristic is equivalent to stopping the branch and bound algorithm when the first solution has been found and is called the steepest descent method.

(2) Sort the transformations according to the cost of the arc inserted with minimum cost arcs being inserted first. If the expansion network is to have minimum weight, it seems reasonable to construct it out of arcs having the least cost. This is called the greedy algorithm.

These heuristics have been tried on a number of problems and the results are given in table 4.2 which compares the costs of the solutions located by the optimal algorithm and the two heuristics for the same problems of Section 4.3.6. Temporarily ignoring the awkward cases of graphs numbers 10,13,15, in all cases the steepest descent algorithm located a better solution than the greedy algorithm. In 90% of cases the cost of the solution was within 10% of the cost of the optimum solution. These results are typical of a large number of experiments performed. This shows that the branching rule chosen for the branch and bound algorithm, which is the same as the heuristic rule which guides the steepest descent algorithm is a good one in that it quickly locates a close to optimal network. In about 50% of cases, the solution found by the steepest descent method is the optimum solution. For graphs 10,13,15, the greedy algorithm located a better solution and in two out of three of these cases, the resulting network was of less cost than the best solution found by the optimal algorithm before it

terminated. This indicates that there are a minority of graphs for which the suggested branching rule does not perform well and so perhaps two versions of the algorithm should be available, one which uses the suggested rule and the other using a greedy branching rule.

4.5 Discussion and Conclusion.

The method for synthesising minimum cost networks satisfying a given terminal capacity requirement matrix which has been described in this chapter is a generalised version of the method of Chapter 3. Both began by generating a solution within the feasible space and then move in the space searching for an optimum feasible solution.

The network transformation employed is more complex than in the uniform cost case. Furthermore, it is not possible to specify 'a priori' the order in which the transformations should be applied to minimise the network cost. This added complication means that the optimisation strategy to apply should be a search method to which end a tree search was developed. The tree which would arise by permutating all possible transformations is vast and so three devices were employed to reduce the tree size.

(1) It was shown that only a certain subset of the edges of the graph should appear in the solution and so the search need never consider a solution which included any of the forbidden arcs.

(2) A method to avoid the explicit searching of sub-trees identical with sub-trees in other parts of the tree was developed.

(3) Most importantly, a good bounding function was found which was to solve a relaxed version of the problem amenable to solution by linear programming. The bounds obtained were very effective

in reducing the amount of the tree which needed to be searched explicitly.

It must be admitted that the computer implementation of the algorithm permitted only fairly small graphs to be synthesised with reasonable computer resources. This is partly because the program was not written with efficiency in mind but only to demonstrate that the optimisation technique worked. But also the problem is itself very complex and so a large computer program would be expected. These difficulties led to the investigation of heuristics which locate sub-optimal solutions. In most cases these heuristics identified good solutions but unfortunately there is no way of determining whether the problem one is trying to solve is one in which a good solution is located by the heuristics or one of the rarer cases in which a good solution is not found.

A possible improvement to the algorithm was considered but not implemented. Additional constraints on the relaxed version of the problem can be generated by realising that the value of a transformation cannot be more than the value of that transformation applied to the original expanded network independently of any other transformation. Hence a new set of constraints

$$b_m \leq b_m^*$$

where b_m^* is the value of the transformation in the initial network can be added to the relaxed version of the problem which may speed the convergence of the algorithm to the solution be generating tighter bounds for the tree search.

For some of the problems, the amount of computation required to locate the optimum solution was very large. If a sub-optimal

solution is acceptable then the following test can be used for early termination of the algorithm. If the best solution so far known has cost U and the lowest bound of any live node has cost L then the algorithm is within (U-L)/L of the optimum. When this parameter falls below an acceptable value then the algorithm may be terminated in the knowledge that a good sub-optimal solution has been located.

Graph	Number of Graph Nodes	Number of Tree Nodes Searched	Total Number Tree Nodes	Total Time	Time to Reach Optimum
1	5	6	10	0.10	0.08
2	5	6	10^{3}	0.11	0.10
3	5	87	10^{5}	0.53	0.27
4	5	58	10^{6}	0.44	0.15
5	8	1344	10^{25}	36	6.7
6	8	276	10^{13}	5.1	4.2
7	8	222	10^{12}	3.3	2.7
8	8	16	10^{13}	0.57	0.53
9	10	12	10^{9}	0.45	0.39
10	10	*	10^{48}	*	76
11	10	589	10^{23}	17	1.2
12	10	451	10^{23}	9.2	0.8
13	11	*	10^{51}	*	46
14	11	2526	10^{38}	107	72
15	11	*	10^{35}	*	24
16	11	116	10^{26}	38	4.9
17	12	436	10^{34}	2.3	1.8
18	12	1236	10^{51}	42	2.9
19	12	44	10^{32}	6.0	4.6
20	12	34	10^{38}	3.4	2.6

Table 4.1 Performance of Branch and Bound Algorithm.

* For these graphs, the computer job time limit was reached before the optimal solution was located. The time given is that to locate the best solution obtained.

Graph	Optimal Algorithm	Steepest Descent Algorithm	Greedy Algorithm
1	52	52	52
2	76	76	76
3	51	57	57
4	24	24	24
5	82	100	100
6	118	118	119
7	111	117	120
8	48	48	48
9	106	106	108
10	120	170	156
11	142	142	157
12	62	62	62
13	78	88	68
14	93	104	101
15	128	130	124
16	108	113	113
17	101	101	101
18	89	90	94
19	117	117	128
20	158	158	168

Table 4.2 Comparison of Costs of Networks Located by Optimal Algorithm, Steepest Descent Algorithm and Greedy Algorithm.

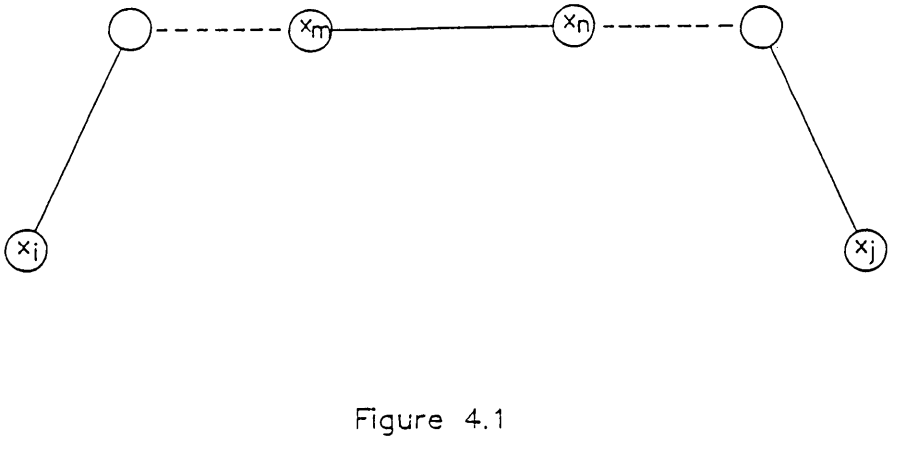

Figure 4.1

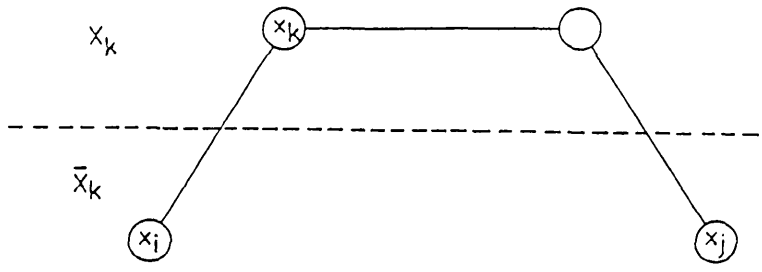

Figure 4.2(a)

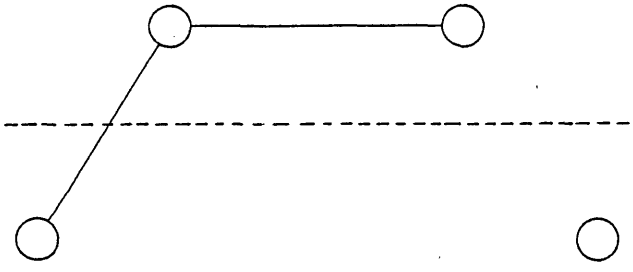

Figure 4.2(b)

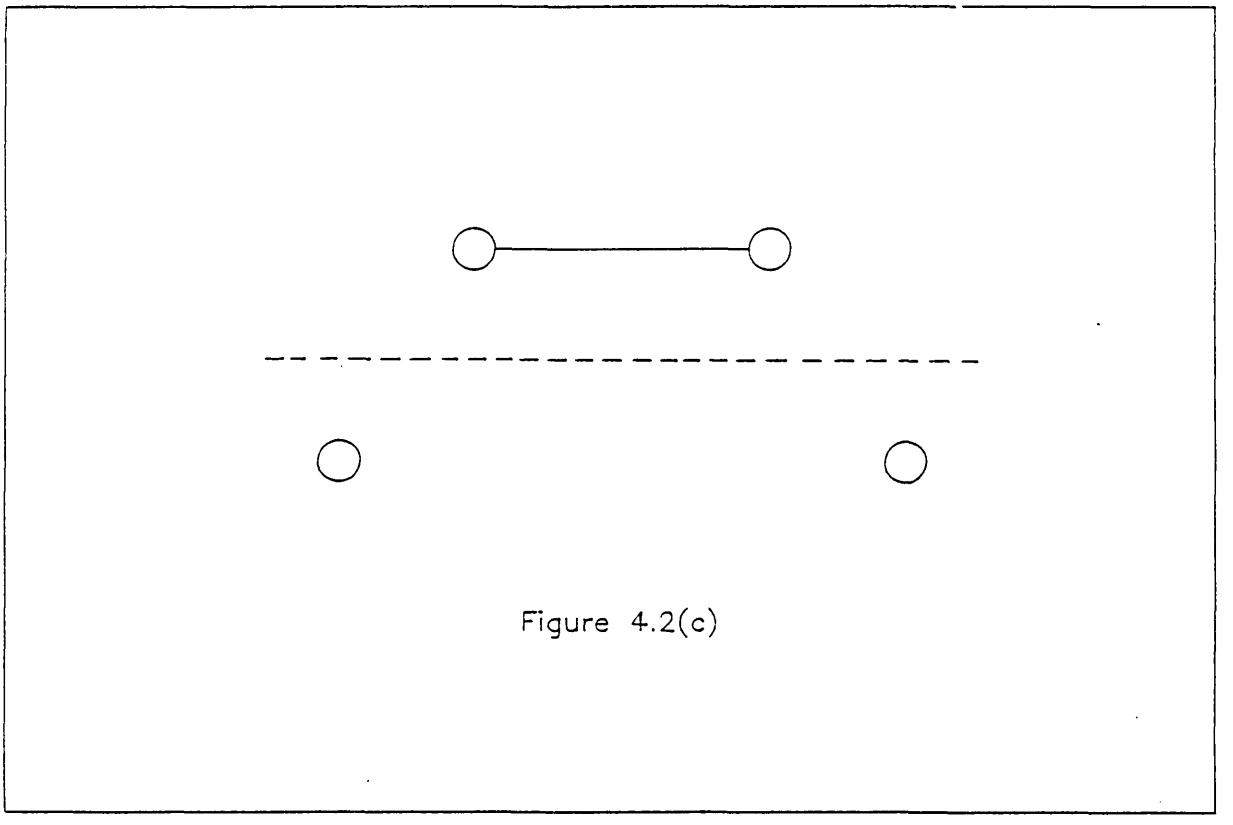

Figure 4.2(c)

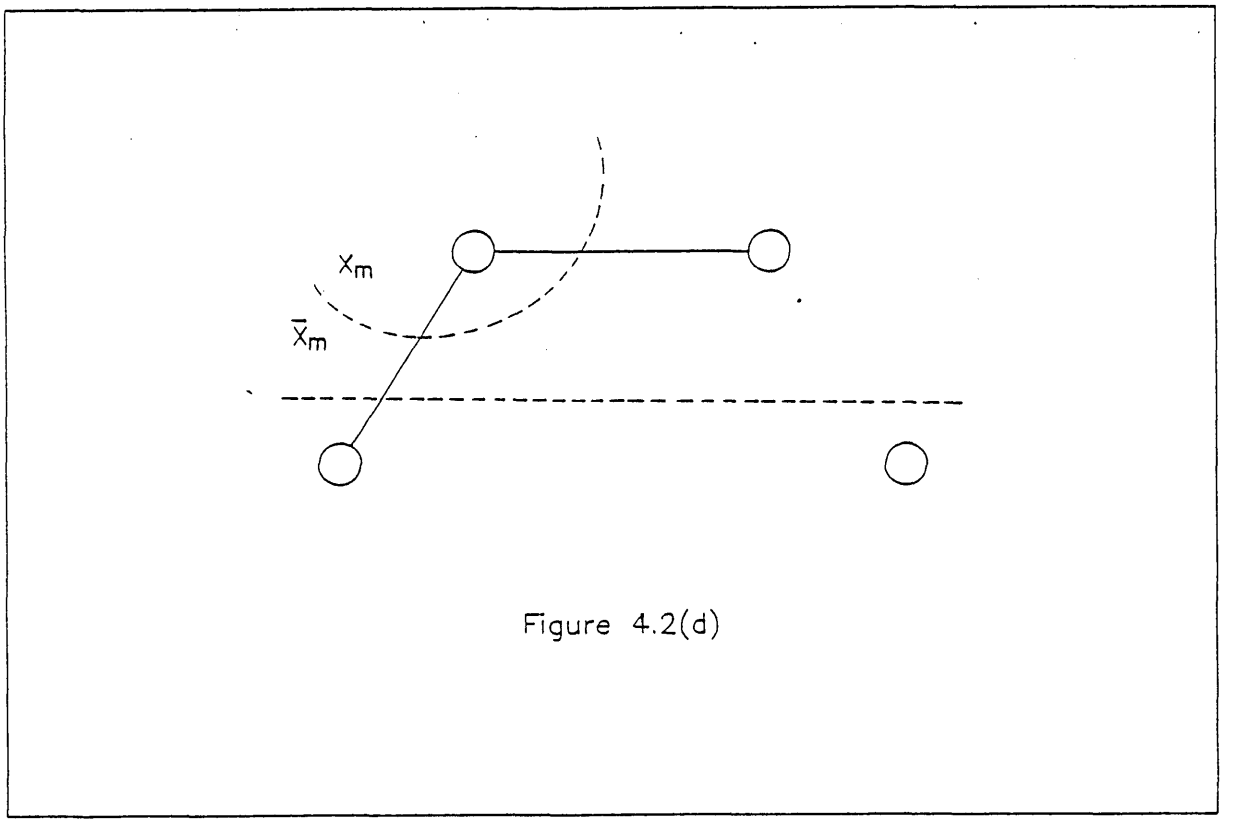

Figure 4.2(d)

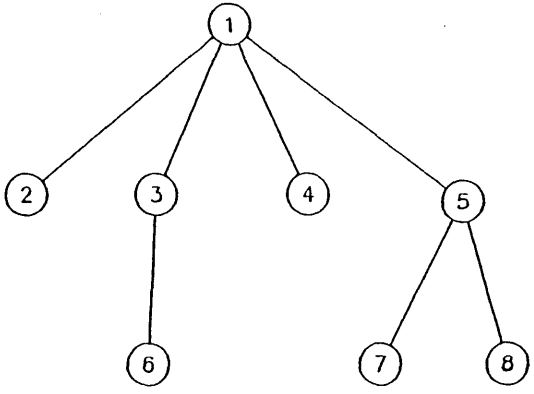

Figure 4.3

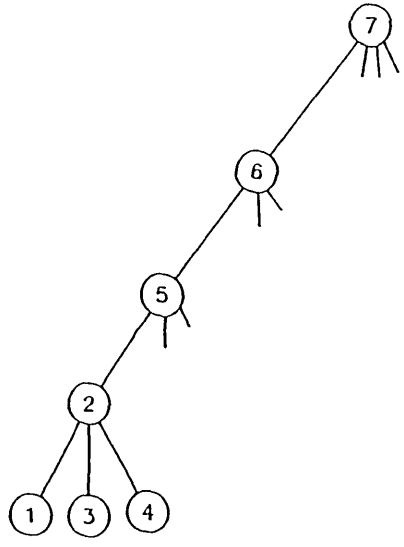

Figure 4.4

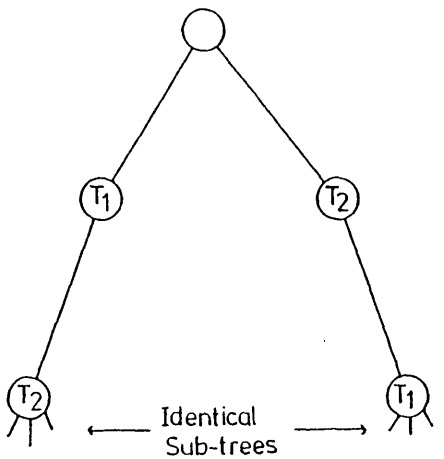

Figure 4.5

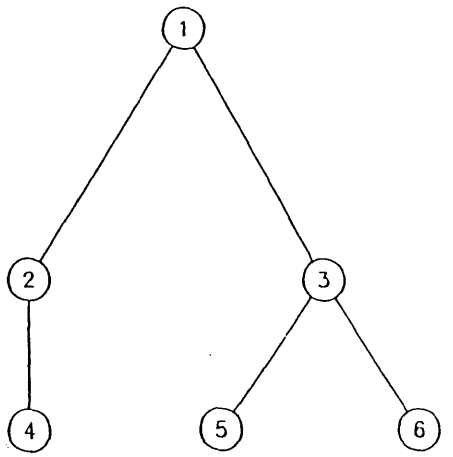

Figure 4.6

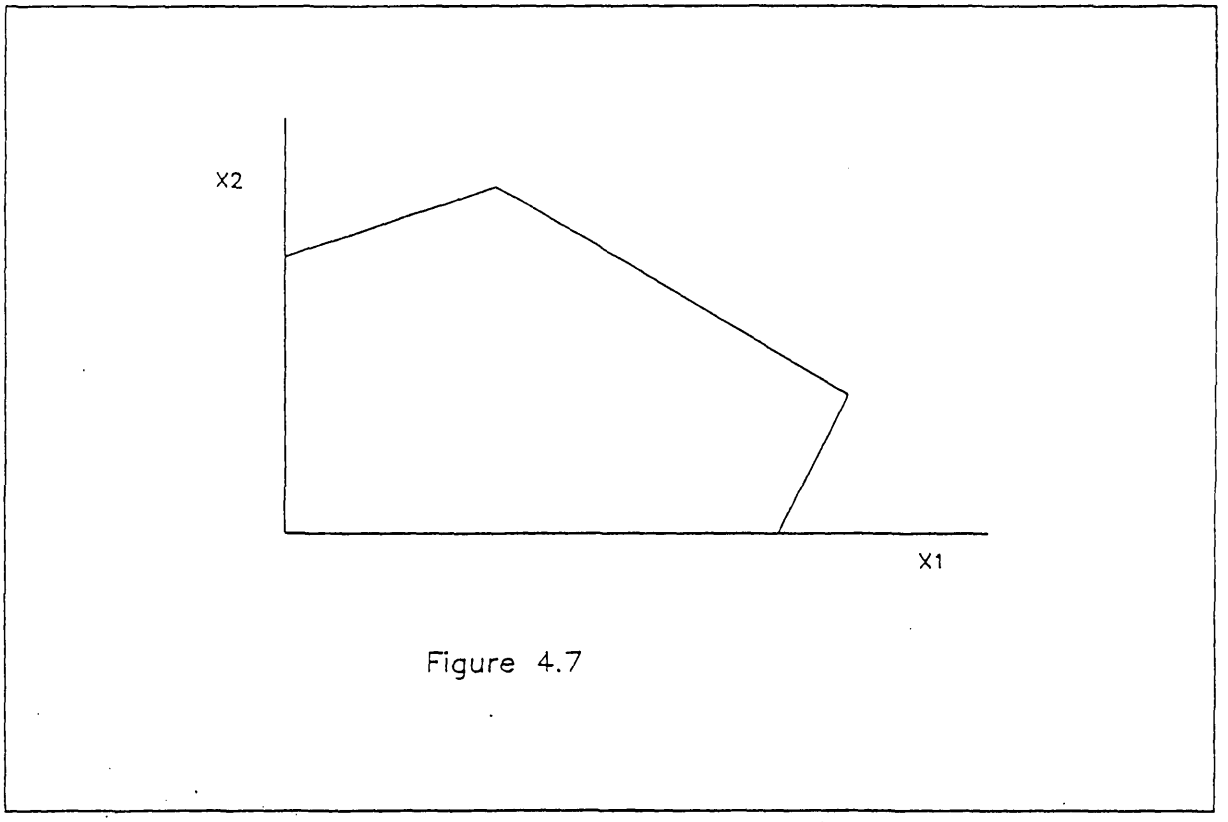

Figure 4.7

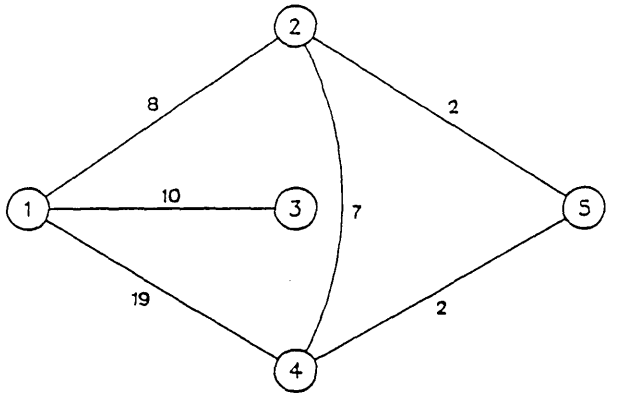

Figure 4.8(a)

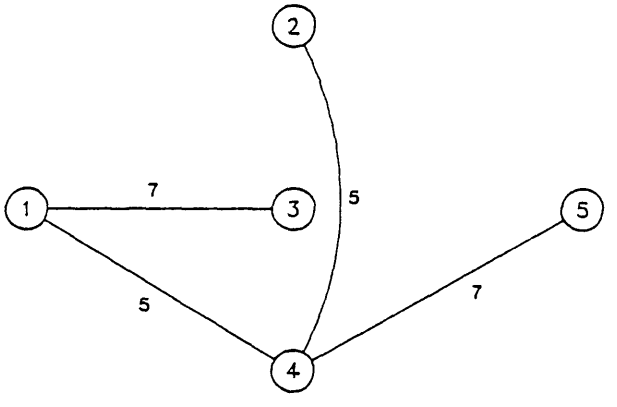

Figure 4.8(b)

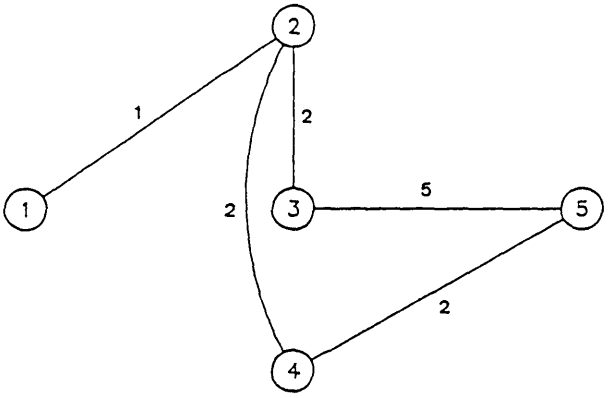

Figure 4.8(c)

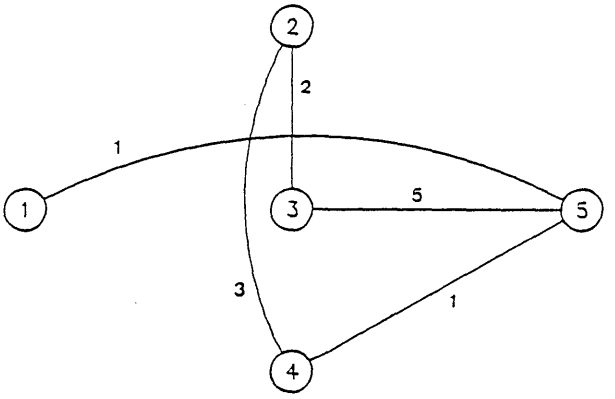

Figure 4.8(d)

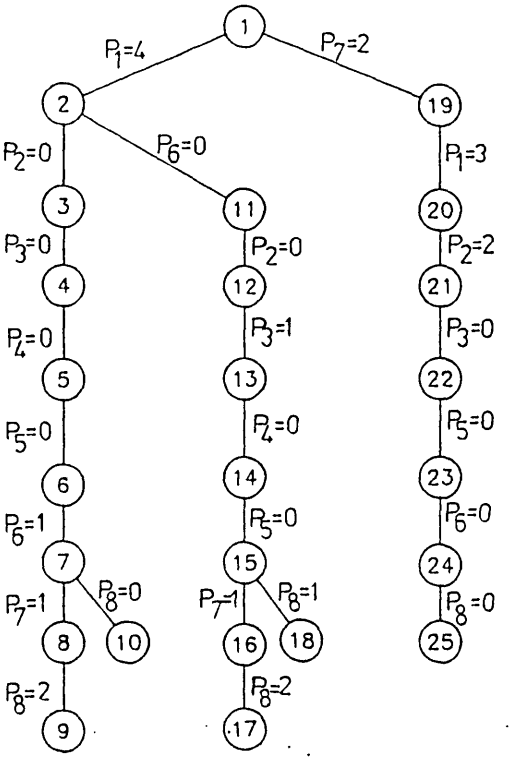
1
P1=4
P7=2
2
19
P2=0
P6=0
P1=3
3
11
20
P3=0
P2=0
P2=2
4
12
21
P4=0
P3=1
P3=0
5
13
22
P5=0
P4=0
P5=0
6
14
23
P6=1
P5=0
P6=0
7
15
24
P7=1
P8=0
P7=1
P8=1
P8=0
8
10
16
18
25
P8=2
P8=2
9
17

Figure 4.9

NODE EXPANSION OF NETWORKS

5.1 Introduction

So far, the discussions of the expansion of networks have been confined to the question of adding branches to a network so as to simultaneously increase the minimum cuts between all pairs of nodes. As was explained in Chapter 2, the physical motivation for examining this problem was from the point of view of increasing the traffic handling capacity of a network. A further extension of the concept of increasing the size and complexity of a network is to allow for the possibility that additional subscribers may wish to be connected to the network and request a certain communication capacity with each of the other subscribers to the network The problem may be posed in graph theoretic terms as the optimal addition of nodes to a network such that the minimum cuts between the new nodes and each of the other nodes in the network are greater than stipulated values. We show that this problem can be solved by a simple generalisation of the methods given in Chapters 3 and 4 of this thesis and thus give a method for tackling the general network expansion problem in which nodes and edges are simultaneously added to a network so as to meet new requirement matrix constraints.

5.2 Node Expansion of Networks

We shall now give a formal statement of the problem to be examined in this chapter.

Let $G=(X,A)$ be an undirected graph with branch capacity matrix B^n and terminal capacity matrix T^n where the superscript n denotes an n by n matrix. Let R^m be a realisable requirement terminal capacity matrix with $m>n$, such that $r^m_{i,j} \geq t^n_{i,j}$ for $i \leq n$ and $j \leq n$ and

$r^m_{i,j} \geq 0$ if $i>n$ or $j>n$. How may a graph $G'=(X',A')$ with branch capacity matrix B'^m, terminal capacity T'^m be synthesised such that $t'_{i,j} \geq r_{i,j}$ $\forall$ (i,j), $b'_{i,j} \geq b_{i,j}$ $i \leq n$ and $j \leq n$, $b'_{i,j} \geq 0$ $i>n$ or $j>n$ and a cost function is minimised. The two cost functions to be considered are total network capacity $\sum b'_{i,j}$ and total network cost $\sum c_{i,j} b'_{i,j}$.

It can be seen that this is a generalisation of the problems considered in Chapters 3 and 4 in that in earlier problems it was only edge set A which was changed to a new set A' but now also the node set X is increased to the new set X'.

As before the problem will be divided into two stages (1) generation of suboptimal network and (2) optimisation of this network for minimum cost.

5.2.1 Generation of Suboptimal Network

The generation of a suboptimal network which meets the requirement matrix constraints is an extension of the idea presented in Chapter 3 for the generation of suboptimal networks. There are essentially two parts

(1) The realisation of the flow requirements between the nodes of the set X of the original graph.

(2) The addition of extra edges to join the extra nodes in the set X'-X to nodes of the set X with arcs of sufficient capacity to meet the requirement matrix constraints.

The method of solving the first section has already been discussed in Section 3.3.2 which is to calculate the flow equivalent tree of the original network and increase the capacity of the branches of this tree until each element of the branch capacity

matrix of the modified tree is greater than or equal to the corresponding element in the requirement matrix.

We now discuss the method of extending the expansion network to join the additional nodes to the network. The new network should have the following properties.

(1) The network should be connected.

(2) The extension should not affect the values of the minimum cuts established by the previous calculations to realise the cuts between the nodes in the original network.

The simplest and most obvious network which satisfies these conditions is a spanning tree. The values of the cuts between the nodes of the original network cannot be affected because the added network being a tree has only one path between any pair of nodes and the capacities of the minimum capacity arcs between pairs of nodes of the original network have already been specified.

The positioning and capacity of the additional edges is determined as follows. Considering the requirement matrix as a distance matrix and, taking the cut tree of the original network as a starting point, edges from the requirement matrix should be added to the network in such a fashion as to maximise the length of the tree using a greedy algorithm [KR1], [PR1].

It can be seen that this longest spanning tree satisfies the requirement matrix constraints as follows. Only those cuts in which at least one of the nodes is an additional node need be considered since all other cuts have already been satisfied. Consider any cut-matrix of the requirement matrix and the corresponding cut in

the cut-tree. If the largest entry in the cut-matrix is larger than the cut in the cut-tree then this flow constraint could be satisfied by removing the edge from the tree and replacing it with an arc of larger capacity between the nodes corresponding to the element in the cut-matrix. But this would increase the length of the tree, contradicting the fact that the tree was a longest spanning tree. Thus the longest spanning tree is a suitable expansion network.

5.2.2 Minimum Capacity Network

Next to be considered is the modification of the expansion network in such a way as to minimise the total capacity of the network. It is clear that we have exactly the same problem that was presented in Chapter 3. All that is necessary is to cluster the nodes hierarchically according to the minimum cuts of the network and minimise the capacities of the resulting star-trees according to the method of Chapter 3.

Example

Consider the network shown in figure 5.1(a) which has branch capacity matrix

$$B = \begin{bmatrix} - & 5 & 11 & 1 & 0 & 0 \\ 5 & - & 14 & 0 & 0 & 0 \\ 11 & 14 & - & 5 & 5 & 0 \\ 1 & 0 & 5 & - & 0 & 5 \\ 0 & 0 & 5 & 0 & - & 0 \\ 0 & 0 & 0 & 5 & 0 & - \end{bmatrix}$$

We wish to expand this network by the addition of two nodes and edges such that it meets the requirement matrix

$$R=\begin{bmatrix} - & 26 & 40 & 40 & 22 & 6 & 22 & 20 \\ 26 & - & 26 & 26 & 22 & 6 & 22 & 20 \\ 40 & 26 & - & 42 & 22 & 6 & 22 & 20 \\ 40 & 26 & 42 & - & 22 & 6 & 22 & 20 \\ 22 & 22 & 22 & 22 & - & 6 & 25 & 20 \\ 6 & 6 & 6 & 6 & 6 & - & 6 & 6 \\ 22 & 22 & 22 & 22 & 25 & 6 & - & 20 \\ 20 & 20 & 20 & 20 & 20 & 6 & 20 & - \end{bmatrix}$$

The sub-optimal expansion which attains this requirement matrix is shown in fig 5.1(b). The branches between nodes 1 to 6 are determined as before. The capacities of the edges (7,8) and (5,7) are determined by the longest spanning tree method described in Section 5.2.1 above. When the optimisation has been carried out, the minimum capacity network which results is shown in fig 5.1(c). The terminal capacity matrix of this network is identical with the requirement matrix.

5.2.3 Minimum Cost Network

The construction of a minimum cost network from the sub-optimal expansion network can again be seen as a generalisation of the method of Chapter 4 for which the branch and bound algorithm presented there gives a solution.

With the cost matrix

$$C=\begin{bmatrix} - & 3 & 3 & 10 & 6 & 5 & 6 & 8 \\ 3 & - & 10 & 6 & 8 & 9 & 5 & 9 \\ 3 & 10 & - & 1 & 4 & 8 & 10 & 3 \\ 10 & 6 & 1 & - & 1 & 5 & 5 & 2 \\ 6 & 8 & 4 & 1 & - & 8 & 5 & 1 \\ 5 & 9 & 8 & 5 & 8 & - & 10 & 2 \\ 6 & 5 & 10 & 5 & 5 & 10 & - & 4 \\ 8 & 9 & 3 & 2 & 1 & 2 & 4 & - \end{bmatrix}$$

the minimum cost network which satsfies the requirement matrix of the previous section is shown in fig 5.1(d).

5.3 Conclusion

The ideas presented in this chapter have been simple extensions of the concepts of Chapters Three and Four where the generalisation has been so as to handle the situation in which both nodes and branches are to be added to a network to optimally increase its terminal capacity matrix. In fact, the extensions are surprisingly so slight that the term generalisation is scarcely warranted. Almost no modification to the computer programs written to implement the earlier algorithms was required to enable the more general problem to be solved. Since the performance of the algorithm was as before, no computational results have been presented in this chapter. The major contribution of this chapter was to show how to generate a sub-optimal network which would satisfy a requirement terminal capacity matrix when both edges and node are to be added.

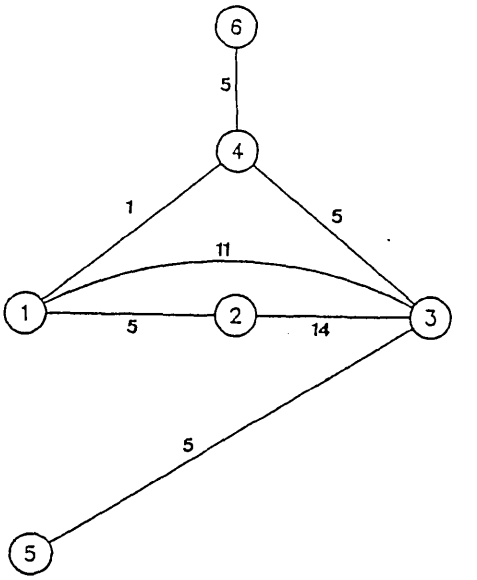

Figure 5.1(a)

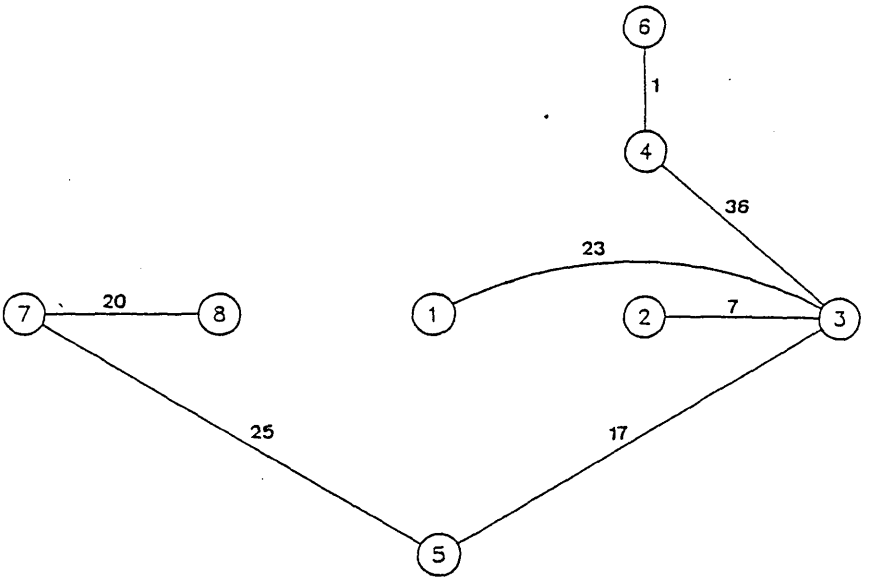

Figure 5.1(b)

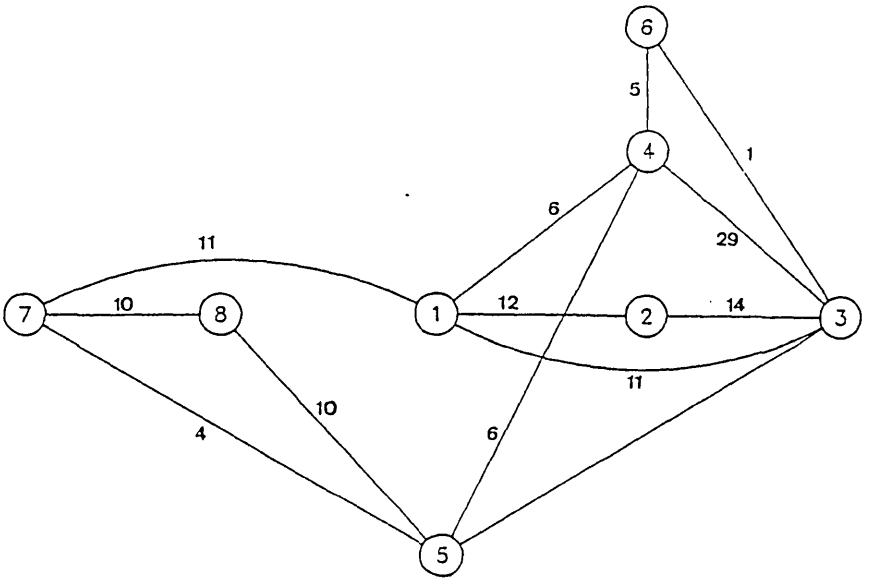

Figure 5.1(c)

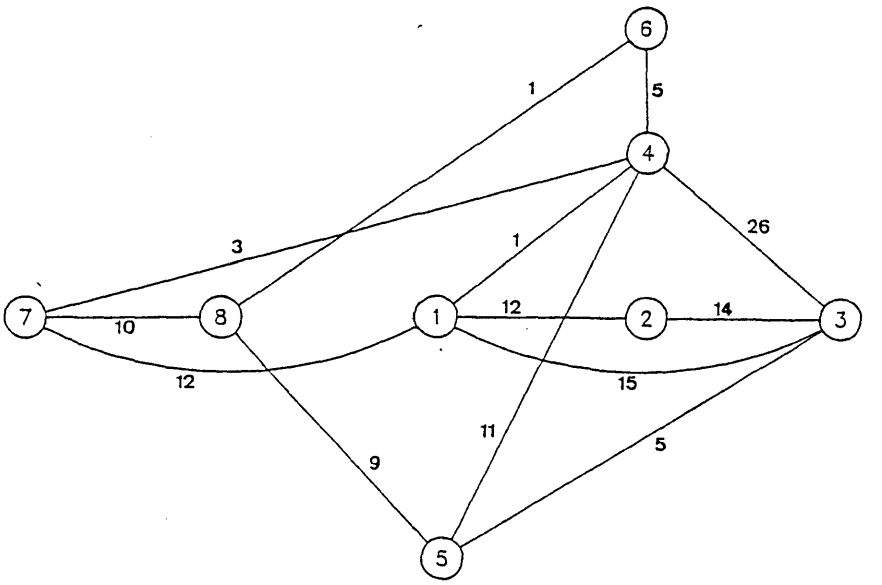

Figure 5.1(d)

CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

6.1 Conclusions

In this thesis an attempt has been made to shed some light on certain problems in network synthesis which have not been previously considered viz. the optimal addition of nodes and branches to an undirected network so as to satisfy multi-terminal flow requirements. This work has come close to completing a sequence of results in the theory of network flows begun thirty years ago. The areas that have been previously studied are the analysis of single source/sink networks, multi-terminal network analysis, multi-terminal network synthesis and single source/sink expansion.

A number of algorithms were developed to solve network expansion problems, which in some respects resemble the known methods for synthesis of undirected networks. These similarities were discussed in the conclusion to Chapter Three. Each algorithm has two distinct phases, first the synthesis of a sub-optimal network and second the optimisation of the network by branch exchanges.

The problem of synthesising the sub-optimal network is tackled through the concept of the cut-tree whereby all the information concerning the terminal capacities of the network can be compressed into a tree. This greatly reduces the dimensionality of the problem and enables an expansion network to be calculated with little difficulty. From this aspect of the work, the conditions which determine whether an expansion is realisable can be derived. This section of the algorithm requires little computational effort.

Cost minimisation of the network is through the technique of

branch exchange, but in contrast to most branch exchange algorithms, an optimal solution rather than a local minimum can be obtained. This is the most computationally expensive part of the algorithms.

Turning first to the algorithm for uniform cost network exapansion, the following general comments can be made concerning it. The computation of the optimal network is made in a well defined manner in the sense that after a known number of steps (or less) the solution is generated. This upper bound on the computational time was calculated by an analysis of the computational complexity of the algorithm which led to the result that the algorithm is of complexity $O(n^5)$. Although the algorithm is polynomially bounded in time, the high order of the polynomial leads to a rapid increase in computational time with the size of the graph which limits the size of network which can be synthesised. The inherent complexity of the problem would lead us to expect this type of result, but as has been pointed out by Papadimitriou and Steiglitz [PA1],

> "For most problems, once any polynomial-time algorithm is discovered, the degree of the polynomial quickly undergoes a series of descents as various researchers improve the idea. Usually, the final rate of growth is $O(n^3)$ or better."

Such a good result cannot be expected for this problem since a single flow calculation is $O(n^3)$ but an improvement on $O(n^5)$ may be attainable.

The algorithm for synthesising a minimum capacity network is, like all branch and bound algorithms, somewhat unpredictable in its performance, with computation times varying by orders of magnitude for graphs of the same size. As is discussed in Chapter Four, in the majority of cases the algorithm performs well, but a minority of

examples are difficult to solve. It is suggested that an examination of alternative branching rules and the inclusion of the extra constraints mentioned in the conclusion of Chapter 4 would be a useful exercise. There are no obvious reasons why some networks should prove difficult to synthesise optimally, and an investigation of this point would be interesting.

As a practical approach to the problem of synthesising networks, if a good but not optimal solution is sufficient, then it would seem to be best to apply the two heuristics approaches and choose the solution which has the lesser cost. Thus even if the network were of the type that is difficult to synthesise optimally, it is likely that a good low cost solution would be generated by one or other of the algorithms. As was seen, such a solution could be better even than the solution located by the optimal algorithm in a reasonable time. Also, towards the end of a search, the optimal algorithm can spend large amounts of time searching for a solution only slightly better than the current best known solution and so, provided that it is not essential to locate the globally optimum solution, an early termination based on the lower bounds of the live nodes may be a sensible approach.

The algorithm for the simultaneous addition of nodes and branches is an important generalisation of the above algorithms because it provides a comprehensive approach to network expansion and covers the very necessary case for the communication engineer which is the addition of extra subscribers to an existing network.

6.2 Suggestions for Further Research

As with many graph theoretic problems, there are a number of possible variations on the basic idea discussed in this thesis. We have here been concerned with the problems of synthesing undirected networks of flow with uniform branch cost and branch cost proportional to the capacity of the branch. A further type of cost function, and the most difficult to deal with, is that where the edge cost is a non-linear function of the edge capacity. The paper of Christofides and Brooker [CH3] contains an algorithm for the solution of an expansion problem with this type of cost function but for the single source/sink case. An approach along their lines would seem to be most likely to produce results in the multi-terminal case, but the method would need to be extended to examine cuts between all pairs of nodes as arcs are included and excluded from the solution network.

Another aspect of the expansion problem is the fact that all edges have been considered undirected. As can be seen from the discussion of network synthesis in Chapter Two, the problem of directed network synthesis is much more complex than undirected network synthesis; it is likewise so with network expansion. Again, a two stage approach seems promisimg- sub-optimal expansion followed by cost minimisation. The problem of sub-optimal expansion is greatly eased in the undirected case because all of the $n(n-1)/2$ elements of the requirement matrix can be represented by the $(n-1)$ edges of a tree, whereas for directed networks, all of the cuts must be considered independently. The concept of a semi-graph will probably be useful in enabling the cuts to be considered in

isolation so as to synthesise a sub-optimal network. Once this network has been synthesised, a cost minimisation procedure along the lines of those suggested in this thesis can be applied. The realisability conditions will also be complex and probably only expressible in terms of the algorithm as in the simple synthesis case.

REFERENCES

[AG1] A. K. Agarwal and S. R. Arora, Synthesis of Multiterminal Communication Nets: Finding One or All Solutions, IEEE Trans. Circuits and Systems, vol CAS-27, 141-146 (1976).

[CH1] N. Christofides, Graph Theory: An Algorithmic Approach, Academic Press, New York, 1975.

[CH2] R. T. Chein, Synthesis of a Communication Net, IBM J. Res. Develop., vol 4, 311-320 (1960).

[CH3] N. Christofides and P. Brooker, Optimal Expansion of an Existing Network, Mathematical Programming, vol 6, 197-211 (1974).

[DA1] G. B. Dantzig and D. R. Fulkerson, On the Max-Flow Min-Cut Theorem of Networks, in Linear Inequalities and Related Systems, Ann. Math. Studies, vol 38, 215-221 (1956).

[DA2] G. B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, NJ, 1963.

[DI1] E. A. Dinic, Algorithm for Solution of a Problem of Maximal Flow in a Network with Power Estimation, Soviet Math. Dokl., vol 11, 1277-1280 (1970).

[ED1] J. Edmonds and R. M. Karp, Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems, J.ACM, vol 19, 248-264 (1972).

[EL1] P. Elias, A. Feinstein and C. E. Shannon, A Note on the Maximium Flow Through a Network, IRE Trans. Inform. Theory, vol IT-2, 117-119 (1956).

[FL1] R. W. Floyd, Algorithm 97- Shortest Path, Comm. ACM, vol 5, 345 (1962).

[FO1] L. R. Ford and D. R. Fulkerson, Maximal Flow through a Network, Can. J. Math., vol 18, 399-404 (1956).

[FO2] L. R. Ford and D. R. Fulkerson, Flows in Networks, Princeton University Press, Princeton (1962).

[FR1] H. Frank and I. T. Frisch, Communication, Transmission and Transportation Networks, Addison-Wesley, Reading, Mass., 1971.

[FR2] I. T. Frisch and D. K. Sen, Algorithms for Synthesis of Orientated Communication Nets, IEEE Trans. Circuit Theory, vol CT-14, 370-379 (1967).

[FR3] H. Frank and W. Chou, Topological Optimization of Computer Networks, Proc. IEEE, vol 60, 1385-1397 (1972).

[FR4] H. Frank, R. E. Kahn and L. Kleinrock, Computer Communication Network Design: Experience with Theory and Practice, Networks, vol 2, 135-166 (1972).

[FU1] D. R. Fulkerson, Increasing the Capacity of a Network: The Parametric Budget Problem, Management Science, vol 5, 472-483 (1959).

[GO1] R. E. Gomory and T. C. Hu, Multiterminal Network Flows, J.SIAM, vol 9, 551-570 (1961).

[GO2] R. E. Gomory and T. C. Hu, An Application of Generalised Linear Programming to Network Flows, J.SIAM, vol 10, 260-283 (1962).

[GU1] R. P. Gupta, On Flows in Pseudosymmetric Networks, J.SIAM, vol 14, 215-225 (1966).

[HU1] T. C. Hu, Integer Programming and Network Flows, Addison-Wesley, Reading, Mass., 1970.

[HU2] T. C. Hu, Minimum Convex Cost Flows, Naval Research Logistics Quarterly, vol 13, 1-9, (1966).

[HU3] T. C. Hu, Optimum Communication Spanning Trees, SIAM J.

Comput., vol 3, 188-195 (1974).

[JO1] E. L. Johnson, Networks and Basic Solutions, J.ORSA, vol 14, 619-623 (1966).

[KR1] J. B. Kruskal, On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem, Proc. Amer. Math. Soc., vol 7, 48-50 (1956).

[LA1] E. L. Lawler and D. E. Wood, Branch and Bound Methods: A Survey, OR, vol 14, 699-719 (1966).

[LI1] S. Lim, Computer Solutions to the Traveling Salesman Problem, BSTJ, vol 44, no 10, 2245-69 (1965).

[MA1] W. Mayeda, Terminal and Branch Capacity Matrices of a Communication Net, IRE Trans. Circuit Theory, vol CT-7, 261-269 (1960).

[MA2] W. Mayeda, On Orientated Communication Nets, IRE Trans. Circuit Theory, vol CT-9, 261-267 (1962).

[PA1] C. H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[PR1] R. Prim, Shortest Connection Network and Some Generalisations, BSTJ, vol 36, 1389-1401 (1957).

[SE1] D. K. Sen and I. T. Frisch, Synthesis of Oriented Communication Nets, IEEE Symposium on Signal Transmission and Processing, New York, 1965, 90-101.

[ST1] K. Steiglitz, P. Weiner and D. J. Kleitman, The Design of Minimal Cost Survivable Networks, IEEE Trans Circuit Theory, vol CT-16, no 4, 455-60 (1969).

[TA1] D. T. Tang and R. T. Chien, Analysis and Synthesis of Orientated Communication Nets, IRE Trans. Circuit Theory, vol

CT-8, 39-44 (1961).

[TA2] R. E. Tarjan, Depth-first Search and Linear Graph Algorithms, J SIAM Comp, vol 1, no 2, 146-160 (1972).

[TS1] S. Toueg and K. Steiglitz, The Design of Small Diameter Networks by Local Search, IEEE Trans Comput, vol C-28, no 7, 537-42 (1979).

[WI1] O. Wing and R. T. Chein, Optimal Synthesis of a Communication Net, IRE Trans. Circuit Theory, vol CT-8, 44-49 (1961).