TWO DIMENSIONAL STOCK CUTTING AND NETWORK SYNTHESIS PROBLEMS

by

Charles Ashley Whitlock

Imperial College of Science and Technology

Thesis submitted to the University of London for the award of Doctor of Philosophy degree

March, 1979

ABSTRACT

In this thesis two operations research problems are discussed. These are stock cutting problems and network synthesis problems.

(i) Stock Cutting Problems: A classification for all types of stock cutting problems is given in conjunction with a literature survey. This classification generates some problems which have not been solved in the literature. One such problem is the constrained two dimensional stock cutting problem. In this problem one stock rectangle is to be cut into a number of smaller rectangular pieces each with an associated value and an upper bound on the number of that type of piece that can be produced. The objective is to maximise the value of the pieces cut whilst satisfying the constraints. A tree search algorithm for the solution of this problem is described and computational results given. Problems with up to 20 different types of piece to be cut have been solved.

(ii) Network Synthesis Problems: Network synthesis problems are problems in which it is required to synthesize a least cost graph which has certain specified connectivity properties. In the thesis a survey of network synthesis problems is given. Many different types of synthesis problems are discussed, some of which have not been well solved in the literature. A particular problem for graphs with non-uniform arc costs is one in which a uniform arc connectivity is required. This problem is discussed in detail and a number of different tree-search methods developed for its solution. These methods use lagrangean relaxation, matching and linear programming to obtain good bounds to be embedded in the tree search. Problems with up to 50 vertices have been solved.

The travelling salesman problem (TSP) is a member of the class of network synthesis problems. A linear programming based tree search algorithm is developed for the TSP. This uses network flow to reduce the number of constraints involved, and also incorporates variable reduction techniques. Problems with up to 70 vertices have been solved.

ACKNOWLEDGEMENTS

I should like to express my grateful thanks to my supervisor, Dr. Nicos Christofides, of Imperial College for his help and encouragement over the preparation of this thesis. I should also like to thank my many friends amongst the staff and students of the Department of Management Science, Imperial College, who have assisted me. The thesis has been expertly typed by Anne Usher, Susan Hayden and Glenda Nickel of the London School of Economics and Political Science. Finally, I should like to thank the Science Research Council who provided the finance for this research.

LIST OF CONTENTS

	Page No.
Chapter 1 - Introduction	13
1.1 Contents of thesis	14
1.2 Survey of Stock Cutting Problems	14
1.3 Stock Cutting Algorithms	14
1.4 Network Synthesis Survey	15
1.5 Network Synthesis Algorithms	15
1.6 The Travelling Salesman Problem	16
Chapter 2 - Survey of Stock Cutting Problems	17
1 Introduction	18
1.1 Notation	19
2 Classification of Stock Cutting Problems	20
2.1 The Problem Type Parameter P	21
2.2 The Dimension Parameter D	21
2.3 The Stock Size Parameter S	21
2.4 The Constraint Parameter CON	22
3 Summary of Problems in the Literature	22
3.1 Classification	22
3.2 $(B,1,K^1,-)$	23
3.3 $(B,1,K^1,LE)$	23
3.4 (A,1,K,GE)	23
3.5 (C,1,R,GE)	23
3.6 $(B,2,K^1,-)$	24
3.7 $(B,2,K^1,LE)$	24
3.8 $(C,2,P^1,EQ)$	24
3.9 (A,2,K,GE)	25
3.10 (C,2,R,GE)	25
4 Literature Survey	26

		Page No.
4.1	(B,1,K^1,-) The One-Dimensional Knapsack Problem	26
4.2	(B,1,K^1,LE) The Constrained One-Dimensional Knapsack Problem	29
4.3	(A,1,K,GE) The One-Dimensional Stock Cutting Problem	29
4.3.1	The Linear Programming Approach	29
4.4	(C,1,R,GE) The One-Dimensional Assortment Problem	35
4.5	(B,2,K^1,-) The Two-Dimensional Knapsack Problem	37
4.6	(C,2,P^1,GE) The Two-Dimensional Assortment Problem with a Continuous Length of Stock	49
4.7	(A,2,K,GE) The Two-Dimensional Trim Problem	50
4.8	(C,2,R,GE) The Two-Dimensional Assortment Problem	51
	References	53
Chapter 3 - Stock Cutting Algorithm		56
1	Introduction	57
2	Problem	59
3	Enumerative Procedure	60
3.1	Effects of Symmetry	61
3.2	Effects of Cut Ordering	63
3.3	Normal Cuts	64
3.4	Description of Enumerative Algorithm	69
4	Tree-Search Algorithm	71
4.1	Upper Bound on Value of Cutting Pattern at Node	71
4.2	Branching Strategies	75
5	Computational Results	76
	References	81
	Appendix	83

		Page No.
Chapter 4 - Survey of Vulnerability		85
1	Introduction	86
1.1	Graph Theory Basic Definitions	86
1.2	Measures of Vulnerability - Basic Definitions	88
1.3	Analysis and Synthesis of Vulnerability Problems in Graphs	90
1.4	Properties of Vulnerability Measures	91
2	Methods for Determining Vulnerability in Graphs	96
2.1	Arc Vulnerability	96
2.2	Vertex Vulnerability	97
2.3	Computation of ω	99
3	Synthesis of Invulnerable Networks	100
3.1	Uniform Cost and Uniform Arc Connectivity	102
	3.1.1 G is not Complete	102
	3.1.2 G is complete	103
3.2	Uniform Cost and Uniform Vertex Connectivity	105
	3.2.1 Nondirected Graphs	105
	3.2.2 Directed Graphs	108
	3.2.3 Bipartite Graphs	110
	3.2.4 Diameters and Optimal Graphs	111
3.3	Problem P with Uniform Costs and Non-unfiorm Arc Connectivity	114
3.4	Problem P with Uniform Cost and Non-uniform Vertex Connectivity	121
4	Problems with Non-uniform Arc Cost	121
4.1	Uniform Arc Connectivity	122
4.2	Non-uniform Vertex Connectivity	123
	Appendix	128
	References	134

		Page No.
Chapter 5 - Network Synthesis Algorithms		138
1	Introduction	139
1.1	The Problems	139
1.2	Analysis of the Optimal Solution to Problem P^u	142
2	Branch and Bound Algorithm A for the Solution of S^u	148
2.1	The Calculation of Lower Bounds on S^u	149
2.2	Iterative Procedure to Improve the Lower Bound	151
2.3	Branching Procedure	151
2.4	Selection to the Set A'_M of Arcs to Branch on	153
3	Branch and Bound Algorithm B for the Solution of Q^u	154
3.1	Method for Finding if a Graph Contains u/2 Disjoint Hamiltonian Circuits Using Implicit Enumeration	156
3.2	Method of Constructing the Graph of Hamiltonian Circuits	157
3.3	Inclusion of a Disjoint Hamiltonian Circuits Procedure (H or I) Into a Branch and Bound Algorithm B	160
4	Algorithm C for Solving P^u and S^u	161
4.1	Problem Formulation	161
4.2	Solution of the LP Relaxation of P^u	163
	4.2.1 Procedure L for Solving P^u	165
4.3	Tree Search Algorithm	167
	4.3.1 Enumeration Tree	167
5	Computational Results for Algorithms A and B	169
5.1	Matching Code	169
	5.1.1 Results	170
5.2	Computational Results for Algorithm C	171

		Page No.
	References	176
Chapter 6 -	The Travelling Salesman Problem	177
1	Introduction	178
1.1	Problem Formulation	178
2	Brief Literature Survey	181
2.1	All Integer Tree Search Methods	181
	2.1.1 Branching Rules for the TSP	181
	2.1.2 Bounding Procedures for the TSP	182
2.2	Linear Programming Based Methods	189
3	Method P for Solving the Travelling Salesman Problem	191
3.1	Variable Reduction Techniques	192
3.2	Use of the Shortest-Spanning Tree Algorithm for Variable Reduction	193
3.3	Dynamic Variable Reduction	195
3.4	Algorithm D	198
4	Computational Results	199
4.1	Bound on the TSP	199
4.2	Tests on Algorithm D	200
	Appendix	205
	References	211
Chapter 7 -	Conclusions	214

LIST OF ILLUSTRATIONS

Figure No.		Page No.
	Chapter 2	
1(a)	Non-Guillotine Cut	38
1(b)	Guillotine Cut	38
2	Staged Cutting with 3 Stages	38
3	Feasible Cutting Pattern for a Rectangle (x,y)	39
4	Figure Illustrating the Use of Functions $\ell(x,y)$ and $w(x,y)$	42
5	Tree Generated from Figure 4 using ℓ and w	43
6	Illustration of Where $F^*(x,y) = F(x,y)$	46
7	Two Stage Cutting	48
	Chapter 3	
1(a)	Cutting Pattern Infeasible with Guillotine Cuts	58
1(b)	Cutting Pattern Made with Guillotine Cuts	58
2	Enumeration of all Possible Cutting Patterns for 3×2 Rectangle	62
3	Illustration of the Effects of Cut Ordering	65
4(a)	Non-normalised Cutting Pattern	66
4(b)	Normalised Form of Cutting Pattern	67
5	Flowchart of the Algorithm	72
6	Optimal Cutting Pattern for Problem 2	80
	Chapter 4	
1	Diagram for Proof of Theorem 2	92
2	Graph Containing Two Disjoint Hamiltonian Circuits with $\theta=4$ and w=2	94
3	Transformation for Finding ω_{st}	98

Figure No.		Page No.
4	Graph Containg Two Disjoint Hamiltonian Circuits such that the Removal of One Particular Circuit Leaves the Graph with no other Hamiltonian Circuit	104
5	An Optimal Damage Resistant Graph for u=4 and n=10	106
6	An Optimal Damage Resistant Graph for u=3 and n=8	107
7	An Optimal Graph for u=3 and n=7	108
8	An Optimal Directed Graph for u=2 and n=10	108
9(a)	An Optimal Directed Graph for u=3 and n=8	109
9(b)	An Optimal Directed Graph for u=3 and n=7	109
10	An Optimal Bipartite Graph for u=3 and n=10	110
11	Friedman's Construction	112
12	Korn's Construction	113
13(a)	First Stage of Degree Algorithm	117
13(b)	First Arcs Added by Degree Algorithm	117
13(c)	Arcs Incident on x_3 Added to Graph	118
13(d)	Arcs Incident on x_4 Added to Graph	118
13(e)	Final Graph Produced by the Degree Algorithm	119
14	The Optimal Solution to Problem P with $\{u_i^{max}\} = \{5,5,2,2,2,2,1,1\}$	121
15(a)	Cost Matrix for the Graphs in Figures 15(b) and 15(c)	124
15(b)	The Optimum Solution for Problem P with u=4 over the Cost Matrix in Figure 15(a)	125
15(c)	The Least Cost Pair of Disjoint Hamiltonian Circuits for the Cost Matrix in Figure 15(a)	125
16	A 4-Regular 4-Connected Graph Having no Hamiltonian Circuits	126

Figure No.		Page No.
17(a)	Initial Flow for ω_{st} Labelling Algorithm	131
17(b)	The Graph after Some Labels have been Added	131
17(c)	Labels before Augmentation Begins	132
17(d)	Flow Pattern at Termination of the Algorithm	133
	Chapter 5	
1	Relationships Between Problems P^u, M^u, S^u and Q^u	142
2(a)	Cost Matrix for the Graphs in Figures 2(b) and 2(c)	145
2(b)	The Optimum Solution for Problem P^4 over the Cost Matrix in Figure 2(a)	146
2(c)	The Optimum Solution for Problems Q^4 and S^4 over the Cost Matrix in Figure 2(a)	146
3	A Graph which is an Optimum Solution to S^4 but not to Q^4	147
4	Structure of the Solution Graph G_p to Problem P^u	148
5	Flowchart for Algorithm A	155
6	Illustration of the Tree Generated by Procedure H for u=8	158
7	Graph G^H Produced by Procedure I	159
8	Illustration of Transformation T	170
	Chapter 6	
1	Illustration of How to Calculate Reduced Costs for the SST	194
2(a)	Graph for Non-Symmetric Random Problems	201
2(b)	Graph for Symmetric Random Problems	201
2(c)	Graph for Euclidean Problems	205
3(a)	The O-graph G_0	209
3(b)	The Graph G_0^v	209

LIST OF TABLES

Table No.		Page No.
	Chapter 3	
1	Performance of Algorithm	78
2	Details of Three Test Problems	79
	Chapter 5	
1	Results for Randomly Generated Problems Using Algorithms A and B	173
2	Results for Algorithm C	174 & 175
	Chapter 6	
1(a)	Performance of Algorithm D on Random Symmetric Cost Matrices	204
1(b)	Performance of Algorithm D on Euclidean Cost Matrices	204

CHAPTER 1

INTRODUCTION

Introduction

1.1 Contents of the Thesis

In this thesis two problems in the area of operations research are discussed. These are stock cutting problems and network synthesis problems. For each class of problem a literature survey and classification is given, and new algorithms are described for specific problems within the two classes. It is shown that the travelling salesman problem (TSP) is a member of the class of network synthesis problems and a new algorithm is given for the TSP which is a particularisation of the network synthesis algorithms. The computational performance for the algorithms presented is investigated. A brief description of the contents of each chapter follows.

1.2 Survey of Stock Cutting Problems

Chapter 2 contains a classification and survey of stock cutting problems. The general types of stock cutting problem are defined. A set of four parameters is then introduced which generate a classification of stock cutting problems. The various types of stock cutting problem are defined in terms of these parameters, and previous work done on these problems is described. This classification generates a number of problems which have not been previously described in the literature.

1.3 Stock Cutting Algorithm

In Chapter 3 a tree search algorithm is described for the solution of one of the practically most important stock cutting problems classified in Chapter 2. This is a problem in which rectangular pieces are to be cut from a larger stock rectangle so as to maximise the value of the pieces obtained (or minimise the wastage), subject to constraints on the allowable number of rectangular pieces to be cut.

This algorithm implicitly generates all possible cutting patterns for the stock rectangle. It contains a bound calculation on the value of each cutting pattern that uses dynamic programming combined with a transportation algorithm. The size of the tree is limited by imposing symmetry and other conditions on the cutting patterns generated. Computational results for this procedure are given which show that problems with up to 20 different types of pieces to be cut can be solved.

1.4 Network Synthesis Survey

Chapter 4 contains a survey and classification of network synthesis problems. The problems considered in this chapter are concerned with synthesising a graph of minimum cost that conforms to certain specified deterministic connectivity conditions.

Methods for the analysis of connectivity are surveyed, and some algorithms described. The class of network synthesis problems is classified. This is done by defining the general connectivity synthesis problem. Parameters associated with this problem are identified that - when varied - generate the class of problems. Methods from the literature that have been developed to solve some of these problems are described. The classification also generates some problems which have not been previously considered.

1.5 Network Synthesis Algorithms

In Chapter 5 some algorithms are given for connectivity synthesis problems which have not been solved optimally before. These problems are discussed in detail and the possible structures of their solutions are given.

The methods described to solve these problems are tree search algorithms. These divide into two types according to which

bounding procedure they are based on. The first type of algorithm is based on the use of the solution to a u-matching problem as a bound. This is combined to a Lagrangean relaxation and ascent procedure which produces an effective bound. The second type of algorithm is based on the use of linear programming as a bound. The synthesis problem can be formulated as an integer program with a very large number of constraints. A technique is described which reduces the number of constraints so that the LP relaxation of this integer program is solvable. This is based on the use of network flow techniques. Computational results for both types of algorithm are given, problems up to 50 vertices are solved.

1.6 The Travelling Salesman Problem

In Chapter 6 a method for the solution of the travelling salesman problem (TSP) is described. The TSP is a member of the class of network synthesis problems and the algorithms produced for these problems can be particularised for the TSP. A brief survey of optimal methods for the TSP is given first.

The method described in this chapter is a tree search based on the use of LP as a bound. The LP relaxation of the TSP contains a very large number of constraints and a technique similar to that in Chapter 5 - based on network flow - is used to reduce the number of these constraints. In addition a technique is developed for reducing the number of variables in the LP. This is based on the use of reduced costs obtained from the solution to the assignment problem after a Lagrangean ascent has been performed to further improve the assignment problem bound. These reduced costs are used in conjunction with a good upper bound to remove a priori a large number of variables from the formulation. Computational results for this procedure are given, problems up to 70 vertices being solved.

CHAPTER 2

SURVEY OF STOCK CUTTING PROBLEMS

1. INTRODUCTION

The problem of cutting a set S_o of one (or more) dimensional stock objects into a set S_p of smaller one (or more) dimensional pieces of specified dimensions in such a way as to minimise wastage or maximise the value of the pieces cut is called the stock cutting problem. This problem appears in two forms:

Either:

Problem A: All pieces in S_p can be produced from the set S_o of objects to be cut, in which case the objective is to produce all the pieces in S_p and to minimise "wastage". In this case "wastage" is defined as the total value of those objects of S_o that are cut to produce S_p minus the total value of the pieces in S_p.

or

Problem B: Not all pieces in S_p can be cut from S_o in which case the objective is to maximise the value of pieces produced from S_o, or to minimise "wastage". In this case "wastage" is defined as the total value of the objects in S_o minus the total value of the pieces that are produced.

Another closely related problem is as follows:

Problem C: Given a set of smaller pieces to be produced by cutting, find the optimum number and sizes of the stock objects from a given set of possible stock sizes so as to minimise wastage. This is known in the literature as the assortment problem.

These problems have been examined by a number of authors [1, 3, 5-34] and methods for their exact solution have been developed

using dynamic programming [e.g. 13, 19, 33] and tree search techniques [3, 9, 14, 18, 20]. In addition L.P. as well as many heuristics have been developed to give approximate solutions [e.g. 11, 12, 21, 22, 23, 25, 31]. This attention is motivated by the many practical problems that can be formulated as stock cutting problems. For example, the steel bar cutting stock problem [6], the problem of cutting rolls of paper or metal sheeting into smaller rolls [31], the division of work into time shifts, the optimal partitioning of computer storage space for a given set of programs (batch) to be run in minimum time; the packing problem in which a set of objects has to be packed into the smallest space, are all examples of special cases of the stock cutting problem.

In this chapter the various types of stock cutting problem will be described and classified. Some of the methods developed for their solution will be briefly surveyed. For the 2 dimensional stock cutting problem, new methods are described and discussed in detail in the next chapter.

1.1 Notation

In this section we shall describe the notation to be used in this and the next chapter.

In the survey the terms 'lengths' (for the one-dimensional case), rectangles (for the two-dimensional case) and 'objects' (in general) will be used to refer to the stock in a stock cutting problem and also anything produced by cutting up the stock, whereas the general term 'pieces' (it will be clear from the text whether the 'pieces' are one or two-dimensional) will be used to refer to the items that are to be produced from or fitted into th stock objects. The total number of stock objects is n and their dimensions are $L_i, i=1,\ldots,n$ (for the one-dimensional case) and $(L_i, W_i), i=1,\ldots,n$ (for the two-dimensional case).

The total number of pieces to be cut is m and their dimensions are $\ell_j, j=1,\ldots,m$ (for the one-dimensional case) and (ℓ_j, w_j), $j = 1,\ldots,m$ (for the two-dimensional case). The value of each piece to be cut is $v_j, j=1,\ldots,m$, these values may or may not be proportional to the dimensions of the pieces, and the constraint (if any) associated with each piece is b_j, $j = 1,\ldots,m$.

A 'cutting pattern' is some method for dividing up the stock objects so that each object produced by the division has either nothing fitted into it, in which case the value of that object is zero, or one only of the pieces k (say) fitted into it, in which case the value of that object is v_k. This value is not necessarily unique as there may be other types of pieces which can fit into the object produced by division. Thus a cutting pattern is uniquely defined by:

(i) The way in which the stock is divided up.

(ii) The choice of pieces to fit the objects produced by the division.

The value of a cutting pattern is equal to the total value of the objects produced by the division.

The objective of the stock cutting and assortment problems is to find the cutting pattern of maximum value that satisfies the constraints (if any). For the stock cutting problem the constraints are the demands on the pieces to be cut. For the assortment problem the constraints are the demands and also constraints on the stock objects (i.e. maximum size, maximum number of sizes, etc.).

2. CLASSIFICATION OF STOCK CUTTING PROBLEMS

In order to classify all types of stock cutting problems it is necessary to define a set of parameters, such that each set of states for these parameters uniquely describes a particular stock cutting

problem. It is sufficient to use the following four parameters to describe all possible stock cutting problems:

2.1 The Problem Type Parameter P

In the introduction the three different types of stock cutting problems were defined. It is necessary to include in the classification of stock cutting problems a parameter P (say) to define the problem type. The parameter P can take three values A, B or C thus describing whether the problem being considered is Problem A, Problem B or Problem C respectively.

If P equals A or B then the problem is a pure stock cutting problem. If P equals C then the problem is an assortment problem.

2.2 The Dimension Parameter D

The second parameter to consider is the dimension parameter D (say). One, two and three dimensional problems have been considered in the literature. However, the greater the dimension the more complex the problem. Very little work has been done on two and almost none at all on three dimensional problems [12]. In this survey one and two dimensional problems are considered, hence D can take two values 1 or 2. If D equals 1 the problem is one-dimensional. If D equals 2 the problem is two-dimensional.

2.3 The Stock Size Parameter S

In normal stock cutting problems the size of the stock to be cut is known. However, for the assortment problem the sizes of the stock objects is either not fixed, and has to be chosen from some restricted set of possible stock sizes, or partially known. For example in the two-dimensional case where stock come in the form of a number of continuous lengths of material of known width, the stock size is only partially known.

From the above it is necessary to define three values for the

stock size parameter S, these are K, R, and P. For S equals K the stock size is known, for S equals R the stock sizes are not fixed but are to be chosen from some restricted set of stock sizes, and for S equals P the stock size is partially known. It is convenient to describe two additional values of S, these are K^1 and P^1. For S equals K^1, the stock size is known and there is only one stock object. Similarly for S equals P^1, the stock size is partially known and there is only one stock object.

2.4 The Constraint Parameter CON (on the pieces to be cut)

In problems of type A and C the various stock sizes are considered to be available in unlimited supply, and it is desired to find the optimal cutting pattern that will at least satisfy demand. The constraints for these problems are of the greater than or equal to type (GE). For the problems of type B, the stock to be cut is considered to be a limited resource, and it is desired to at most satisfy demand, since not all the demands for all the pieces can be satisfied. Hence the constraints for this type of problem are normally of the less than or equal to type (LE). It is also possible to formulate problems of type B in which there is no demand (-) and it is simply required to optimum cutting pattern for the limited stocks available , or in which demand must be exactly satisfied (EQ).

From the above the constraint parameter CON has four possible values GE, LE,- and EQ.If CON equals GE the constraints are of the greater than or equal to type. If CON equals LE the constraints are of the less than or equal to type. If CON equals - the problem is unconstrainted, and if CON equals EQ the problem is equality constrained.

3. SUMMARY OF PROBLEMS IN THE LITERATURE

3.1 Classification

In the previous sections the parameters that are used to classify

stock cutting problems have been defined. All stock cutting problems can be classified using the following quad-tuple:

(P, D, S, CON).

where;

P can take the values A , B , C ,

D can take the values 1 , 2 (problems with D>2 are not discussed in this survey),

S can take the values K, K^1, R, P, P^1

and CON can take the values GE , LE , EQ, - .

Using this classification it is now possible to define some of the stock cutting problems that exist in the literature.

3.2 (B , 1 , K^1, -)

This is the simplest form of stock cutting problem, in the literature mainly problems with one length of stock to be cut have been considered. There are no constraints on the numbers of each type of piece that is to be produced. This type of problem is often called the 'one-dimensional knapsack problem' [13].

3.3 (B , 1 , K^1, LE)

This problem is similar to the (B, 1, K, -) problem except that there are specified demands on the maximum numbers of each type of piece that is to be produced, this is often called the 'contrained one-dimensional knapsack problem'.

3.4 (A , 1 , K , GE)

This is the standard type of one-dimensional stock cutting problem for which there is an unlimited supply of n sizes of lengths of stock with specified minimum demands on the numbers of each type piece to be cut. This is often called a trim problem.

3.5 (C , 1 , R , GE)

This is the standard form of the one-dimensional assortment problem,

in which the minimum demands on the pieces to be cut are known and it is required to find the least amount of stock to hold in order to minimise wastage. In this type of problem the number of different stock sizes is given or if not given solutions are required for a range of the number of stock sizes. Clearly an optimal solution without such constraints would be to stock all the pieces that are required to be cut in the amounts that they are demanded and hence require no cutting at all.

3.6 (B , 2 , K^1, -)

This is the simplest form of two-dimensional stock cutting problem, the two dimensional equivalent of the one-dimensional knapsack problem, where there is no specified demand on each piece to be cut. In the literature mainly problems with one stock rectangle have been considered. In this thesis these problems will be called 'two-dimensional knapsack problems', thus using this term in a way similar to Gilmore and Gomory [13]. There is however some ambiguity in the literature, as the term 'multi-dimensional knapsack problem' has been used to describe one-dimensional knapsack problems in which the stock object and pieces to be cut have additional characteristics (e.g. weight, volume, etc.) which have to be taken into account.

3.7 (B , 2 , K^1, LE)

This is the constrained version of the problem (B, 2, K, -), very little work has been done on this problem in the literature, and no optimal methods have been developed for the problem. It is considered in greater detail in the next chapter. This problem has many practical applications to the cutting of plastic and steel sheets and the cutting of wood.

3.8 (C , 2 , P^1, EQ)

In this problem the stock consists of one long roll of material

of fixed width and it is necessary to find the shortest lengths of material to meet minimum demand and minimise wastage. The problem has applications to the scheduling of a batch of jobs on a computer where the fixed dimension is one resource (the size of the core store) and the dimension to be minimised is the other resource (time). There are a number of programs to be run each with various core and time demands and it is necessary to find the best way of sequencing the jobs in order to maximise the utilisation the computer resources. This is often called the 'packing problem'. In this type of problem the demands on the pieces to be cut are not usually lower bounds on the number of each piece that is to be produced, but must be satisfied with equality.

3.9 (A , 2 , K , GE)

This is the general two-dimensional trim problem, where there are n stock rectangles and a minimum demand on all the pieces to be cut. Problems of this type are too complicated to solve optimally and only certain subproblems with extra constraints have been considered in the literature by using heuristics. For example there may be a restriction on the number of cuts that can be made on any one piece [13, 17].

3.10 (C , 2 , R , GE)

This is the general form of the two-dimensional assortment problem with unknown numbers of stock objects to be chosen from a given set, as in the one-dimensional case the problem is normally formulated with some additional constraints in order to limit the number of distinct stock sizes.

The rest of this chapter will survey the approaches in the literature to the problems classifed above. The survey will describe in greater detail some of the approaches to the simpler two-dimensional problems, i.e. (B, 2, K, -), as these are of greater relevance

to the next chapter in which an optimal method is described for the (B, 2, K^1, LE) problem with restrictions on the type of cuts that can be used.

4. LITERATURE SURVEY

We shall now consider some of the above defined problems in greater detail:

4.1 (B , 1 , K^1, -) The One-Dimensional Knapsack Problem

The one-dimensional knapsack problem has been studied by a great many authors, it is a problem of considerable importance to mathematical programming. For a comprehensive survey on knapsack problems the reader is referred to Toth [30], and Salkin and DeKluyver [27]. In view of the excellent existing surveys on this problem we will not survey the problem here any further, but will instead consider only its dynamic programming formulation since it relates very closely to the dynamic programming formulations of the 2D problems considered in later sections.

The Gilmore and Gomory Method for the One-Dimensional Knapsack Problem

A detailed study of knapsack problems was undertaken by Gilmore and Gomory [13]. They suggest methods for solving the one-dimensional knapsack problem using dynamic programming, and develop the idea of a knapsack function; they formulate the problem as: Given positive integer lengths $\ell_1,\ldots,\ell_m$ with corresponding non-negative values $v_1,\ldots v_m$ find for a given integer x the value $F(x)$, where $F(x)$ is the maximum of $\sum_{i=1}^{m} v_i q_i$, where $q_i,\ldots q_m$ are non-negative integer satisfying:

$$\sum_{i=1}^{m} \ell_i q_i \leq x.$$

The function $F(x)$ is known as a knapsack function. Clearly if a function $F_o(x)$ is defined such that:

$$F_o(x) = \max \{0, v_i \mid \ell_i \leq x\}.$$

Then F(x) satisfies:

$$F(x) = \max \{F_o(x), F(x_1) + F(x_2) \mid x \geq x_1 + x_2, 0 < x_1 \leq x_2\}.$$

From this definition of F(x) Gilmore and Gomory have developed the following dynamic program.

Let us first assume that the pieces to be cut are ordered in decreasing order of length thus

$$\ell_1 \geq \ell_2 \geq \cdots \geq \ell_m ,$$

and consider an additional dummy length ℓ_{m+1} to be cut, such that $\ell_{m+1} = 1$ and $v_{m+1} = 0$. This piece then fills any wastage that may be left, but does not change the value of F(x). Now introduce a function L(x):

$$L(x) = \max \{i \mid F(x) = F(x - \ell_i) + v_i\} \qquad (1)$$

Theorem [13]: The knapsack function F(x) is defined by the equation:

$$F(x) = \max \{0, F(x - \ell_i) + v_i \mid i \geq L(x - \ell_i), x \geq \ell_i\}$$

where L(x) is defined in (1).

From the above theorem the dynamic program for the one-dimensional knapsack problem follows immediately. Two temporary functions F* and L* are used and at the termination of the procedure F* and L* become F and L. Call the length of the stock object L_1.

1.1 Initialise $F^*(x) = F_o(x)$ and $L^*(x) = \max\{i \mid v_i = F_o(x)\}$ for $0 \leq x \leq L_1$. Let $x_2 = 1$.

2.1 Let $x_1 = 1$.

2.2 If $x_1 + x_2 \leq L_1$ then let $V = F^*(x_1) + F^*(x_2)$ and go to 2.3. Otherwise go to 3.1.

2.3 If $V > F^*(x_1 + x_2)$, then let $F^*(x_1 + x_2) = V$, let $L^*(x_1 + x_2) = \max\{L^*(x_1), L^*(x_2)\}$, and goto 2.4. Otherwise goto 2.4

2.4 If $x_1 < x_2$ then let $x_1 = x_1 + 1$ and go to 2.2. Otherwise go to 3.1.

3.1 If $x_2 < L_1$ then let $x_2 = x_2 + 1$ and go to 2.1. Otherwise stop.

At the termination of the computation $F^*(x)$ and $L^*(x)$ contain the values of $F(x)$ and $L(x)$ for $0 \leq x \leq L_1$, where L_1 is the length of the stock object to be cut. $F(x)$ is the value of the optimal solution. From the values of $L(x)$ it is possible to deduce the structure of the optimal solution in terms of how many of each piece are cut from the stock length. For any x, $L(x)$ contains the index of the smallest piece (say i_1) that makes up the optimal solution. In order to find the index of another piece in the optimal solution it is necessary to obtain the value of $L(x - \ell_{i_1})$, (say i_2). The value of the next index is then found from $L(x - \ell_{i_1} - \ell_{i_2})$, and the process continues until all pieces in the optimal solution are found.

The method described above suffers from the main disadvantage of dynamic programming, i.e. that for large scale problems a large amount of storage is required. This has lead to the development of much better tree search algorithms to solve the one-dimentionsal knapsack problem which, however, will not be discussed here, since they are not generalisable to 2 or more dimensions. The interested reader is referred to [27, 30].

The Multi-Parameter Knapsack Problem

A variation on the knapsack problem has been referred to in the literature as the multi-dimensional 0-1 knapsack problem, this leads to some ambiguity as the multi-dimensionality does not refer to Euclidean space, contrary to Gilmore and Gomory's [13] use of the term two-dimensional knapsack problem to refer to problems in two-

dimensional Euclidean space. In this thesis the term two-dimensional knapsack problem will refer to problems in two-dimensional Euclidean space, and problems referred to in the literature as multi-dimensional, where the dimensionality refers to the vector of parameters for each piece, will be referred to here as multiparameter knapsack problems. The main method that has been used to solve this problem is branch and bound. See for example [9] and [32].

4.2 (B , 1 , K^1, LE) The Constrained One-Dimensional Knapsack Problem

A function for the solution of this problem was given by Gilmore and Gomory [14] as a variation of their unconstrained knapsack function. The function is $G_k(x)$ and gives the value for the optimum solution to the constrained knapsack problem when the constraints of the pieces to be cut are given by:

b_j for $j = 1,\ldots, k$, and are.

0 for $j = k+1,\ldots, m$.

$G_k(x)$ can be defined as follows:

$$G_o(x) = 0 \text{ for all } x \qquad (1)$$

$$G_{k+1}(x) = \max_n \{G_k(x - n.\ell_{k+1}) + n.\, v_{k+1}\} \qquad (2)$$

where:

$$x \geq n.\ell_{k+1} \text{ and } b_{k+1} \geq n \geq 0.$$

4.3 (A , 1 , K , GE) The One-Dimensional Stock Cutting Problem

Mostly approximate methods have been used to find a solution to this problem. These are linear programming then rounding up or down for an integer solution and various other heuristics. In this section both types of heuristic methods will be discussed.

4.3.1 The Linear Programming Approach

Linear programming has been used to find an approximate solution to the one-dimensional stock cutting problem. This is done by solving the L.P. relaxation of the integer programming formulation of the

problem, in order to obtain a continuous solution. A heuristic is then used to round the solution up or down.

In this section two integer programming formulations of the problem will be given. This first one is the simpler of the two. However, it has not appeared in the literature before. This is may be because the second more complex formulation is likely to give a more nearly integer solution than the simpler formulation, as has been experienced with other combinatorial problems [2].

First Formulation

Let y_i be the number of times stock length L_i is used. Let x_{ijk} be the number of pieces of length ℓ_j cut from the k_{th} use of stock length L_i, and v_j and b_j be the value and demand respectively, on ℓ_j. The one-dimensional stock cutting problem can be formulated as follows:

$$\text{Max} \sum_{\substack{i=1..n \\ j=1..m \\ k=1..B_i}} v_j \, x_{ijk}$$

such that:

$$\sum_{j=1..m} \ell_j \, x_{ijk} \leq L_i, \quad i = 1..n, \quad k = 1..B_i \; ;$$

$$\sum_{\substack{j=1..m \\ k=1..B_i}} \ell_j \, x_{ijk} \leq L_i \, y_i, \quad i = 1..n \; ;$$

$$\sum_{\substack{i=1..n \\ k=1..B_i}} x_{ijk} \geq b_j, \quad j = 1..m \; ;$$

$$0 \leq y_i \leq B_i \; ; \quad x_{ijk} \geq 0; \quad y_i, \; x_{ijk} \text{ integer}$$

where B_i is an upper bound on the number of stock lengths L_i that are used.

B_i can be estimated by heuristic methods taking into account the demand (b_i) on each piece, or alternatively can be decided by the user.

Second Formulation

The second I.P. formulation of this problem was given by Paull

and Walters [22, 23] this was in relation to the cutting of large roll of paper into smaller rolls for newspaper production, a similar approach is also given by Metzger [21], it can be stated as follows:

Given n lengths of stock $L_1,\ldots, L_n$, and m types of lengths $\ell,\ldots,\ell_m$ for the pieces to be cut, with a lower bound b_j on the demand associated with each length ℓ_j find the optimum cutting patterns.

An I.P. for this problem can be formulated by considering all the possible ways of cutting up the stock lengths into the pieces to be cut. For each stock length there will be a large number of ways in which it can be cut up to produce a combination or cutting pattern of the pieces to be cut. Suppose that there are N of these combinations over all stock lengths*. With each combination h (say) associate m integers t_{hj}, $j = 1,\ldots,m$ such that for combination h there are t_{hj} pieces of length ℓ_j to be cut. Also associate with each combination h a cost c_h, this could be the cost of the stock length that the combination uses, or some measure of the wastage generated by that combination, or an inverse function of the value of the lengths produced by the combination. Then the general one-dimensional stock cutting problem is:

$$\text{minimise} \quad \sum_{h=1}^{N} c_h x_h$$

subject to

$$\sum_{h=1}^{N} t_{hj} x_h \geq b_j \qquad j = 1, m$$

$$x_h \geq 0 \quad , \quad x_h \text{ integer}$$

where x_h is the number of times combination h is used in the solution.

---

* Note that a "combination" is used here to imply the actual cutting pattern and also the stock from which this pattern is to be cut.

In practice the number of possible combinations is so large that it is impossible to obtain an integer solution and in fact difficult to obtain an L.P. solution, however if an L.P. solution is obtained then it is often sufficient to round up or round down the non-integer values. In the case of rounding up there may be an over-supply of some of the demand, in the case of rounding down then some of the demand may not be satisfied, in which case an additional problem may have to be solved in order to meet demand.

As the number of variables in the L.P. for the stock cutting problem is so large, most of the work that has been done in this area has been to try and reduce the number of variables involved, in order to make the L.P. more practical for large problems. Metzger [21] only considers combinations which generate less than a certain amount of wastage. Eilon [6] only allows combinations which are 'practical', for example it may not be practical, because of the nature of the cutting equipment, to cut more than a certain number of pieces from one stock length, and any combinations that contain more than that number of pieces need not be considered.

A considerable increase in the size of problem that could be solved was achieved by Gilmore and Gomory [11, 12]. In their method an initial basic feasible solution is found by an ad hoc method, then the L.P. is set up using as variables only the combinations that occur in the initial solution, then in order to find if there are any combinations which should be introduced into the L.P. they solve an auxilliary problem to generate any combinations that will improve the solution. For an optimal improvement the dynamic program for the unconstrained one-dimensional problem described in section 4.1 is used to solve the auxilliary problem, however they also describe an ad hoc method for generating a combination that improves the solution, if an

improvement exists, this is faster than the D.P. although it will not necessarily give the best improvement. Also any combinations which do not contribute to the solution are removed, in order to keep the size of the L.P. down. They also introduced the idea of terminating the L.P. if a certain number of consecutive pivots do not produce at least a certain percentage improvement in the optimal solution.

Other Heuristics

Many practical problems are too large even to use the Gilmore and Gomory method, for example Pierce [25] found that such an approach was not practical for problems in the paper industry that he was considering. Stainton [29] considers that the methods are not practical for use in his problems where the stock is steel reinforcement bars. Many of these difficulties are also introduced by the peculiar nature of the problem, for example, the sequence of cutting patterns produced by the Gilmore and Gomory method may not be practical; Stainton found that it may be desirable to attach some value to the wastage from a particular cutting pattern in case it can be returned to stock instead of being treated as scrap. A number of authors have as a result of these problems developed a heuristic approach to the problem.

Pierce [25] attempts to heuristically find a 'good' cutting pattern for cutting up a stock length, he then procedes to cut up that stock length until the demand for at least one of the ordered pieces is satisfied, he then generates a new 'good' combination without the pieces for which demand has been satisfied, and uses this to satisfy demand for another ordered piece, the procedure continues until all demand has been satisfied.

Haessler [15, 16] in examining problems associated with the paper industry adds extra constraints to the problem. He puts a bound on the total number of pieces that can be cut from any one stock length,

this is associated with the maximum number of knives that the cutting machine may have. He also introduces a fixed charge for any particular cutting pattern thus making it desirable to reduce the total number of cutting patterns used, this is because of the delays introduced by changing the cutting pattern on the cutting machine, which can be expensive in the high speed production of paper. He uses an approach somewhat similar to that of Pierce [25]. First he analyses the orders for cut lengths and from this analysis produces two 'descriptors' of the current problem situation. One descriptor gives an estimate of the number of stock lengths needed to satisfy the remaining requirements. The other gives the average number of ordered lengths to be obtained from each stock length. From these descriptors he produces some 'aspiration levels' for a cutting pattern to achieve these are:

(1) The maximum allowable trim loss

(2) The minimum and maximum number of ordered lengths permitted in a pattern

(3) The minimum number of stock lengths to be cut using the cutting pattern.

A search is then carried out on a restricted set of cutting patterns, in order to find one that satisfies the 'aspiration levels', if one is not found then the aspiration levels are altered and the search repeated. When a cutting pattern is found that satisfies the aspiration levels then it is given maximum possible usage until demand for some ordered length is satisfied, the problem descriptors are then updated and the whole process repeated. This continues until all demand has been satisfied. Haessler found that if the various parameters involved were correctly 'tuned' it was possible to obtain an average 25% improvement in trim loss over manual methods for a set of actual production problems.

Stainton [29] developed a heuristic for the cutting of steel reinforcement bars, in this case consideration is given to the possible utilisation of wastage. The procedure has two phases, the first phase is a 'cutting phase' in which stock length are cut, the second phase a 'reduction of wastage phase' in which the off-cuts are examined to see if that can be used to fulfil the order or whether they must be thrown away.

Finally the work of Eisenmann [7] should be mentioned. He considered the one-dimensional stock cutting problem for the case when there are two cutting machines used to do the cutting and they have to be scheduled to have production times that do not differ by more than a specified number of hours. He uses L.P. to find a fractional solution, and then manually rounds up or down.

4.4 (C , 1 , R , GE) The One-Dimensional Assortment Problem

The last and potentially the most complex of the one-dimensional stock cutting problems is the one-dimensional assortment problem, this has been discussed by a number of authors [8, 24, 26, 33, 34] and the formulation presented is in general a considerably simplified version of the complete problem expressed by the quadtuple (C, 1, R, GE).

The main restriction that is imposed on the problem is that a piece is produced by cutting one stock length, and anything left after the stock length is cut to produce that piece is treated as "waste". Thus, only one piece is produced from one stock length. This considerably simplifies the problem as clearly the dimensions of the stock will be a subset of the set of dimensions of the pieces demanded. In particular the longest length amongst the pieces demanded will have to be one of the stock lengths.

Sadowski [26] first used the name assortment problem. He developed a dynamic programming formulation of the problem and improvements to

this formulation were made by Frank [8] and Wolfson [34]. The formulation given here is that of White [33]. Suppose that the lengths of the pieces to be cut are ℓ_i $i = 1,\ldots,m$ these lengths being ordered so that $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_m$ and the demand associated with each ordered length is b_i. Let us define a function $\sigma(x, z)$ to be the cost of satisfying the demands of all the pieces with lengths in the range x to z (call this subset of pieces $S(x, z)$), from a stock object of length z. This function can be easily calculated given the relation between wastage and cost. For example if cost is equal to wastage and the set of pieces $S(x, z)$ is $\{\ell_j,\ldots, \ell_{j+k}\}$ then:

$$\sigma(x, z) = \sum_{i=o}^{k} \left[(z - \ell_{j+i}) \quad b_{j+i} \right].$$

Also, let us define a function $C_N(x)$ to be the minimal cost of satisfying demand of all the pieces with lengths greater or equal to x, using at most N different lengths for the stock. Then the following recursive equation relates the two functions C_N and σ:

$$C_N(x) = \min_{z \geq x} \left[\sigma(x, z) + C_{N-1} (z+1) \right]$$

where C_o is defined $C_o(z) = 0$ for $z > \ell_m$ and $C_o(z) = \infty$ for $z \leq \ell_m$.

From the above recursion, a dynamic program can be used to find $C_N(x)$ for $1 \leq N \leq m$ and $0 \leq x \leq \ell_m$. It is possible to considerably limit the values of x which need to be used in the recursion. Thus, x need only take values equal to the lengths of the pieces to be cut. Clearly, if x is not equal to one of these lengths, say $x = \ell_{i*} + y$ and $\ell_{i*+1} > \ell_{i*} + y$ then a stock of length ℓ_{i*} can be used to produce the same pieces as a stock of length x. However, the stock of length x will produce y more wastage each time it is cut, and hence will cost more.

The dynamic program is used to generate the values for $C_N(\ell_m)$ for $1 \leq N \leq m$, and from these values a decision can be made about which is the best value of N. No algorithm has been suggested for making this decision. Wolfson [3, 4] gives a practical application for the cutting of steel billets in which he says that the results of the dynamic program were used by management to make the decision.

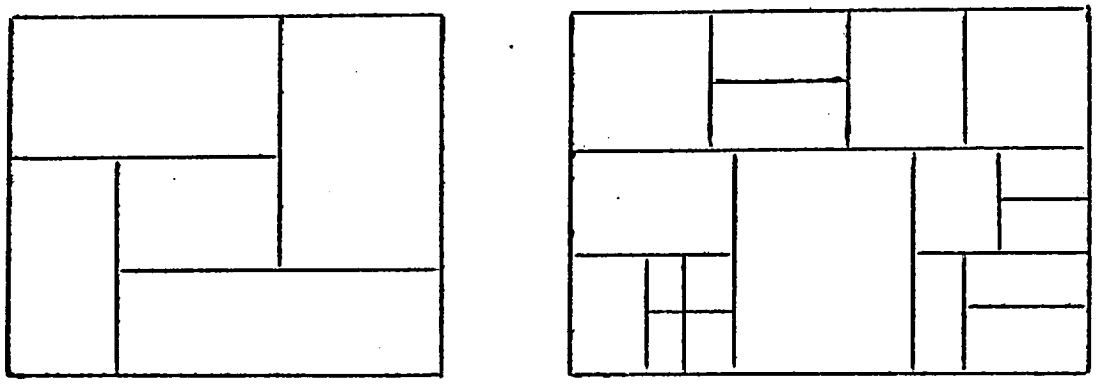
4.5 (B , 2 , K^1, -) The Two-Dimensional Knapsack Problem

The two-dimensional unconstrained knapsack problem has been considered by a number of authors with various extra constraints to simplify the problem. The main way in which the problem is simplified is to put some restriction on the ways in which the stock rectangle is cut. There are two types of restrictions that are used.

(i) The first type is to impose the restriction of guillotine cuts, these are cuts which go from one edge of the rectangle to be cut to the opposite edge, parallel to the two other edges, an illustration of guillotine and non-guillotine type cuts is given in Figures 1 (a) and 1 (b). The restriction to guillotine cuts often occurs in practice for example in the cutting of paper or glass.

(ii) The second type of restriction is to limit the cutting to a number of 'stages'. Thus, if two adjacent edges of the rectangle to be cut are considered to be the x and y axes then the first stage cuts would be cut parallel to the x (or y) axes, the second stage parallel to the y (or x) axes, the third stage parallel to the x (or y) and so on. Thus, a restriction can be imposed so that cuts only go up to the third stage for example. The staging of cuts also can occur in practice; thus, if the rectangle to be cut is being processed along a production line there may be a number of cutting machines along the line each corresponding to a stage. Staged cuts are illustrated in Figure 2, note that these are always guillotine cuts otherwise it would

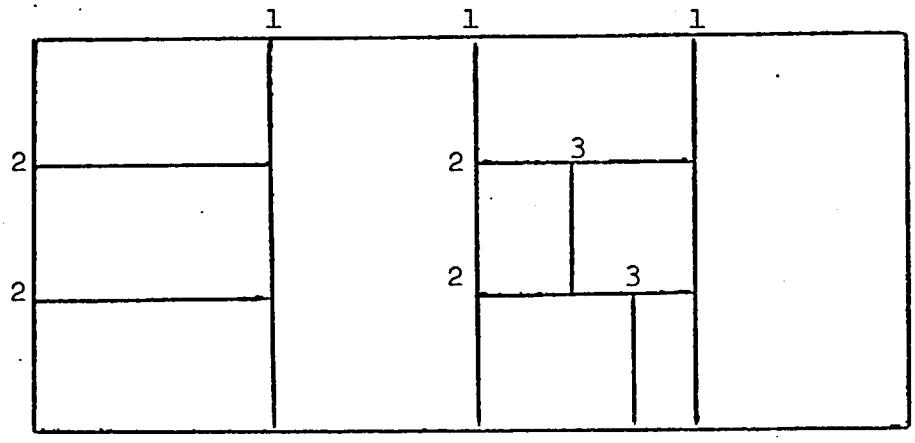
not be possible to define stages. In this survey the first problem to be discussed will be the two-dimensional knapsack problem with guillotine cuts imposed.



(a) Non-guillotine cut

(b) Guillotine cut

Figure 1



Staged cutting with 3 stages

Figure 2

A. Guillotine Cuts

There have been two optimal methods developed to solve the two-dimensional knapsack problem with guillotine cuts imposed, one method uses dynamic programming and was developed by Gilmore and Gomory [13], the other method uses tree search techniques and was developed by Herz [18]. In this section the dynamic proqramming method of Gilmore and Gomory will be described in preference to the method of Herz as in the next chapter the Gilmore and Gomory method has been used in the

algorithm dealing with the constrained version of this problem.

In the paper by Gilmore and Gomory two methods are described for solving the two-dimensional knapsack problem a simple method and a second more sophisticated method. However during the course of the work for this thesis it was found that the second method was incorrect, and it is therefore the first method of Gilmore and Gomory which will be described here. This fault was also found independently by Herz and an explanation of what is wrong with their second method can be found in [18].

In order to develop a dynamic program for solving the two-dimensional knapsack problem Gilmore and Gomory introduce a knapsack function similar to the one-dimensional case. This two-dimensional knapsack function $F(x, y)$ is defined as follows:

> Given m rectangular pieces $r_i = (\ell_i, w_i)$ with non-negative values v_i associated with them, then
>
> $$F(x, y) = \max \sum_{i=1}^{m} \xi_i v_i$$
>
> where $\xi_1, \ldots, \xi_m$ are non-negative integers such that there exists a way of dividing, by the use of guillotine cuts, a rectangle (x, y) into ξ_i rectangles (ℓ_i, w_i) for $i = 1, \ldots, m$.

For example consider Figure 3 the rectangle (x, y) contains one

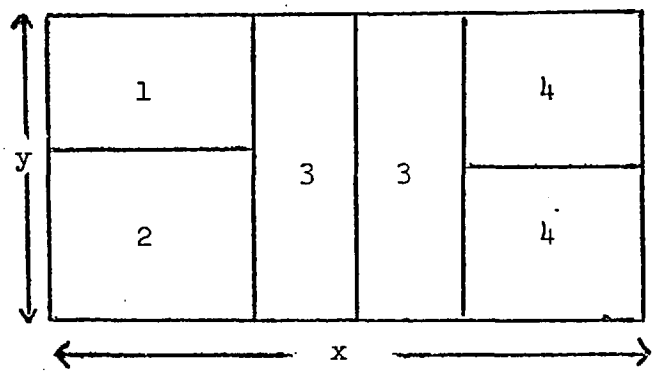


Figure 3

Feasible Cutting Pattern for a Rectangle (x,y)

rectangle r_1, one rectangle r_2, two rectangles r_3 and two rectangles r_4 hence for this particular rectangle it can be said that:

$$F(x, y) \geq v_1 + v_2 + 2v_3 + 2v_4.$$

The function $F(x, y)$ has a number of properties:

$$F(x, y) \geq 0 \tag{1}$$

$$F(x_1+x_2,y) \geq F(x_1, y) + F(x_2, y) \tag{2a}$$

$$F(x,y_1+y_2) \geq F(x,y_1) + F(x, y_x) \tag{2b}$$

$$F(\ell_i, w_i) \geq v_i \quad i = 1,\ldots, m \tag{3}$$

Property (1) is a consequence of the values of the rectangles being non-negative. Property (3) is obvious, the value of $F(\ell_i, w_i)$ will be equal to v_i unless there is a rectangle r_j such that $\ell_j \leq \ell_i$ and $w_j \leq w_i$ such that $v_j > v_i$ in which case $F(\ell_i, w_i)$ may equal v_j, or there is a set of rectangles that will fit into (ℓ_i, w_i) whose sum of values is greater than v_i, in either case $F(\ell_i, w_i) > v_i$. Properties (2a) and (2b) are also simple. If there existed an x_1^* and x_2^* such that $x^* = x_1^* + x_2^*$

$$F(x^*, y^*) < F(x_1^*, y^*) + F(x_2^*, y^*)$$

then $F(x^*, y^*)$ can be increased, for it is possible to make a cut on (x^*, y^*) at $x = x_1^*$ so that it is divided into two rectangles (x_1^*, y^*) and (x_2^*, y^*) and as $F(x^*, y^*)$ is the maximum sum of the values of the rectangles it can be cut into it can at least be updated to $F(x^*, y^*) = F(x_1^*, y^*) + F(x_2^*, y^*)$.

A dynamic program, which uses the above properties of $F(x, y)$, can be developed for calculating $F(x, y)$. The function $F^*(x, y)$ used in the dynamic program (and which becomes equal to $F(x, y)$ when the

procedure is finished) can be initialised from properties (1) and (3). Thus for at least x or y less than the x or y dimensions of any of the pieces to be cut $F^*(x, y)$ is initialised to zero, if x and y are such that the x and y dimension of at least one piece to be cut can fit into x, y than F(x, y) is initialised to the value of the highest value piece that will fit in (x, y). Note that in the Gilmore and Gomory procedure the orientation of any piece to be cut is considered to be fixed. Thus a rectangle (p, q) is considered different to a rectangle (q, p). This has practical relevance in some cases. For example in the cutting of cloth and the cutting of wood. However, if for any problem being considered the orintation is not important then each piece to be cut will have to be considered twice once with one orientation and a second time rotated through 90°. The dynamic program uses properties (2a) and (2b). Thus, for any (x^*, y^*), $F^*(x^*, y^*)$ is found by considering all possible guillotine cuts on (x^*, y^*), i.e. all possible cuts parallel to the y axis are considered from x = 0 to $x = \left\lfloor \frac{x^*}{2} \right\rfloor$ and the value of $F^*(x^* - x, y^*) + F^*(x, y^*)$ found for each cut. This type of cut is called an x-cut. Similarly, the y-cuts, which are parallel to the x axis are considered, from y = 0 to $y = \left\lfloor \frac{y^*}{2} \right\rfloor$ and the value of $F^*(x^*, y^* - y) + F^*(x^*, y)$ found for each cut. From all the values for the x and y cuts, the maximum is selected and $F^*(x^*, y^*)$ is set equal to that value. The dynamic programming procedure given above can be expressed in the following functional equations given by Gilmore and Gomory [13]:

$$F(x, y) = \max \{F_o(x, y), F(x_1, y) + F(x_2, y), F(x, y_1) + F(x, y_2);$$
$$x \geq x_1 + x_2,\ 0 < x_1 \leq x_2,\ y \geq y_1 + y_2, \text{ and}$$
$$0 < y_1 \leq y_2 \} \qquad (4)$$

where $$F_o(x, y) = \max_j \{0, V_j : \ell_j \leq x \text{ and } w_j \leq y\} \qquad (5)$$

Equations (4) and (5) give the value of F(x, y) for any x and y. They do not however give the structure of the cutting pattern that achieves this value. Gilmore and Gomory introduced two functions $\ell(x, y)$ and w(x, y) that give the cutting pattern for F(x, y) for any x and y. The way the functions $\ell(x, y)$ and w(x, y) work is best illustrated by an example. Consider Figure 4; suppose that this figure contains the optimal cutting pattern for the two-dimensional

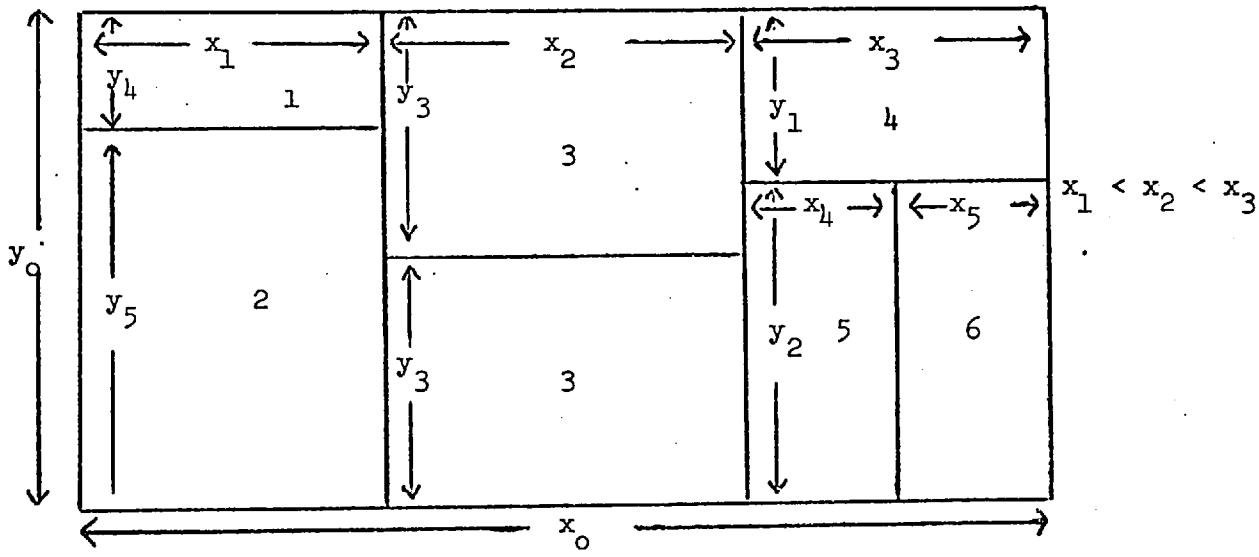


Figure 4

Figure Illustrating the Use of Functions $\ell(x,y)$ and w(x,y)

knapsack problem, that is the sum of the values of the rectangular pieces in the cutting pattern has a value $F(x_o, y_o)$. The functions $\ell(x, y)$ and w(x, y) are set to equal the size of the smallest x-cut and y-cut respectively that gives the value $F(x_o, y_o)$. Thus for (x_o, y_o) the smallest x-cut on (x_o, y_o) in Figure 4 is at $x = x_1$ hence $\ell(x_o, y_o) = x_1$. There are no y-cuts on (x_o, y_o) in Figure 4. The cut that obtains the value $F(x_o, y_o)$ is an x-cut and this fact is recorded by setting $w(x_o, y_o) = y_o$. From the values of ℓ and w at (x_o, y_o) it can be deduced that $F(x_o, y_o)$ was obtained by making an

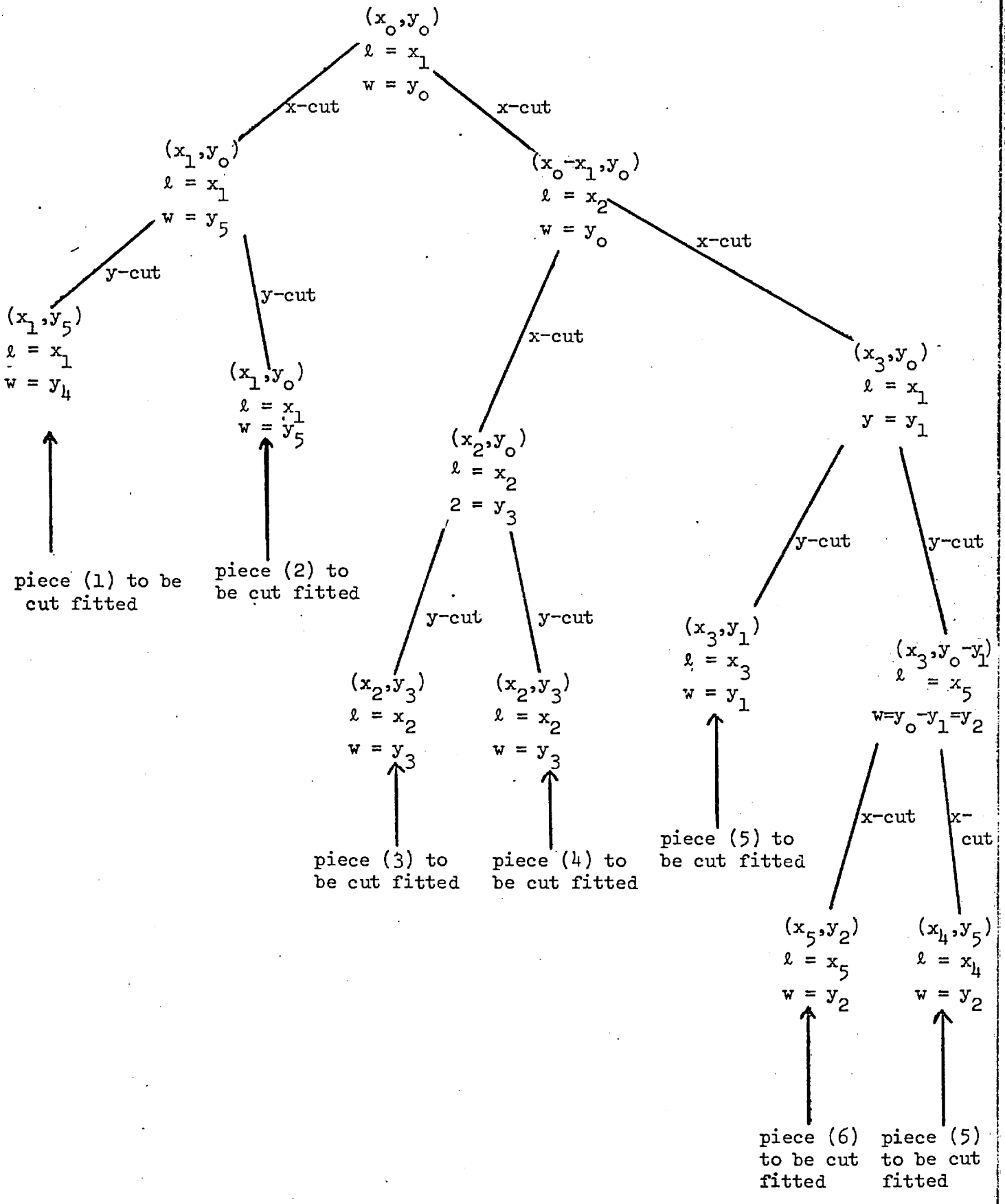


Figure 5 Tree Generated from Figure 4 using ℓ and w

x-cut at $x = x_1$, thus the next rectangles whose cutting patterns must be evaluated are (x_2, y_o) and $(x_o - x_1, y_o)$. Consider (x_1, y_o); this will have $\ell(x_1, y_o) = x_1$ and $w(x_1, y_o) = y_5$, indicating that the cut made on (x_1, y_o) is a y-cut at $y = y_5$. This cut produces two more rectangles (x_1, y_5) and $(x_1, y_o - y_5) \equiv (x_1, y_6)$ these rectangles are not cut in Figure 4, but have stock pieces (1) and (2) respectively filled into them, thus the values of ℓ and w will equal the x and y dimensions respectively of the rectangles, i.e. $\ell(x_1, y_5) = x_1$, $w(x_1, y_5) = y_5$ and $\ell(x_1, y_6) = x_1$, $w(x_1, y_6) = y_6$, this indicates that these rectangles are not cut further but have a piece (to be cut) fitted into them. Thus, a searchof the pieces to be cut which will fit (x_1, y_5) and (x_1, y_6) is carried out and the ones of maximum value selected. (This value equals the value of F for each rectangle.) In this case the rectangles are r_1 and r_2 respectively, and are added to a list of rectangles which fit into $F(x_o, y_o)$.

In general, the process described above is that of generating a binary tree. A tree for the complete analysis of Figure 4 is given in Figure 5. At each node the size of the rectangle represented at that node is given, and the values of ℓ and w for that rectangle are also given. On each branch it is shown whether the branch represents an x-cut or a y-cut. The functions $\ell(x, y)$ and $w(x, y)$ are defined as follows:

$$\ell(x, y) = \min \{x_1, x \mid 0<x_1 \leq x-x_1,\ F(x,y) = F(x_1,y) + F(x-x_1,y)\}$$

$$w(x, y) = \min \{y_1, y \mid 0<y_1 \leq y-y_1,\ F(x,y) = F(x,y_1) + F(x,y-y_1)\}.$$

The Gilmore and Gomory algorithm for the two-dimensional knapsack problem will now be formally stated. It uses three temporary functions F^*, ℓ^* and w^* which become F, ℓ, and w at the end of computation. The

dimensions of the stock rectangle are considered to be L and W:

1. $F^*(x, y) = F_o(x, y)$ for $0 \leq x \leq L$ and $0 \leq y \leq W$

   $\ell^*(\ell_i, w_i) = \ell_i$ and $w^*(\ell_i, w_i) = w_i$ for $i = 1, \ldots, m$

   $\ell^*(0,0) = w^*(0,0) = 0$

   $x_2 = y_2 = 1$ goto 2.1

2.1 $x_1 = 1$

2.2 If $x_1 + x_2 \leq L$, let $V = F^*(x_1, y_2) + F^*(x_2, y_2)$ goto 2.3, otherwise goto 3.1.

2.3 If $V > F^*(x_1+x_2, y_2)$ then let $F^*(x_1+x_2, y_2) = V$

let $\ell^*(x_1+x_2, y_2) = x_1$ and $w^*(x_1+x_2, y_2) = y_2$ and goto 2.4

If $V = F^*(x_1+x_2, y_2)$ then let $\ell^*(x_1+x_2, y_2) = x_1$ and goto 2.4

otherwise goto 2.4

2.4 If $x_1 < x_2$ then let $x_1 = x_1+1$ and goto 2.2. Otherwise goto 3.1

3.1 $y_1 = 1$

3.2 If $y_1+y_2 \leq W$ let $V = F^*(x_2, y_1) + F^*(x_2, y_2)$ and goto 3.3, otherwise goto 4.1

3.3 If $V > F^*(x_2, y_1+y_2)$ then $F^*(x_2, y_1+y_2) = V$,

let $w^*(x_2, y_1+y_2) = y_1$; let $\ell^*(x_2, y_1+y_2) = x_2$ and goto 3.4

If $V = F^*(x_2, y_1+y_2)$ then $F^*(x_2, y, +y_2) = V$ then

let $w^*(x_2, y_1+y_2) = y_1$ and goto 3.4

Otherwise goto 3.4

3.4 If $y_1 < y_2$ set $y_1 = y_1+1$ and goto 3.2. Otherwise goto 4.1

4.1 If $x_2 < L$ then let $x_2 = x_2 + 1$ and goto 2.1. Otherwise goto 4.2

4.2 If $y_2 < W$ then let $y_2 = y_2 + 1$ and $x_2 = 1$ and goto 2.1.

Otherwise stop.

This algorithm has four main parts. The first part (step 1) is the initialisation stage, this corresponds to making sure that properties (1) and (3) hold for F^* and indicating in ℓ^* and w^* that no cut is necessary for a rectangle with size equal to the size of a piece to be cut, it also initialises the 'step off' point from which to search for improvements in F^*. The second stage (step 2) sequentially looks for x-cuts at $x = x_1$ which will improve the value of $F^*(x_1+x_2, y_2)$, thus making sure that property (2a) holds for F^*. If any improvements are found then F^*, ℓ^* and w^* are altered, if an x-cut at x_1 is found that give the same value of $F^*(x_1+x_2,y_2)$ then ℓ^* is altered to x_1 as x_1 will be less than the current value of $\ell^*(x_1+x_2,y_2)$ because $\ell^*(x_1+x_2,y_2)$ will have been last altered at a lower value of x_2. The third stage (step 3) is the y-cut equivalent of the second stage this time making sure that property (2b) holds for F^*. The fourth stage (step 4) increments the 'step off' point.

The algorithm works in such a way that just before using (x_2,y_2) as a step off-point $F^*(x, y) = F(x, y)$ for x and y such that $0 \leq x \leq x_2$ and $0 \leq y \leq y_2$ and also $x_2 < x \leq L$ and $0 \leq y < y_2 - 1$, this situation is illustrated in Figure 6, the shaded area illustrates the points where $F^*(x, y) = F(x, y)$.

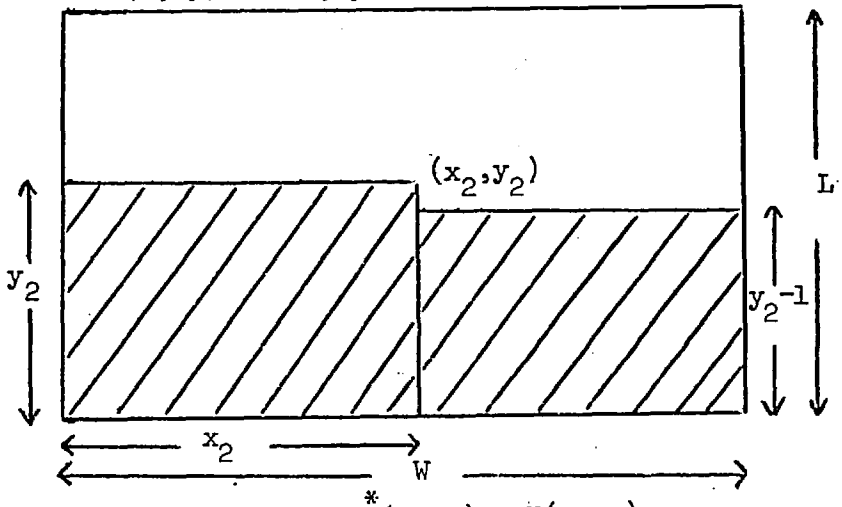


Figure 6 Illustration of where $F^*(x, y) = F(x, y)$

At each step off point (x_2,y_2) a series of rectangles (x_1+x_2,y_2) $0 < x_1 \leq x_2$, $x_1 + x_2 \leq L$ are examined to see if a cut at $x = x_1$ improves the value of $F^*(x_1+x_2,y_2)$, and if it does F^* is increased. (This is done in the second stage). Also for each step off point the rectangles (x_2,y_1+y_2) $0 < y_1 \leq y_2$, $y_1 + y_2 \leq W$ are examined to see if a cut at $y = y_1$ will improve $F^*(x_2,y_1+y_2)$. (This is done in the third stage). Once all improvements have been examined it is possible to move the step off point, if $x_2 < L$ this is done by moving it to (x_2+1,y_2), if $x_2 = L$ then it is moved to $(1,y_2+1)$, the procedure terminates when the step off point moves to (L, W).

B. Staged Cutting

Methods for solving the (B, 2, K^1, -) stock cutting problem in which the cutting is considered to be staged, have been described by Gilmore and Gomory [13] and Hahn [17], Gilmore and Gomory primarily consider problems in which there are two stages of cutting. They developed two methods for the staged problem one using dynamic programming and the other using linear programming.

Their dynamic programming method uses the a dynamic program similar in structure to the program for the one-dimensional knapsack problem, but uses the dynamic program in two stages - for each stage in the cutting process. The two stages are:

(1) For all widths w_i calculate v_i^*, the optimum value obtainable by fitting rectangles (ℓ_j, w_j), where $w_j \leq w_i$, end to end into a strip of width w_i and length L. For each i this is a one-dimensional knapsack problem.

(2) The optimum value of the objective function $\sum_{i=1}^{m} \xi_i v_i$ is then obtained by solving a one-dimensional knapsack problem: maximise $\sum_{i=1}^{m} \xi_i' v_i^*$ subject to $W \geq \sum_{i=1}^{m} \xi_i' w_i$

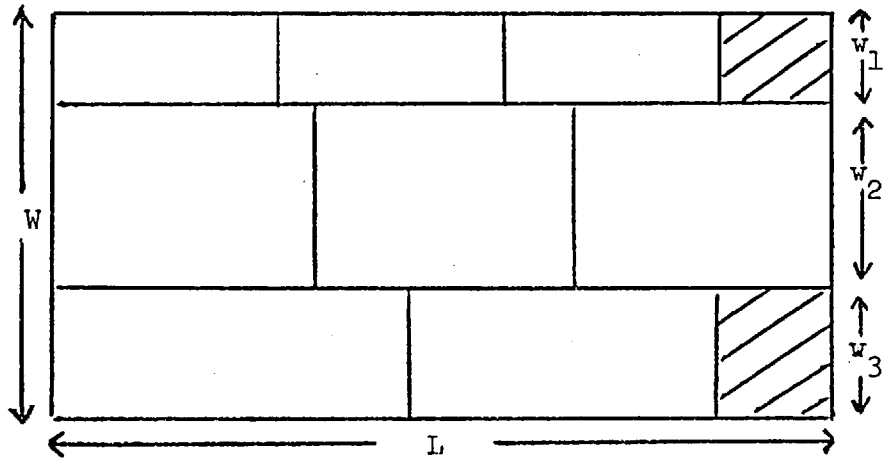


Figure 7 Two Stage Cutting

Figure 7 illustrates a typical cutting pattern produced by two stage cutting.

They also formulate this problem as a 'staged linear program', having two stages corresponding to the two cutting stages. Their approach combines the linear programming approach they used for the one-dimensional stock cutting problem with the decomposition method of Dantzig and Wolfe [4].

Hahn [17] shows how the above procedure can be extended to more than two stages. Although she is primarily concerned with a three stage cutting problem in which the cutting at the third stage only produces rectangles of equal size. She also extends the method to include the possibility of having defects in the stock rectangle which cannot be included in the pieces to be cut. The method was designed for the glass industry and she discusses ways of altering the method so as the find a near optimal solution quickly.

4.6 (C , 2 , P^1, GE) The Two-Dimensional Assortment Problem with a Continuous Length of Stock

Most of the work that has been done on this problem is for a considerably restricted version. However, Short [28] studied the problem without any additional restrictions imposed, introducing various heuristics to find a good solution and comparing their performance.

In other studies [10, 11, 12, 31] the additional restrictions that are imposed are:

1. Only staged cutting is considered with a maximum of two stages (note: staged cuts are guillotine cuts)
2. If the dimensions of a piece to be cut are (ℓ_k, w_k), then (ℓ_k, w_k) need not be supplied whole. Thus, in order to satisfy a demand of one on (ℓ_k, w_k), it is possible to supply r pieces (ℓ_k^p, w_k) : p = 1,...,r such that $\sum_{p=1}^{r} \ell_k^p = \ell_k$. Note that the width of these r pieces is equal to w_k.

This last type of situation occurs when the lengths of the pieces to be cut are much greater than their widths and it therefore may not be so important for each piece to be whole. Another situation in which this can occur is in executing a batch of jobs on a computer. Time can be regarded as the continuous dimension and the size of the computer storage as the fixed dimension. Programs requiring a certain amount of core and time do not require to run continuously but can be swapped in and out of core whenever convenient.

The problem with both simplifications 1 and 2 described above has been studied by a number of authors [11, 12, 13, 31] and L.P. can be used to find a solution. The L.P. formulation has many similarities

to the one-dimensional stock cutting problem. For each combination of widths of the pieces to be cut that will fit across the width of the stock, a variable x_i is used to denote the length of that combination used in the optimal solution. With each combination there is associated a waste a_i per unit of length, for example if combination 1 uses two widths w_1 one width w_2 and four widths w_3 then:

$$a_1 = W - 2w_1 - w_2 - 4w_3$$

where W is the width of the stock. Then, as the objective is to minimise wastage the objective function is:

$$\sum_{\substack{\text{all} \\ \text{combinations} \\ i}} a_i x_i .$$

Since it is not important that the pieces to be cut are produced whole, but can be supplied in smaller pieces whose sum is the size of the required length, demand can be though of as a total length required for each width w_j. Suppose that the demand length for each width w_j is d_j, and that the number of times width w_i occurs in combination i is t_{ij} then the L.P. constraints are

$$\sum_{\substack{i \\ \text{all} \\ \text{combination } i}} t_{ij} x_i \geq d_j \qquad i = 1, \ldots, m.$$

4.7 (A , 2 , K , GE) The Two-Dimensional Trim Problem

The two-dimensional trim problem can be solved approximately in a very similar manner to the heuristic method used for the one-dimensional stock cutting problem. Thus for all N feasible combinations of the two-dimensional pieces to be cut from stock*, there is a variable c_h $h = 1, \ldots, n$ which gives the cost of that combination and there are t_{hj} pieces of type j in combination h then the objective

* Note that a 'combination" is used here to imply the actual cutting pattern and also the stock from which this pattern is to be cut.

is

$$\text{minimise} \quad \sum_{h=1}^{N} c_h x_h$$

subject to:

$$\sum_{h=1}^{N} t_{hj} x_h \geq b_j \qquad j = 1, \ldots, m$$

This formulation is identical to that for the one-dimensional problem. However, it is much more complex to implement, since the size of N even for small problems is enormous, and is also not a simple matter to generate the cutting combinations.

Gilmore and Gomory [13] developed an improvement to the above method, for when guillotine cuts are imposed. This is of similar form to their improvement to the one-dimensional algorithm. A basic feasible solution is first found using a heuristic with only a limited number of the variables, corresponding to the cutting combinations, being considered. The algorithm for the two-dimensional knapsack problem (B, 2, K^1, -) (see Section 4.5) is then used to generate any combinations that should be included in the L.P. in order to improve the solution. Any combination that becomes 'redundant' is removed in order to keep the size of the L.P. as low as possible. This process for the one-dimensional problem is described in [11, 12], its use for two-dimensional problems is given in [13].

4.8 (C , 2 , R , GE) The Two-Dimensional Assortment Problem

Chambers and Dyson [1] have considered a heuristic for the two-dimensional assortment problem. As in the methods for the one-dimensional problem a cost is evaluated for a range of numbers of stock rectangles and a decision is then made on the correct number of stock rectangles to keep. The heuristic makes use of the solution to problem (A, 2, K, GE) described in Section 4.7 for a variety of potential stock sizes, and then uses a heuristic procedure for

choosing those potential stock sizes that are most highly utilised to keep as stock.

REFERENCES

1. Chambers, M.L. and Dyson, R.G., "The Cutting Stock Problem in the Flat Glass Industry - Selection of Stock Sizes", Opl. Res. Q., Vol. 27, 1976, pp. 949-957.
2. Christofides, N., Mingozzi, A., and Toth, P., "Loading Problems", Presented at the Summer School on Combinatorial Optimisation, Urbino, Italy, 1978, Proceedings to be published by Wiley.
3. Christofides, N. and Whitlock, C., "An Algorithm for Two-Dimensional Cutting Problems", Opns. Res., Vol. 25, 1977, pp. 30-44.
4. Dantzig, G.B., and Wolfe, P., "Decomposition Principle for Linear Programs", Opns. Res., Vol. 8, 1960, pp.101-111.
5. Dyson, R.G. and Gregory, A.S., "The Cutting Stock Problem in the Flat Glass Industry", Opl. Res. Q., Vol. 25, 1974, pp. 41-53.
6. Eilon, S., "Optimizing the Shearing of Steel Bars", J. Mech. Eng. Sci., Vol. 2, 1960, pp. 129-142.
7. Elseman, K., "The Trim Problem", Mgmt. Sci., Vol. 3, 1957, pp. 279-284.
8. Frank, C.R., "A Note on the Assortment Problem", Mgmt. Sci., Vol. 11, 1965, pp. 724-726.
9. Ghare, P.M., and Walters, L.E., "A Branch and Bound Algorithm for the Multidimensional Knapsack Problem", presented at a joint meeting of the 33rd national meeting of the Operations Research Society of America, and American meeting of the Institute of Management Science, 1968.
10. Gilmore, P.C. and Gomory, R.E., "A Linear Programming Approach to the Cutting Stock Problem", Opns. Res., Vol. 9, 1961, pp. 849-859.
11. Gilmore, P.C. and Gomory, R.E., "A Linear Programming Approach to the Cutting Stock Problem - Part II", Opns. Res., Vol. 11, 1963, pp. 863-888.

12. Gilmore, P.C., and Gomory, R.E., "Multistage Cutting Stock Problems of Two and More Dimensions", Opns. Res., Vol. 13, 1965, pp. 94-120.

13. Gilmore, P.C., and Gomory, R.E., "The Theory and Computation of Knapsack Functions", Opns. Res., Vol. 14, 1966, pp. 1045-1074.

14. Greenberg, H., and Hegerich, R.L., "A Branch Search Algorithm for the Knapsack Problem", Mgmt. Sci., Vol. 16, 1970, pp. 327-332.

15. Haessler, R.W., "A Heuristic Programming Solution to a Non-Linear Cutting Stock Problem", Mgmt. Sci., Vol. 17, 1971, pp. B-793-B-802.

16. Haessler, R.W., "Controlling Cutting Pattern Changes in One-Dimensional Trim Problems", Opns. Res., Vol. 23, 1975, pp. 483-493.

17. Hahn, S.G., "On the Optimal Cutting of Defective Sheets", Opns. Res., Vol. 16, 1968, pp. 110-1114.

18. Herz, J.C., "Recursive Computational Procedure for Two-Dimensional Stock Cutting", IBM J. of Res. and Dev., Vol. 16, 1972, pp. 462-469.

19. Ingargiola, G., and Korsh, J., "Reduction Algorithm of 0-1 Single Knapsack Problems", Mgmt. Sci., Vol. 20, 1973, pp. 460-463.

20. Kolesar, P.J., "A Branch and Bound Algorithm for the Knapsack Problem", Mgmt. Sci., Vol. 13, 1967, pp. 723-735.

21. Metzger, R.W., "Stock Slitting", Chap. 8 of Elementary Mathematical Programming, John Wiley and Sons Inc., New York, 1958.

22. Paull, A.E., and Walter, J.R., "The Trim Problem: An Application of Linear Programming to the Manufacture of Newsprint Paper", Paper presented at the Annual Meeting of the Econometric Society, Montreal 1954, Abstract in Econometrica, Vol. 23, p. 336.

23. Paull, A.E., "Linear Programming: A Key to Optimum Newsprint Production", Pulp Paper Mag. Can., Vol. 57, 1956, pp. 145-150.

24. Pentico, D.W., "The Assortment Problem with Probabilistic Demands", Mgmt. Sci., Vol. 21, 1974, pp. 286-290.

25. Pierce, J.F., "Some Large Scale Production Scheduling Problems in the Paper Industry", Prentice Hall, New Jersey, 1964.

26. Sadowski, W., "A Few Remarks on the Assortment Problem", Mgmt. Sci., Vol. 6, 1959, pp. 13-24.

27. Salkin, H.M., and Dekluyver, C.A., "The Knapsack Problem: A Survey", Nav. Res. Log. Quart., Vol. 22, 1975, pp. 127-144.

28. Short, R., "The Application of Two-Dimensional Packing to Computer Batch Scheduling", M.Sc. Thesis, Dept. of Management Science, Imperial College, 1972.

29. Stainton, R.S., "The Cutting Stock Problem for the Stockholder of Steel Reinforcement Bars", Opl. Res. Q., Vol. 28, 1977, pp. 139-149.

30. Toth, P., and Martello, S., "A Survey of Knapsack Problems", To be published in the proceeding of The Summer School on Combinatorial Optimisation, Urbino, Italy 1977, John Wiley.

31. Vajda, S., "Trim Loss Reduction", Chap. 21 of Readings in Linear Programming, John Wiley and Sons Inc., New York, 1958.

32. Weingartner, H.M., and Ness, D.N., "Methods for the Solution of the Multi-Dimensional 0/1 Knapsack Problem", Opns. Res., Vol. 15, 1967, pp. 83-103.

33. White, D.J., "Dynamic Programming", Oliver and Boyd, Edinburgh, 1969.

34. Wolfson, M.L., "Selecting the Best Lengths to Stock", Opns. Res., Vol. 13, pp. 570-585.

CHAPTER 3

STOCK CUTTING ALGORITHM *

* The contents of this chapter have been published in;
Operations Research, Vol 25, No 1, 1977, pp30-44

An Algorithm for the Two-dimensional Constrained Knapsack Problem (B, 2, K, LE)

1. Introduction

In this chapter an algorithm is developed for the solution of the constrained two-dimensional knapsack problem (B, 2, K, LE). The algorithm uses the search techniques and limits the size of the tree generated by deriving and imposing necessary conditions for the cutting pattern to be optimal. The dynamic programming procedure of Gilmore and Gomory [8] for the solution of the unconstrained problem, and a node evaluation method based on a transportation routine, are used to produce upper bounds during the search. The computational performance of the algorithm is illustrated by tests performed on a large number of randomly generated problems with constraints of varying 'tightness'. The results indicate that the algorithm is an effective procedure for solving cutting problems of medium size.

The two-dimensional cutting problem requires cutting a plane rectangle into smaller rectangular pieces of given sizes and values to maximize the sum of the values of the pieces cut. This version of the problem appears in the problem of cutting steel or glass plates [10] into required stock sizes to minimize waste. By taking the value of a piece to be proportional to its area, we can formulate the waste minimization problem as one of maximizing the value of the pieces cut. The problem also appears in cutting wood plates to make furniture and paper board to make boxes.

A special case of the general two-dimensional cutting problem is one in which all cuts must go from one edge of the rectangle to be cut to the opposite edge, i.e., the cut has to be of a

"guillotine" type. Some cutting patterns - e.g., that shown in

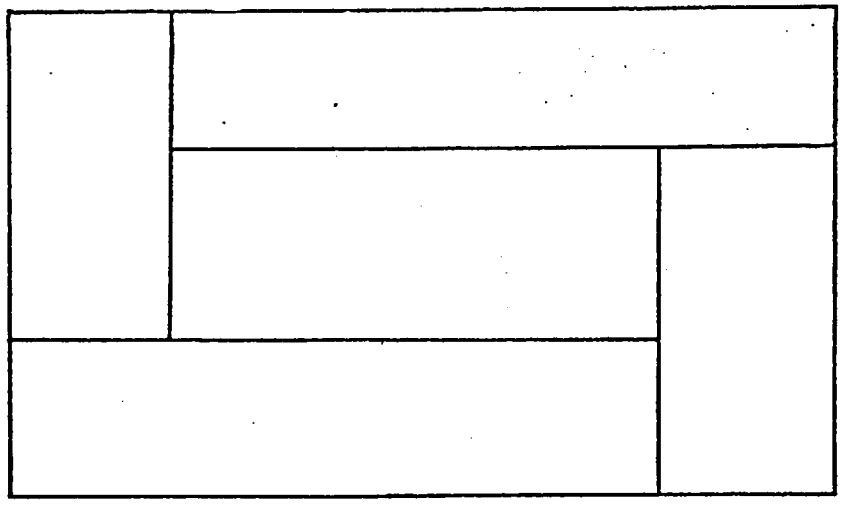


Figure 1(a). Cutting pattern infeasible with guillotine cuts

Figure 1(a) - could not be produced by this type of cut. However, the restriction of "guillotine" cuts appears very often in practice especially for the cutting of paper and glass. Figure 1(b) shows a possible cutting pattern using guillotine cuts where the cuts are numbered in the order in which they could

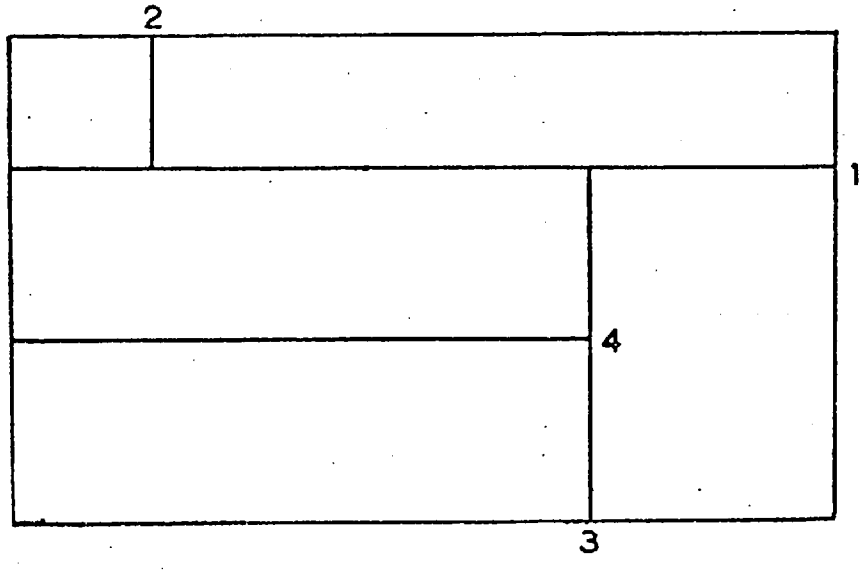


Figure 1(b). Cutting pattern made with guillotine cuts

be made, although other sequences are obviously also possible. In this chapter only "guillotine" type cuts are considered, and when the word "cut" is used this type of cut is implied. The more general cutting problem appears less frequently in practical problems and is more difficult to solve.

In practice cutting problems appear in a constrained form, the most usual constraint being the one that restricts the maximum number of pieces of each type to be cut. In the one-dimensional problem upper bound constraints on the variables can easily be included in a dynamic programming algorithm. However, the two-dimensional problem is not amenable to efficient solution by these means. We present a tree-search algorithm for the solution of the two-dimensional constrained cutting problem. The algorithm described limits the size of the tree search by deriving and imposing necessary conditions for the cutting pattern to be optimal and by using the dynamic programming procedure for the unconstrained problem together with a node evaluation method based on a transportation routine as subalgorithms to produce upper bounds during the search. The computational performance of the algorithm is illustrated by tests performed on a large number of randomly generated problems with constraints of varying "tightness". The results indicate that the algorithm is an effective procedure for solving cutting problems of medium size.

2. Problem

The constrained two-dimensional cutting problem can be defined as follows. Let a large rectangle $A_0 = (L_0, W_0)$ (i.e., the length of L_0 and width W_0 units) be given, together with a set R of m smaller rectangular pieces $R = \{(\ell_1, w_1), (\ell_2, w_2), \ldots, (\ell_m, w_m)\}$, each piece in R having associated with it a value v_i and a maximum

number b_i of pieces of type i that can be cut from A_0. The problem is to find the maximum value of $z = \sum_{i=1}^{m} \xi_i v_i$ so that $\xi_i \leq b_i$, i = 1,...,m, and there exists a series of cuts on A_0 so that ξ_i pieces of type i in R can be cut from A_0, the ξ_i being nonnegative integer variables.

In order to distinguish between the given pieces in set R and the rectangles produced by the cuts on A_0 at any stage during the cutting process, we will refer to the former as "pieces" and the latter as "rectangles". It will be assumed throughout the chapter that L_0, W_0 and ℓ_i, w_i, i = 1,...,m are integers and that the cuts on the rectangles are to be made in integer steps along the x and y axes. This limitation is not serious since in practice the actual dimensions can be scaled up. It should also be noted that the orientation of the pieces is considered to be fixed, i.e., a piece of length ℓ and width w is not the same as a peice of length w and width ℓ.

3. Enumerative Procedure

We will first describe a procedure that generates all possible cutting patterns without duplication and then describe methods for implicity enumerating these patterns in a general tree-search algorithm.

All possible cutting patterns can be generated by developing a tree, where branchings represent cuts on a rectangle. Thus. the branches emanating from the root-node of the tree correspond to all possible cuts on A_0, and each node at the end of a branch represents the rectangles produced by the corresponding cut on A_0.

To generate all possible cutting patterns (including the original uncut rectangle), we must include among "all possible cuts" an artificial "cut" that leaves the rectangle intact. This

"cut" plays an important role in the algorithm and will be referred to as a 0-cut. Thus, a rectangle that has been "cut" by a 0-cut must not be a candidate for future cutting and must - from that node onward - be considered fixed.

At successive nodes a rectangle is selected from the list of rectangles represented by a node, and branching occurs from that node by making all possible cuts on the chosen rectangle.

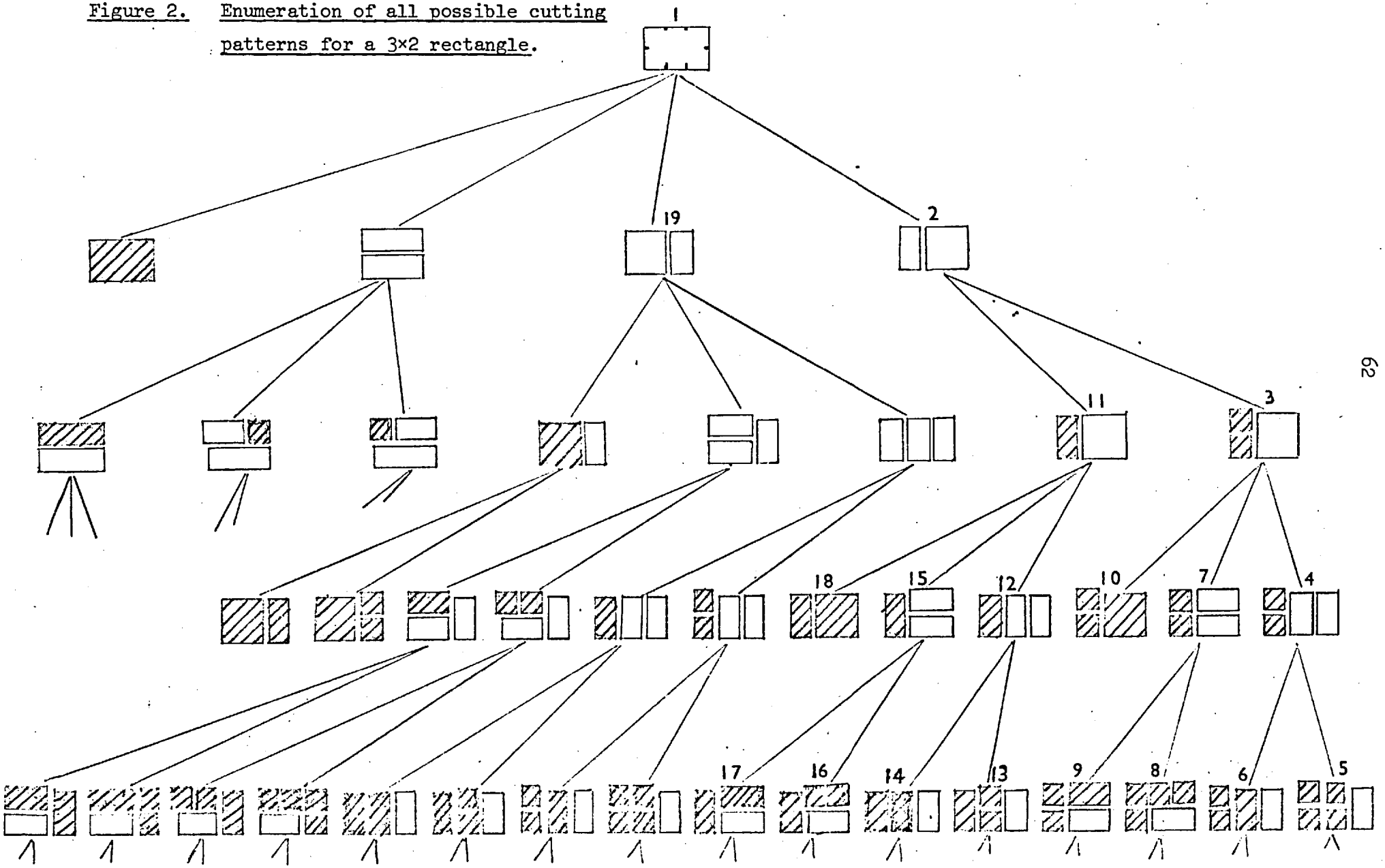
Part of the tree for a 3×2 rectangle is illustrated in Figure 2. At each node rectangle is selected for cutting from the rectangles that have not been 0-cut (a 0-cut rectangle is shaded in Figure 2). In Figure 2 the smallest of the non-0-cut rectangles is selected for cutting at each node. Branching is ordered so that the cuts parellel to the y-axis are made first in increasing order of x. Cuts parallel to the x-axis in increasing order of y are made next. The final cut is the 0-cut. The tree shown in Figure 2 is not complete as the tree generated by this process is too large. In order to develop an effective method for solving the problem, it is necessary to develop some ways for reducing the size of the tree, which we shall now describe.

3.1 Effects of Symmetry

Given a rectangle with dimensions (p,q), there are p+q-1 cuts that can be made on it - along $x = 1,\ldots,(p-1)$, along $y = 1,\ldots,(q-1)$, and the 0-cut. If, however, all these cuts are made, producing p+q-1 branches, then the set of rectangles represented at successor nodes will be duplicated at several nodes because of the appearance of symmetrical cutting patterns. These duplications can easily be removed as follows.

Given a rectangle (p,q) a cut at $x = a$ on (p,q) produces two

Figure 2. Enumeration of all possible cutting patterns for a 3×2 rectangle.



rectangles (a,q) and (p-a,q). Clearly, these same two rectangles could have been produced by a cut at x=p-a, which is symmetrically opposite to the cut at x=a with respect to (p,q). This can be avoided without missing any unique cutting pattern by making cuts only up to half way along the x-side of the rectangle. Thus, the range of x-cuts is limited instead of $1 \leq x \leq (p-1)$ to $1 \leq x \leq [p/2]$ where [p/2] means "the greatest integer not greater than p/2". Similarly, the rante of the y-cuts can be redefined to be $1 \leq y \leq [q/2]$.

An example of this type of duplication is shown in Figure 2, the rectangle cut up at node 1 is cut at x=1 and x=2 to produce nodes 2 and 19. The sets of rectangles represented by these nodes are identical. Thus by implementing the improvement described above it is possible to eliminate node 19 and all nodes beneath it, as well as many other nodes in the tree.

3.2 Effects of Cut Ordering

Consider a rectangle (p,q) and suppose that at some node (p,q) is cut at x=a, producing two rectangles (a,q) and (p-a,q). Then suppose at some successor node (p-a,q) is cut at b, $a < b \leq [(p-a)/2]$ producing three rectangles (a,q), (b,q) (p-a-b,q) from (p,q). These same three rectangles could have been produced by first making the cut x=b on (p,q) and then making the cut x=a on (p-b,q). This type of duplication can obviously be removed without missing any unique cutting patterns by introducing an arbitrary cut ordering so that if a rectangle is cut at, say, $x=\alpha$, then all subsequent x-cuts on the two resultant rectangles must be greater than or equal to α. This restriction, together with the restriction imposed by symmetry in the last section, implies

that for the larger of the two resultant rectangles, $(p-\alpha,q)$, the range of x-cuts is now limited to $\alpha \leq x \leq [(p-\alpha)/2]$ and, in particular, if $[(p-\alpha)/2] < \alpha$ no further x-cut on that rectangle need be made. For the smaller of the two resultant rectangles, (α,q), the restriction imposed by the cut ordering implies that no further x-cut is possible.

The consequence of cut ordering as explained above is to eliminate from explicit consideration different sequences of cuts when these lead to the same final cutting pattern. A similar kind of restriction can be imposed on the y-cuts.

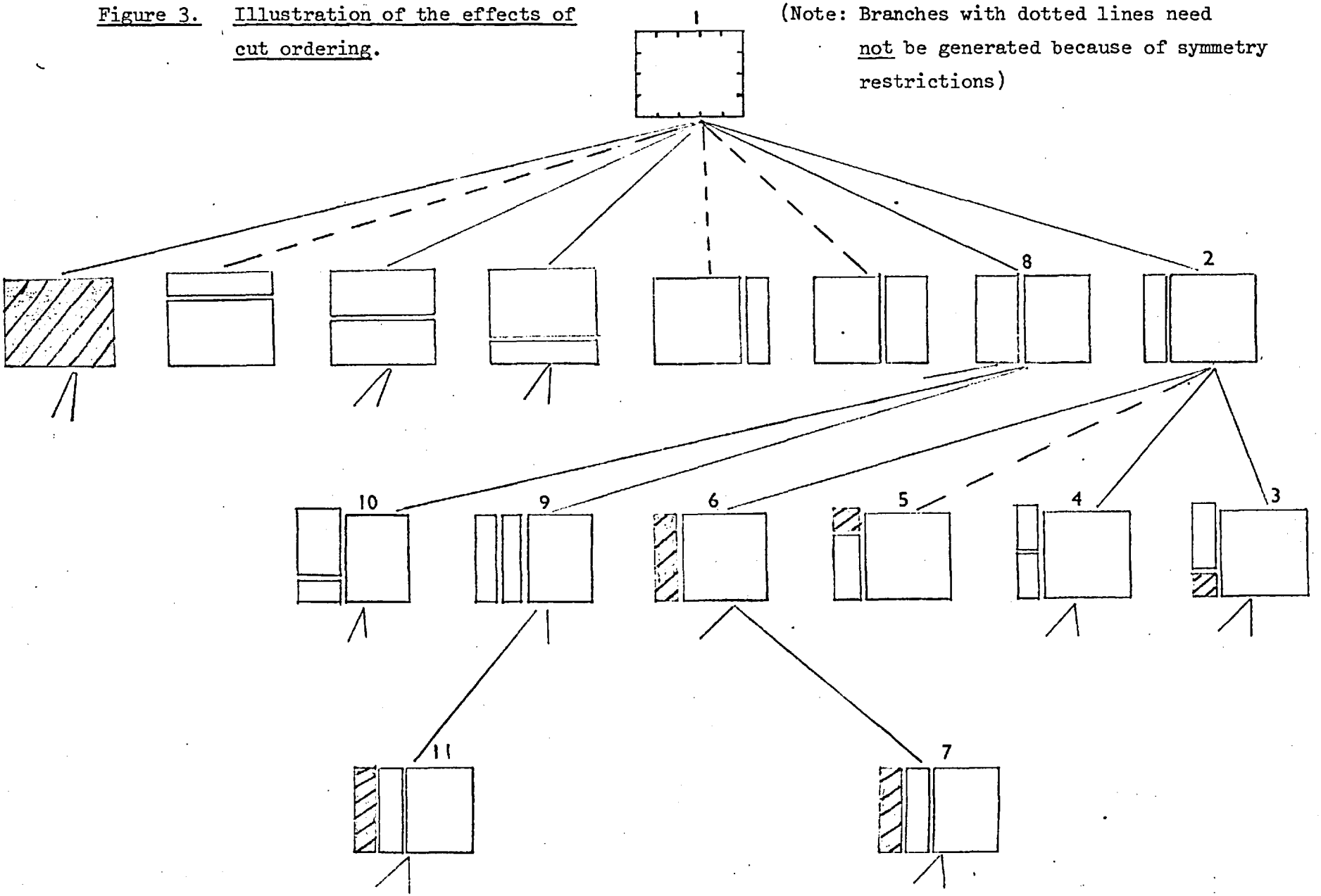
This can be seen in Figure 3 which illustrates part of the enumeration tree, with the symmetry restrictions imposed, for a 5×4 rectangle. At node 8 a (2,4) rectangle is cut at x=1 contrary to the above rule, as this rectangle was produced by a cut at x=2 on the (5,4) rectangle at node 1. It should therefore not be cut with an x-cut of value less than 2. As a result of cutting the (2,4) rectangle at node 2 at x=1 node 9 is produced. However, this node is redundant, as any node that is produced by branching from node 9 can also be produced in branching from a node below node 2. This occurs if the 4×4 rectangle at these nodes is cut at x=1. For example nodes 7 and 11 are identical.

3.3 Normal Cuts

If a rectangle (p,q) is to be cut by, say, an x-cut at some position x, then in the final cutting pattern there must be some combination of the lengths ℓ_i of the available pieces, for which $w_i \leq q$, whose sum must be exactly x. If this were not so (i.e., as shown in Figure 4(a) for the x-cut marked a-a), then an alternative cut at position $x' < x$ could also lead to a final

Figure 3. Illustration of the effects of cut ordering.

(Note: Branches with dotted lines need not be generated because of symmetry restrictions)



cutting pattern (as shown in Figure 4(b)), involving the same pieces and hence having the same value as the pattern in Figure 4(a). The pattern of Figure 4(b) will be called normal, and it is apparent that for any pattern there is a normal equivalent. It should, however, be pointed out that, contrary to the cut restrictions imposed previously, normality is a property of a cutting pattern that is relative to the set of pieces available for cutting. It should also be noted that a direct consequence of normality is that "waste" - such as the shaded area in Figure 4(b) - is never cut away by the algorithm

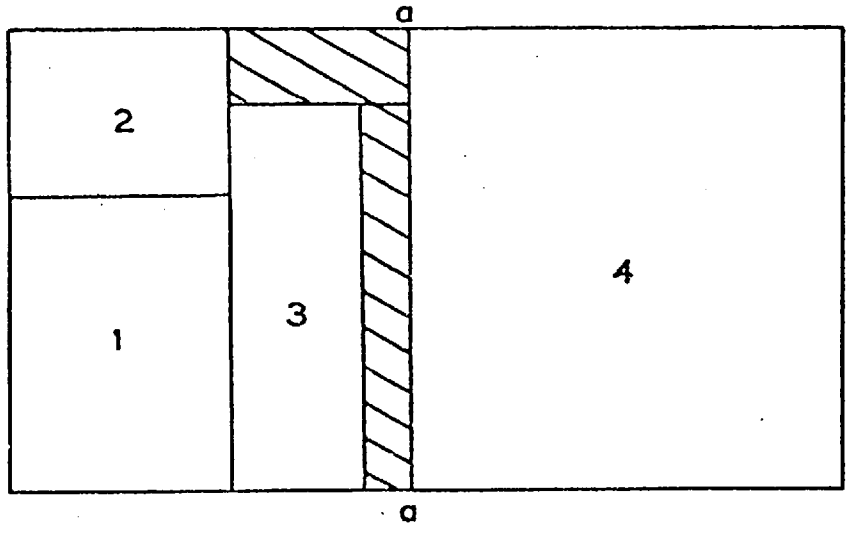


Figure 4(a). Nonnormalized cutting pattern

but is left attached to a rectangle from which some piece will later be produced.

Thus, one can (without loss of optimality) limit the x-cuts only to those values of x leading to normal patterns; and similarly for the y-cuts. The feasible x-cuts on a rectangle (p,q), given the set R of pieces (ℓ_i, w_i) to be cut, must then be at values of x that are the elements of some set S^q.

In Figure 3 if the pieces to be cut from the initial rectangle

at node 1 have x-dimensions greater than one, then the value x=1 will not be in the set S^q, and hence it is not necessary to branch on cuts at x=1. Thus nodes (2) and (9) and their successors need not be generated and the size of the tree is considerably reduced.

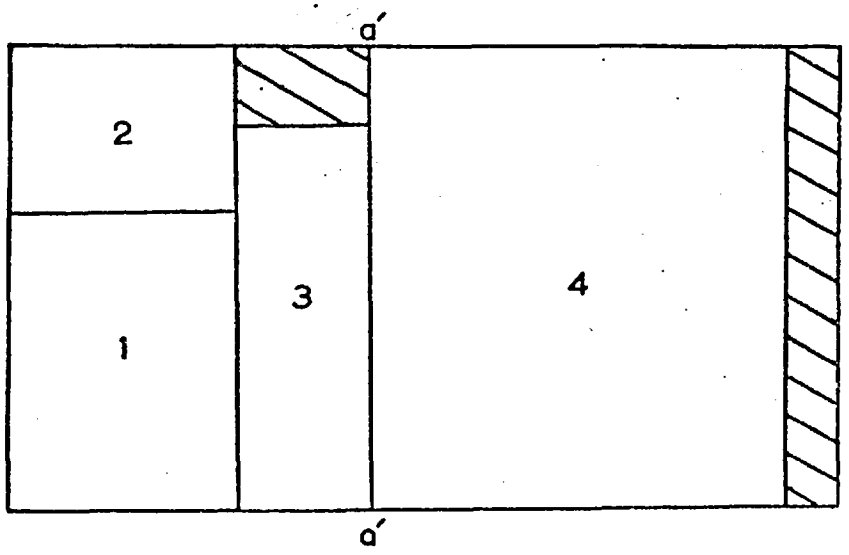


Figure 4(b). Normalized form of cutting pattern in Figure 4(a).

The sets S^q can be calculated for all values of q by a single iterative equation in one pass. First assume that the m pieces are ordered in non-decreasing values of w_i. We define a function $f_r(x)$ as follows.

If there exists one or more integer-valued vectors $\overline{\xi}$ that satisfy

$$x = \sum_{i=1}^{r} \overline{\xi}_i \ell_i, \qquad 0 \leq \overline{\xi}_i \leq b_i \ , \tag{1}$$

then set $f_r(x) = w_{i^*}$, where

$$i^* = \min \overline{\xi} [\max \{i \mid \overline{\xi}_i \neq 0\}] \ . \tag{2}$$

If no such vector $\overline{\xi}$ satisfying (1) exists, then set $f_r(x) = \infty$.

It follows from the above definition of $f_r(x)$ that if $f_m(x) \leq q$, then x is the sum of lengths ℓ_i of some combination of

pieces (satisfying the b_i constraints) all of whose widths w_i are less than q, and x must therefore be in the set S^q. Thus, once the tableau $f_m(x)$ is generated, the sets S^q can be produced for any q.

The tableaux $f_r(x)$, $1 \leq r \leq m$, $0 \leq x \leq [L_0/2]$ can be generated simply by the recursion: $f_i(x) = \min \{f_{i-1}(x), \max \{w_i, \min_k(f_{i-1}(x - k\ell_i))\}\}$ m where $1 \leq k \leq \min \{b_i, [x/\ell_i]\}$, k integer, $x \geq \ell_i$, $f_i(x) = f_{i-1}(x)$, $x < \ell_i$, and $f_o(x) = \infty$ for all x.

The corresponding sets T^p for the y-cuts can be generated in a similar manner.

The use of the function $f_r(x)$ is best illustrated by an example:

Suppose that the dimensions of the stock rectangle (L_0, W_0) are (18,12) and the pieces to be cut and the constraints associated with them are,

$$(\ell_1, w_2) = (3, 3) \qquad b_1 = 2$$
$$(\ell_2, w_2) = (5, 4) \qquad b_2 = 2$$
$$(\ell_3, w_3) = (4, 2) \qquad b_3 = 2 \ .$$

Note that they are ordered in non-decreasing value of w_i, then the values of $f_x(x)$ are as follows.

Clearly for $0 \leq x \leq 2$ $\qquad f_3(x) = \infty$.

For x = 3 there is only one vector $\overline{\xi}$ (1, 0, 0), hence

$f_3(3) = w_1 = 3$.

Similarly for x=4 and x=5 the only vectors are (0, 0, 1) and (0, 1, 0) respectively hence

$$f_3(4) = w_3 = 12 \quad \text{and} \quad f_3(5) = w_2 = 4 \ .$$

For x=6 the only vector is (2, 0, 0) hence

$$f_3(6) = W_1 = 3 .$$

For x=7 the vector is (1, 0, 1) hence $f_3(7) = w_3 = 12$.
For x=8 there are two vectors (1, 1, 0) and (0, 0, 2) hence $i^* = \min(2,3) = 2$ and $f_3(8) = w_2 = 4$. For x=9 there is only one vector (0, 1, 1) hence $f_3(9) = 12$, if the constraint on piece 1 were greater than 2 then another vector would be (3, 0, 0) and $f_3(9)$ would equal 3.

Thus for $0 \leq x \leq [L_0/2] = 9$, $f_3(x)$ has the values:

x	0	1	2	3	4	5	6	7	8	9
$f_3(x)$	∞	∞	∞	3	12	4	3	12	4	12

From this table the sets S^q that contain the normal cuts are:

$$S^q = \emptyset \qquad 0 \leq q \leq 2 \qquad (3)$$

$$S^q = \{3,6\} \qquad q = 3 \qquad (4)$$

$$S^q = \{3,5,6,8\} \qquad 4 \leq q \leq 11 \qquad (5)$$

$$S^q = \{3,4,5,6,7,8,9\} \qquad q = 12 \qquad (6)$$

($q > 12$ need not be considered as $W_0 = 12$).

The above analysis will considerably reduce the size of the tree, for example, from (5) at most four cuts need to be made on any rectangle of width less than 12.

3.4 Description of Enumerative Algorithm

We now describe an algorithm that generates all normal cutting patterns on a rectangle $A_0 = (L_0, W_0)$, given a set R of pieces to be cut. In this algorithm each node represents a state of the rectangle after cutting has taken place, and a tree branching from one node to another represents a cut.

The state at a node n is described by the list L of rectangles

produced by the sequence of cuts corresponding to the path that leads from the root of the tree to node n. In list L each rectangle is represented by a four-part label (p,q,x,y), where p and q are the rectangle's length and width and x and y are integers that can take values in the ranges $0 \leq x \leq [p/2] + 1$, $1 \leq y \leq [q/2] + 1$.

The meaning of x and y is as follows. If $1 \leq x \leq [p/2]$, then the next cut to be considered on rectangle (p,q) - if this rectangle is chosen for cutting - is an x-cut at position x. If $x = [p/2] + 1$ and $1 \leq y \leq [q/2]$, then the next cut to be made on (p,q) is a y-cut at position y. If $x = [p/2] + 1$ and $y = [q/2] + 1$, all feasible x-cuts on rectangle (p,q) have been performed and the next cut to be made is an 0-cut. If x = 0, we infer that an 0-cut on (p,q) has been made and this rectangle is not to be cut further by any branching following node n.

A few comments on the mechanics of the search procedure are necessary before the algorithm is described.

(i) Only one rectangle is cut at a node, and the procedure is exhaustive because of the introduction of the 0-cut, as explained earlier.

(ii) When a rectangle (p,q) is produced by an x-cut, future x-cuts on (p,q) must be at values of $x \geq X$, where X is calculated as described earlier. Similarly, Y is calculated if rectangle (p,q) was produced by a y-cut.

(iii) The state of the search is represented by:

(a) The list L of four-part labels corresponding to rectangles produced by the cuts so far. This list is updated for forward and backward branchings.

(b) Say that at node j a rectangle (p_j, q_j) from list L was chosen for cutting and the four-part label for

that rectangle was (p_j, q_j, x_j, y_j). Then a vector Q(j) of six-part labels $(p_j, q_j, x_j, y_j, X_j, Y_j)$ is stored for all j, $0 < j < n$, where X_j and Y_j are as explained in (ii) above.

A diagrammatic description of the algorithm is shown in Figure 5, and a detailed description is given in the appendix.

4. Tree-Search Algorithm

Section 3 gave an enumerative procedure that could generate all normal cutting patterns with respect to a set $R = \{(\ell_i, w_i), i = 1,\ldots,m\}$ and maximum numbers b_i, $i = 1,\ldots,m$ of pieces to be cut. The procedure generates all cutting patterns without symmetric duplications and without explicitly considering different sequences of cuts when these lead to the same cutting pattern. However, up to now no consideration has been given to the fact that the pieces in R have values v_i associated with them, and the purpose of this section is to examine ways of limiting the tree search described earlier in order to solve the optimization problem of Section 2.

4.1 Upper Bound on Value of Cutting Pattern at Node

In the normalized optimal cutting pattern there is exactly one piece from the set R fitted into each rectangle of this pattern because (as mentioned earlier) the "waste" - i.e., rectangles with nothing fitted in them - is not cut away by the algorithm itself. Thus, at some node of the tree when the list of rectangles already cut is given by the set L, let those rectangles that have had an O-cut made on them form the subset $H_0 \subseteq L$. The rectangles in H_0 will not be cut at any node below the current node and in the final cutting pattern will have some piece fitted in them. Hence,

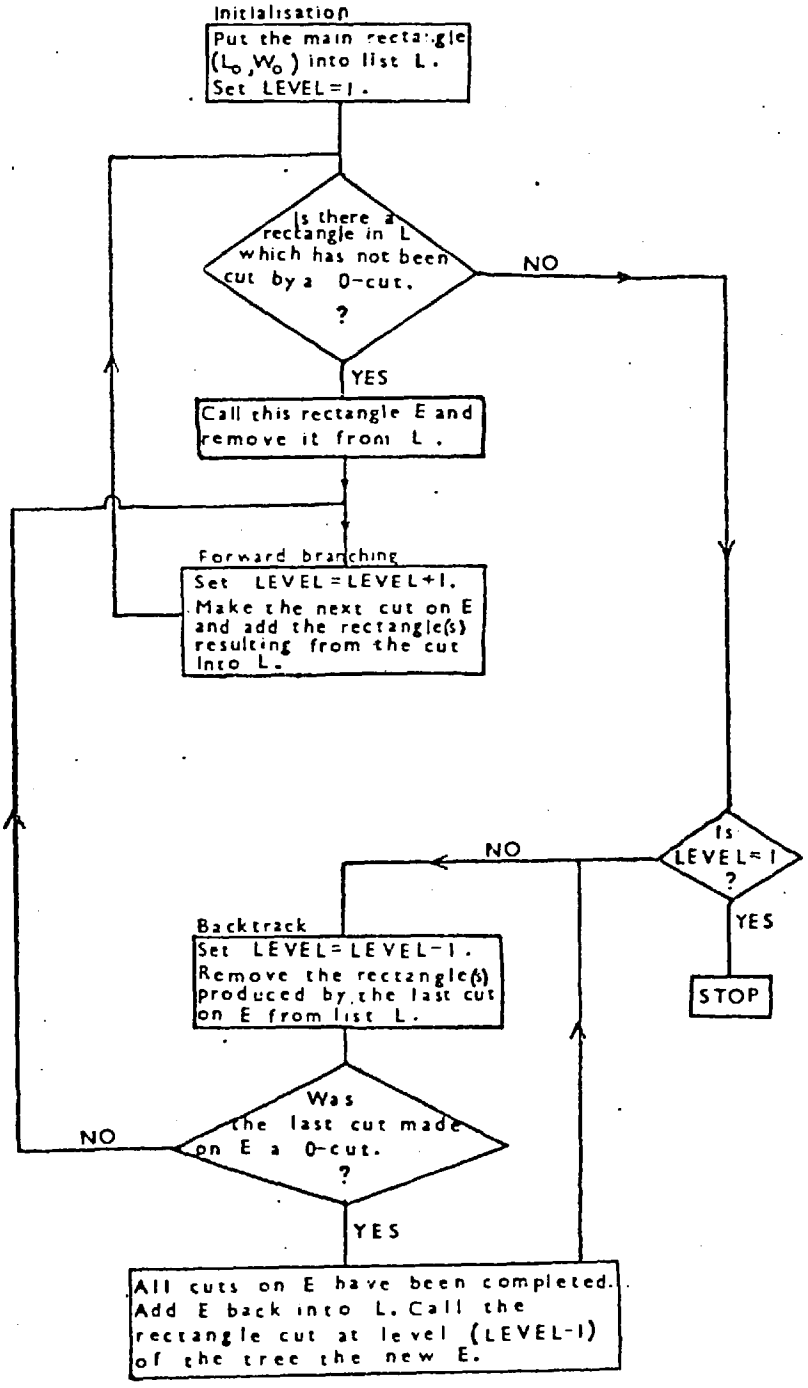


Figure 5. Flow chart of the algorithm

an upper bound on the value obtainable from the rectangles in H_0 can be obtained at any node of the tree by allocating pieces to these rectangles from the set R in an optimal fashion. This allocation can be done quite simply as follows:

Form a matrix $[a_{ik}]$ with m rows corresponding to the pieces in R and u columns corresponding to the u (say) rectangles (p_k, q_k) in H_0. Set $a_{ik} = v_i$, if $\ell_i \leq p_k$ and $w_i \leq q_k$ and $a_{ik} = -\infty$ otherwise.

The best feasible allocation of pieces to rectangles is then given by a solution to the transportation problem:

$$\max z = \sum_{k=1}^{u} \sum_{i=1}^{m} a_{ik} x_{ik} \tag{7}$$

$$\sum_{i=1}^{m} x_{ik} \leq 1 \tag{8}$$

$$\sum_{k=1}^{u} x_{ik} \leq b_i \tag{9}$$

$$x_{ik} \geq 0 .$$

The solution to this problem is made very simple by the special structure of the $[a_{ik}]$ matrix, and the problem can be solved by a particularly efficient version of the transportation routines [2].

The rectangles in L that have not had O-cuts made on them are liable, at future branchings, to be cut further into smaller rectangles and hence may be allocated several pieces from R in the final solution. It is therefore not possible to use the transportation routine to calculate an upper bound for these rectangles, as this only allocates one piece per rectangle. However, an upper bound on the value obtainable from each rectangle in $L - H_0$ can be derived by solving, for every such rectangle, the unconstrained two-dimensional cutting problem using the first of the dynamic programming procedures given by Gilmore and

Gomory [8]. The second method given in [8] is incorrect, as shown in [11].

The procedure for calculating an upper bound at any node is therefore to solve the transportation problem for all rectangles in H_0 and to solve the unconstrained two-dimensional cutting problem for all other rectangles.

If at any node the value z^* of the upper bound is greater than the value $\bar{z}$ of the best solution so far, and if the number of each piece i used to generate this value is not greater than the constraint b_i, then an improved solution has been obtained. z^* can then replace $\bar{z}$, and the backtracking can occur. If on the other hand $z^* \leq \bar{z}$, then further branching from the current node can be discontinued and backtracking can again take place.

If neither of these cases occurs, then forward branching can take place until a terminal node is reached (i.e., a node at which all rectangles in the list L have had 0-cuts made on them), in which case the value of the solution of the transportation problem is the value of the cutting pattern corresponding to that node.

We note that the solution to the unconstrained problems need not be calcualted at each node but that a single dynamic programming table corresponding to the solution for the initial rectangle (L_0, W_0) and calculated once at the beginning of the tree search could be used to obtain directly the unconstrained solutions to any rectangle (p,q), $1 \leq p \leq L_0$, $1 \leq q \leq W_0$ in the list L. The major part of the computation of the upper bound is therefore the solution of the transportation problem of (3), (4) and (5), and this solution need take place only at nodes resulting from some 0-cut, i.e., only when the set H_0 of rectangles changes.

4.2 Branching Strategies

The choice of which rectangle from the list L of available rectangles is to be cut has been left unspecified. There are many ways in which this rectangle can be chosen, and a number of these have been tried.

(i) One simple method is to select that rectangle n with the minimum x-dimension p_n and if more than one such rectangle exists, then choose among these the one with the smallest y-dimension q_n. This corresponds to choosing the "smallest" rectangle; or similarly, the "largest" rectangle could be chosen.

(ii) An alternative method is to select that rectangle for which the unconstrained problem gives the highest value. This is another simple and computationally inexpensive method since the value of the unconstrained solution is used in the calculation of the bounds.

(iii) A slightly more complex branching strategy that aims to obtain a feasible solution early on in the search is as follows. At each node of the tree the number r_i' of pieces of type i that have been allocated to rectangles by the transportation subroutine is available. Also available is the number r_i'' of pieces of type i used by all the unconstrained dynamic programming solutions to the rectangles still available for cutting (i.e., the ones that have not as yet had an O-cut made on them). If $r_i = r_i' + r_i''$, then obviously $r_i > b_i$ for at least one $i = 1,\ldots,m$; otherwise, a feasible solution would have been obtained and backtracking would have occurred. Let i* be that piece i for which $(r_i - b_i)$ is maximum.

It is then reasonable to try to reduce the number of pieces used for any type i for which $r_i > b_i$ and in particular that of type i* since this would produce the largest step toward feasibility. With this in mind, one could then choose to cut the rectangle that uses the largest number of pieces of type i* in the unconstrained solution. One hopes that, after this rectangle is cut, fewer pieces of type i* will be used.

The computational results given in the next section have been obtained from a computer program using branching strategy (iii).

5. Computational Results

The algorithm described in the last two sections was tested on a large number of randomly generated two-dimensional cutting problems and, as can be seen from Table 1, the method can be used to solve practical problems of quite reasonable size. This algorithm has solved actual wood cutting problems in the manufacture of furniture.

The random problems were produced as follows. Given the initial rectangle A_0 with area $\alpha_0 = L_0 \cdot W_0$, then m random numbers α_i, $i = 1,\ldots,m$, corresponding to the areas of the m rectangles in R, were generated by sampling from the uniform distribution in the range 0 to $0.25\ \alpha_0$. The x dimensions ℓ_i of the rectangles in R were generated by sampling from the uniform distribution in the range 0 to α_i and rounded upward to the nearest integer, and the y dimensions w_i were then calculated using the formula $w_i = \alpha_i/\ell_i$, again rounding the number upward.

The constraints b_i on each piece in R were deliberately chosen so that the unconstrained solution obtained from the dynamic

program was infeasible. Table 1 gives details of the size of the problems tested and the time needed for their solution. The computer code was written in FORTRAN IV for the CDC 7600 computer and was run using the FTN compiler of that machine. The total memory requirements of the code used for the solution of all problems in Table 1 was 30 K words.

Table 2 gives the exact details of three of the problems solved, including the sizes, values, and constraints of the pieces in R, and the values of the constrained and unconstrained solutions. Figure 6 shows the cutting pattern that gives the constrained optimal solution to problem 2 in Table 2.

Table 1

Performance of Algorithm

m	(L_0, W_0)	No. of problems solved	Times (CDC 7600 sec)			No. of nodes in tree		
			Max.	Av.	Min.	Max.	Av.	Min.
5	(40, 70)	4	1.1	0.96	0.79	1141	835	654
5	(53, 65)	3	0.79	0.73	0.76	194	131	85
5	(50, 100)	6	2.32	1.60	1.31	1943	1024	170
6	(15, 10)	4	14.31	7.15	1.25	11051	5310	1085
7	(40, 70)	4	15.23	7.48	1.18	18013	8866	1109
10	(40, 70)	6	25.43	15.29	0.7	18602	12175	230
20	(40, 70)	5	181.0	130.18	66.14	57284	38807	22184

Table 2

Details of Three Test Problems

Problem No. 1 : Initial rectangle A_0 = (15, 10), m = 7

Details of pieces:

Dimension vector: $[\ell_i, w_i]$ = [(8,4), (3,7) (8,2), (3,4), (3,3), (3,2), (2,1)]
Value vector: $[v_i]$ = [66, 35, 24, 17, 11, 8, 2]
Constraint vector: $[b_i]$ = [2, 1, 3, 5, 2, 2, 1]
Solution value: constrained by $[b_i]$, 244 (unconstrained, 249)
Solution time: 2.47 sec. Number of nodes in tree: 3,794

Problem No. 2 : Initial rectangle A_0 = (40, 70), m = 10

Details of pieces

Dimension vector: $[\ell_i, w_i]$ = [(21,22), (31,13), (9,35), (9,24), (30,7), (11,13), (10,14), (14,8), (12,8), (13,7)]
Vector vector: $[v_i]$ = [582, 403, 315, 216, 210, 143, 140, 110, 94, 90]
Constraint vector: $[b_i]$ = [1, 1, 3, 3, 2, 3, 1, 3, 3, 3]
Solution value: constrained by $[b_i]$, 2892 (unconstrained, 3006)
Solution time: 24.07 sec. Number of nodes in tree: 18,602

Problem No. 3 : Initial rectangle A_0 = (40, 70), m = 20

Details of pieces

Dimension vector: $[\ell_i, w_i]$ = [(31,43), (30,41), (29,39), (28,38), (27,37), (26,36), (25,35), (24,34), (33,23), (22,32), (31,21), (29,18), (17,27), (15,24), (16,25), (15,24), (23,14), (21,12), (19,11), (9,17)]
Value vector: $[v_i]$ = [500, 480, 460, 440, 420, 410, 400, 380, 360, 340, 320, 300, 280, 240, 260, 240, 220, 180, 160, 140]
Constraint vector: $[b_i]$ = [4, 2, 4, 4, 3, 4, 3, 4, 4, 3, 3, 3, 2, 2, 4, 1, 4, 3, 4, 1]
Solution value: constrained by $[b_i]$, 1860 (unconstrained, 2240)
Solution time: 66.14 sec. Number of nodes in tree: 22,184

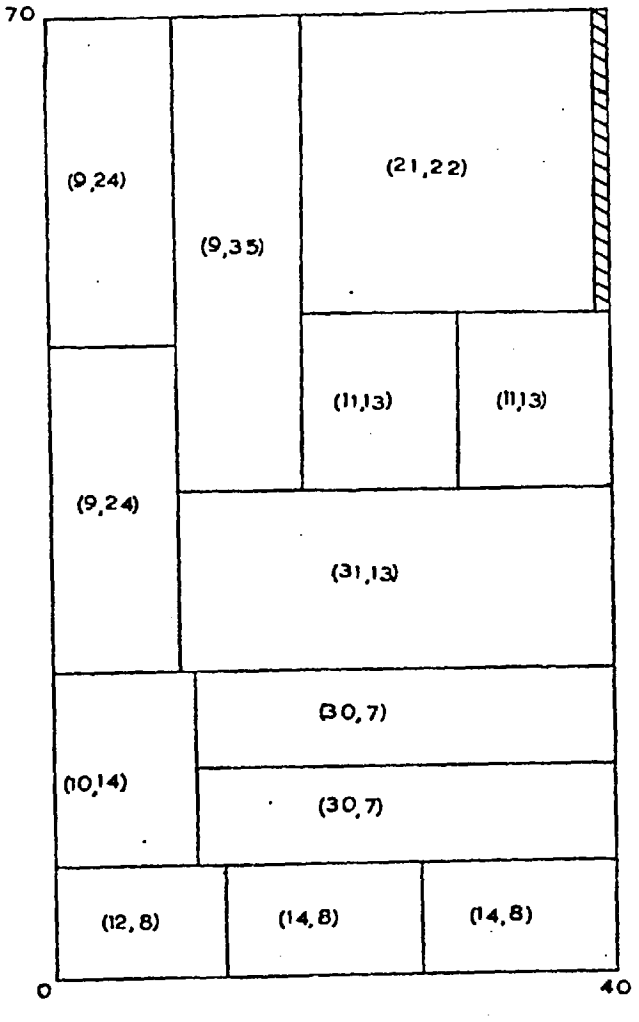


Figure 6. Optimal cutting pattern for Problem 2

REFERENCES

1. R. ART, "An Approach to the Two-Dimensional, Irregular Cutting Stock Problem", I.B.M. Cambridge Scientific Centre Report No. 320-2006, 1966.

2. J.F. DESLER, AND S.L. HAKIMI, "A Graph-Theoretic Approach to a Class of Integer-Programming Problems", Opns. Res. 17, 1017-1033 (1969).

3. S. EILON AND N. CHRISTOFIDES, "The Loading Problem", Management Sci. 17, 259-268 (1971).

4. L.F. ESCUDERO AND E. GARBAYO, "The Cutting Stock Problem : Application of Combinational Techniques and Mixed Integer Programming", presented at 8th Mathematical Programming Symposium, Stanford University, August 1973.

5. P.M. GHARE AND L.E. WALTERS, "A Branch-and-Bound Algorithm for the Multi-Dimensional Knapsack Problem", presented at a joint meeting of the 33rd national meeting of the Operations Research Society of America, and American meeting of the Institute of Management Science, 1968.

6. P.C. GILMORE AND R.E. COMORY, "A Linear Programming Approach to the Cutting-Stock Problem", Opns. Res. 9, 849-859 (1961).

7. P.C. GILMORE AND R.E. GOMORY, "Multistage Cutting Problems of Two and More Dimensions", Opns. Res. 13, 94-120 (1965).

8. P.C. GILMORE AND R.E. GOMORY, "The Theory and Computation of Knapsack Functions", Opns. Res. 15, 1045-1075 (1967).

9. H. GREENBERG AND R.L. HEGERICK, "A Branch Search Algorithm for the Knapsack Problem", Management Sci. 16, 327-332 (1970).

10. S. HAHN, "On the Optimal Cutting of Defective Glass Sheets", I.B.M. New York Scientific Center Report No. 320-2916, 1967.

11. J.C. HERZ, "A Recursive Computing Procedure for Two-Dimensional Stock Cutting", I.B.M. J. Res. Dev. 16, 462-469 (1972).

12. G. INGARGIOLA AND J. KORSCH, "Reduction Algorithm for 0-1 Single Knapsack Problems", Management Sci. 20, 460-463 (1973).

13. H.M. SALKIN AND C.A. DeKLUYVER, "The Knapsack Problem : A Survey", Nav. Res. Log. Quart. 22, 127-144 (1975).

APPENDIX

Let c_i be the number of pieces of type i in R used in the upper bound calculation described in Section 3. Thus c_i is the sum of the number of pieces used in the solution of the transportation problem for H_0 and the number of pieces used in the dynamic programming solution for the remaining rectangles. The description of the complete algorithm, including the calculation of the bounds, is then as follows:

Initialization:

1.1 Set: $n = 1$, $L = \{(L_0, W_0, 1, 1)\}$, $\bar{z} = 0$.

Calculation of the bound:

2.1 Calculate the value of z^* and c_i, $i = 1, \ldots, m$ for the list L.

2.2 If $z^* \leq \bar{z}$, set $K = 0$ and go to 6.1; otherwise, continue.

2.3 If $b_i \geq c_i$, for all $i = 1, \ldots, m$, set $\bar{z} = z^*$, $K = 0$ and go to 6.1; otherwise, continue.

Choose a rectangle for cutting:

2.4 Choose a rectangle (p_n, q_n, x_n, y_n) from the list L with $x_n \neq 0$. If none exists set $K = 0$ and go to 6.1; otherwise, set $K = 1$ and go to step 2.5.

2.5 Set $X_n = x_n$ and $Y_n = y_n$.

2.6 Set $Q(n) = (p_n, q_n, x_n, y_n, X_n, Y_n)$ and $L = L - \{(p_n, q_n, x_n, y_n)\}$.

Forward branching (x-cut):

3.1 Set $x = x_n$.

3.2 If $x \geq [p_n/2] + 1$, set $x_n = x + 1$ and go to 4.1.

3.3 If $x \in S^{q_n}$, go to 3.4; otherwise, set: $x = x + 1$ and go to 3.2.

3.4 Set $L = L \cup \{(x, q_n, x, 1), (p_n - x, q_n, x, 1)\}$.

3.5 Set $x_n = x + 1$, $n = n + 1$, and go to 2.1.

Forward branching (y-cut):

4.1 Set $y = y_n$.

4.2 If $y \geq [q_n/2] + 1$, go to 5.1.

4.3 If $y \in T^{p_n}$ go to 4.4; otherwise, set $y = y + 1$ and go to 4.2.

4.4 Set $L = L \cup \{(p_n, y, 1, y), (p_n, q_n - y, 1, y)\}$.

4.5 Set $y_n = y + 1$, $n = n + 1$ and go to 2.1.

Forward branching (0-cut):

5.1 Set $L = L \cup \{(p_n, q_n, 0, y_n)\}$.

5.2 Set $x_n = 0$, $n = n + 1$ and go to 2.1.

Backtracking:

6.1 If $n = 1$, stop; all normal cutting patterns have been generated.

6.2 If $K = 0$, go to 6.4.

6.3 $L = L \cup \{(p_n, q_n, X_n, Y_n)\}$.

6.4 Set $n = n - 1$.

6.5 If $x_n > [p_n/2] + 1$, go to 6.5.2. If $x_n = 0$, go to 6.5.3, otherwise, continue.

(Last cut made was an x-cut):

6.5.1 Set $L = L - \{(x_n - 1, q_n, x_n - 1, 1), (p_n - x_n + 1, q_n, x_n - 1, 1)\}$ and go to 3.1.

(Last cut made was a y-cut):

6.5.2 Set $L = L - \{(p_n, y_n - 1, 1, y_n - 1), (p_n, q_n - y_n + 1, 1, y_n - 1)\}$ and go to 4.1.

(Last cut made was a 0-cut):

6.5.3 Set $L = L - \{(p_n, q_n, 0, y_n)\}$, $K = 1$ and go to 6.1.

CHAPTER 4

SURVEY OF VULNERABILITY

SURVEY OF VULNERABILITY

1. Introduction

Graph theoretic techniques provide a useful tool for the investigation of many aspects of a communication network. This chapter investigates the vulnerability properties of a network.* Problems of vulnerability have received a great deal of attention in the literature and problems involving the analysis and synthesis of directed and undirected networks with minimal vulnerability have been extensively investigated because of the many practical applications in the fields of communications and computer networks design.

1.1 Graph Theory Basic Definitions

Before introducing some definitions on connectivity and vulnerability we will introduce some basic definitions from graph theory. Definitions not given here are as used in [9].

A graph $G = (X, A)$ consists of a set of n vertices $X = \{x_1, x_2, \ldots, x_n\}$+ and a set of m arcs $A = \{a_1, a_2, \ldots, a_m\}$.+ If for each arc $a_k = (x_{i_k}, x_{j_k})$ the pair of vertices is ordered, i.e. each arc has a direction shown by an arrow, then the graph G is a 'directed' graph, if the arcs do not have an implied direction then the graph is a 'nondirected' one. Any nondirected graph can be converted into a directed one by replacing each arc represented by an unordered pair (x_i, x_j), by two directed arcs (x_i, x_j), (x_j, x_i). If (x_i, x_j) exists, x_i and x_j are said to be adjacent

---

* In this chapter and the following ones we use the terms 'graph' and 'network' interchangeably.

\+ For brevity, vertices and arcs are referred to by subscript in certain sections of the thesis. The meaning will be clear from the text.

vertices and the arc (x_i, x_j) is incident at these vertices. An arc will be denoted either by a_k, meaning the kth arc, or by (x_i, x_j). It will be assumed for the sake of simplicity that graphs do not have parallel arcs, i.e. (x_i, x_j) is unique, nor self-loops, i.e. $(x_i, x_j) \notin A$. Most of the theory of vulnerability has been developed for nondirected graphs and the current description will concentrate mainly on these graphs. However some of the methods for analysis of vulnerability are also applicable to directed graphs as there are many similarities between the analysis of vulnerability and network flow problems.

For a nondirected graph the set of vertices adjacent to a vertex x_i is denoted by $\Gamma(x_i)$. The degree of x_i is denoted by $d_i = |\Gamma(x_i)|$, as each arc is incident at exactly two vertices then clearly the following relation must be true

$$\sum_{i=1}^{n} d_i = 2m \qquad (1)$$

A nondirected graph G is said to be regular if all its vertices have the same degree i.e. $d_i = d$ for all i. From (1), (for a regular graph), $d = \frac{2m}{n}$. A graph is said to be complete if every vertex has an arc going from it to every other vertex. A sequence of adjacent arcs (x_{i_1}, x_{i_2}), (x_{i_2}, x_{i_3}), ... $(x_{i_k}, x_{i_{k+1}})$ is called a path, which can also be represented by a sequence of adjacent vertices, e.g. $(x_{i_1}, x_{i_2}, x_{i_3}, \ldots x_{i_{k+1}})$. A path is called simple if it does not use the same arc more than once and elementary if it does not use the same vertex more than once. Two paths both going from vertex x_i to vertex x_j are said to be arc disjoint if they have no arcs in common, and vertex disjoint if they have no vertices in common except x_i and x_j.

In a graph G = (X, A) a vertex x_i is said to be reachable from a vertex x_j if there exists a path from vertex x_j to vertex x_i. A graph is said to be connected if (ignoring the directions of arcs)

every vertex is reachable from every other vertex, if this is not the case then it is said to be disconnected.

A partial graph of a graph $G = (X, A)$ is a graph $G_p = (X_p, A_p)$ with $X_p = X$ and $A_p \subseteq A$, i.e. it has the same set of vertices as G but only a subset of its arcs. A subgraph of a graph $G = (X, A)$ is a graph $G_s = (X_s, A_s)$ such that $X_s \subseteq X$ and for every $x_i \in X_s$:

$$\Gamma_s(x_i) = \Gamma(x_i) \cap X_s \ ,$$

thus a subgraph consists of a subset of the vertices of G but contains all arcs in G between those vertices in the subset. G_s is also written as (X_s). Let a graph $G = (X, A)$ be connected. A set of arcs which when removed from the graph make it disconnected is referred to as an "arc cut-set". Similarly, if a set of vertices, when removed from a graph, make it disconnected is referred to as a "vertex cut-set". A set of both vertices and arcs, which when removed from a graph, make it disconnected is called a "mixed cut-set". Note that the removal of a vertex from a graph automatically implies the removal of all arcs incident on that vertex. If the removal of a cut-set makes a vertex x_t no longer reachable from a vertex x_s then the cut-set can be called an "s-t cut-set".

1.2 Measures of Vulnerability - Basic definitions

For the purpose of analysing the vulnerability of a graph G it is normally assumed that the graph is connected and the "destruction" of the network means that it is in some way disconnected, or that the number of arcs in the shortest paths between certain sets of vertices become so long as to render communication impractical. In the literature the main criterion for the "destruction" of a graph has been that the graph becomes disconnected [5, 6, 8, 15, 26, 42].

However, more complicated forms of disconnectedness have been considered, for example Boesch & Thomas [3] define "destruction" for nondirected graphs as that condition when the largest connected subgraph of G left after destruction has less than a specified number of vertices. The prime measure of "destruction" considered in this thesis will be simple disconnectedness. There are three possible ways in which a graph can become disconnected; by damage to its arcs, by damage to its vertices, and by damage to both its arcs and vertices. Corresponding to these three ways of damaging a network three variables can be defined, θ, ω and ν.

In order to develop methods for analysing the various types of vulnerability in a graph it is necessary to define θ, ω and ν more formally. In particular it can be shown that ν is in fact redundant and is always equal to ω. In order to define θ and ω, we first define θ_{st} and ω_{st} for two specific vertices s and t of G = (X, A).

$$\theta_{st} = |\text{The smallest s-t arc cut set}|$$

$$\omega_{st} = |\text{The smallest s-t vertex cut set}| \quad ; \text{ for } (x_s, x_t) \notin A$$

$$\equiv (n-1) \quad ; \text{ for } (x_s, x_t) \in A.$$

The arc connectivity* of a graph G = (X, A) is θ, where

$$\underset{\substack{x_s, x_t \in X \\ s \neq t}}{\text{Min}} [\theta_{st}].$$

The arc vulnerability of G is defined as $1/\theta$. The vertex connectivity* of a graph G = (X, A) is ω, where:

$$\underset{\substack{x_s, x_t \in X \\ x_s \neq x_t}}{\text{Min}} [\omega_{st}]$$

---

* For directed graphs θ and ω give the minimum number of arcs and vertices respectively that have to be removed so that the graph is no longer strongly connected [9, p.52]. It may be unilaterally or weakly connected after the removal of these arcs or vertices.

The vertex vulnerability of G is defined as $^1/\omega$.

θ and ω respectively measure the number of arcs or vertices that have to be removed from a graph to make it disconnected (in the case of a nondirected graph) or partially disconnected (in the case of a directed graph). Thus disconnectedness is used as a criterion for 'destruction' of a graph. Call this criterion (1). A number of other criteria for destruction can be postulated [16, p.301].

(2) The graph G contains at least two components. This is equivalent to (1) for nondirected graphs.

(3) There are no directed $x_{s_k} - x_{t_k}$ paths for specified sets of vertices $\{x_{s_k} \ldots\}$ and $\{x_{t_k} \ldots\}$.

(4) The number of vertices in the largest component of G is less than some specified number. This is the criterion used by Boesch and Thomas [3].

(5) The shortest $x_s - x_t$ path is longer than some specified number.

1.3 Analysis and Sythesis of Vulnerability Problems in Graphs

Two types of graphs are used when vulnerability is considered in networks: deterministic graphs and probabilistic ones. For deterministic graphs the object of analysis is to find the minimum number of arcs and/or vertices that have to be removed in order to damage the graph according to some vulnerability criterion. If we choose $1/\theta$ or $1/\omega$ as the criterion, the problem is to find θ and ω. In the case of the measure used by Boesch and Thomas, (criterion (4)), the problem is to find the least number of vertices or arcs to be removed from G, so that the largest connected component of G resulting after this removal is less than a given number. The object of synthesis is to produce a graph with the least number of arcs or of least cost which will have vulnerability (according to the chosen criterion) less than a given permissible threshold value say V, or

alternatively given a budget B, to synthesize a network of cost less than B with minimal vulnerability.

For probabalistic graphs in which each vertex or arc has a specific probability of failure associated with it, the problem is to investigate the reliability (i.e. probability that a graph will remain connected) of the network according to some criterion, or to design a 'maximally reliable network' subject to constraints on cost etc.

Which type of graph is used depends on the system being modelled. In this thesis only deterministic graphs are considered.

1.4 Properties of Vulnerability Measures

One of the basic theorems to do with connectivity in graphs is due to Menger and Whitney [12, 36, 45]. Let $P^a_{st} \equiv \{p^a_{st}\}$ represent any set of arc-disjoint paths from x_s to x_t and let $P^v_{st} \equiv \{p^v_{st}\}$ be any set of vertex disjoint paths from x_s to x_t. The Menger-Whitney theorem then states:

$$\theta_{st} = \max_{\forall p^a_{st}} |P^a_{st}|$$

$$\omega_{st} = \max_{\forall p^v_{st}} |P^v_{st}|$$

i.e. the value of θ_{st} is equal to the maximum number of arc-disjoint paths from x_s to x_t and the value of ω_{st} is equal to the maximum number of vertex-disjoint paths from x_s to x_t.

A result given by Boesch and Thomas [3] follows from this theorem:

Theorem 1 For any two vertices $x_s, x_t \in X$ with $(x_s, x_t) \notin A$,

$$\omega_{st} \leq \theta_{st}.$$

Proof Since all vertex disjoint paths are edge disjoint, the theorem follows immediately from the Menger-Whitney theorem.

It does not follow from theorem 1 that $\omega \leq \theta$ as θ and ω are obtained by minimising over all pairs of vertices and theorem 1 only holds for adjacent vertices. This result requires additional proof.

Theorem 2 If $G = (X, A)$ is a directed (or nondirected) graph then for any pair of adjacent vertices x_s, x_t there exists a pair of vertices x_i, x_j such that:

$$\omega_{ij} \leq \theta_{st} . \qquad (2)$$

Proof Case (i): $\theta_{st} = n - 1$. For adjacent vertices $\omega_{st} = n - 1$ hence for $i = s$ and $j = t$ relation (2) is true.

Case (ii): $\theta_{st} < n - 1$. Let K be the minimum arc cut set producing θ_{st}. Let K partition G into two subgraphs G' and G'' as shown in figure 1. Clearly $G' \neq \phi$ and $G'' \neq \phi$,

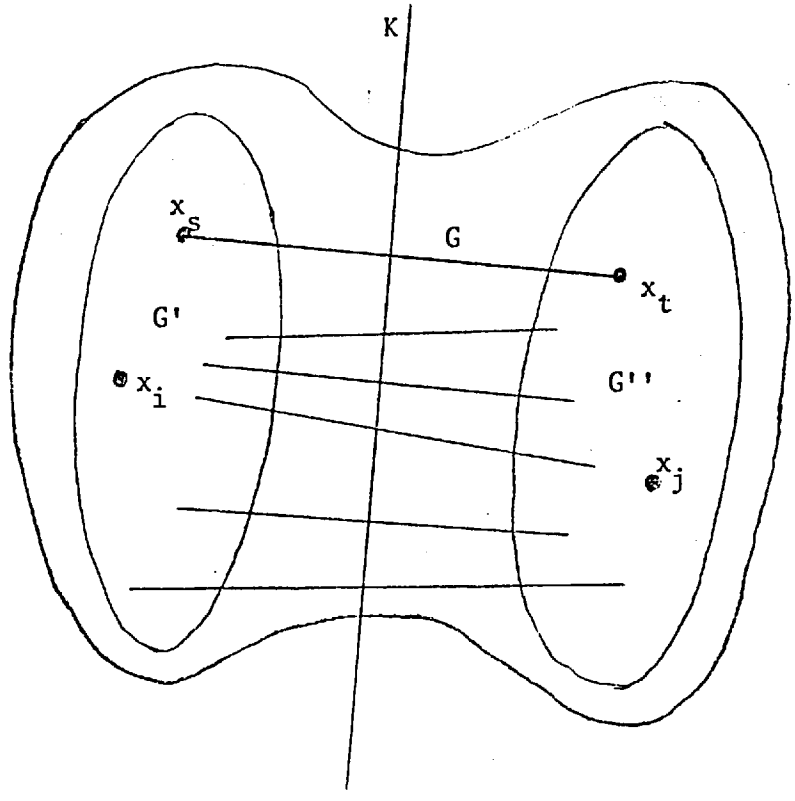


Figure 1 Diagram for proof of Theorem 2

as $x_s \in G'$ and $x_t \in G''$. Also $\exists\ x_i, x_j$ such that $x_i \in G'$, $x_j \in G''$ and $(x_i, x_j) \notin A$; for suppose $\forall x_i \in G'$ and $\forall x_j \in G''$ $(x_i, x_j) \in A$ then $|K| = |G'| \cdot |G''| \geq (n-1)$, but $|K| = \theta_{st} < (n-1)$ hence such x_i and x_j must exist.

Hence, it is possible to make x_j not reachable from x_i by removing θ_{st} end vertices $\{x_k^e\}$ of the arcs in K such that $x_k^e \neq x_i$ or x_j $k = 1\ \theta_{st}$.

Hence, $\omega_{ij} \leq \theta_{st}$ Q.E.D.

Theorem 3 $\omega \leq \theta$

Proof This result follows from theorem 2 and the definitions of ω and θ.

From theorem 3, ω is bounded from above by θ, and it would be of interest to have an upper bound for θ which can be obtained for any graph. For any pair of vertices x_s and x_t, θ_{st} is bounded by the degrees of vertices x_s and x_t as there cannot be more arc disjoint paths between a pair of vertices than there are arcs leaving the vertices. Thus:

$$\theta_{st} \leq \min[d_s, d_t]$$

hence

$$\theta = \min_{\substack{x_s, x_t \in X \\ s \neq t}} [\theta_{st}] \leq \min_{\substack{x_s, x_t \in X \\ s \neq t}} [\min\{d_s, d_t\}]$$

$$= \min_{x_i \in X} (d_i) \leq \frac{1}{n} \sum_{i=1}^{n} d_i = \frac{2m}{n}$$

Thus for any graph G = (X, A) with n vertices and m arcs $\frac{2m}{n}$ is an upper bound on the arc or vertex connectivity of that graph is

$$\omega \leq \theta \leq \min_{x_i\ X} (d_i) \leq \frac{2m}{n} \qquad (3)$$

The above inequality without the last term is often called the 'Whitney inequality', and it has been shown by Chartand and Harary [7]

that for any set of integers a, b, c such that $a \leq b \leq c$ a graph can be constructed such that $a = \omega$, $b = \theta$ and $c = \min_{x_i \in X} (d_i)$.

The existence of upper bounds has led to a number of definitions for types of graphs with vulnerabilities equal to the upper bounds. One, given by Boesch and Thomas [3], is for an optimal damage resistant graph:

Defn: An optimal damage resistant graph is one in which:

$$\omega = \theta = \frac{2m}{n} .$$

They also define an arc-optimal graph as being one in which:

$$\theta = \frac{2m}{n} ;$$

clearly all arc-optimal and optimal graphs are regular. Boesch & Thomas also prove the following theorem.

Theorem 4 Any arc-disjoint union of Hamiltonian circuits is an arc-optimal graph.

However, it does not follow that an arc disjoint union of Hamiltonian circuits is an optimal damage resistant graph. Figure 2 illustrates an arc-disjoint union of Hamiltonian circuits which is not an optimal damage resistant graph. This figure contains

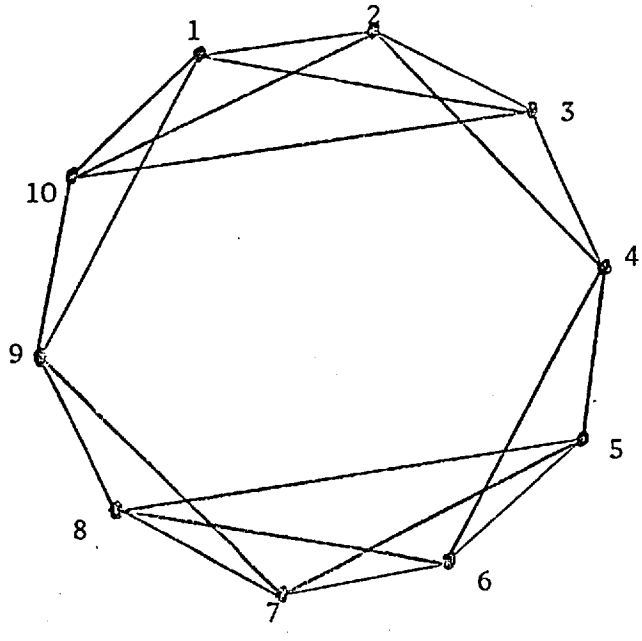


Figure 2 Graph containing two disjoint hamiltonian circuits with $\theta=4$ and $\omega=2$

two disjoint Hamiltonian circuits (and is therefore arc-optimal) with $\theta = 4$, however it only has $\omega = 2$, since the removal of vertices 1 and 6 will make the graph disconnected and is therefore not optimal damage resistant.

Other definitons associated with the bounds on connectivity are:

Defn: A maximally vertex connected graph is one in which:

$$\omega_{ij} = \min\{d_i, d_j\}, \forall_{i,j} (x_i, x_j) \notin A .$$

Defn: A maximally arc connected graph is one in which

$$\theta_{ij} = \min\{d_i, d_j\}.$$

Clearly all maximally vertex connected graphs are maximally arc connected since

$$\theta_{ij} \geq \omega_{ij} \text{ and } \theta_{ij} \leq \min\{d_i, d_j\}.$$

Connectivity in graphs is a subject which has been studied independently of vulnerability theory by many authors, in particular Tutte [43].

One of the prime objectives of vulnerability theory is to facilitate the design of a network with certain vulnerability at minimum cost (or having the minimum number of arcs). It is thus useful to define classes of networks such that the removal of any arc or vertex from that network will increase the vulnerability of that network. Halin [27] has given the following definition.

Defn: A graph G with vertex-connectivity p is said to be arc critical (with respect to vertex connectivity) if the removal of any arc from G reduces its vertex-connectivity to (p-1). Lick [34] has used a similar type of definition for a vertex-critical graph:

Defn: A graph G with vertex-connectivity is said to be vertex-critical (with respect to vertex connectivity) if the removal of any vertex from G reduces it vertex-connectivity to (p-1).

A number of results have been shown for arc and vertex critical graphs. Halin [27] has shown that a necessary and sufficient condition for a p-vertex connected graph G to be arc-critical is that for every pair of adjacent vertices x_s, $x_t \in X$ there are exactly p vertex disjoint s, t-paths in G. In [28] he shows that every finite arc-critical p-vertex connected graph contains more than $\frac{1}{2}\sqrt{(p+1}$ vertices of degree p. Lick [34] has shown that for any vertex-critical p-vertex connected graph the vertex with minimum degree d_{min} has $d_{min} < (3p-1)/2$. Chartrand [6] has shown that for any graph G with n vertices such that $d_{min} \geq \left\lfloor \frac{n}{2} \right\rfloor$* then $\theta = d_{min}$.

2. Methods for determining vulnerability in graphs

The methods used for calculation of the arc vulnerability of a graph are adaptations fo the methods developed for network-flow problems [13]. The methods for determining the vertex vulnerability of a graph have been developed directly. Both methods use the result of the Menger-Whitney theorem stated earlier.

In network flow problems only directed graphs are considered and each arc has a positive number associated with it which represents the capacity of the arc. The maximum flow between any pair of vertices x_s, x_t is equal to the capacity of the minimum s-t cut; the capacity of the cut being the sum of the capacities of the arcs contained in that cut.

2.1 Arc vulnerability

Ford and Fulkerson [13] developed the well known labelling algorithm for finding the maximum flow between any pair of vertices x_s, x_t, which can also be used for finding the arc vulnerability of a graph. If the

---

* $\lfloor a \rfloor$ means the largest integer less than or equal to a.

capacity of each arc is set to one then the value of the minimum flow between a pair of vertices x_s, x_t is equal to the number of arcs crossing the minimum s-t cut, which is equivalent to θ_{st}.

If the algorithm is to be used for a nondirected graph then this must first be transformed into a directed one. This is done by substituting two directed arcs for each nondirected arc, so that a nondirected arc $(\overline{x_s, x_t})$ becomes two directed arcs (x_s, x_t) and (x_t, x_s). To find the value of θ would then require $n(n-1)/2$ iterations of the Ford-Fulkerson network flow algorithm. Alternatively the Gomory Hu [23] algorithm which is only applicable for non-directed graphs can be used and this requires only $(n-1)$ iterations.

2.2 Vertex vulnerability

The vertex vulnerability of a graph G can also be found using the network flow algorithms if a transformation is first carried out on the graph G to produce a graph G'. Let every vertex x_i in G be replaced by two vertices x_i^{in} and x_i^{out} in G' with a directed arc (x_i^{in}, x_i^{out}) with capacity 1 between them. The arcs incident on a vertex x_i in G are divided into two classes, those directed into x_i and those directed out of x_i. In G' the arcs directed into x_i in G are incident on x_i^{in} and those directed out of x_i in G are incident on x_i^{out}. All these arcs are given capacity of value ∞. This transformation is shown in figures 3a and 3b. Fig. 3a shows the original graph G and 3b the graph G' produced by the transformation described above. The capacity of each arc in G' is given in 3b, and the additional arcs introduced by the transformation are shown with double arrows. In order to compute ω_{st} for vertices x_s, x_t in G the maximum flow between x_s^{out} and x_t^{in} is calculated in G', the value of the flow is equal to ω_{st}. The transformation described above does not provide a very efficient method for computing vertex vulnerability as it doubles the number of vertices

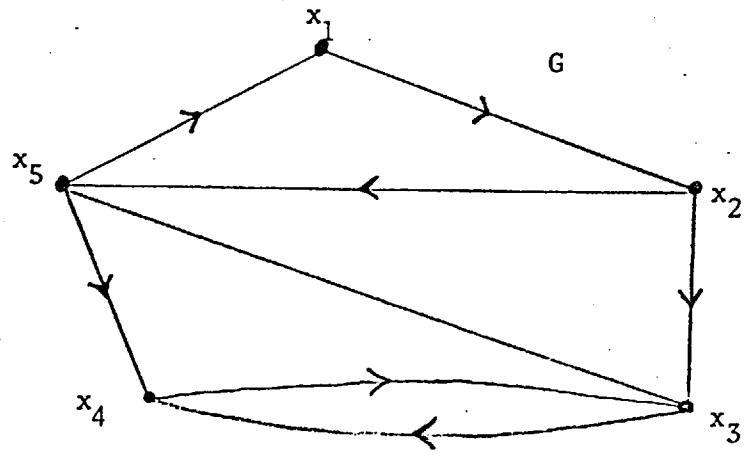


Figure 3(a)

Figure 3 Transformation for finding ω_{st}

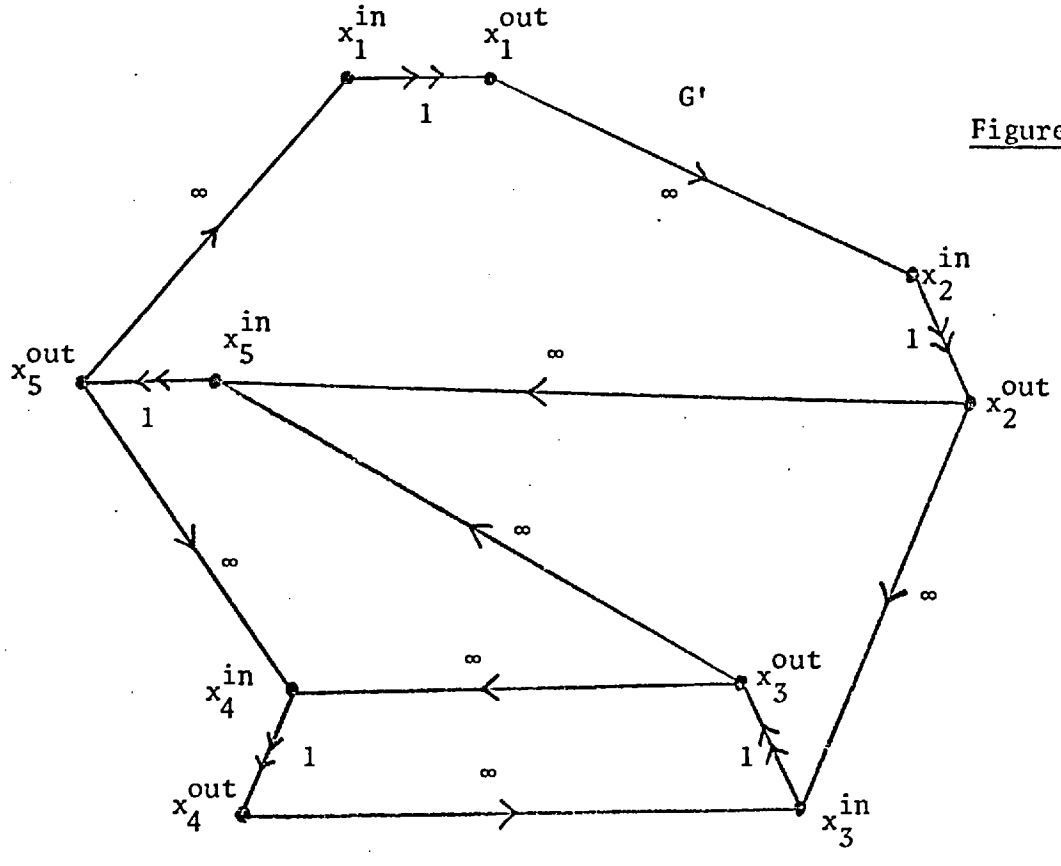


Figure 3(b)

in the graph and increases the number of arcs by n. A direct method for finding ω_{st} for a graph has been developed by Frisch [18] this is based on the Ford and Fulkerson labelling algorithm, but has more and larger labels to supply more information in order to avoid having to do the graph transformations. This is called the ω_{st} labelling algorithm and is described in the Appendix.

2.3 The computation of ω

The calculation of ω is not as straightforward as the calculation of θ using the Gomory Hu algorithm. In the Gomory Hu algorithm the technique of 'vertex condensation' is used. Sets of vertices are condensed for the purpose of the flow calculations and this enables only $n - 1$ flow calculations to be done instead of $\frac{n(n-1)}{2}$. The technique of vertex condensation cannot be used when calculating ω, and hence it is not possible to develop an algorithm for ω in an analogous way to the development of the Gomory-Hu algorithm.

Frisch [20] has shown that the results obtained in calculating the flow from x_s to x_t can be used in calculating the flow from $x_{s'}$ to x_t for any other vertex $x_{s'} \in X$. The technique he developed can be used for directed graphs unlike vertex condensation which can only be used on non-directed graphs. This technique can be used to considerably reduce the effort involved in calculating ω. Frisch [20] has developed an algorithm based on this idea, called the "flow variation algorithm". This algorithm can be adapted to find ω for directed graphs in less time than the direct application of the ω_{st} labelling algorithm in $n(n-1)/2$ iterations.

Kleitman [32] has shown that in order to prove that a nondirected graph G with n vertices has a vertex connectivity $\omega = p$ the maximum number of pairs of vertices (x_s, x_t) for which ω_{st} need be calculated is $p(p-1)/2 + p(n-p)$ out of a total number of pairs which is $n(n-1)/2$.

In the same paper [32] it has also been shown that if for a nondirected graph G

$$\omega_{st_i} = p \qquad i = 1, k \quad \text{for some set of k nodes} \quad x_{t_1} \ldots x_{t_k}$$

which are all successors of some node $x_{t'}$.

Then in order to prove that $w_{st'} = p$ it is only necessary to show that $w_{st'} = p-k$, in the graph formed by removing vertices $x_{t_1} \ldots x_{t_k}$ from G.

3. Synthesis of Invulnerable Networks

The objective of vulnerability synthesis is to develop methods for generating invulnerable graphs according to certain criteria. The prime interest is to produce graphs of least cost, or, if the cost of the arcs is uniform, then produce a graph with the least number of arcs.

The graph to be generated can be considered either from the point of view of arc or vertex connectivity, and can be expressed either as an overall connectivity requirement or as a matrix of s-t vulnerabilities. Thus a generalised connectivity synthesis problem can be expressed as follows:

Problem P: Given a graph G = (X,A) in which every arc $(x_i, x_j) \in A$ has a cost C_{ij} associated with it, and given an integer n × n matrix U of required arc (or vertex) connectivities U_{ij}, find a partial graph $G_p = (X,A_p)$ of G such that

$$\theta_{ij} \text{ (or } \omega_{ij}) \geq U_{ij} \; ; \; \forall \, x_i, x_j; \in X \tag{4}$$

and

$$\sum_{(x_i,x_j)\in A_p} C_{ij} \text{ is a minimum} \tag{5}$$

For problems for which a uniform connectivity is required then the matrix U can be replaced by an integer u and (4) becomes:

$$\theta(\text{or } \omega) \geq u \text{ for } G_p \qquad (6)$$

Using this general problem it is possible to classify most of the work that has been done in connectivity synthesis although some methods fall outside the sphere of this problem definition. For example, [46, 47] has investigated methods for solving the connectivity synthesis problem for uniform cost and uniform vertex connectivity, but with the additional requirement that the diameter (the value of the maximum overall shortest path in a graph) of the graph be less than a prescribed quantity.

In Problem P the required connectivity can be arc connectivity or vertex connectivity, the cost matrix can be uniform or non-uniform and the matrix of required connectivities can be uniform or non-uniform. Hence, it is possible to define eight types of connectivity synthesis problems as follows:

(i) Problem P with uniform cost and uniform arc connectivity. Little work has been done on this problem. However, Boesch and Thomas [3] has shown that if the required arc connectivity is even, then the solution to this problem is composed of $\frac{u}{2}$ edge disjoint Hamiltonian circuits.

(ii) Problem P with uniform cost and uniform vertex connectivity. This problem has been studied by a number of authors including Fulkerson and Shapley [21], Harary [29] and Hakimi [26], and their results are discussed in a later section.

(iii) Problem P with uniform cost and non-uniform arc connectivity. This problem has been studied by Frank and Chow [8, 14, 15], it will be discussed in a later section.

(iv) Problem P with uniform cost and non-uniform vertex connectivity. This problem has not been discussed very much in the literature. The only method that has been developed for its solution is a heuristic described by Wilkov [48]. The problem will be mentioned later.

(v) Problem P with non-uniform cost and uniform arc connectivity. Not much work has been done on this problem, and no optimal algorithms exist in the literature. A heuristic has been suggested by Frisch [19] which will be discussed later. This problem is the subject of the next chapter where an optimal procedure for its solution is developed.

(vi) Problem P with non-uniform cost and uniform vertex connectivity. No work has been done on this problem. However, the heuristic of Steiglitz, Weiner and Kleitman [41] for non-uniform vertex connectivity could be adapted to find an approximate solution. The method developed in the next chapter for non-uniform cost and uniform arc connectivity can also be adapted to solve this problem.

(viii) Problem P with non-uniform cost and non-uniform vertex connectivity. A heuristic has been developed for this problem by Steiglitz, Weiner and Kleitman [41] for finding an approximate solution. It will be discussed in a later section.

3.1 Uniform cost and Uniform Arc Connectivity

Problem P with uniform cost and uniform arc connectivity will be considered in this section. Two possible cases can occur for this problem, (1) when graph G is not complete and (2) when graph G is complete. In order to avoid excessive length we discuss here only the case of non-directed graphs. A similar argument applies to directed graphs.

3.1.1 G not complete

The case when G is not complete has not been considered in the literature. However it will be discussed here as it is interesting to note the complexity of the problem when G is a general graph.

It has been shown by Boesch and Thomas [3] that if the required

connectivity is even then the solution to this problem is composed of $\frac{u}{2}$ edge disjoint Hamiltonian circuits. Hence, one method to solve this problem for u even would be to generate edge disjoint Haniltonian circuits of G. However, it is not sufficient to generate these circuits sequentially in an arbitrary manner since a situation may arise in which at some stage $k < \left|\frac{u}{2}\right|$ when k arc-disjoint Hamiltonian circuits have been enumerated, it may not be possible to generate a $(k+1)^{\underline{st}}$ arc disjoint Hamiltonian circuit, whereas a different choice of the initial k Hamiltonian circuits may permit a $(k+1)^{\underline{st}}$, $(k+2)^{\underline{nd}}$ etc. arc-disjoint, Hamiltonian circuits to be subsequently generated. For example, consider Figure 4a; and suppose that a least cost partial graph with $\theta = 4$ is required. A solution to this problem would be to find two edge disjoint Hamiltonian circuits. One procedure to obtain this solution would be to find one Hamiltonian circuit, remove it from the graph and then look for another one in the remaining graph. One Hamiltonian circuit is (1, 10, 2, 4, 5, 3, 6, 8, 7, 9, 1). The partial graph in which this circuit has been removed from G is shown in Figure 4b, and it is clear that this graph contains no further Hamiltonian circuits. However, the graph in Figure 4a does contain two arc disjoint Hamiltonian circuits, i.e. (1, 2, 4, 3, 5, 7, 6, 8, 9, 10, 1) and (1, 4, 5, 6, 3, 2, 10, 8, 7, 9, 1). Hence in order to find a partial graph with two arc disjoint Hamiltonian circuits a search procedure is required.

3.1.2 G is complete

When G is complete Problem P can be solved by construction. The procedure will not be given in this section but will be postponed until the next section since the same method used to construct a minimal graph with vertex connectivity U - and which is described in the next section - can also be used here, as will now be shown.

Figure 4(a)

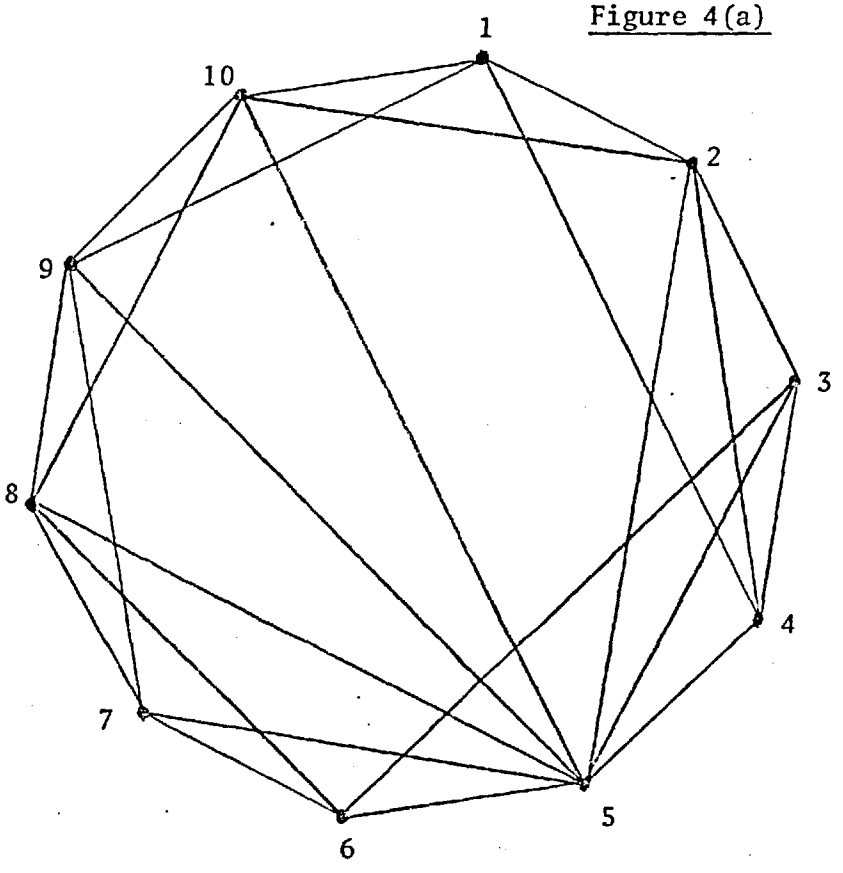
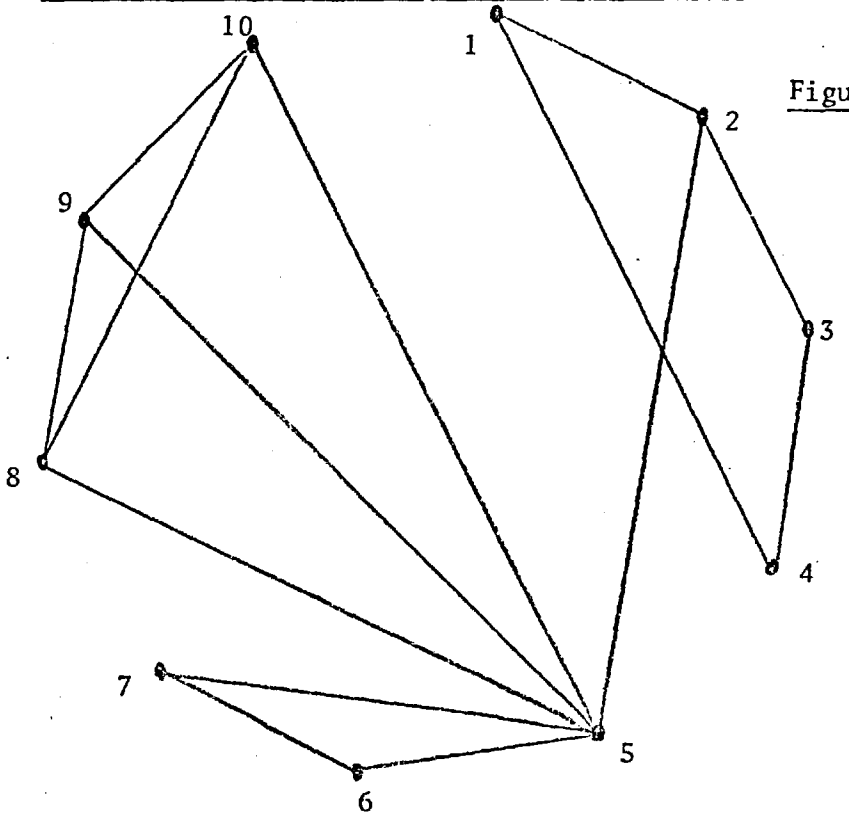


Figure 4 Graph containing two disjoint hamiltonian circuits such that the removal of one particular circuit leaves the graph with no other hamiltonian circuit

Figure 4(b)



From the properties of vertex and arc connectivity described in Section 1.4,

$$\omega \leq \theta \leq d_{min}$$

Hence, since the constructions described in the next section generate graphs with n vertices and with $\omega = d_{min} = u$, they also have $\theta = u$. These constructions generate graphs with $\frac{nu}{2}$ arcs (for nu even) and $\frac{nu+1}{2}$ arcs (for nu odd). It only remains to show that these constructions generate graphs with numbers of arcs equal to the lower bound on the number of arcs in a graph with n vertices and arc connectivity u.

The minimum degree of the solution graph must be u, and to have the least number of arcs every vertex should have degree u, i.e. the minimum number of arcs must be $\left(\frac{nu}{2}\right)$. If nu is even $\left(\frac{nu}{2}\right)$ is integer and is hence the lower bound on the number of arcs. If nu is odd $\frac{nu}{2}$ is fractional and hence the next largest integer i.e. $\left(\frac{nu+1}{2}\right)$ is the lower bound.

3.2 Uniform cost and Uniform Vertex Connectivity

In this section we consider Problem P (defined at the beginning of Section 3) for the case of uniform cost and uniform vertex connectivity. In addition we restrict our attention to the case where graph G - in the definition of Problem P - is a complete graph. The case when the required minimum cost graph G_p must be a partial graph of an arbitrary given graph G is not considered in the literature and will not be mentioned here.

3.2.1 Nondirected Graphs

The problem to be discussed in this section is how to generate graphs with a specified vertex connectivity $\omega = u$ at minimum cost. As the costs are uniform this means with least number of arcs. The problem can be restated as: Given a set X of n vertices to be connected with

vertex connectivity u, find the minimum number of arcs m_{min} and generate a graph $G_m = (X, A_{min})$ with m_{min} arcs, so that $\omega = u$ for G_m.

A lower bound on m_{min} can be obtained from equation (3) as:

$$m_{min} \geq \frac{\omega n}{2} \qquad (7)$$

Two possibilities arising from (7) are:

Case 1 n.u is even in which case $m_{min} = \frac{nu}{2}$ and G_{min} is a regular graph of degree U.

Case 2 n.u is odd in which case $m_{min} = \frac{nu+1}{2}$ and G_{min} is not regular.

A. Case 1 (nu even) For Case 1 when nu is even, Harary [29] and Hakimi [26] describe two classes of optimal damage resistant graphs which are solutions to the problem:

(i) The first class is for the case when the required vertex connectivity u is even (in which case n could be either odd or even), in this case vertex x_i is adjacent to all vertices with indices $j = i \pm k + n(\text{modn } n)$* where $1 \leq k \leq u/2$. An example of such a graph for $u = 4$ and $n = 10$ is given in Figure 5. For this class of graphs the diameter is [n/u], where [a] means the smallest integer greater than or equal to a, the diameter in Figure 5 is 3.

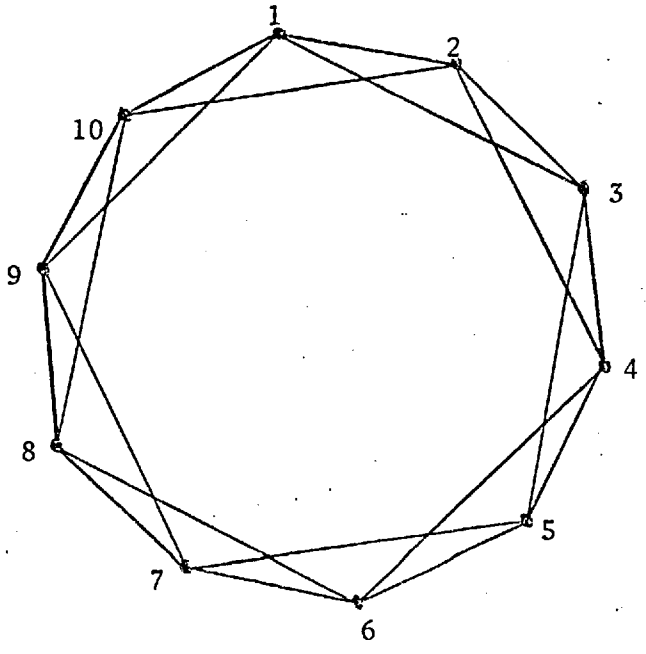


Figure 5 An optimal damage resistant graph for u=4 and n=10

*Footnote (modn n) is equivalent to (modulo n) if the number (modulo n) is non-zero. For (modulo n) equals zero (modn n) equals n.

(ii) The second class of graphs is when u is odd (and hence n is even), in this case any vertex x_i is adjacent to all vertices with indices $j = i \pm k + n(\text{modn } n)$ where $1 \leq k \leq u/2$ and also $j = (i + n/2)$ (modulo n), an example of this type of graph is given in Figure 6, where u = 3 and n = 8. The diameter for this class of graph is $[n/2(u-1)]$. In Figure 6 the diameter can be seen to be 2.

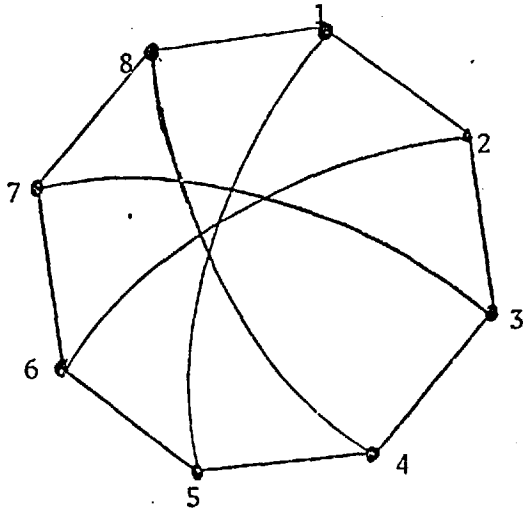


Figure 6 An optimal damage resistant graph for u=3 and n=8

B. Case 2 (nu is odd) For Case 2 in which nu is odd it is not possible to generate an optimal damage resistant graph, since the optimal graph for this problem is not regular. The procedure for generating the optimal graph G_{min} is given in Harary [29], Hakimi [26], and Fulkerson and Shapley [21], and is as follows:

The regular graph for n vertices and required vertex connectivity (u-1) is first produced, then all vertices x_i, x_j such that $|i-j| = (n-1)/2$ are joined by arcs. The resulting graph has n vertices of degree u and one of degree (u+1). An example of this type of graph for n = 7 and u = 3 is shown in Figure 6, in this figure all vertices except vertex 4 have degree 3 and vertex 4 has degree 4.

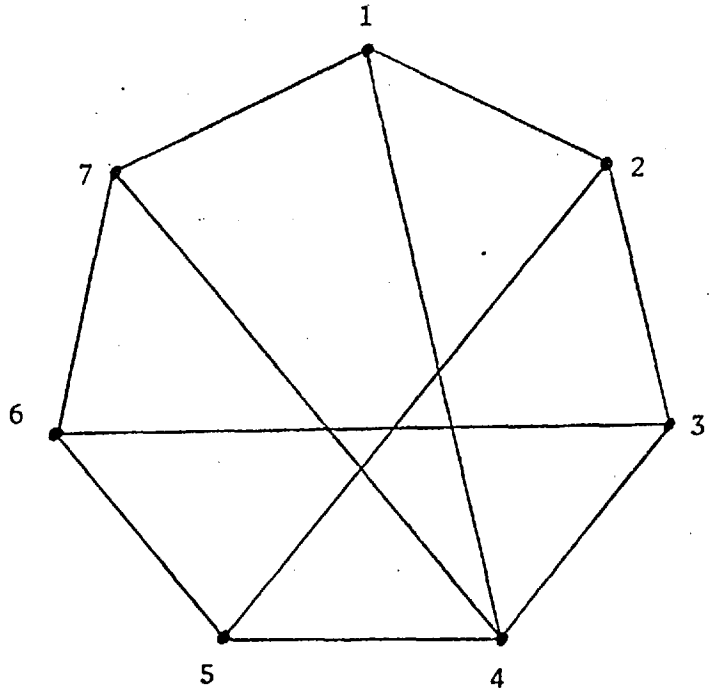


Figure 7. An optimal graph for u=3 and n=7

3.2.2 Directed graphs

The generation of optimal directed graphs for this problem has been studied by Ayoub and Frisch [1]. These graphs are very similar to the non-directed graphs in the last section. Thus, a directed graph of n vertices and required vertex connectivity u is generated by making each vertex x_i adjacent to vertices with indices j = (i+k) (modulo n) where $1 \leq k \leq u$. An example of an optimal graph with u = 2 and n = 10 is shown in Figure 8, note that this is similar in structure to Figure 5, which is an optimal non-directed graph for n = 10 and u = 4.

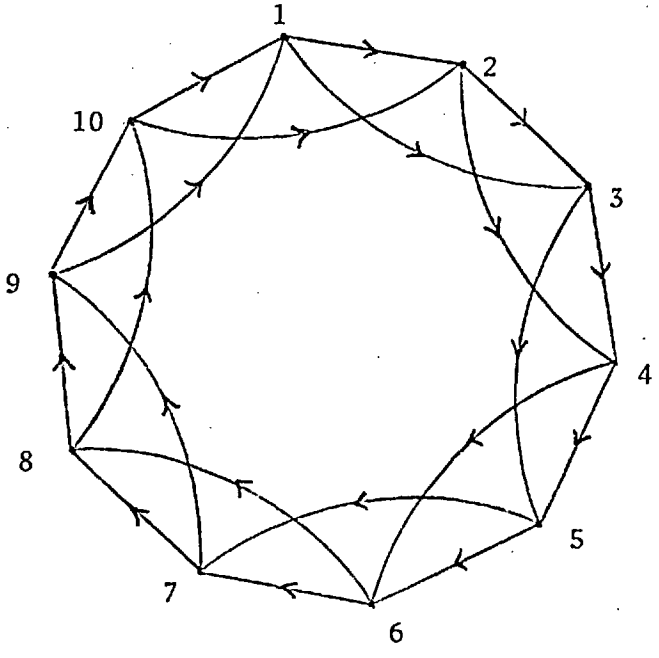


Figure 8. An optimal directed graph for u=2 and n=10

This construction also works for the case of u odd and n even and nu odd, for example consider Figures 9(a) and 9(b). Figure 9(a) contains an optimally invulnerable graph for n = 8 and u = 3, and Figure 9(b) contains one for n = 7 and u = 3.

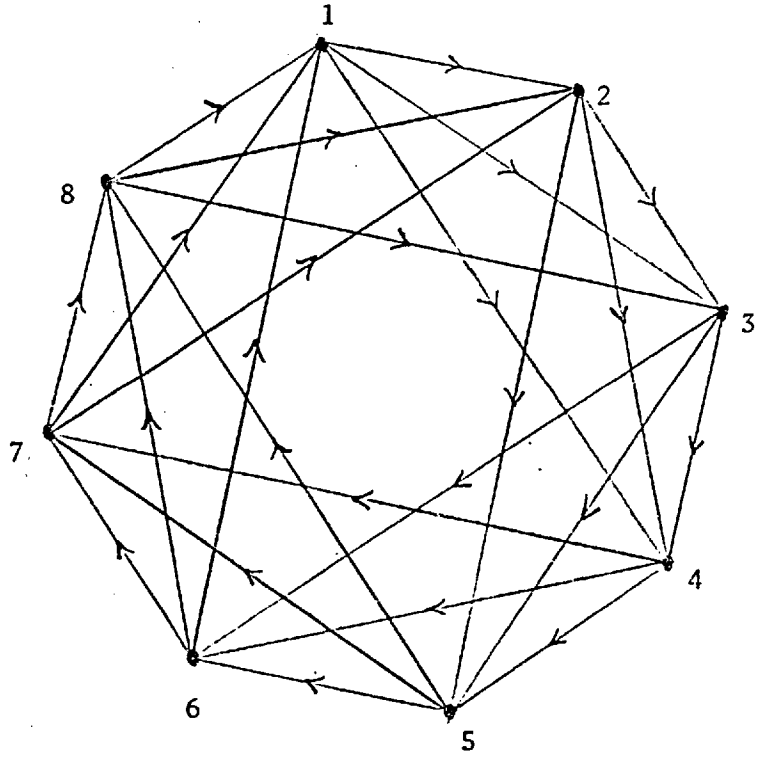


Figure 9(a). An optimal directed graph for u=3 and n=8

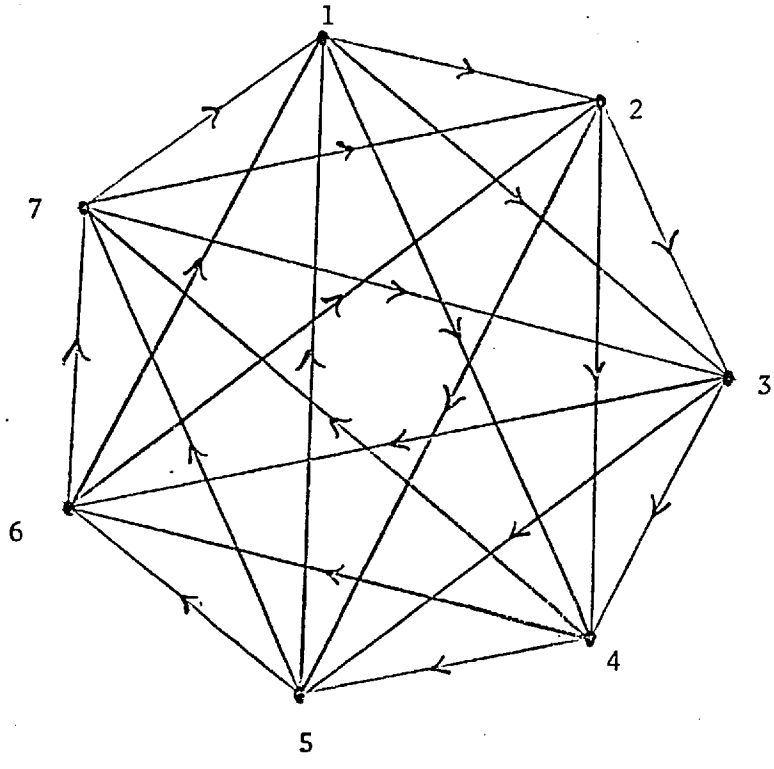


Figure 9(b). An optimal directed graph for u=3 and n=7

3.2.3 Bipartite Graphs

The generation of optimal bipartite graphs has been studied by Boesch and Thomas [3]. A bipartite graph is a nondirected graph $G = (X,A)$ in which the set of vertices X can be partitioned into two distinct sets of vertices X_1, X_2 such that the set of arcs A only connects vertices in X_1 to vertices in X_2. They have shown that for n even an optimal bipartite graph can be generated by setting vertex x_i adjacent to vertices x_j such athat $j = (i + 2k - 1)$ (modulo n) for every k, $1 \leq k \leq u$. An example of an optimum bipartite graph for $u = 3$ and $n = 10$ is shown in Figure 10, the two distinct sets of vertices are the odd vertices and the even vertices, no odd vertex is adjacent to another odd vertex and similarly for even vertices. The approximate diameter of this type of bipartite graph has been shown by Wilkov [46] to be $[(n+1)/2(u-1)]$.

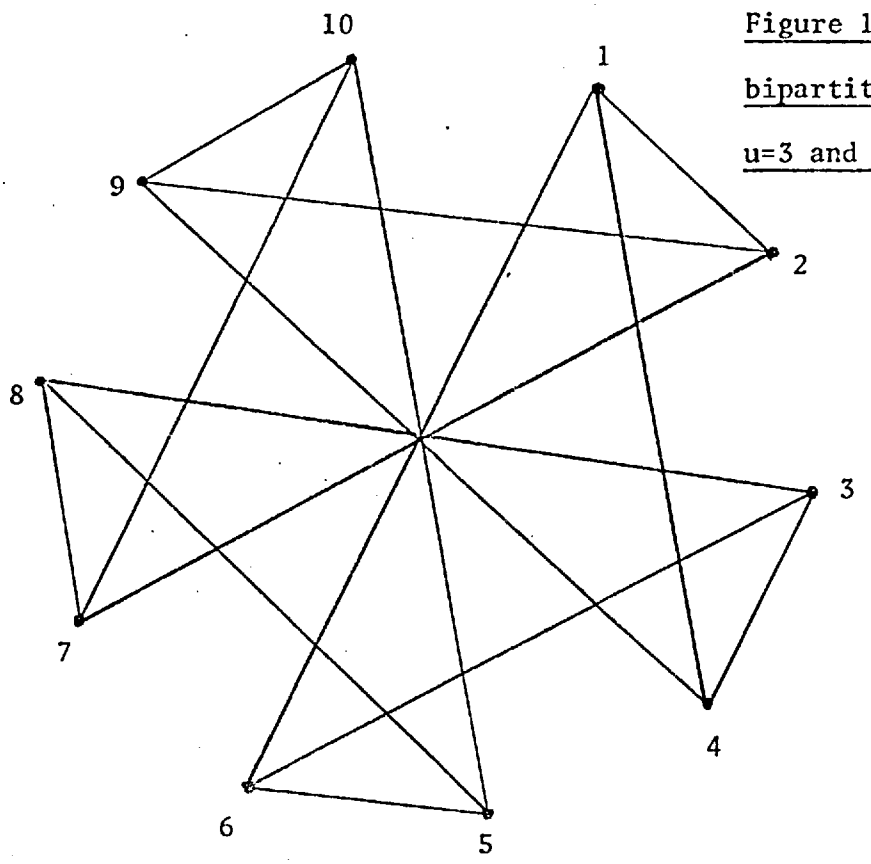


Figure 10. An optimal bipartite graph for u=3 and n=10

3.2.4 Diameters and Optimal Graphs

The above methods generate optimal graphs for the solution of the general vulnerability synthesis problem with uniform cost and uniform vertex connectivity. As metnioned previously they pay no attention to the diameter of the graph generated. It is desirable for some purposes to generate solution graphs with as small a diameter as possible, for example in a communications network if messages have to travel through a large number of intermediate vertices then it is possible that errors may occur. A number of people have investigated diameters in graphs [17, 31, 33, 38, 40, 46, 47]. Friedman [17] and Korn [33] have generated a particular class of optimal damage resistant graphs with n vertices vertex connectivity u = d (the degree of each vertex) and diameter δ for the particular case where n is given by:

$$n = \frac{2d}{d-2}\left((d-1)^{\delta/2} - 1\right); \text{ for } \delta \text{ even and } u > 2 \qquad (8)$$

and

$$n = \frac{2}{d-2}\left(2(d-1)^{\frac{\delta+1}{2}} - d\right) \text{ for } \delta \text{ odd and } u > 2 \qquad (9)$$

Friedman developed the construction for the graphs with even diameter for which n is given in (8) and Korn developed the construction for the case where n is given by (9).

Given an even value of δ and a value for d Friedman's construction is as follows:

Form a tree with $\delta/2 + 1$ levels such that the root node has d successors and all other nodes except those at level $\delta/2 + 1$ have (d-1) successors, the nodes at level $\delta/2 + 1$ are terminal nodes and have no successors. The graph is then constructed by forming d - 1 more such trees all sharing the same terminal nodes. Thus the vertices of this graph all have degree d, because the nodes at levels less than $\delta/2 + 1$ in each tree have degree d and the nodes at level $\delta/2 + 1$ have degree d because they are shared by d trees.

An example of this construction for d = 5 and $\delta = 4$ is shown in Figures 11(a) and (b). In Figure 11(a) the tree for d = 5 and $\delta = 4$ is given, the graph formed by joining 5 such trees is shown in 3-dimensional form in Figure 11(b). Clearly the vertex connectivity of these graphs equal d.

Figure 11(a)

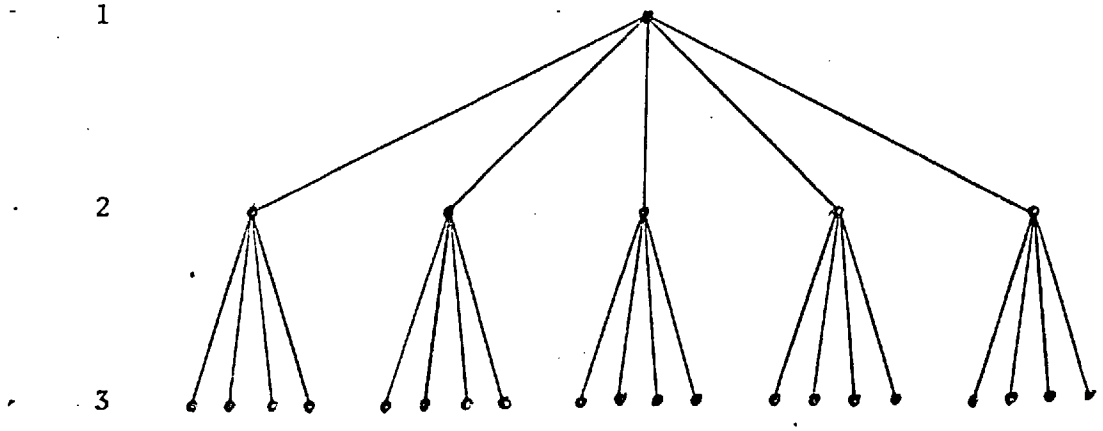
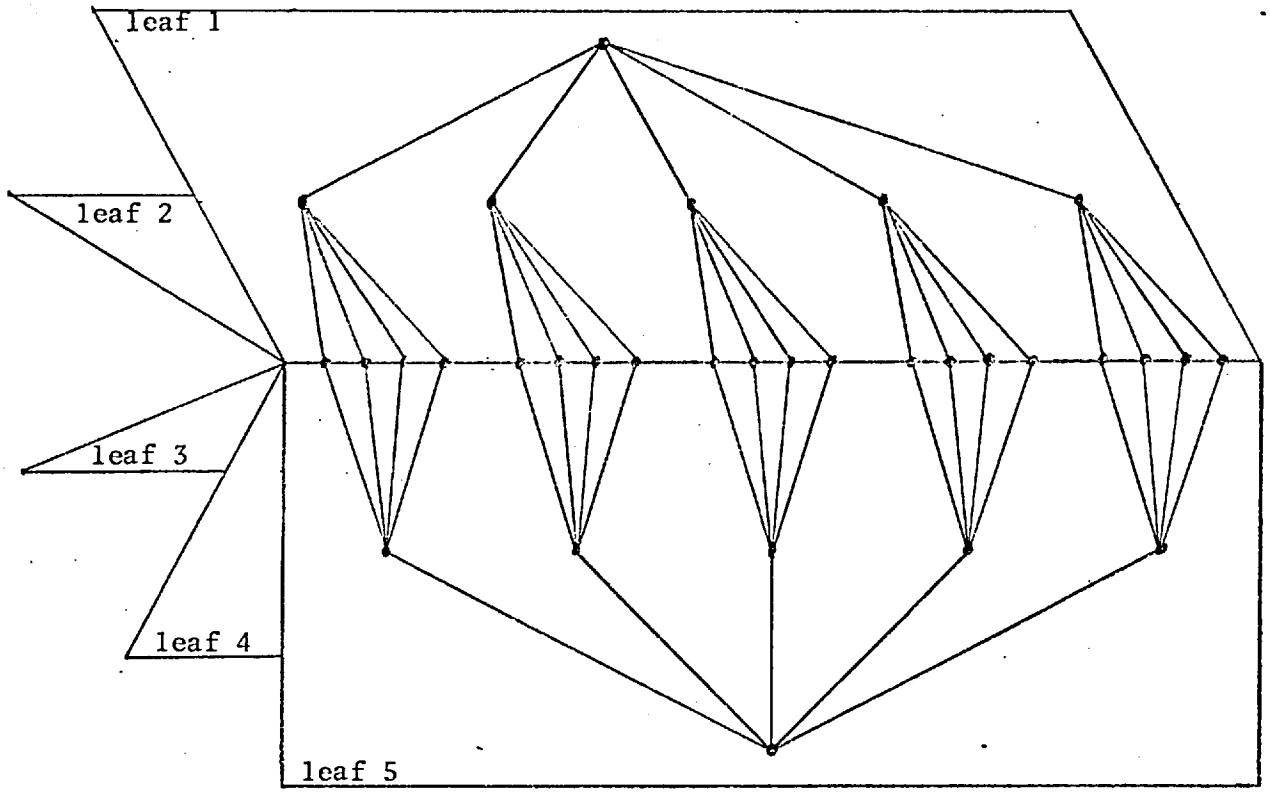


Figure 11. Friedman's construction

Figure 11(b)



At level L (the level of the root in L = 1) in each tree the total number of nodes N_L is given by

$$N_L = d(d-1)^{L-2} \quad , \quad 2 \leq L \leq \left(\frac{\delta}{2} - 1\right)$$

$$N_1 = 1 \ ;$$

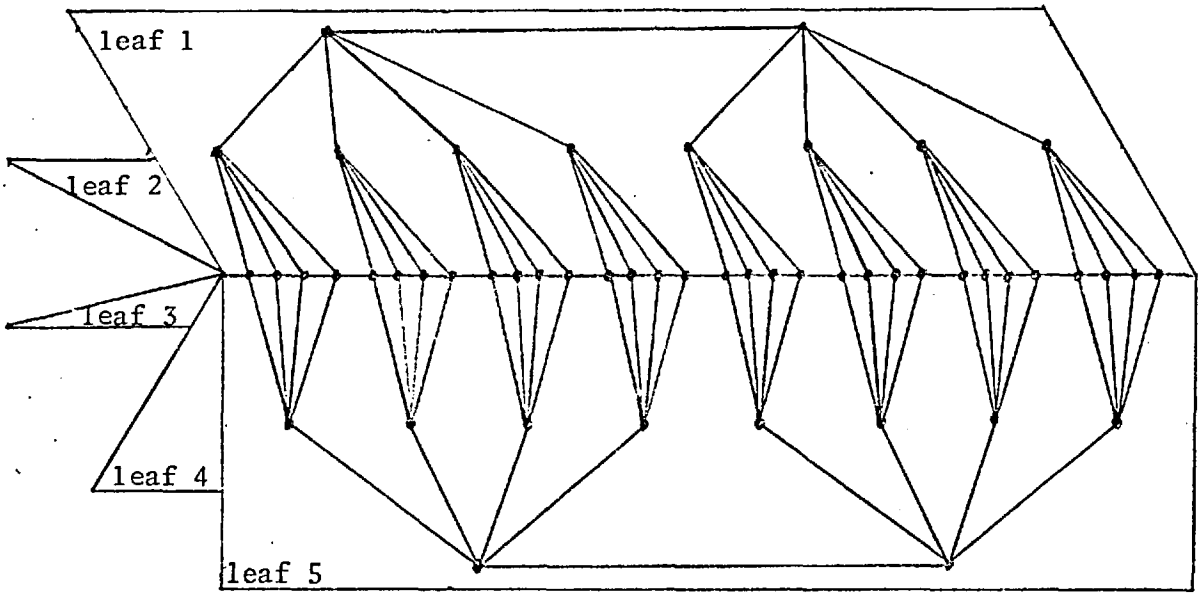
counting all but the terminal nodes n times, the total number of vertices n in the graph is given by:

$$n = \frac{2d}{d-2}\left[(d-1)^{k/2} - 1\right] .$$

Korn [33] developed a similar construction for δ odd. Two trees are drawn in which every node has (d-1) successors except the terminal nodes which are at level $r = \frac{(k-1)}{2} + 1$. A new structure is formed by joining the roots of these trees by an edge. The graph is formed by constructing d - 1 more of these structures all sharing the same terminal nodes as described earlier. The structure for d = 5 and δ = 5 is shown in Figure 12. The total number of vertices n in the graph generated using this structure can be shown to be

$$n = \frac{2}{d-2}\left[2(d-1)^{\frac{\delta+1}{2}} - d\right]$$

Figure 12. Korn's construction



Other studies have been carried out into the relation between the diameter of a graph and its connectivity. In particular Quaife [40] discusses another class of optimal damage resistant graphs with vertex degree $d \geq 3$ and a specific diameter $\delta = j + 4\ell$, where $j = 1,2,3,4$ and ℓ in a "non-negative integer" where the number of vertices n is given by:

$$n = 2(2j + 1)(d-1)^{(\delta-j)/4}$$

However, no algorithm for generating an optimal damage resistant graph with a given n, given $n = d$ and having minimum diameter has yet been produced.

Rubin [41] considered graphs with a minimum average path length in place of using diameter as in the above described studies.

3.3 Problem P with Uniform Cost and Non-Uniform Arc Connectivity

If in a network, communications between the vertices of that network is of varying importance then it is desirable to synthesize a graph in which the connectivity between the various vertices is non-uniform. For example in a computer network the communication between two large computers may be of greater importance than that between two small computers, or in an electrical transmission network it is more important for a large city to have a reliable source of electricity than it is for a small village. In this section we consider the problem P in which a partial graph G_p of a complete graph G, with specified non-uniform arc connectivities, is required.

In order to satisfy the connectivity requirement as specified by a symmetric matrix U of connectivity requirements, we must have:

$$\theta_{ij} \geq u_{ij}$$

in G_p. Since

$$d_i \text{ or } d_j \geq \min(d_i, d_j) \geq \theta_{ij} \qquad i = 1, \ldots, n,\ j = 1, \ldots, n,\ j \neq i$$

it clearly must be the case in G_p that

$$d_i \geq \max_k(u_{ik}) \ .$$

From this inequality we can obtain a lower bound m' on the number of arcs in the optimal solution. Let $u_i^{max} = \max_k(u_{ik})$ then clearly one lower bound is:

$$m' = \left\lceil \left(\sum_{i=1}^{n} u_i^{max} + 1 \right) / 2 \right\rceil$$

If it is possible to realise a graph with m' arcs then that graph will have:

$$d_i = u_i^{max} \ , \quad \text{if} \quad \sum_{i=1}^{n} u_i^{max} \quad \text{is even}$$

and

$$d_i = u_i^{max} \ , \ i \neq k \text{ and } d_k = u_k^{max} + 1, \text{ for some k, if } \sum_{i=1}^{n} u_i^{max} \text{ is odd.}$$

Thus, in order to solve this problem it is of interest to know when a set of numbers (in this case u_i^{max}, $i = 1, \ldots, n$, or u_i^{max}, $i = 1, \ldots, n$, $i \neq k$ and $u_k^{max} + 1$) can be realised as the degrees of a graph. This is a problem which we shall now discuss.

Degree realisability

The necessary and sufficient conditions for a set of integers to be realizable as the degrees of a graph have been studied by a number of authors [2, 22, 24, 25]. Hakimi [24, 25] defined the compressibility of a set of integers as an aid to finding if a set of integers can be realised as the degrees of a graph:

A set of non-negative integers $u_1 \geq u_2 \geq \ldots \geq u_n$, with r of the integers positive, is said to be compressible if $u_1 \leq r - 1$. For example the set of integers {3,2,1,0,0} is not compressible because r = 3 hence

$u_1 = 3 > r-1 = 2$, but {2,2,1,0,0} is compressible. A set of integers is compressed by the following procedure:

(1) Subtract 1 from $u_2, u_3, \ldots, u_{u_1+1}$

(2) Let $u_1 = 0$

(3) Relabel the resulting integers as $u_1 \geq u_2 \geq \ldots \geq u_n$

Thus the compression of {2,2,1,0,0} produces {1,0,0,0,0}. The resulting set of integers is said to be the compressed set or set obtained by compression.

Hakimi then proves the following theorem about compressible sets of integers:

Theorem 5: A set of non-negative integers $u_1 \geq u_2 \geq \ldots \geq u_n$ is realisable as the degrees of an n-vertex non-directed graph G, if and only if the given set of integers $\{u_1 \ldots u_n\}$ is compressible and every set of integers resulting from successive compressions is again compressible, until all integers are zero.

From the proof of this theorem are algorithm can be developed to generate a graph with the degrees of a compressible set of integers. This algorithm is called the degree algorithm, from a compressible set of numbers $(u_1 \ldots u_n)$ it generates a graph G with n vertices $\{x_1 \ldots x_n\}$ such that $d_{(x_i)} = u_i$.

Degree Algorithm

Initialisation: 1 Initialise the graph G so that $A = \phi$ and $X = \{x_1 \ldots x_n\}$, each number u_i corresponds to vertex x_i in X.

2.1 Reorder the number into ordering $\{u_{j_1}, u_{j_2}, \ldots u_{j_n}\}$ so that $u_{j_1} \geq u_{j_2} \geq \ldots \geq u_{j_n}$.

2.2 If $u_{j_1} = 0$ stop; otherwise continue.

3.1 Set $A = AU\{(x_{j_1}, x_{j_2}), (x_{j_1}, x_{j_3}), \ldots, (x_{j_1}, x_{j_{(u_{j_1}+1)}})\}$

3.2 For $\ell = 2$ to $(u_{j_1}+1)$, set $u_\ell = u_\ell - 1$.

3.3 Set $u_{j_1} = 0$ and go to 2.1.

The above can be made clearer with the aid of an example.

Example Consider the compressible set of six numbers $\{u_i\} = \{5,2,3,3,2,3\}$. The first stage of the algorithm is to set up a graph with six vertices and no arcs as shown in Figure 13(a). Each number has associated with

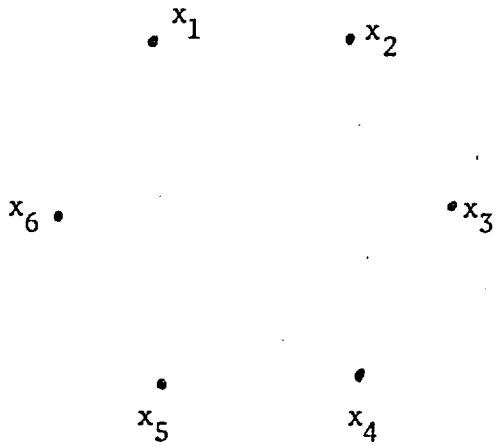


Figure 13(a). First stage of degree algorithm

it a vertex as follows:

x_1	x_2	x_3	x_4	x_5	x_6
5	2	3	3	2	3

The numbers are then put in descending order the vertices being reordered as well:

x_1	x_3	x_4	x_6	x_2	x_5
5	3	3	3	2	2

Vertex x_1 corresponds to the first number in this ordering and $u_{j_1} = u_1 = 5$, hence the five arcs (x_1,x_3), (x_1,x_4), (x_1,x_6), (x_1,x_2), (x_1,x_5) are added to the graph, as shown in Figure 13(b). The numbers

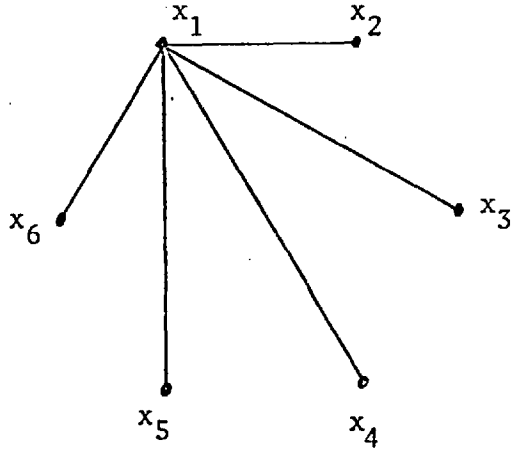


Figure 13(b). First arcs added by degree algorithm

are then compressed the ordering after compression is:

x_3	x_4	x_6	x_2	x_5	x_1
2	2	2	1	1	0

Vertex x_3 corresponds to the first number in this ordering and $u_{j_1} = u_3 = 2$ hence the two arcs $(x_3,x_4),(x_3,x_6)$ are added to the graph as shown in Figure 13(c).

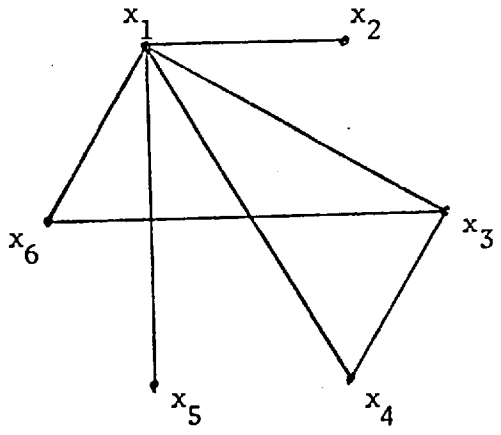


Figure 13(c). Arcs incident on x_3 added to graph

The numbers are again compressed and the new ordering is:

x_4	x_6	x_2	x_5	x_1	x_3
1	1	1	1	0	0

Vertex x_4 corresponds to the first number in the ordering and $u_4 = 1$ hence arc (x_4,x_6) is added to the graph, this is shown in Figure 13(d).

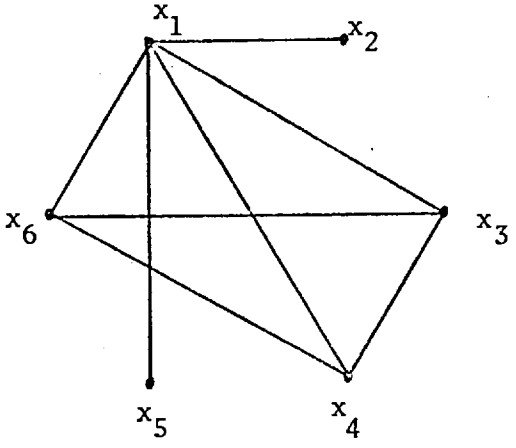


Figure 13(d). Arcs incident on x_4 added to graph

After compression the new ordering is:

x_2	x_5	x_1	x_2	x_4	x_6
1	1	0	0	0	0

Thus arc (x_2,x_5) is added to the graph, after the next compression all numbers go to zero and the algorith m terminates. The final graph is shown in Figure 13(e)

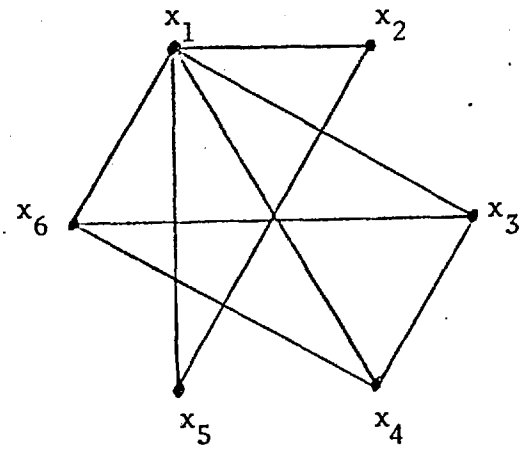


Figure 13(e). Final graph produced by the degree algorithm

Frank and Chou Algorithm

From the above results it follows that in order to generate a graph with vertex degrees specified by a set of numbers $u_1^{max}, \ldots, u_n^{max}$ then the numbers must be compressible. Frank and Chou [8, 14, 15] have shown that for any set of compressible numbers $u_1^{max} > 1, \ldots, u_n^{max} > 1$ it is possible to generate a maximally arc connected graph (see Section 1.4), that is a graph in which

$$\theta_{ij} = \min(d_i, d_j) = \min(u_i^{max}, u_j^{max}) .$$

Thus, in order to solve the problem P with a non-uniform symmetric matrix U of arc connectivity requirements in which the maximum entries in each row u_i^{max}, i = 1 ... n are compressible and $u_i^{max} > 1$, i=1,....,n, it is only necessary to use the u_i^{max} i=1,..,n to construct a maximally arc conected graph. In this graph all u_{ij} are satisfied.

For the case when the u_i^{max} are not compressible it is still possible to construct an optimal solution using only the u_i^{max}. The lower bound on the number of arcs,

$$m' = \left| \left(\sum_{i=1}^{n} u_i^{max} + 1 \right) /2 \right|$$

cannot be realised as the numbers u_i^{max} are not compressible. The solution that is required in this case is a maximally arc connected graph G_p with vertex degrees d_i^p, i = 1 ... n, such that:

$$d_i^p \geq u_i^{max} \qquad i = 1 \ldots n$$

and

$$\sum_{i=1}^{n} d_i^p$$

is a minimum over all possible sets of d_i^p.

Using the above principles Frank and Chou have developed an algorithm to solve this problem with $u_i^{max} > 1$, $i = 1 \ldots n$, [8, 14, 15]. It generates a maximally arc-connected graph with the least possible number of arcs. This algorithm contains three distinct parts. Each part is a special case algorithm which is used according to the properties of the set of n integers $\{u_i^{max}\}$, between them they cover all possibilities (with $u_i^{max} \neq 1$ for any $i = 1, \ldots, n$). The proptrties of $\{u_i^{max}\}$ which decide which part of the algorithm is used are as follows:

(i) When $\max_i[u_i^{max}] = n - 1$, and the u_i^{max} compressible or otherwise the first part of the algorithm is used. This part of the algorithm is similar in structure to the degree realisation algorithm described previously. The solution produced is a maximally arc connected graph. If the $\{u_i^{max}\}$ are compressible then a graph with degrees $\{u_i^{max}\}$ is realised.

(ii) The second part of the algorithm is used when $u_1^{max} < n - 1$ and the first $(u_i^{max} + 1)$ integers $u_1^{max} \geq u_2^{max} \geq \ldots \geq u_{u_1^{max}+1}^{max}$ are successively compressible with the remaining integers compressible or not. This uses the first part of the algorithm to produce a maximally arc connected graph of $u_1^{max} + 1$ vertices, and then adds the remaining $(n-(u_i^{max} + 1))$ vertices. The final solution is also a maximally arc connected graph.

(iii) The third part of the algorithm is used when $u_1^{max} < n - 1$ and the integers $u_1^{max} \geq u_2^{max} \geq \ldots > u_{u_1^{max}+1}^{max}$ are not successively compressible [14].

For the case when $u_i^{max} = 1$ for some i then a maximally arc connected graph may not be the structure of the optimal solution, and algorithms (i) and (ii) do not apply. For example consider the sequence of u_i^{max} :

5, 5, 2, 2, 2, 2,1,1 which is compressible. The sequence is realisable as the degrees of the graph shown by the solid lines in Figure 14. The optimum solution to problem P with the above sequence $\{u_i^{max}\}$ is clearly the graph of Fig. 14 with arc (2, 7) added. This final graph can be seen to not be maximally arc connected as the connectivity $\theta_{2,3} = 2 < \min(3,5)$ for example.

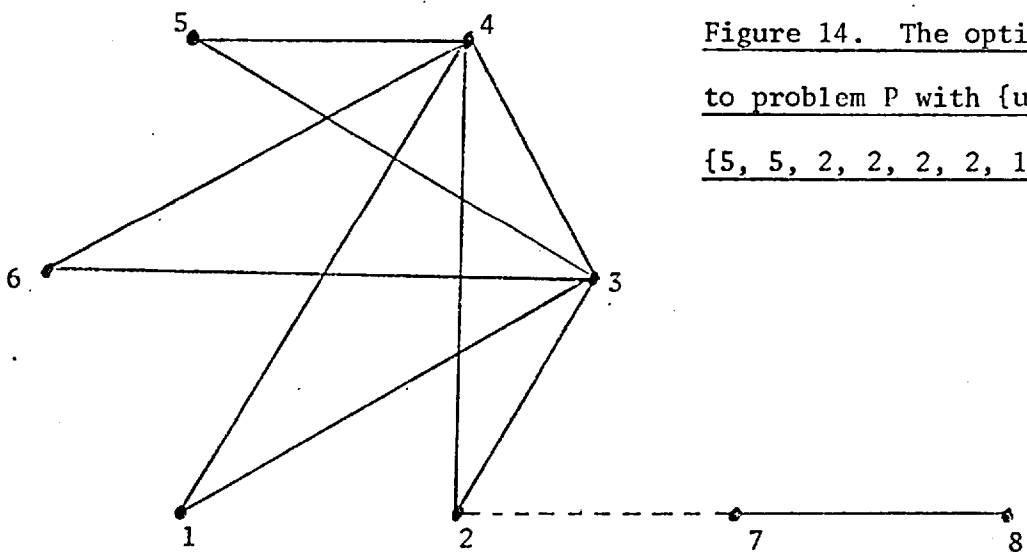


Figure 14. The optimum solution to problem P with $\{u_i^{max}\}$ = {5, 5, 2, 2, 2, 2, 1, 1 }

3.4 Problem P with Uniform Cost and Non-Uniform Vertex Connectivity

A similar analysis applies to the case of uniform cost and non-uniform vertex connectivity. Thus, it is possible to use the maximum entries in each row of the matrix of vertex connectivity requirements to generate a maximally vertex connected graph (see Section 1.4) to solve this problem. No exact procedures for solving this problem exist and the only method described in the literature is an approximate algorithm given by Wilkov [48], to which the interested reader is referred.

4. Problems with Non-uniform arc cost

In the discussion so far only problems with a uniform cost matrix have been considered, that is problems in which all arcs are of equal importance. In the real world this is very often not the case, since

the cost of installing arcs between the vertices can vary enormously. It is therefore desirable to develop methods which can synthesise invulnerable networks of least cost, and these will be discussed in this section.

Because of the complexity of the problem not a great deal of work has been done in this area, and most of the methods that have been developed are heuristics that do not guarantee optimal solutions.

4.1 Uniform Arc Connectivity

The first problem to be considered is that in which a uniform arc connectivity matrix is given as a minimum requirement. A heuristic for this problem was suggested by Frisch [19], and can only be used when the required arc connectivity is even say 2u. This is based on methods for finding the solution to the travelling salesman problem. The travelling salesman problem is the problem of finding the shortest Hamiltonian circuit of a graph and a number of approximate [10, 35] and optimal methods [9] have been developed to solve it. This problem and a new method for solving it will be discussed in greater detail in Chapter 6. The method for finding an approximate solution to the synthesis problem with uniform arc connectivity 2u in a graph G with cost matrix $\{C_{ij}\}$ is as follows:

1. Set n = 1; $A_1 = A$; $G_1 = G = (X,A_1)$; $(A_s=\phi)$; $P_s = (X,A_s)$.

2.1 Solve the travelling salesman problem for G_n, call the set of arcs in the solution T.

2.2 Set $A_s = A_s \cup T$.

3.1 Set n = n + 1.

3.2 If n > u stop (G_s is the approximate solution); else continue.

3.3 Set $A_n = A_{n-1} - T$ and go to 2.1

Clearly at termination G_s is the union of u edge disjoint Hamiltonian circuits and hence by Theorem 4 (see Section 1.4) has $\theta = 2u$.

The method used to solve the travelling salesman problem can either be approximate if large problems have to be solved or exact if greater accuracy is required. This method will not necessarily generate an optimal solution to the problem as the solution may not be two disjoint Hamiltonian circuits, in fact it may not even be regular. For example consider the cost matrix in Figure 15(a). Figure 15(b) shows the least cost subgraph of the graph for the costs in Figure 15(a) with $\theta = 4$, it has a cost of 107. Figure 15(c) shows the least cost subgraph containing two disjoint Hamiltonial circuits which has a value of 108.

Moreover, even if the solution is regular it is not necessarily composed of edge disjoint Hamiltonian circuits. For example consider the graph in Figure 16, this is the optimal solution to the problem where G is complete and every arc in G has cost 2 except the arcs shown in Figure 16. The graph in Figure 16 contains no Hamiltonian circuits [37].

4.2 Non-Uniform Vertex Connectivity

Another problem for which heuristic methods have been developed is that in which a non-uniform matrix of vertex connectivities is given as a minimum requirement. A heuristic has been developed by Steiglitz, Weiner and Kleitman [42] for finding an appropriate solution to this problem. This is a method in which a random initial solution is generated so that for the random graph $G^R = (X,A^R)$

$$\omega^R_{ij} \geq u_{ij} \qquad \forall_{i,j} \in X$$

An arc exchange procedure is the n used in which pairs of arcs are exchanged for other pairs of arcs with a lower cost, in such a way as not to violate the inequality:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
x_1	∞	6	20	15	2	20	2	20	5	16
x_2	6	∞	10	3	2	10	15	20	14	10
x_3	20	10	∞	11	11	12	16	16	1	13
x_4	15	3	11	∞	17	7	12	7	20	3
x_5	2	2	11	17	∞	2	3	4	11	2
x_6	20	10	12	7	2	∞	18	11	19	3
x_7	2	15	6	12	3	18	∞	1	19	4
x_8	20	20	16	7	4	11	1	∞	6	11
x_9	5	14	1	20	11	19	19	6	∞	14
x_{10}	16	10	13	3	2	3	4	11	14	∞

Figure 15(a). Cost matrix for the graphs in figures 15(b) and 15(c)

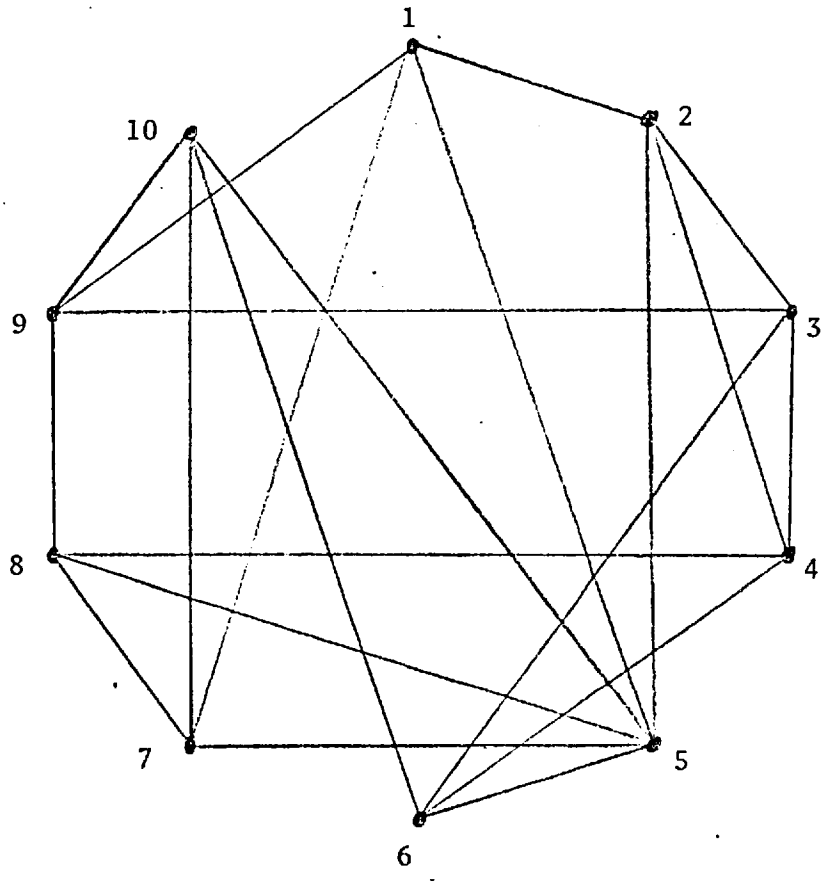


Figure 15(b). The optimum solution for problem P with u=4 over the cost matrix in figure 15(a)

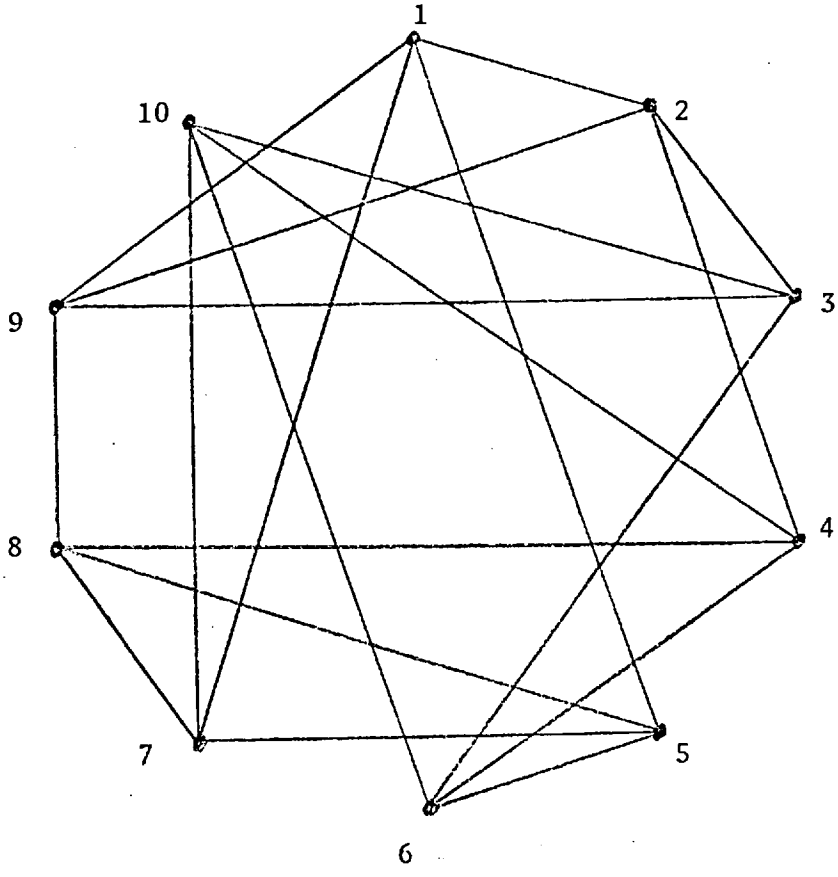


Figure 15(c). The least cost pair of disjoint hamiltonian circuits for the cost matrix in figure 15(a)

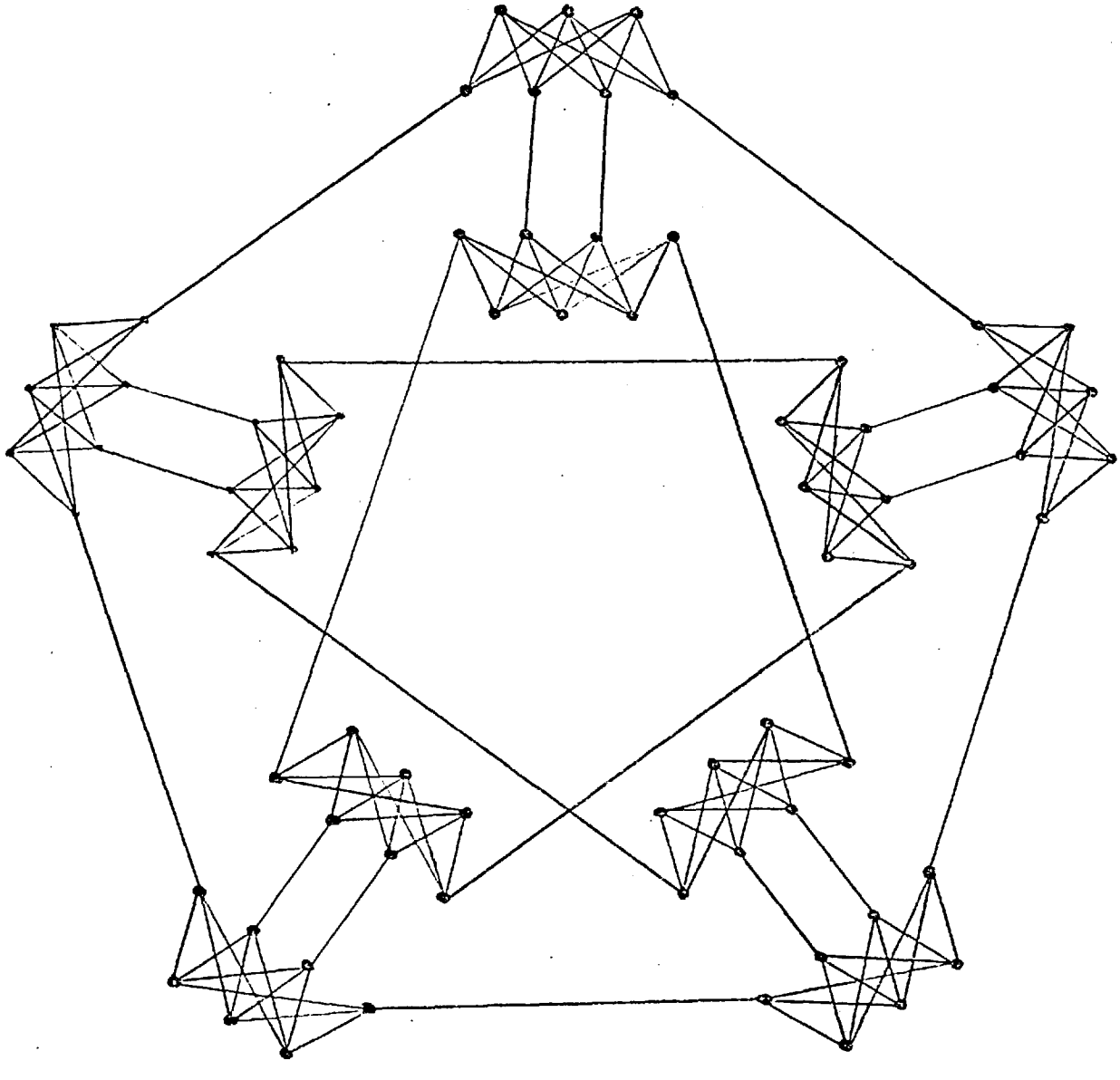


Figure 16. A 4-regular 4-connected graph having no hamiltonian circuits

$$\omega_{ij} \geq u_{ij} .$$

This procedure is carried out until a local optimum is obtained. If greater accuracy is required then the procedure can be repeated several times for different initial solutions and the best local optimum can be used. Note that this procedure belongs to the family of r-optimal algorithms [10, 11, 30, 35] used successfully for a number of other combinatorial problems.

The efficiency of this procedure is dependent on the efficiency of the method used to generate the initial random solution. An alternative method to the one suggested in [42] was developed by Owens [39]. This method generally results in starting graphs of lower cost.

One optimal method which should be mentioned in this section is due to Suurballe [43]. This method solves a simpler but related problem, i.e. it is concerned with finding the least cost set of u vertex-disjoint paths between a given pair of vertices s and t. Thus, it can be used in cases in which a specific connectivity u is required only between two vertices. The algorithm is based on a shortest path labelling algorithm.

APPENDIX

The ω_{st} labelling algorithm

In the ω_{st} algorithm each vertex is assigned up to two labels, the labels that are used being ordered triples. The first part of the label represents the vertex scanned in order to produce that label. The second part is a + or - which indicates whether the flow through the scanned vertex can be increased or decreased by one unit, a label with a + is called 'weak' and a label with a - is called'strong'. The third part of the label is a variable M which can take the values 0 and 1, its value being fixed by the nature of the vertex that was scanned to produce the labels. If the vertex that was scanned in order to produce that label had two labels when it was scanned then M = 1 otherwise M = 0. This variable is used when finding an augmenting path normally in augmentation if a vertex has two labels the weak label is used for finding the augmenting path, as the strong label may give circuits, but if a value of M = 1 is encountered then the strong label is given priority. It is possible for both labels of a vertex to be used during augmentation.

In the algorithm the 'scanning' of a vertex has a more complex meaning than in the Ford and Fulkerson labelling algorithm. As each vertex can have two labels it is possible to scan a vertex twice during the course of the procedure, once for each label. It is therefore convenient to distinguish between the scanning of vertices and labels, a vertex being considered 'scanned' if it contains no unscanned labels. It is possible for a scanned vertex with one scanned label to become unscanned by the addition of a second label. In the algorithm only a strong label can be added to a vertex as a second label.

The ω_{st} labelling algorithm for finding vertex connectivity between vertices x_s and x_t in a graph G is as follows:

initialisation 1.0 Initialise the flows $\xi_{ij} = \xi(x_i, x_j)$ in the graph G. (They can be set to 0 or to arbitrary integer values such that $\sum_j \xi_{ij} = 0$ or 1 for all $x_i \in X$, $i \neq s$ or t).

labelling routine

1.1 Assign a weak label (s, +, 0) to vertex x_s. x_s is now labelled and unscanned. Set all other vertices are unlabelled and unscanned.

1.2.1 Select any labelled and unscanned vertex x_i, if there are non stop. Otherwise continue.

1.2.2 If x_i has a strong label go to 1.4.1, otherwise continue. (Note: It is in fact not possible for a vertex to have a scanned strong label and an unscanned weak label. Thus if a vertex is unscanned and contains a strong label then the strong label must be unscanned).

1.3.1 If $\sum_{j \in X} \xi_{ij} \neq 0$ go to 1.3.2. Otherwise weakly label by (i, +, 0) all <u>unlabelled</u> x_j such that $x_j \in T(x_i)$. These x_j are now weakly labelled and unscanned. Go to 1.3.3.

1.3.2 Strongly label by (i, −, 0) all x_j <u>that are not strongly labelled</u> such that $x_i \in T(x_j)$, $\xi_{ji} = 1$ and $j \neq s$. These x_j are now strongly labelled and unscanned. Continue.

1.3.3 Mark as scanned the weak label on x_i. Go to 1.5.

1.4.1 If x_i has two labels set M = 1, else set M = 0. Continue.

1.4.2 Weakly label by (i, +, M) all <u>unlabelled</u> x_j such that $x_j \in T(x_i)$. These x_j are now weakly labelled and unscanned. Continue.

1.4.3 Strongly label by (i, -, M) all x_j <u>that are not strongly labelled</u> such that $x_i \in T(x_j)$, $\xi_{ji} = 1$ and $j \neq s$. These x_j now strongly labelled and unscanned. (Although they may contain a scanned or unscanned weak label.) Continue.

1.4.4 Mark as scanned <u>all</u> labels on x_i. Continue.

1.5 If x_t is unlabelled go to 1.2.1, else continue.

Augmentation routine

2.1 Set k = t. Continue.

2.2.1 If x_k is weakly labelled by (ℓ, +, M), set $\xi_{\ell k} = 1$ and go to 2.3. Else continue.

2.2.2 If x_k is strongly labelled by (ℓ, -, M), set $\xi_{k\ell} = 0$. Continue.

2.3 If $\ell = s$ go to 1.1, else continue.

2.4 Set $k = \ell$. Continue

2.5 If M = 1 go to 2.2.2, else go to 2.2.1.

At termination the flow between x_s and x_t is equal to ω_{st} and the minimum vertex cut set can be found by examining the sets of labelled and unlabelled vertices. If the set of labelled vertices is called L and the set of unlabelled vertices is called U, clearly neither L nor U is empty as $x_s \in L$ and $x_t \in U$, also $L \cup U = X$, and we define a set of vertices L_u at the interface of the labelled and unlabelled vertices such that

$$L_u = \{x_i : (x_i, x_j) \in A,\ x_i \in L,\ x_j \in U\}$$

then the minimum vertex cut set X^{st}_{min} between x_s and x_t has been given by Frisch [16]:

$$X^{st}_{min} = L_u \qquad \text{for} \quad x_s \notin L_u \tag{12}$$

$$X^{st}_{min} = \{L_u - \{x_s\}\} \cup \{x_j : (x_s, x_j) \in A,\ x_j \in U\} \tag{13}$$

$$\text{for } x_s \in L_u .$$

Example Consider the graph in Figure 17(a) with the arcs directed as shown, the initial flow in each arc is also shown in this figure.

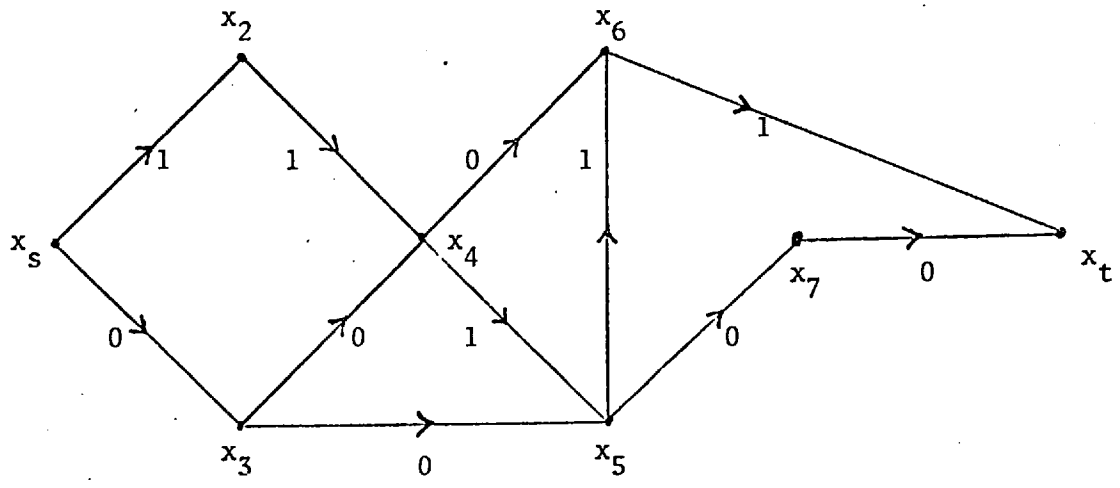


Figure 17(a). Initial flow for ω_{st} labelling algorithm

In the first stage of the labelling routine x_s is given label (s, +, 0), x_s is then scanned to produce a weak label (s, +, 0) on x_3. At this stage x_s is labelled and scanned x_3 is labelled and unscanned, all other vertices are unlabelled and unscanned.

The next label to be scanned is x_3, this produces labels (3, +, 0) on x_4 and x_5. If x_4 is then scanned this produces label (4, −, 0) on x_2, the scanning of x_2 produces no new labels. The labels at this point are shown in Figure 17(b), the signs in the labels are circled if that label is scanned.

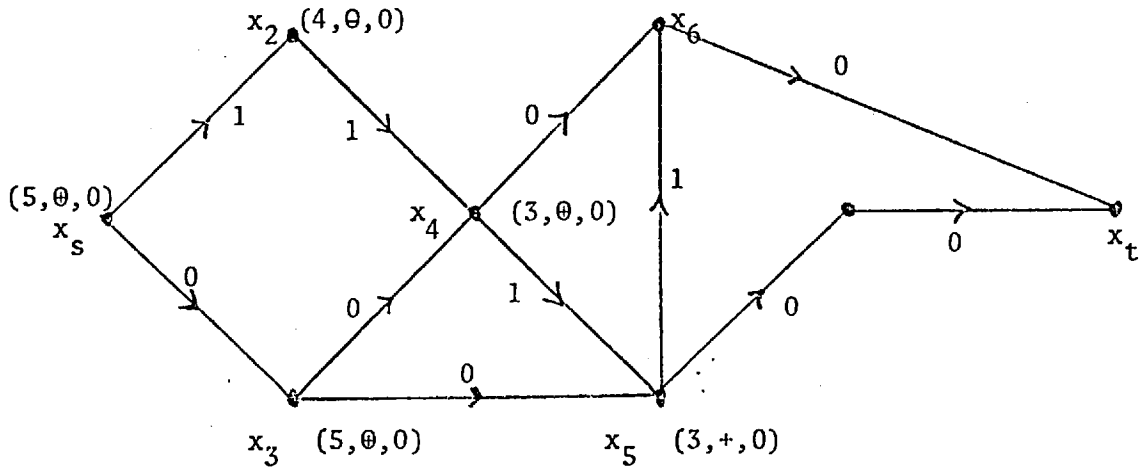


Figure 17(b). The graph after some labels have been added

From Figure 17(b) it can be seen that the only unscanned labelled vertex is x_5, this is scanned and produces a second label (5, -, 0) on x_4, hence x_4 becomes unscanned. x_4 is then scanned for a second time to produce a label (4, +, 1) on x_6. The scanning of x_6 produces a second label (6, -, 0) on x_5, and the second scanning of x_5 produces a label (5, +, 1) on x_7, this in turn produces a label (7, +, 0) on x_t, and augmentation begins. The labels at this stage are shown in Figure 17(c).

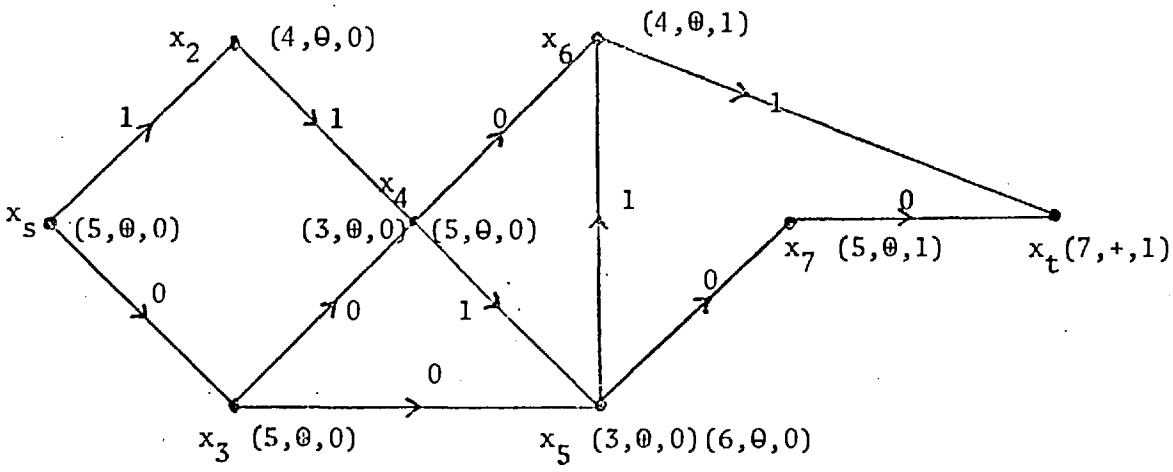


Figure 17(c). Labels before augmentation begins

Augmentation begins by setting the flow along (x_7, x_t) and (x_5, x_7) to 1. M = 1 at x_7 hence the strong label on x_5 is considered this reduces the flow on (x_5, x_6) to 0, the flow is then increased to 1 on (x_4, x_6). M = 1 at x_6, hence the strong label at x_4 is used to reduce the flow on (x_4, x_5) to 0. The weak label on x_5 increased the flow on (x_3, x_5) to 1 and finally the flow on (x_5, x_3) is set to 1. The new flow pattern is shown in Figure 17(d), the flow from x_5 to x_t is 2.

A second application of the labelling routine produces only a label on x_5, thus $w_{st} = 2$ and from (13) $X^{st}_{min} = x_2, x_3)$.

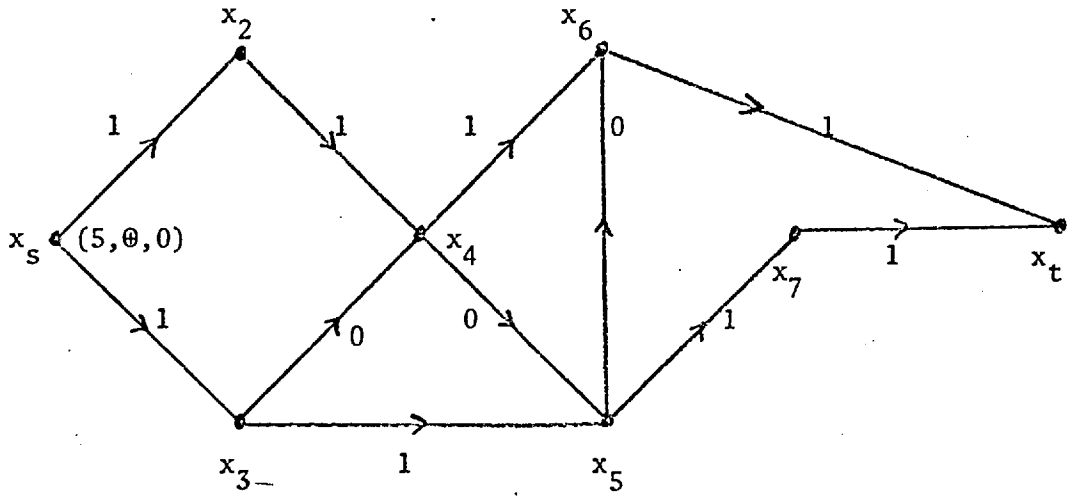


Figure 17(d). Flow pattern at termination of the algorithm

REFERENCES

1. AYOUB, J.N., and FRISCH, I.T., "Optimally Invulnerable Directed Communications Networks", IEEE Trans on Comm Tech, Vol.CCM-18, 1970, pp.484-489.

2. BERGE, C., "The Theory of Graphs and Its Applications", Ch.9, Wiley, New York, 1962.

3. BOESCH, F.T. and THOMAS, R.E., "On graph of invulnerable communication nets", IEEE Trans. Circuit Theory, Vol.CT-17, 1970, pp.183-192.

4. BOESCH, F.T. and FRISCH, I.T., "On the Smallest Disconnecting Set in a Graph", IEEE Trans. Circuit Theory, Vol. CT-15, 1968, pp.286-288.

5. BOESCH, F.T. and FELZER, A.P., "A General Class of Invulnerable Graphs", Networks, Vol.2, 1972, pp.261-283.

6. CHARTRAND, G., "A graph theoretic approach to a communication problem", S.I.A.M. J. Appl. Math., Vol. 14, 1966, pp.778-781.

7. CHARTRAND, G. and HARARY, "Graphs with prescribed connectivities", in Theory of Graphs, Ed. P. Erdos and G. Katona, New York, Academic Press 1968, pp.61-63.

8. CHOU, W. and FRANK, H., "Survivable Communication Networks and the Terminal Capacity Matrix", IEEE Trans. Circuit Theory, Vol. CT-17, 1970, pp.192-197.

9. CHRISTOFIDES, N., "An Algorithmic Approach to Graph Theory", Academic Press, 1975.

10. CHRISTOFIDES, N. and EILON, S., "Algorithms for Large Scale Travelling Salesman Problems", Operations Research Quarterly, Vol.23, 1972, pp.511-518.

11. CHRISTOFIDES, N. and VIOLA, P., "The Optimum Location of Multi-centers on a Graph", Operations Research Quarterly, Vol.22, 1971, pp.145-154.

12. DIRAC, G.A., "Extensions fo Menger's Theorem", Journal London Math. Soc., Vol.38, 1963, pp.148-161.

13. FORD, L.R. and FULKERSON, D.R., "Flows in Networks", Princeton University Press, 1962.

14. FRANK, H. and CHOU, W., "Connectivity Considerations in the Design of Survivable Networks", Office of Emergency Preparedness, Washington, D.C., Tech. Memo, Jan. 1969.

15. FRANK, H. and CHOU, W., "Connectivity Considerations in the Design of Survivable Networks", IEEE Trans. Circuit Theory, Vol. CT-17, 1970, pp.486-490.

16. FRANK, H. and FRISCH, I.T., "Communications, Transmission and Transportation Networks", Addison Wesley, 1971.

17. FRIEDMAN, H., "A design for (d,k) graphs", IEEE Trans. Electron Computers, Vol. EC-15, 1966, pp.253-254.

18. FRISCH, I.T., "An Algorithm for Vertex Pair Connectivity", Intern. J. Control, Vol.6, 1967, pp.579-593.

19. FRISCH, I.T., "A Computer Technique for Design and Analysis of Hard Communications Systems", Digest of th e 1967 IEEE Intern. Conf. on Communications, Minneapolis 1967, pp.96.

20. FRISCH, I.T., "Flow variation in multiple min-cut calculations", J. Franklin Inst., Vol.287, 1969, pp.61-72.

21. FULKERSON, D.R. and SHAPLEY, L.S., "Minimal K-Arc Connected Graphs", Networks, Vol.1, pp.91-98.

22. FULKERSON, D.R., HOFFMAN, A.J. and McANDREW, H., "Some Properties of Graphs with Multiple Edges", Can. J. Maths, Vol. 17, 1965, pp.496-506.

23. GOMORY, R.E. and HU, T.C., "Multiterminal Network Flows", SIAM J. Appl. Maths., Vol.9, 1961, pp.551-570.

24. HAKIMI, S.I., "On the Realizability of Integers as the Degrees of the Vertices of a Linear Graph - I", J. SIAM, Vol.10, 1962, pp.496-506.

25. HAKIMI, S.I., "On the Realizability of a Set of Integers as the Degrees of the Vertices of a Linear Graph - II Uniqueness", J. SIAM, Vol.11, 1963, pp.135-147.

26. HAKIMI, S.L., "An Algorithm for Construction of the Least Vulnerable Communication Network or the Graph with the Maximum Connectivity", IEEE Trans. Circuit Theory, Vol. CT-16, 1969, pp.229-230.

27. HALIN, R., "A theorem on n-connected graphs", Journal of Combinatorial Theory, Vol.7, pp.150-154.

28. HALIN, R., "On the structure of n-connected graphs", from Recent Progress in Combinatorics, Ed. Tutte, W.T., New York, Academic Press, 1969, pp.91-162.

29. HARARY, F., "The Maximum Connectivity of a Graph", Proc. Nat. Acad. Sci., Vol.48, 1962, pp.1142-1146.

30. KERNINGHAN, B.W. and LIN, S., "An Efficient Heuristic Procedure for Partitioning Graphs", Bell Systems Technical Journal, Vol.49, 1970, pp.291-307.

31. KLEE, V. and QUAIFE, H., "Minimum Graphs of Specified Diameter, Connectivity and Valence. I", Mathematics of Operations Research, Vol.1, 1978, pp.28-31.

32. KLEITMAN, D.J., "Methods for investigating connectivity of large graphs", IEEE Trans. Circuit Theory, Vol. CT-16, 1969, pp.232-233.

33. KORN, I., "On (d,k) Graphs", IEEE Trans. Electron. Computers, Vol. EC-16, 1967, pp.90.

34. LICK, D., "Critically and minimally n-connected graphs", from The Many Facets of Graph Theory, Ed. Chartrand, G., and Kapoor, S., New York, Springer, 1969, pp.199-205.

35. LIN, S., "Computer Solutions of the Travelling Salesman Problem", Bell Systems Technical Journal, Vol.44, 1965, pp.2245-2269.

36. MENGER, K., "Aur allgememen Kurventheorie", Funetam. Math., Vol.10, 1927, pp.96-115.

37. MEREDITH, G.H.J., "Regular n-Valent n-Connected Non-Hamiltonian Non-n-Colorable Graphs". Journal of Combinatorial Theory, Vol.14B, 1973, pp.55-60.

38. ORE, O., "Diameters in Graphs", J. Combinatorial Theory, Vol.5, 1968, pp.75-81.

39. OWENS, A.B., "A Starting Algorithm for Minimal Cost Survivable Networks", Naval Res. Lab., Washington D.C., Rep.7272.

40. QUAIFE, H., "On (d,k,u) graphs", IEEE Trans. Computers, Vol. C-18, 1969, pp.270-272.

41. RUBIN, I., "On Reliable Topological Structures for Massage-Switching Communications Networks", IEEE Trans. on Communications, Vol. CDM-26, 1978, pp.52-74.

42. STEIGLITZ, K., WEINER, P. and KLEITMAN, D.J., "The Design of Minimum-Cost Survivable Networks", IEEE Trans. Circuit Theory, Vol. CT-16, 1969, pp.455-460.

43. SUURBALLE, J.W., "Disjoint Paths in a Network", Networks, Vol.4, 1974, pp.125-145.

44. TUTTE, W.T., "Connectivity in Graphs", London: Oxford University Press, 1966.

45. WHITNEY, H., "Congruent Graphs and the Connectivity of Graphs", American J. Maths., Vol.54, 1932, pp.150-168.

46. WILKOV, R.S., "Construction of maximally reliable communications networks with minimum transmission delay", Proc. IEEE Int. Conf. Communications, Vol.6, 1970, pp.4210-4215.

47. WILKOV, R.S., "On maximally connected graphs with minimal diameter", Proc. 1970 IEEE Int. Symp. Circuit Theory, 1970, pp.25-26.

48. WILKOV, R.S., "On the Design of High Speed Communications Networks", presented 5th Annual Princeton Conf. Information Sciences and Systems, March 1971.

49. WILKOV, R.S., "Analysis and Design of Relaible Computer Networks", Vol. COM-20, 1972, pp.660-678.

CHAPTER 5

NETWORK SYNTHESIS ALGORITHMS

1. Introduction

In this chapter a more detailed discussion is given of the graph synthesis problem for uniform arc connectivity and non-uniform arc costs, as mentioned in the last chapter.

The most general synthesis problem (P^u) is formulated. The formulations of two related problems (S^u and Q^u) are then given, and the relationship between these three problems and the u-matching problem (M^u) are discussed.

The effect of the type of costs in use (i.e. random, obeying triangularity conditions, euclidean distances etc.) on the structure of the optimal solution is discussed. It is shown for example, that the optimal solution to the problem with uniform arc costs is a regular graph.

1.1 The Problems

The graph synthesis problem P^u for uniform arc connectivity and non-uniform arc costs can be defined as follows:

Problem P^u: Given a graph $G = (X,A)$, with X as the set of vertices and A the set of arcs, with every arc $(x_i,x_j) \in A$ having an associated cost c_{ij}; and given the required arc connectivity u, find a partial graph $G_p = (X,A_p)$ of G such that:

$$\theta_p = u \tag{1}$$

and which minimises;

$$\sum_{(x_i,x_j)\in A_p} c_{ij} \tag{2}$$

where θ_p is the arc connectivity of G_p.

Quite clearly a feasible solution to the above problem exists only if $\theta(G) \geq u$.

The second more constrained network synthesis problem is problem S^u. This is defined when the product of the number of vertices n and u is even, and is as follows:

Problem S^u: Given a graph $G(X,A)$, with X as the set of n vertices and A the set of arcs, with every arc $(x_i,x_j) \in A$ having an associated cost c_{ij}; and given the required arc connectivity u with the product n.u. even, find a partial graph $G_s = (X,A_s)$ of G such that:

$$\theta_s = u \tag{1}$$

$$d_i = u \quad \forall x_i \in X \text{ in } G_s \tag{3}$$

and which minimises;

$$\sum_{(x_i,x_j)\in A_s} c_{ij} \tag{2}$$

where θ_s is the arc connectivity of G_s.

(Note: if n.u is odd a graph with the above properties does not exist).

From the above definition it can be seen that problem S^u is problem P^u with the additional constraint that the degrees of the vertices of the solution graph G_s are all equal to u.

A third more constrained problem than the above two is problem Q^u which is defined for u even as follows:

Problem Q^u: Given a graph $G = (X,A)$, with X as the set of vertices and A the set of arcs, with every arc $(x_i,x_j) \in A$ having an associated cost c_{ij}; and given the required arc connectivity u (even), find a partial graph $G_Q = (X,A_Q)$ of G such that:

G_Q contains u/2 arc disjoint hamiltonian circuits (4)

and which minimises;

$$\sum_{(x_i,x_j)\in A_Q} c_{ij} \qquad (2)$$

Clearly the solution graph G_Q to problem Q^u is regular of degree u. From Theorem 4 of Chapter 4;

$$\theta_Q = u \ ,$$

where θ_Q is the arc connectivity G_Q.

Problem Q^u is the problem for which the Frisch [6] heuristic, described in Chapter 4, Section 4.1, finds an approximate solution which is than also proposed as approximate solution to problem P^u.

The solution graphs G_s and G_Q for problems S^u and Q^u respectively are u-regular. The u-matching problem M^u finds a least cost partial graph with every vertex of regular degree u and hence, it is a relaxation of problems S^u and Q^u. For completeness we give the definition of the u matching problem as follows:

Problem M^u: Given a graph G = (X,A), with X as the set of vertices and A the set of arcs, with every arc $(x_i,x_j) \in A$ having an associated cost c_{ij}; and given the required vertex degree u, find a partial graph $G_M = (X,A_M)$ of G such that:

$$d_i = u \quad \forall x_i \in X \ \text{ in } G_M \qquad (3)$$

and which minimises:

$$\sum_{(x_i,x_j)\in A_M} c_{ij} \qquad (2)$$

From the above, problem Q^u is the most constrained problem of the four. It is also - as will be shown later - the most difficult problem to solve. Problem S^u is a relaxation of Q^u. The hamiltonian circuit constraints are relaxed in Q^u to produce S^u. Problems P^u and M^u are both relaxations of S^u. In P^u the degree constraints have been relaxed from S^u. In M^u the connectivity

constraints have been relaxed. The relationships between these four problems is summed up in Figure 1. In this figure the arrows mean 'is a relaxation of'.

The first two methods (A and B) described in this chapter are designed to solve problems, S^u and Q^u. They use Lagrangean penalties

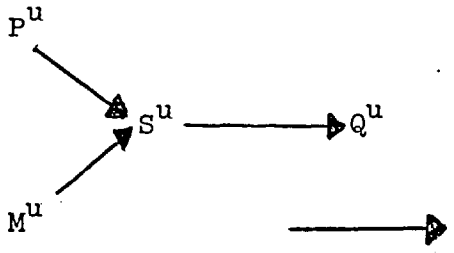


Figure 1. Relationships between problems P^u, M^u, S^u and Q^u

applied to the solution of the matching problem M^u to derive bounds and proceed by implicitly enumerating all feasible solutions.

The third method C uses an integer programming approach to the problem based on an LP relaxation in which most of the constraints are not included. The constraints that are not used but which are violated in the LP solution are found using a network flow sub-algorithm. These constraints are then added to the LP which is then resolved. The process is repeated until an LP solution is found which does not violate any of the constraints. Integrality is obtained by using a branch and bound procedure. Method C optimally solves both problems P^u and S^u. Computational result are given for all three methods.

1.2 Analysis of the Optimal Solution to the Problem P^u

Before discussing these methods it is helpful to consider the possible structures for the optimal solution to problem P^u, and to relate this to the structures of the solutions of problems S^u and Q^u.

The only method in the literature for finding a reasonably good feasible solution to problem P^u is that of Frisch [6]. This method

finds an approximate solution to problem Q^u, and it is only defined for u-even.

If a subgraph of a graph G with $\theta = u$ is required, where u is even, then the method first solves the travelling salesman problem for G - producing a subgraph $G^1 = (X,T_1)$, where T_1 is the set of arcs in the optimal tour. (Note: It is possible to use a heuristic to solve the travelling salesman problem since the entire procedure is approximate.) The travelling salesman problem is then resolved for graph $G_1 = (X,(A-T_1))$ to produce a second tour T_2 and a second graph $G^2 = (X,T_1 \cup T_2)$. Thus, in general the travelling salesman problem is solved u/2 times over the graph:

$$G_n = \left[X, \left[A - \bigcup_{i=0}^{n-1} T_i\right]\right] \qquad 0 \leq n \leq \left[\frac{u}{2} - 1\right]$$

where $T_0 = \phi$, and the graph $G^{u/2}$, where:

$$G^{u/2} = \left[X, \bigcup_{i=1}^{u/2} T_i\right],$$

i.e. the union of the hamiltonian tours that have been generated, is considered to be a good approximate solution to the synthesis problem P^u.

The graph $G^{u/2}$ generated by the Frisch heuristic is of similar structure to the structure of the solutions to problems S^u and Q^u. It obeys constraints on the degree of each vertex (i.e. the degree of each vertex is u) as required for problem S^u. It also obeys the disjoint hamiltonian circuits constraint (i.e. it contains u/2 disjoint hamiltonian circuits) as required for problem Q^u. It is of interest to know if the optimal solution to problem P^u, for any particular cost matrix will have a similar structure to the solution of problems S^u and Q^u, as this structure might be exploited in developing an algorithm.

For graphs in which the costs are random it can be shown that in general G_P, G_S and G_Q do not have the same structure. For example, consider Figures 2(a), 2(b) and 2(c). Figure 2(a) shows a random symmetric cost matrix in which the costs are uniformly distributed between 1 and 20. The solution graph G_P for problem P^4 over the cost matrix in Figure 2(a) is given in Figure 2(b). This contains only one arc disjoint hamiltonian curcuit and is not regular. Hence, G_P does not have the same structure as G_S or G_Q over this cost matrix. The value of G_p is 107. The solutions to problems S^4 and Q^4 over the cost matrix in Figure 2(a) are given in Figure 2(c) (both solutions are the same). Although this graph has less arcs than G_P it has a larger cost (of 108).

It has been shown in the above example that in general for <u>random</u> arc costs G_P differs from G_S and G_Q. It can also be shown that in general for <u>triangular</u> arc costs G_S differs from G_Q. For example consider Figure 3 from Meredith [7]. This contains a 4-regular graph which is the optimal solution to S^4 (and also P^4) for the complete graph with a triangular cost matrix in which every arc in Figure 3 has a cost of 1 and every arc not in Figure 3 has a cost of 2. The graph in Figure 3 contains no hamiltonian circuits. Hence, the solution to Q^4 over this cost matrix is different from the solutions to S^4 (and P^4).

In the above two examples it has been shown that in general for triangular (and therefore also for random) arc costs the solution graphs to S^u and Q^u do not have the same structure. For random arc costs the structures of P^u and S^u (and Q^u) differ. However, it has not been shown that P^u and S^u differ for triangular costs (note: S^u is only defined when the product n.u is even. Clearly G_P will not have the property of being u-regular when n.u is odd as such as a graph cannot be realised). It is postulated that P^u and S^u have the same

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
x_1	∞	6	20	15	2	20	2	20	5	16
x_2	6	∞	10	3	2	10	15	20	14	10
x_3	20	10	∞	11	11	12	16	16	1	13
x_4	15	3	11	∞	17	7	12	7	20	3
x_5	2	2	11	17	∞	2	3	4	11	2
x_6	20	10	12	7	2	∞	18	11	19	3
x_7	2	15	6	12	3	18	∞	1	19	4
x_8	20	20	16	7	4	11	1	∞	6	11
x_9	5	14	1	20	11	19	19	6	∞	14
x_{10}	16	10	13	3	2	3	4	11	14	∞

Figure 2(a). Cost matrix for the graphs in Figures 2(b) and 2(c)

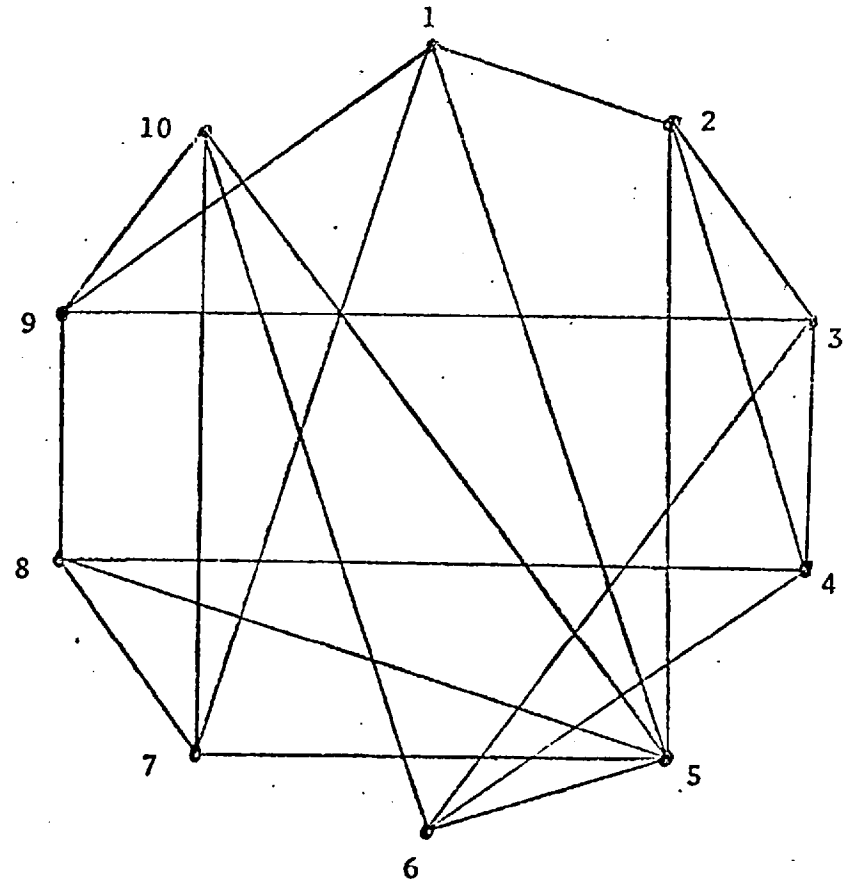


Figure 2(b). The optimum solution for problem P^4 over the cost matrix in Figure 2(a)

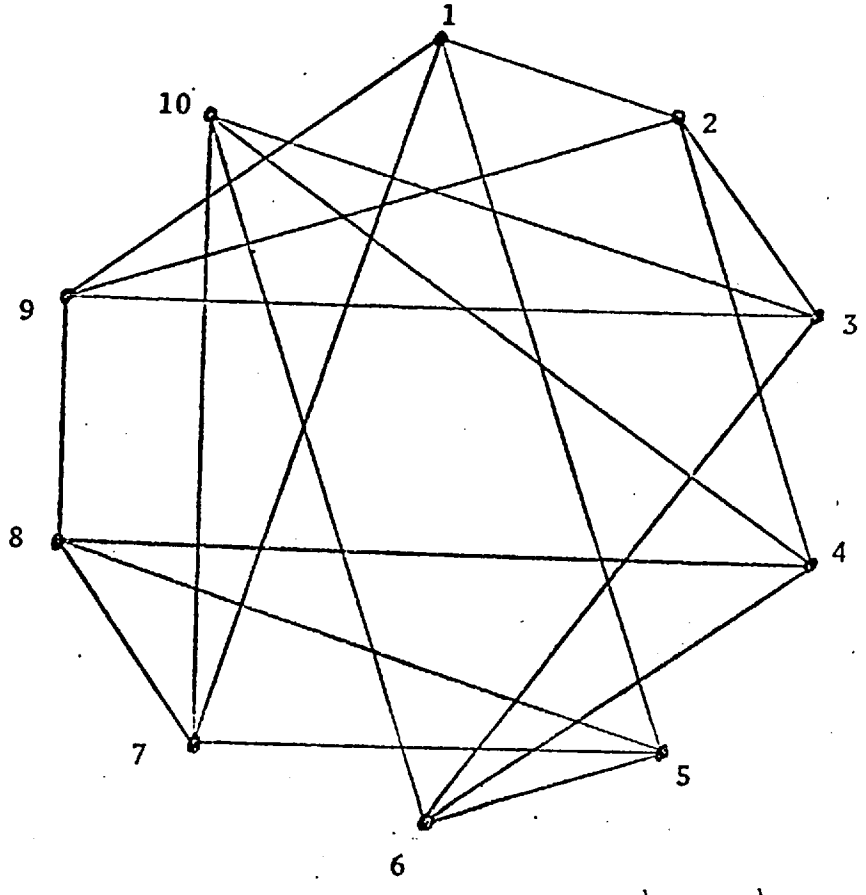


Figure 2(c). The optimum solution for problems Q^4 and S^4 over the cost matrix in Figure 2(a)

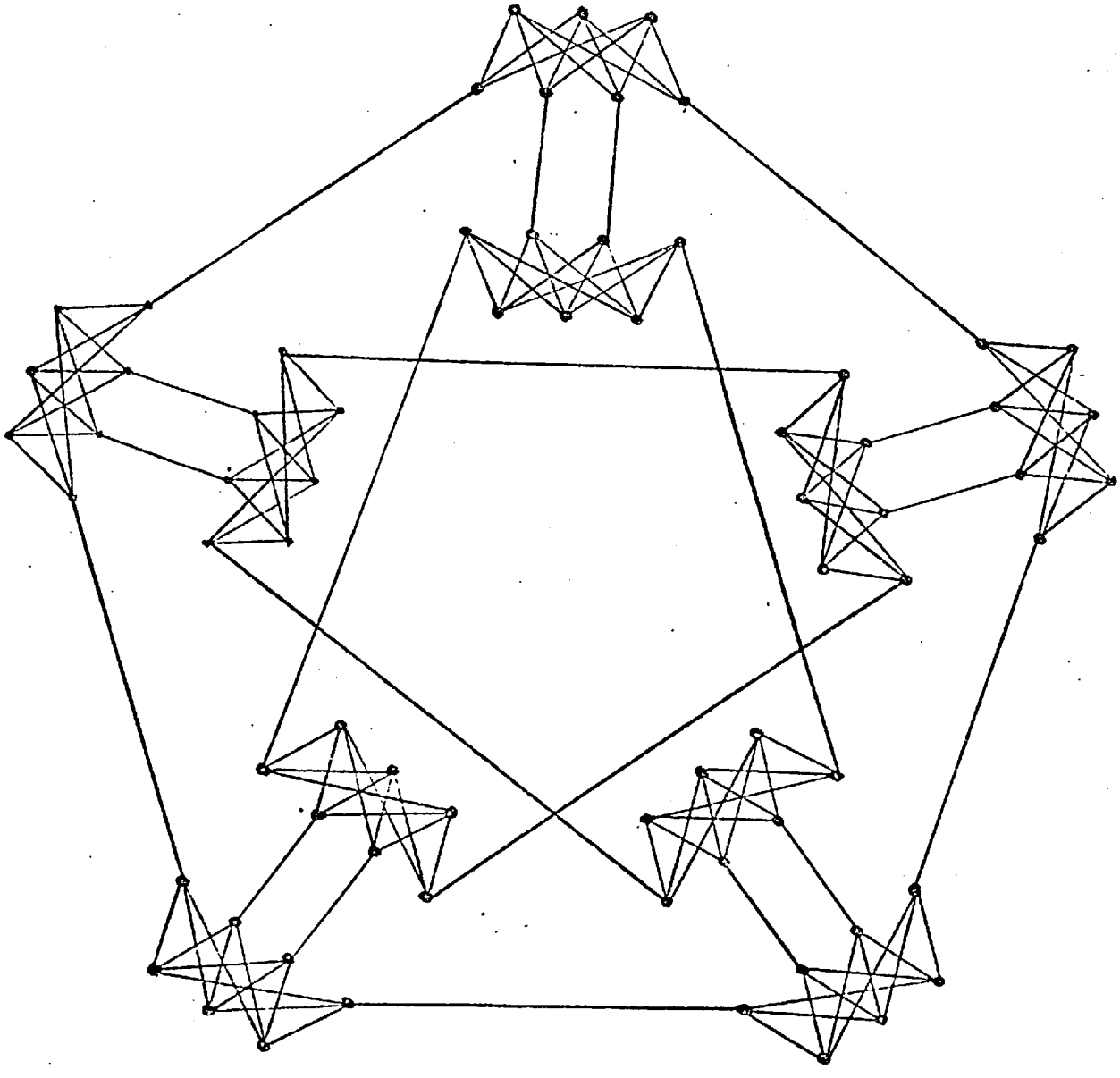


Figure 3. A graph which is an optimum solution to S^4 but not to Q^4

solution when the arc costs are triangular and n.u is even. The above results in terms of the properties of G_p are summarised in Figure 4.

From the above results it is clear that, in general, very little can be said about the properties of the solution to problem P^u. However, in many tests it has been found that a large proportion of the solutions are u-regular and that if u is even they contain u/2

	n.u even	n.u. odd
random c_{ij}	Not UR Not u/2 DHC	Not UR Not u/2 DHC
triangular c_{ij}	Postulated UR Not u/2 DHC	Not UR Not u/2 DHC
uniform	UR u/2 DHC - u even not u/2 DHC - odd	Not UR Not u/2 DHC

Figure 4. Structure of the solution graph G_p to problem P^u.

UR ≡ U-regular

u/2 DHC ≡ Contains u/2 disjoint hamiltonian circuits

disjoint hamiltonian circuits.

2. Branch and Bound Algorithm A for the Solution of S^u

In this section we shall describe a branch and bound algorithm for the solution of problem S^u. This method uses a routine for solving the matching problem M^u, which is a relaxation of S^u, implanted in a branch and bound procedure. The Gomory H_u algorithm for network flow is used to test the solutions of M^u for feasibility. The use of this algorithm for evaluating θ for a

graph is described in Chapter 4, Section 2.1.

Call the family of u-regular partial graphs of a graph G $\mathcal{G}_M^u$, clearly $G_M \in \mathcal{G}_M^u$ and $G_S \in \mathcal{G}_M^u$. Call the family of u-regular u-arc connected, partial graphs of a graph G $\mathcal{G}_S^u$. Clearly $\mathcal{G}_S^u$ is a subfamily of $\mathcal{G}_M^u$, and G_S is the minimum cost member of $\mathcal{G}_S^u$. One way of finding the minimum cost member of the family $\mathcal{G}_S^u$ would be to implicitly enumerate all members of $\mathcal{G}_M^u$, checking if they are in $\mathcal{G}_S^u$. The problem of finding the minimum cost member of $\mathcal{G}_M^u$ for any graph G is the matching problem M^u. An efficient method to solve the matching problem has been developed by Edmonds [2, 3, 4, 5]. Thus, it is possible to embed an algorithm for M^u in a branch and bound procedure to find the minimum cost member of $\mathcal{G}_S^u$.

The value of the solution to M^u can be used as a lower bound on the value of the solution to S^u. For constraints which are violated by the matching solution Lagrangean penalties can be applied to the arc costs in order to obtain a better lower bound. We shall now describe the bounding procedure.

2.1 The Calculation of Lower Bounds on S^u

If $G_M = (X, A_M)$ is the solution graph for M^u then, we know that $G_M \notin \mathcal{G}_S^u$ if there exists a cut $K_t = (S_t, \bar{S}_t)$ in G_M with cardinality $|K_t| < u$, where K_t is defined as

$$K_t = \{(x_i, x_j) \mid x_i \in S_t,\ x_j \in \bar{S}_t\}$$

where $S_t \subset X$ and $\bar{S}_t = X - S_t$.

The set of all minimum cuts between every pair of vertices of M can be computed by the Gomory-Hu algorithm in $O(n^3)$ step.

Problem S^u can be formulated as follows:

$$\text{Min} \sum_{\ell=1}^{m} c_\ell \xi_\ell \qquad (5)$$

such that:

$$d_i = u, \qquad \forall x_i \in X \tag{6}$$

$$\sum_{\ell \in K_t} \xi_\ell \geq u, \qquad \forall \text{ cuts } K_t \tag{7}$$

$$\xi_\ell \in \{0,1\} \tag{8}$$

where c_ℓ is the cost of arc ℓ (m is the total number of arcs in the graph), $\xi_\ell = 1$ if arc ℓ is in the solution, $\xi_\ell = 0$ otherwise.

Let us assume that constraints (7) are relaxed, the resulting problem solved as a matching problem, and that the application of the Gomory-Hu algorithm has identified cuts K_t, $t = 1,\ldots,T'$ for which $|K_t| < u$.

The lagrangian relaxation with multipliers λ' on these cuts produces the problem $M^u(\lambda')$ defined as follows:

$$\text{Min} \sum_{\ell=1}^{m} c_\ell \xi_\ell - \sum_{t=1}^{T'} \lambda'_t \sum_{\ell \in K_t} \xi_\ell + \sum_{t=1}^{T'} \lambda'_t \tag{9}$$

such that

$$d_i = u \qquad \forall x_i \in X \tag{6}$$

$$\xi_\ell \in \{0,1\} \; . \tag{8}$$

The solution to $M^u(\lambda')$ - for a given λ' - can be found by using the matching algirithm. This solution can also be tested for u-connectivity which could in turn generate more cuts T'+1, T'+2, etc.

Thus, in general we have to solve a matching problem $M^u(\lambda)$ with lagrange multipliers λ for a set $t=1,\ldots,T$ of cuts K_t.

A lower bound on the value of the solution to problem S^u is then given by:

$$V(M^u(\lambda^*)) = \max_{\lambda} [V(M^u(\lambda))] \tag{10}$$

where V(a) is the value of the optimal solution to problem a. Subgradient optimisation can be used to solve approximately the problem given in (10) to find λ^*.

2.2 Iterative Procedure to Improve the Lower Bound

Two situations can arise at the end of the solution to equation (10).

(i) The solution corresponding to $V(M^u(\lambda^*))$ is a feasible solution to S^u. This occurs under the following two conditions:

(a) If the relaxed constraints are satisfied with equality then this solution is optimal.

(b) If the relaxed constraints are not satisfied with equality the solution provides an upper bound on the optimal solution.

or

(ii) The solution corresponding to $V(M^u(\lambda^*))$ is not a feasible solution to S^u and therefore at least one constraint 7 is not satisfied for some cut $K_{\overline{t}}$.

In case i(a) the procedure stops (Backtracks in the general case tree search). In case i(b) the best solution so far may be updated - but branching forward must occur. In case (ii) a lower bound given by equation (10) is obtained to be used in the tree search.

2.3 Branching Procedure

We will now describe the branching procedure for this algorithm.

If a u-matching problem M^u is solved for a graph G then the resulting partial graph G_M may also be the solution graph G_S for problem S^u, i.e. it is the minimum cost member of $\mathcal{G}_S^u$. If that is the case the problem is finished. If however $G_M \notin \mathcal{G}_S^u$ a tree search is necessary.

If $G_M \notin \mathcal{G}_S^u$, then there must be at least one arc in G_M which must be replaced by some other arc, not in G_M. A tree search can be developed on the principle of removing arcs that occur in G_M from the graph and resolving M^u for the new graph produced. Thus, if at the initial node of the tree $G_M \notin \mathcal{G}_M^u$ and there is a set of arcs A_M' at least one of which must be replaced in G_M, for G_M to become a number of $\mathcal{G}_S^u$, then a branch is generated from this node for each arc in A_M'. Each branching corresponds to the elimination of an arc in A_M' from G and the fixing of other arcs to be in the solution. For example if $\{a_1, a_2, \ldots, a_k\}$ are the arcs in A_M' then the nodes of the tree are generated as:

Node 1 : $(\bar{a}_1)$

Node 2 : $(a_1, \bar{a}_2)$

Node 3 : $(a_1, a_2, \bar{a}_3)$

$\vdots$

Node k : $(a_1, a_2, \ldots, a_{k-1}, \bar{a}_k)$

where a_i implies that arc a_i must be in the solution and $\bar{a}_i$ implies that arc a_i must be excluded. The procedure is then repeated for each one of the graphs corresponding to the nodes of the tree.

If at any node, the solution to the p-matching problem for the partial graph G" (say), corresponding to that node, is an element of $\mathcal{G}_S^u$ that node is fathomed and backtracking occurs (with updating of the currently best solution if possible).

If the u-matching solution for G" is not an element of $\mathcal{G}_S^u$ but the bound (the value of the u-matching solution iteratively improved by the lagrangean procedure described above) is greater than or equal to the value of the best solution so far, again the node is fathomed and backtracking occurs. If the bound is less than the value of the best

solution so far, forward branching continues from that node.

A depth first procedure was adopted in the above branch and bound algorithm. The branchings at each node are ordered according to the costs of the arcs in A'_M, the low costs arcs being branched on first.

2.4 Selection of the Set A'_M of Arcs to Branch On

In this section we shall describe how to chose the set A'_M of arcs to branch on. Clearly one choice would be to choose A'_M to be the set of arcs in G_M. However, this would lead to a very large number of branches being generated at each node.

A better approach to this problem is to use the analysis provided by the Gomory-Hu algorithm to choose the set of arcs. Consider a cut K_t for which constraints 7 are not satisfied. K_t partitions the set of vertices of G_M into S_t and $\bar{S}_t$. For G_M to become a member of $\mathcal{G}^u_S$ there must at least be one additional arc crossing K_t. The degrees of the vertices in S_t and $\bar{S}_t$ must be u. Hence there must be at least one arc $a_\ell = (x_i, x_j)$, $x_i, x_j \in S_t$ in G_M that is not in G_S for the given graph G. Similarly for at least one arc of the type $a_\ell = (x_i, x_j)$, $x_i, x_j \in \bar{S}_t$. Thus A'_m can be chosen to be one of the two sets:

$$B_t = \{a_\ell \mid a_\ell = (x_i, x_j) \text{ and } x_i, x_j \in S_t\}$$

or

$$B_t = \{a_\ell \mid a_\ell = (x_i, x_j) \text{ and } x_i, x_j \in \bar{S}_t\} .$$

In order to reduce the number of branches at each node A'_M is chosen as follows

$$A'_M = B_t \quad \text{if} \quad |B_t| = \min\{|B_t|, |B'_t|\}$$

$$A'_M = B'_t \quad \text{if} \quad |B'_t| = \min\{|B_t|, |B'_t|\}.$$

In general there will be T' cuts violating constraints 7 and A'_M is chosen from all these cuts as follows:

$$\left.\begin{aligned} A'_M = B_{t*} \quad &\text{if} \quad |B_{t*}| = \min_{t=1,T'}\{|B_t|, |B'_t|\} \\ A'_M = B'_{t*} \quad &\text{if} \quad |B'_{t*}| = \min_{t=1,T'}\{|B_t|, |B'_t|\} \end{aligned}\right\} \qquad (11)$$

A flowchart for the complete algorithm A described above is given in Figure 5.

3. Branch and Bound Algorithm B for the Solution of Q^u

In this section algorithm B for solving problem Q^u will be described. Q^u is the problem of finding u/2 least cost arc disjoint hamiltonian circuits in a graph. This problem is similar to problem S^u, which is solved by algorithm A, but with the additional constraint that the solution graph G_Q contains u/2 arc disjoint hamiltonian circuits. It is therefore possible to modify algorithm A for problem S^u to include these additional constraints and solve problem Q^u.

Algorithm A works by implicitly enumerating all members of the family of graphs $\mathcal{G}_M^u$ and checking if they are also members of $\mathcal{G}_S^u$. This is done by the use of the efficient Gomory-Hu algorithm. Call the family of partial graphs of a graph G which contain u/2 arc disjoint hamiltonian circuits $\mathcal{G}_Q^u$. Thus, algorithm B could work in a similar manner to algorithm A by implicitly enumerating all members of $\mathcal{G}_M^u$ and checking if they are in $\mathcal{G}_Q^u$. However, to check a graph for membership of $\mathcal{G}_Q^u$ is not easy. It is well known, for example, that the problem of deciding whether a graph is hamiltonian is NP complete. In this section we shall describe two methods for investigating membership of $\mathcal{G}_Q^u$. One method works by implicit enumeration, and the other by forming a graph of hamiltonian circuits.

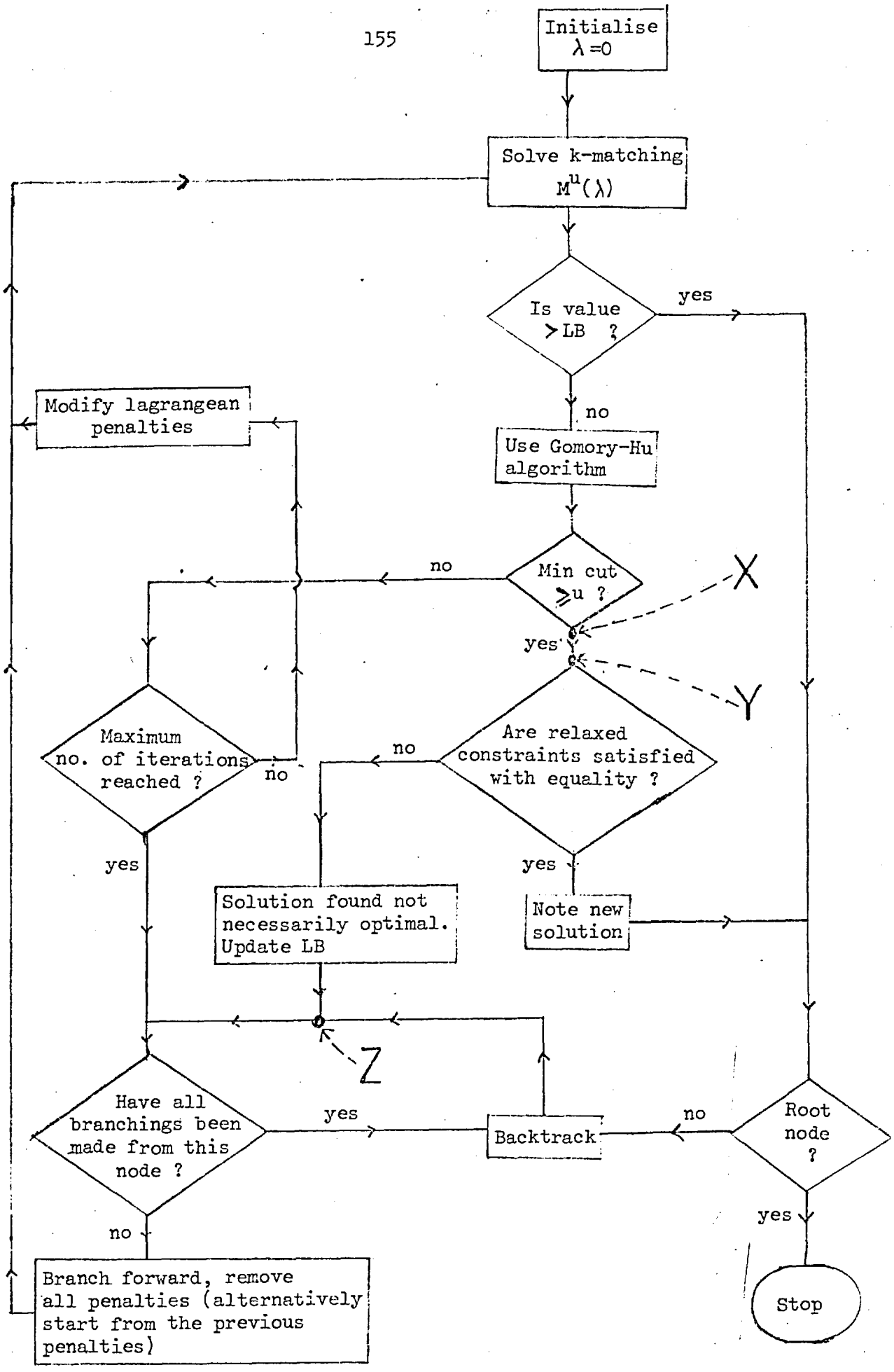


Figure 5 Flowchart for algorithm A

3.1 Method for Finding if a Graph Contains u/2 Disjoint Hamiltonian Circuits Using Implicit Enumeration

The method described in this section is based on the use of an algorithm developed by Selby [8] called the multipath algorithm, which is a procedure for enumerating all the hamiltonian circuits in a graph G. The method of Selby is an effective procedure for generating hamiltonian circuits. However, it is slow at proving that a graph does not have hamiltonian circuits. Hence, it is desirable to do some preprocessing on any graph G to be input to this procedure to ensure that it has a high probability of containing a hamiltonian circuit (or show it to be nonhamiltonian).

In Section 2.1 it was shown that the families of graphs $\mathcal{G}_S^u$ and $\mathcal{G}_Q^u$ are not equivalent. However, in many tests it has been found that a large proportion of members of $\mathcal{G}_S^u$ are also in $\mathcal{G}_Q^u$. Thus, if a graph is a member of $\mathcal{G}_S^u$ there is a high probability of it also being in $\mathcal{G}_Q^u$. Membership of $\mathcal{G}_S^u$ can be checked using the Gomory-Hu algorithm.

The procedure for finding if a graph has u/2 disjoint hamiltonian circuits is as follows:

Procedure H:

1. Set $A_0 = A$, $G_0 = G = (X,A_0)$, $n = 0$.
2. Use Gomory-Hu algorithm on G_n.
3. If $G_n \notin \mathcal{G}_s^{(u-2n)}$ go to 7, else continue.
4. Use multipath algorithms to generate a hamiltonian circuit with arcs H_n in G_n. If none exists go to 7, else continue.
5. If $n = u/2$ stop ($G \in \mathcal{G}_Q^u$), else continue.
6. Set $n = n+1$, $G_n = (X,(A_{n-1}-H_n))$ and go to 2.
7. If $n = 0$ stop ($G \notin \mathcal{G}_Q^u$); else continue.
8. Set $n = n - 1$ and go to 4.

The above procedure H generates a tree in which each node corresponds to the initial graph G with the arcs of a number of arc-disjoint hamiltonian circuits removed. This tree structure is illustrated in Figure 6. The value of u is 8 and the procedure is searching for u/2 = 4 arc disjoint hamiltonian circuits. Note that the superfix on the letters G and H represents the generation number of the graph or hamiltonian circuits with respect to the predecessor graph. Thus the superfix 2 on H_2^2 indicates that H_2^2 is the second hamiltonian circuit to be generated from G_1^1. Also the superfix 1 on G_1^1 indicates that G_1^1 is the first graph to be generated from G_0.

The tree generated by procedure H is potentially very large. However, as membership of $\mathcal{G}_S^u$ is a good test of membership of $\mathcal{G}_Q^u$ in most examples that have been tried, in which the graph G was a member of $\mathcal{G}_M^u$, the procedure has either stopped at n=0 or found u/2 are disjoint hamiltonian circuits in one pass down the tree.

3.2 Method of Constructing the Graph of Hamiltonian Circuits

In this section we describe an alternative to procedure H. This method is also based on the multipath algorithm for finding disjoint hamiltonian circuits.

In this procedure (I say) the Gomory-Hu algorithm is first used to test for membership of $\mathcal{G}_S^u$, if G is not in $\mathcal{G}_S^u$ then clearly $G \notin \mathcal{G}_Q^u$. The multipath algorithm is then used to successively generate all h (say) hamiltonian circuits with arc sets H_i, i = 1,...,h in G. The hamiltonian circuits of G are generated in blocks of u/2 hamiltonian circuits at a time. Procedure I forms a graph G^H of hamiltonian circuits which is updated after each block generation. In G^H each vertex corresponds to a hamiltonian circuit in G. Thus, after the ℓ'th stage of the procedure G^H contains $\frac{u\ell}{2}$ vertices x_i^H,

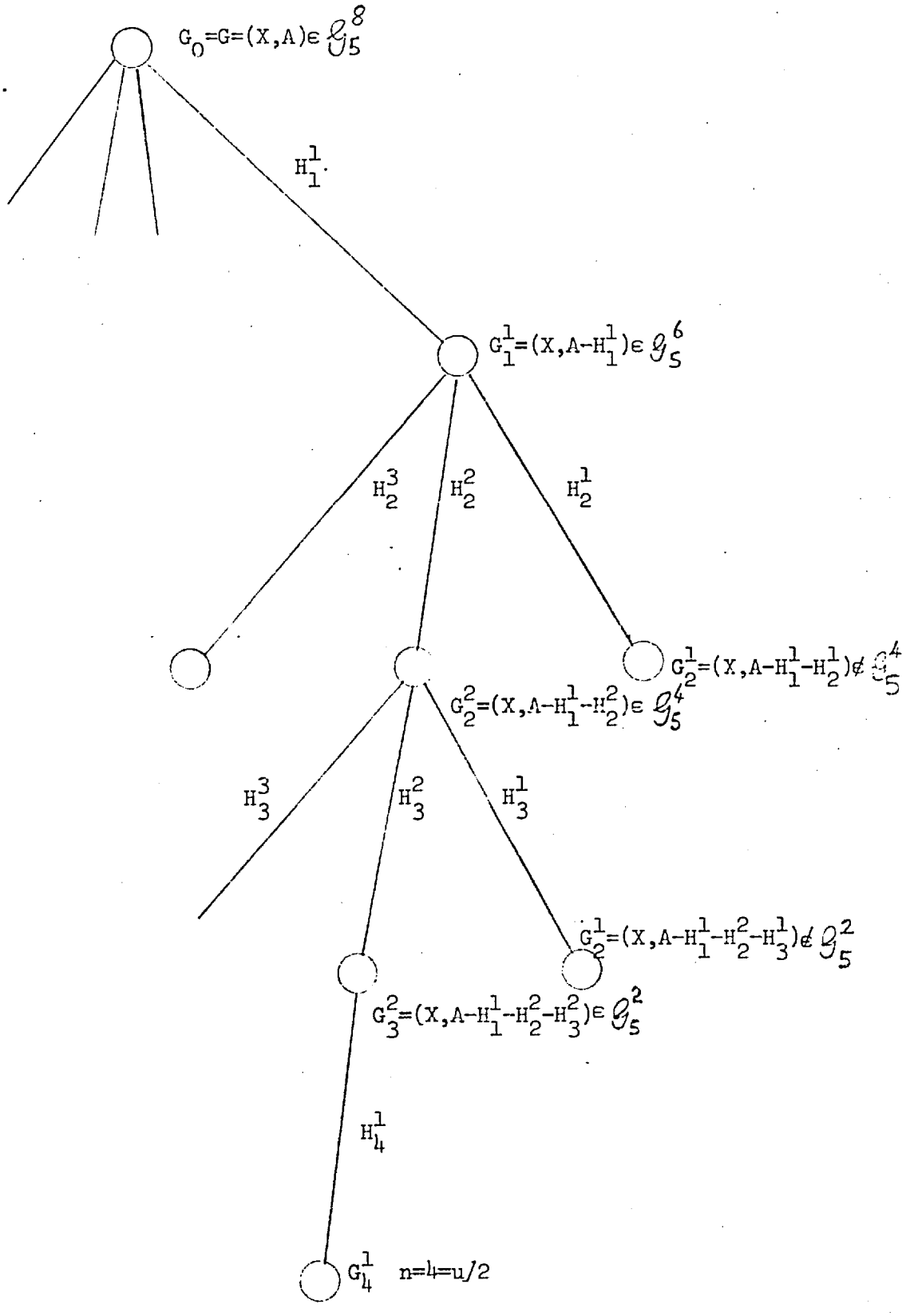


Figure 6 Illustration of the tree generated by procedure H for u=8

$i = 1,\ldots,\ell$. G^H is such that x_i^H and x_j^H are connected iff H_i and H_j have an arc in common. G^H is formally defined at the ℓ'th step of the procedure as follows:

$$G^H = (X,A^H),\ X^H = \{x_i^H\} \quad i = 1,\ldots,u\ell/2$$

$$A^H = \{(x_i^H,x_j^H) \mid H_i \cup H_j \neq \phi\} \tag{12}$$

Clearly the size α of the maximum independent set [1] of G^H at step ℓ is equal to the number of arc-disjoint hamiltonian circuits in G that have been generated up to step ℓ. In n^H is the number of vertices in G^H and m^H is the number of arcs, then we have:

$$\alpha \leq n^H - \lfloor \sqrt{2m^H} + \tfrac{1}{2} \rfloor + 1 .$$

A graph illustrating G^H is given in Figure 7. The size of α in Figure 7 is 4 hence the graph corresponding to G^H has 4 arc disjoint hamiltonian circuits.

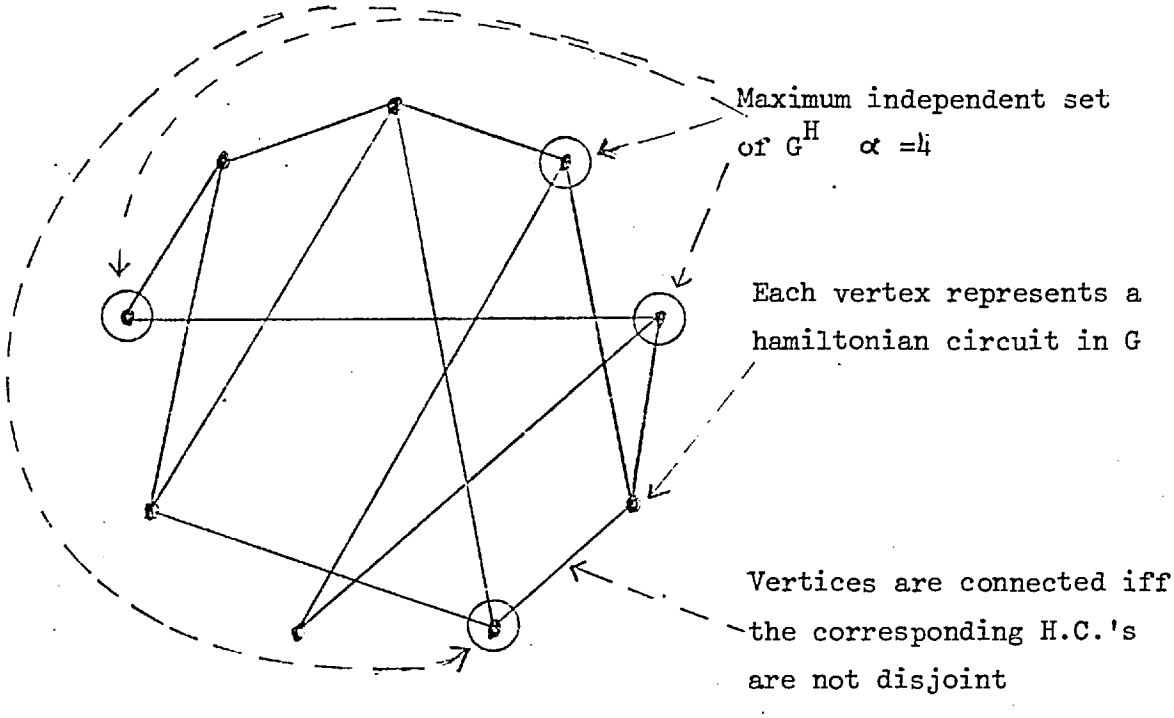


Figure 7 Graph G^H produced by procedure I

At each step of the procedure the value of α for G^H is checked. If it is found that $\alpha \geq u/2$ then the procedure is terminated as u/2 G contains u/2 disjoint hamiltonian circuits. If $\alpha < u/2$ then the next u/2 hamiltonian circuits are generated. If G^H is generated for which $\alpha \geq u/2$ then the procedure terminates when all h hamiltonian circuits of G have been generated.

3.3 Inclusion of a Disjoint Hamiltonian Circuits Procedure (H or I) into a Branch and Bound Algorithm B

In this section we describe algorithm B for solving problem Q^u. This algorithm is very similar to algorithm A, with the addition of a procedure to check if a graph is a member of $\mathcal{G}_Q^u$.

It can be seen from the previous two sections that the test for membership of the family $\mathcal{G}_Q^u$ is not an easy problem to solve. Potentially the two procedures H and I are computationally very expensive (although in tests performed on procedure H it was found to work in one pass for most cases considered). Thus, it is desirable to use such a procedure as little as possible, and use a computationally less expensive but also less accurate procedures wherever possible. This can be achieved by first using the test for membership of $\mathcal{G}_S^u$ before using tests for $\mathcal{G}_Q^u$. Thus, the initial stage in both procedures H and I - which uses the Gomory-Hu algorithm to test for membership of $\mathcal{G}_S^u$ - is essential.

The first stages are the initial tests for membership of $\mathcal{G}_S^u$ and the second stages are the rest of the procedures which test for membership of $\mathcal{G}_Q^u$. Call these second stages H^- and I^- for H and I respectively. Algorithm B can then be formed from Algorithm A (see Figure 5) by inserting either test H^- or I^- at the point marked X in Figure 5 and proceeding to point Y if the solution is found to be a member of $\mathcal{G}_Q^u$ or proceeding to point Z otherwise.

4. Algorithm C for Solving P^u and S^u

In this section we describe an L.P. based Algorithm C for solving problems P^u and S^u. This method uses branch and bound in conjunction with an L.P. relaxation of the problem. The solution to the relaxed L.P. is tested for feasability using the Gomory-Hu algorithm.

4.1 Problem Formulation

An integer programming formulation of problem P^u is as follows:

$$\text{Min} \sum_{j>i} \sum_{i=1}^{m} c_{ij}\, \xi_{ij} \tag{12}$$

such that

$$\sum_{\substack{x_i \in X_k \\ x_j \in \overline{X}_k}} \xi_{ij} \geq u \tag{13}$$

for all partitions $(X_k, \overline{X}_k)$ of the set of vertices X.

$$\xi_{ij} \in \{0,1\} \; . \tag{14}$$

Note that the above formulation is similar in structure to that given in Section 2.1 for S^u. In the formulation for S^u there are equality constraints on the degree of each vertex (expression 6). The objective function in the above is a double summation over vertices, whereas (5) is a summation over arcs the two objectives are, however, the same.

The number of constraints (13) is very large. Let the cardinality of X_k be k. Then $|\overline{X}_k| = n-k$ where n is the number of vertices in the graph. For each value of k there are $\binom{n}{k}$ unique partitions $(X_k, \overline{X}_k)$. The total number of partitions and thus constraints of type (13) is then $\sum_{k=1}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{k}$ which is of the order of 2^{n-1}. Clearly,

any algorithm that is going to be practical will not be able to explicitly use all these constraints. Thus, in order to develop an effective method for solving P^u based on the formulation (12) - (14) given above, it is necessary to consider ways of relaxing the constraints (13) and (14), to produce subproblems which are easier to solve. Two types of relaxations corresponding to the two sets of constraints (13) and (14) can be considered. These are as follows:

(i) Relaxation of constraints (13): If a number of constraints of type (13) were relaxed to produce a problem of reasonable size, then the I.P. could be solved and the additional constraints (13) tested for feasbility. Any violated constraints being added to produce a new problem to be resolved.

(ii) Relaxation of the constraints (14): The integrality constraints (14) can be relaxed so that the ξ_{ij} are continuosuly variable in the range:

$$0 \leq \xi_{ij} \leq 1 \ . \tag{15}$$

This leaves a very large linear program to solve. If it were possible to solve this L.P. then the integer program could be solved by using branch and bound.

The method that has been adopted for Algorithm C is to combine both relaxations (i) and (ii). Constraints (13) and (14) are both relaxed. Constraints of type (13) that are violated by the solution to this L.P. relaxation we first added to obtain an optimal solution to the LP described in (ii) above. Constraints (14) are then imposed by the use of branch and bound.

First we shall describe how to obtain an optimal solution to the L.P. described in (ii) above.

4.2 Solution of the L.P. Relaxation of P^u

The L.P. relaxation of P^u (P^u_L say) in which (14) are relaxed to (15) leads to a very large problem. Relaxations of some of constraints (13) have to be considered. To develop a useful relaxation it is convenient to split up constraints (13) into two sets of constraints. Those for k=1 (16), and those for $\left|\frac{n}{2}\right| \geq k > 1$ (15). The L.P. relaxation of P^u then becomes:

Problem P^u_L

Minimise

$$\sum_{j>i} \sum_{i=1} c_{ij}\, \xi_{ij} \tag{12}$$

such that

$$\sum_{\substack{x_i \in X_1 \\ x_j \in \bar{X}_1}} \xi_{ij} \geq u \tag{16}$$

for all n partitions $(X_1, \bar{X}_1)$ of the set of vertices X,

$$\sum_{\substack{x_i \in X_k \\ x_j \in \bar{X}_k \\ 2 \leq k \leq \left|\frac{n}{2}\right|}} \xi_{ij} \geq u \tag{17}$$

for all partitions $(X_k, \bar{X}_k)$ of the set of vertices X.

$$0 \leq \xi_{ij} \leq 1 \tag{15}$$

Call the constraints (16) the degree constraints. This is because if the ξ_{ij} were integer then (16) would specify the minimum degree of each vertex. Similarly call the constraints (17) the arc cut-set constraints. This is because for ξ_{ij} integer they impose a minimum size for any cut set in the solution graph G_p.

Consider the L.P. defined by (12), (15) and (16) above, i.e. by

the removal of the arc cut-set constraints (17). This is a reasonably small L.P. with only n constraints and m variables which can easily be solved for large graphs. Call this problem P_S^u.

In Algorithm A the solution to P_S^u is used as a starting point for finding the optimal solution to P_L^u. This is done by imposing onto P_S^u the constraints in P_L^u that are violated in the solution of P_S^u. Thus, it is required to check the solution to P_S^u to see if it is a solution to P_L^u, and if it is not which constraints in P_L^u are violated. Clearly this cannot be done by checking each individual constraint (17) for feasibility, and a different approach is necessary.

The Gomory-Hu algorithm can again be used to check if any constraints (17) were violated. This can be done in a way analogous to method described in Chapter 4. In the method in Chapter 4 each arc is given a capacity 1, then the Gomory-Hu algorithm computes the maximum flows which are then equal to the arc connectivities. The relevant minimum cuts are also given. A graph G_ξ which can be analysed by the Gomory-Hu algorithm to check if any constraints (17) are violated is constructed as follows:

$$G_\xi = (X, A_\xi)$$

$$X = (x_1, x_2, \ldots, x_n)$$

$$A_\xi = \{(x_i, x_j) \mid \xi_{ij} \neq 0\}$$

the capacity q_{ij} of arc (x_i, x_j) is given by

$$q_{ij} = \xi_{ij} . \qquad (18)$$

Analysis of G_ξ by the Gomory-Hu algorithm gives the cuts and hence the constraints (17) which are violated by (ξ_{ij}). In this case ξ_{ij} can be considered as representing the <u>fraction</u> of arc (x_i, x_j). In terms of fraction of an arc, constraints (16) and (17) are given a

different interpretation. Constraint (16) implies that the sum of fractions of an arc incident at every vertex should be greater than or equal to u. Similarly (17) implies that the sum of fractions of arcs crossing every cut should be greater than or equal to u.

In practice the Gomory-Hu algorithm expects arc capacities to be integer, not fractional as implied by (18). However, this is easily overcome by multiplying all the values of the ξ_{ij} in the solution to P_S^u by some large scaling factor I (e.g. 1000).

4.2.1 Procedure L for Solving P_L^u

The initial step of procedure L is to solve P_S^u. P_S^u is formulated as follows:

$$\text{minimise} \sum_{j>i} \sum_{i=1}^{n} c_{ij}\, \xi_{ij} \tag{12}$$

such that

$$\sum_{\substack{x_i \in X_1 \\ x_j \in \bar{X}_1}} \xi_{ij} \geq u \tag{16}$$

for all n partitions $(X_1, \bar{X}_1)$ of the set of n vertices X.

$$0 \leq \xi_{ij} \leq 1 \tag{15}$$

Call solution of P_S^u, $\{\xi_{ij}^s\}$. Choose a scaling factor I and form a graph $G_\xi = (X, A_\xi)$ where $(x_i, x_j) \in A_\xi$, if $\xi_{ij} \neq 0$. Set the capacities to the arcs in G to be q_{ij} where,

$$q_{ij} = \lfloor I * \xi_{ij} + 0.5 \rfloor .$$

If there is a cut K_t^* in G for which

$$Q(K_t^*) < u * I$$

where $Q(K_t^*)$ is defined as the capacity of K_t^*, then a constraint (17)

is violated by the $\{\xi^{s}_{ij}\}$. In this case the violated constraints can be added to the tableau for P^{u}_{S} and this new LP - $P^{u}_{S'}$, (say) - solved to produce a new solution $\{\xi^{s'}_{ij}\}$. The new solution $\{\xi^{s'}_{ij}\}$ can again be checked to see if it violates any constraints (17), etc. The optimal solution to P^{u}_{L} is found when none of the constraints (17) are violated.

In practice it has been found that it is not necessary to add a large number of constraints (17) in order to produce an optimal solution to P^{u}_{L}. For most test problems it was not necessary to add more than n additional constraints, often the figure was well below n.

In order to reduce computational effort only one of the violated constraints (17) is added for each use of the Gomory-Hu algorithm. This makes it possible to stop computation of the Gomory-Hu algorithm as soon as one violated constraint is found. The LP tableau for $P^{u}_{S'}$ with just one extra constraint added is also easy to reoptimise. The addition of only one constraint to the LP often caused other constraints to be satisfied in the new LP solution.

The complete procedure L for finding the optimal solution to P^{u}_{L} can be described as follows:

1.0 Set $P^{LP} = P^{u}_{S}$.

2.0 Solve P^{LP}, call the solution $\{\xi^{LP}_{ij}\}$.

3.0 Form graph $G_{\xi} = (X, A_{\xi})$, where $(x_i, x_j) \in A_{\xi}$ if $\xi^{LP}_{ij} \neq 0$.

3.1 Let $q_{ij} = \lfloor I * \xi_{ij} + 0.5 \rfloor$ in G_{ξ}

3.2 Use Gomory-Hu algorithm on G_{ξ} until it generates a minimum cut K_t.

3.3 If $Q(K_t) < u * I$ go to 4.0, else continue.

3.4 If Gomory-Hu algorithm has generated all minimum cuts in G_{ξ} stop (ξ^{LP}_{ij} is the optimal solution to P^{u}_{L}). Otherwise go to 3.2.

4.0 Generate the constraint $C(K_t)$ (say) of type (17), corresponding to K_t.

4.1 Set $P^{LP} = P^{LP} \cup C(K_t)$ and go to 2.0.

4.3 Tree Search Algorithm

In the above sections a practical method for solving problem P_L^u is described. This problem is the LP relaxation of problem P^u which we are interested in solving. The method described in this section for the solution of P^u, embeds procedure L for P_L^u into a tree search algorithm. This method is not described in great detail as it is very similar to branch and bound algorithms for general integer programming. However, branching rules will be described more fully. The choice of branching rules for this algorithm is both more interesting and more important than for most integer programs, as the LP relaxation has an associated graph structure.

4.3.1 Enumeration Tree

At the initial node of the tree P_L^u is solved using Procedure L. Call the solution to P_L^u $\{\xi_{ij}^L\}$. If $\{\xi_{ij}^L\}$ is all integer then P^u is solved. However, if $\{\xi_{ij}^L\}$ is not all integer then it is necessary to generate branches of the enumeration tree. This is done by picking one of the non-integer variables in $\{\xi_{ij}^L\}$, ξ_{ij}^* (say), and generating two subproblems of P_L^u, one in which ξ_{ij}^* is fixed to zero and the other in which it is fixed to one. One or both of these subproblems is then solved using procedure L.

In general there are a number of unsolved subproblems in the enumeration tree. One of these is chosen using procedure L for solution.

If at any node the value of the solution to the LP subproblem at that node is greater than the value of the best solution so far, then the node is fathomed and no branches are generated from it. The complete tree is enumerated when all end nodes in the tree are fathomed or represent subproblems with all integer solutions.

The enumeration tree described above is similar in structure to that used for general 0-1 programs. However, the subproblem represented

at each node is solved by procedure L and not by ordinary linear programming. The methods for choosing which variable to branch on and for searching this tree will now be described.

(i) Branching Rules:

The nodes in the enumeration tree represent subproblems of P_L^u. If the solution $\{\xi_{ij}^L\}$ to one of these subproblems is non-integer then branches are generated by setting one of the non-integer variables ξ_{ij}^* (say) first to zero and then to one. The choice of ξ_{ij}^* is of considerable importance. The analysis provided by the Gomory-Hu algorithm gives information about the minimum cuts and flows in the solution. This can be used for making the selection of ξ_{ij}^*. The following rules can be applied:

Branching Rule 1:

Find the minimum cut K_t' which has the largest number of fractional arcs crossing it and branch on a variable corresponding to a fractional arc in K_t'. If a variable with a large fractional value is chosen then the first branching sets it to one, the second to zero. The reverse is done if it has a small fractional value.

Branching Rule 2:

This is the converse of rule 1. The minimum cut K_t' with the least number of fractional arcs crossing it is found. One of the fractional variables in K_t' is then branched on.

Branching Rule 3:

The cut K_t' with the least sum of fractional arcs crossing it is found. One of the fractional variables in K_t' is then branched on. If there is more than one such K_t' then the one with the most fractional variables is chosen.

(ii) Search Procedure

In the coding of algorithm C a depth first method for searching the branch and bound tree was adopted.

5. Computational Results for Algorithms A and B

5.1 Matching Code

In order to code algorithms A and B it is necessary to use a u-matching code. The one-matching algorithm described by Edmonds [5] was coded. U-matching problems were solved using this one-matching code in conjunction with a graph transformation, also described by Edmonds [5]. This transformation T (say) converts to a graph G, in which a u-matching is to be solved, into a graph T(G). T(G) is such that there is a one to one mapping between arcs in the one-matching solution of T(G) and the u-matching solution of G.

The transformation T is as follows:

Given a graph $G = (X,A)$ with n vertices x_i, $i=1,\ldots,n$ and m arcs a_ℓ, $\ell=1,\ldots,m$ with costs c_ℓ, $\ell=1,\ldots,m$, form the graph $G^u = (X^u,A^u)$ with $(u.n+2m)$ vertices x_i^u, $i=1,\ldots,(u.n+2m)$ and $m(2u+1)$ arcs a_ℓ^u, $=1,\ldots,m(2u+1)$ and costs c_ℓ^u, $=1,\ldots,m(2u+1)$, with the following structure.

(i) For each vertex x_i in X there corresponds u vertices x_{it}^u, $t=1,\ldots,u$ in X^u.

(ii) For each arc $a_\ell = (x_i,x_j)$ of cost c_ℓ in A there corresponds;

(a) Two vertices x_ℓ^{u1} and x_ℓ^{u2}.

(b) (2u+1) arcs,

(x_{it}^u, x_ℓ^{u1}), $t=1,\ldots,u$ each with cost $\frac{c_\ell}{2}$,

$(x_\ell^{u1}, x_\ell^{u2})$, with cost 0,

(x_ℓ^{u2}, x_{jt}^u), $t=1,\ldots,u$ each with cost $\frac{c_\ell}{2}$.

Figures 8(a) and 8(b) illustrate the above transformation for u=4 on a graph G consisting of one arc and two end vertices.

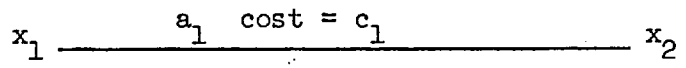


Figure 8(a) Graph G before transformation T

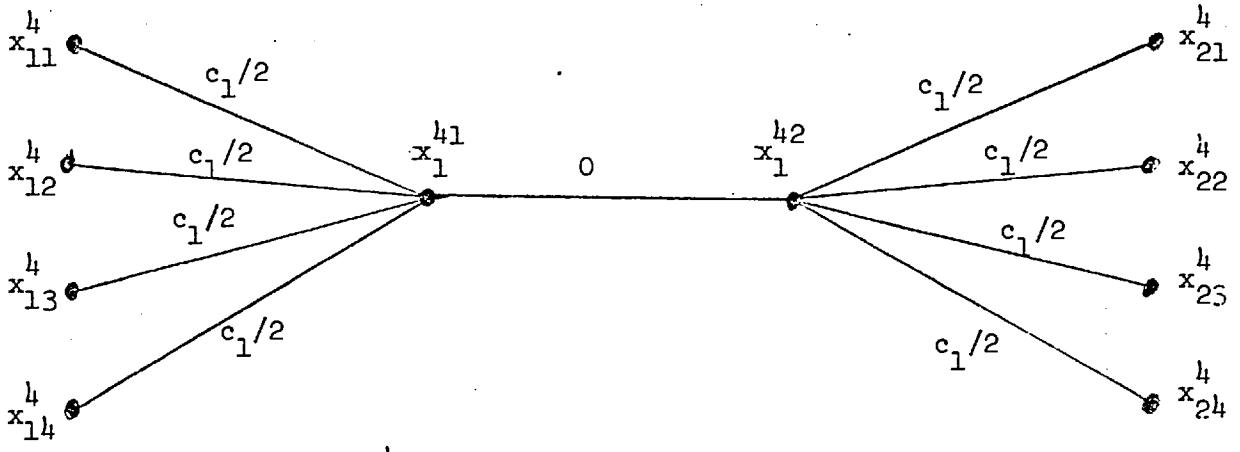


Figure 8(b) Graph G^4 after transformation T with u=4

T is very expensive in terms of computer storage space. For example if n is the number of vertices in G, and m the number of arcs, then T(G) contains un+2m vertices and m(2u+1) arcs.

It would have been possible to write a direct u-matching code [4]. However, it is believed* that at least in terms of computing time, this approach should not be much more efficient than using the graph transformation T with a one-matching algorithm . The objective was to test the general ideas behind the algorithms and not to develop the most efficient matching code, and since writing a u-matching code is a very difficult exercise in its own right, a 1-matching code was written and used in conjunction with the graph transformation T.

5.1.2 Results

In order to test algorithms A and B a large number of randomly generated network synthesis problems with u=4 were solved. The cost

---

* E. Johnson private conversation

matrix for each problem (only complete graphs were considered) was generated by sampling from a uniform integer distribution from the range 1 to 20. For $u=4$ it is not necessary to solve problems with less than 10 vertices. All 4-regular subgraphs of graphs with less than 10 vertices have $\theta=4$. Thus for problems with $5 \leq n \leq 9$ and $u=4$ the 4-matching is the optimal solution. A and B were only tested on graphs with $n \geq 10$ and $u=4$.

The results of these tests were quite surprising. A total of 1196 randomly generated problems were solved. In all problems the first 4-matching solution was the optimal solution. However, the choice of transformation T combined with a 1-matching, lead to severe computer storage problems.

The results demonstrate that the u-matching is an excellent bound for S^u and Q^u. A summary of the results for algorithm A is given in Table 1(a) and for algorithm B in Table 1(b). The algorithms were programmed in Fortran IV. They were run on a CDC 7600 computer and all times are in seconds on that machine.

5.2 Computational Results for Algorithm C

Algorithm C was also programmed in Fortran IV and run on the CDC 7600 computer and all computation times are in CDC 7600 seconds.

Algorithm C was tested on a large number of randomly generated problems. The graphs generated were complete with both euclidean and non-euclidean cost matrices. The non-euclidean costs were generated by sampling from a uniform integer distribution in the range 1 to 100. The costs for the euclidean problems were generated by randomly generating points on a euclidean plane 100 units square. A uniform distribution in the range 1 to 100 was used to generate the x and y coordinates of each point. The values of u used in the tests were 2, 3, 4 and 5.

No. of vertices	No. of problems solved	Average no. of nodes in time	(CDC 7600 seconds) Average time	Max. time	Min. time
10	41	1	1.3	1.898	0.692
11	27	1	1.98	2.741	1.325
12	17	1	3.01	3.755	1.938
13	12	1	4.44	5.611	3.055
14	9	1	5.58	8.137	3.827
15	6	1	8.66	11.55	6.815
16	48	1	10.15	13.495	6.71
17	35	1	13.85	21.138	10.1
18	3	1	17.85	20.491	14.782
19	21	1	22.44	26.328	14.105
20	15	1	31.31	41.977	22.978

Table 1(a): Results for Algorithm A

No. of vertices	No. of problems solved	Average no. of nodes in time	(CDC 7600 seconds) Average time	Max. time	Min. time
10	320	1	1.5	1.906	0.693
11	250	1	1.96	3.123	0.946
12	164	1	2.98	4.249	1.334
13	115	1	4.26	6.526	2.507
14	84	1	5.83	8.233	3.738
15	6	1	8.8	11.731	6.936
16	5	1	10.57	12.496	6.83
17	3	1	15.39	18.424	11.222
18	2	1	18.05	21.116	15.002
19	2	1	25.91	26.811	25.018
20	1	1	30.167	30.167	30.167

Table 1(b): Results for Algorithm B

Table 1: Results for randomly generated problems for algorithms A and B

An LP code was written specially for alogirhtm C as the commerically designed codes to which we had access did not allow for the addition of constraints. The results for algorithm C are shown in Table 2. Table 2(a) shows the results for random problem, Table 2(b) for euclidean problems.

No. of vertices	No. of problems solved				Average no. of nodes in tree				Average time (7600 sec)				Maximum time				Minimum time			
	u				u				u				u				u			
	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5
10	88	75	100	53	1.5	2.3	2	1	0.07	0.05	0.12	0.05	.53	.43	.77	0.06	.05	0.03	.09	0.04
20	50	46	13	32	2.5	2.1	1.8	12	0.28	0.33	0.37	2.17	.63	1.99	.72	14.80	.14	.18	.26	0.31
30	17	33	28	14	15	2.3	2.1	20	1.11	1.15	1.49	4.32	17.41	3.31	3.3	20.91	0.87	.7	.93	1.57
40	1	13	10	8	1	3.9	4.2	15	1.87	4.18	5.62	6.85	1.87	12.73	8.19	23.33	1.87	2.12	3.65	4.52
50	5	7	5	4	21	2.2	3.4	23	12.63	7.37	10.73	12.43	53.75	9.31	16.44	60.87	3.62	5.94	8.24	7.42

Table 2(a) : Performance of Algorithm C on Random Symmetric Cost Matrices

No. of vertices	No. of problems solved				Average no. of nodes in tree				Average time (7600 sec)				Maximum time				Minimum time			
	u				u				u				u				u			
	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5
10	9	7	66	62	1.8	171	1	1	0.13	2.29	0.04	0.05	0.27	5.33	0.05	0.06	0.03	0.92	0.04	0.04
20	23	3	39	33	2.8	233	2.1	26	0.72	6.42	0.43	2.33	1.42	7.84	2.62	19.85	0.44	4.23	.89	0.34
30	17	1	27	5	15	250	4.2	16	3.17	10.31	2.11	5.91	15.31	10.31	11.78	20.62	0.95	10.31	1.1	1.5
40	5	0	13	1	23	250+	3.1	1	14.69	120+	4.28	4.67	22.52	120+	8.29	4.67	2.46	120+	2.48	4.67
50	4	0	17	3	19	200+	7.8	14	14.62	120+	11.35	15.23	31.21	120+	38.3	44.23	3.67	120+	5.3	7.42

Table 2(b) : Performance of Algorithm C on Euclidean Cost Matrices

REFERENCES

1. CHRISTOFIDES, N., "Graph Theory - An Algorithmic Approach", Academic Press, London, 1975.
2. EDMONDS, J., "Paths, Trees and Flowers", Canadian Math. Journal, Vol. 17, 1965, pp. 499-466.
3. EDMONDS, J., "Maximum Matchings and a Polyhedron with 0-1 Vertices", Journal of Research, Nat. Bur. of Standards, Vol. 69B, 1965, pp. 125-130.
4. EDMONDS, J. and JOHNSON, E.L., "Matching : A Well Solved Class of Integer Linear Programs", Proc. of Calgary International Conference on Combinatorial Structures and Their Application, 1970, Gordon and Breach, New York, p. 89.
5. EDMONDS, J., "An Introduction to Matchings" , Summer seminar on Mathematics of the Decision Sciences 1967, Stanford University.
6. FRISCH, I.T., "A Computer Technique for Design on Analysis of Hard Communications Systems", Digest of the 1967 IEEE Itern. Conference on Communications, Minneapolis, 1967, p. 96.
7. MEREDITH, G.H.J., "Regular n-Valent n-Connected Non-Hamiltonian Non-n-Edge Colorable Graphs", Journal of Combinatorial Theory, Vol. 14B, 1973, pp. 55-60.
8. SELBY, G.R., "The Use of Topological Methods in Computer Aided Circuit Layout", Ph.D. Thesis, Imperial College, September 1970.

CHAPTER 6

THE TRAVELLING SALESMAN PROBLEM

1. Introduction

In this chapter we present a branch and bound algorithm (D) for the solution of the travelling salesman problem. The algorithm is similar in structure to algorithm C for problem P^u, presented in Chapter 5. A technique is used in the present algorithm to eliminate variables a priori from the 0-1 formulation of the TSP. This reduction is based on the reduced costs obtained from the solution to:

(a) The assignment problem lagranean relaxation at the end of an ascent procedure,

(b) The shortest spanning tree (SST) lagranean relaxation at the end of an ascent procedure,

together with a good upper bound obtained by a heuristic.

Test results are given for a number of problems in which the costs are:

(i) randomly generated symmetric and

(ii) euclidean.

1.1 Problem Formulation

The travelling salesman problam can be described as follows:

> Find the order in which a salesman must visit each one of a set of n cities once and only once and return to the starting city, so that the total distance travelled is a minimum.

For directed graphs (i.e. for graphs where the distances or costs between the vertices of the graph are not symmetric) there are a number of integer programming formulations of the problem, three of which are as follows:

Problem T_a^D or T_b^D or T_c^D

Given a graph $G = (X,A)$ with costs c_{ij} associated with each arc

(x_i, x_j) in A, there being n vertices in X and m arcs in A, then

$$\text{minimise} \quad \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij}\, \xi_{ij} \tag{1}$$

such that

$$\sum_{j=1}^{n} \xi_{ij} = 1, \qquad i=1,\ldots,n \tag{2}$$

$$\sum_{i=1}^{n} \xi_{ij} = 1, \qquad j=1,\ldots,n \tag{3}$$

and either

$$\sum_{\substack{i\in X_k \\ j\in \bar{X}_k}} \xi_{ij} \geq 1 \tag{4a}$$

or

$$\sum_{j\in X_k} \sum_{i\in X_k} \xi_{ij} \leq |X_k| - 1 \tag{4b}$$

or

$$\sum_{(i,j)\in \Phi_k} \xi_{ij} \leq |X_k| - 1 \tag{4c}$$

for all partitions $(X_k, \bar{X}_k)$ of the set vertices of the graph G, where $|X_k| = k$ and Φ_k is a hamiltonian circuit of X_k (note: for constraint (4a) $2 \leq k \leq n-2$, for constraints (4b) and (4c) $2 \leq k \leq n-1$)

$$\xi_{ij} = 0,1 \tag{5}$$

If the costs associated with each arc in the graph are symmetric i.e. the graph is non-directed, then the problem can be rewritten as follows:

Problem $T^N_{a \text{ or } b \text{ or } c}$

$$\text{minimise} \quad \sum_{j>i} \sum_{i=1}^{n} c_{ij}\, \xi_{ij} \tag{6}$$

such that

$$\sum_{\substack{x_i \in X_1 \\ x_j \in \bar{X}_1}} \xi_{ij} = 2 \qquad (7)$$

for all partitions $(X_1, \bar{X}_1)$ of the set of vertices X,

and either

$$\sum_{\substack{x_i \in X_k \\ x_j \in \bar{X}_k}} \xi_{ij} \geq 2 \qquad (8a)$$

or

$$\sum_{j \in X_k} \sum_{\substack{i \in X_k \\ j > i}} \xi_{ij} \leq |X_k| - 1 \qquad (8b)$$

or

$$\sum_{\substack{(i,j) \in \Phi_k \\ j > i}} \xi_{ij} \leq |X_k| - 1 \qquad (8c)$$

for all partitions $(X_k, \bar{X}_k)$ of the set of vertices X, where $|X_k| = k$ and Φ_k is a hamiltonian circuit of X_k (note: for constraint (8a) $2 \leq k \leq \left\lfloor \frac{n}{2} \right\rfloor$, for constraints (8b) and (8c) $2 \leq k \leq n-1$).

$$\xi_{ij} = 0,1 \qquad (9)$$

Problems T^D and T^N have subscripts a, b or c corresponding to which versions of the constraints 4 and 8 respectively are used.

It is worthwhile to note here that constraints (7, 8a or b or c) are equivalent to the two sets of constraints:

$$\sum_{\ell \in C_k} \xi_\ell \geq 1 \qquad \forall\, C_k \equiv (X_k, \bar{X}_k),\ X_k \subset X \qquad (10a)$$

and

$$\sum_{\ell \in A_i} x_\ell = 2 \qquad i=1,\ldots,n \qquad (10b)$$

where A_i is the set of arcs incident at vertex x_i.

Constraints 4(a,b,c) and 8(a,b,c) are the subtour elimination or connectivity constraints.

2. Brief Literature Survey

The travelling salesman problem has been studied by many authors [1-6, 8, 9, 11, 15-28], and many surveys have been written on this problem [3, 7]. In this section only a brief review of methods developed to solve the problem is given. More attention is paid to the methods which are of relevance to this chapter. In particular a bound developed by Balas and Christofides [1] is described in detail in the appendix. For a more comprehensive survey of the travelling salesman problem the reader is referred to [7].

2.1 All Integer Tree Search Methods

The two essential elements of a tree search algorithm are

(i) a branching rule, and

(ii) the bounding procedure.

For the TSP most attention has been paid to developing good bounds. This approach has given a greater payoff than developing more sophisticated branching rules (this is the case with most combinatorial problems). Most of this survey is therefore devoted to the study of bounds. However, we first give a brief discussion on branching rules.

2.1.1 Branching Rules for the TSP

In general two main branching rules for the TSP have been developed. These are as follows:

(i) Tour Construction Rule: This was first introduced by Little, Murty, Sweeny and Karel [23], their paper is one of the first examples of the use of tree search. The rule produces a binary search tree. At each node in the search tree one arc is chosen (according to some given rule) for branching on. Two successor nodes are generated by alternatively fixing this arc to be in or out of the solution. An arc is only fixed to be in the solution if its addition does not

violate any of the TSP constraints. If n forward branchings are made in which an arc is added to the solution at each branching, then a hamiltonian tour is constructed.

(ii) Subtour elimination rule: This rule is used in conjunction with a bounding procedure that generates subtours. For example the solutions to the assignment or matching problems are both bounds for the TSP and also generate subtours. It was first used by Shapiro [28] and later by Bellmore and Malone [3]. In this procedure the bound at each node of the search tree generates a set of subtours of the graph G. One of these subtours, S_r (say), is chosen for branching on. If the set of arcs in S_r is $\{a_1,a_2,\ldots,a_r\}$ then the following branches are generated from the node:

branch 1 $\{\bar{a}_1\}$

branch 2 $\{a_1,\bar{a}_2\}$

branch 3 $\{a_1,a_2,\bar{a}_3\}$

. .

. .

. .

branch r $\{a_1,a_2,\ldots,a_{r-1},\bar{a}_r\}$

where a_i implies the arc a_i is in the TSP solution and $\bar{a}_i$ implies arc a_i is not in the solution.

2.1.2 Bounding Procedures for the TSP

For all tree search procedures for the TSP the accuracy of the bound is more significant to the effectiveness of the procedure than the branching rule. Some of the bounding procedures which have been used are given in the following.

(i) Assignment and matching based bounds

The assignment problem is formulated for a directed graph G = (X,A) as follows:

Problem AP,

$$\text{minimise} \quad \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij}\, \xi_{ij}$$

such that

$$\sum_{j=1}^{n} \xi_{ij} = 1, \qquad i=1,\ldots,n$$

$$\sum_{i=1}^{n} \xi_{ij} = 1, \qquad j=1,\ldots,n$$

$$\xi_{ij} = 0 \text{ or } 1$$

The 2-matching problem is similar to the assignment problem but it is only formulated for non-directed graphs. For non-directed complete graphs it is as follows:

Problem M2,

$$\text{minimise} \quad \sum_{j>n} \sum_{i=1}^{n} c_{ij}\, \xi_{ij}$$

such that

$$\sum_{j=1}^{n} \xi_{ij} = 2, \qquad i=1,\ldots,n$$

$$\xi_{ij} = 0 \text{ or } 1 .$$

Problem AP is the same as the TSP for directed graphs with no constraints of type 4. Problem M2 is the same as the symmetric TSP with no constraints of type 8. In the solution to the assignment and matching problems subtours occur because these constraints are missing. The additional constraints are included by the tree search.

A more sophisticated bound based on the assignment problem has been developed by Balas and Christofides [1]. This imposes by

lagrangean relaxation constraints (4a), (4b) and (4c) which are violated by the assignment problem solution. We shall not describe this procedure in this section as it is used as part of algorithm D, and is described in the appendix.

(ii) Bounds from the shortest spanning tree

Symmetric TSP

Consider the case when G is a nondirected graph and let $\hat{\Phi}$ be a hamiltonian circuit which is the solution to the TSP. Let v be a distinguished vertex on $\hat{\Phi}$. If the two arcs incident at v together with vertex v are removed, the remaining graph is a path, P, through vertices X-{v}. Clearly, if T_v is the SST of graph $\langle X-\{v\}\rangle$, that is, the subgraph induced by the set of vertices X-{v}, then the cost of T_v is a lower bound on the cost of P.

The two arcs which have been removed from $\hat{\Phi}$ to obtain P, have a total cost at least as great as the sum of the two shortest arcs - say (i_1,v) and (i_2,v) - incident at v. Let the graph composed of the arcs in T_v and arcs (i_1,v) and (i_2,v) be called G_v. It is clear that the total cost of the arcs in G_v is a lower bound on the cost of $\hat{\Phi}$. The mathematical formulation of the problem whose solution is the graph G_v is problem P_T below.

$$
P_T \begin{cases}
\text{Min} \quad \sum_{\ell=1}^{m} c_\ell \, \xi_\ell & (11) \\
\text{s.t} \quad \sum_{\ell=1}^{m} \xi_\ell = n & (12) \\
\sum_{\ell \in C_k} \xi_\ell \geq 1 \qquad \forall C_k \equiv (X_k, \bar{X}_k),\ X_k \subset X & (13) \\
\sum_{\ell \in A_v} \xi_\ell = 2 & (14) \\
\xi_\ell \in \{0,1\} \qquad \ell = 1,\ldots,m & (15)
\end{cases}
$$

Problem P_T is clearly the same as problem T^N+ with constraints (8) replaced by constraints 10(a) and constraints 10(b) ignored.

A better bound on the value of problem T^N can be derived if constraints 10(b) are included in the objective function of P_T by means of lagrange multipliers λ, that is, by solving the problem:

$$\underset{\xi}{\text{Min}} \left(\sum_{\ell=1}^{m} c_\ell \xi_\ell \right) + \left(\sum_{i \in X} \lambda_i \sum_{\ell \in A_i} \xi_\ell - 2 \right)$$

subject to constraints (12) to (15) that is by minimising:

$$P_T(\lambda) \begin{cases} \underset{\xi}{\text{Min}} \left[\sum_{\ell=1}^{m} (c_\ell + \lambda_{i_\ell} + \lambda_{j_\ell}) - 2 \sum_{i \in X} \lambda_i \right] \\ \text{subject to constraints (12) to (15)} \end{cases}$$

where i and j are two terminal vertices of arc ℓ. One should note here that for any given λ, problem $P_T(\lambda)$ is the same as problem P_T but with transformed costs $c_\ell = c_\ell + \lambda_{i_\ell} + \lambda_{j_\ell}$. The best values of the penalties, (λ^* say), are then those which maximise the expression:

$$V(P_T(\lambda^*)) = \underset{\lambda}{\text{Max}} \; [V(P_T(\lambda))] \qquad (16)$$

where V(a) is used to denote the optimum value of the problem a.

$V(P(\lambda^*)$ is then a lower bound to $V(T^N)$. The above penalty procedure was first proposed by Held and Karp [18] and Christofides [5] for slightly different versions of the travelling salesman problem. In addition, Held and Karp gave a very effective iterative subgradient optimisation procedure for solving equation (16). They also noted that if the λ_i are set initially to $-(u_i + v_i)/2$, where u_i and v_i are the optimal dual variables of the assignment problem P_A defined by equations (1) - (3), then $V(P_T(\lambda))$ is at least as large as $V(P_A)$. Improved methods for deriving λ^* were later introduced by Hansen and Krarup [17].

+ Note: The term T^N is used here without subscript a, b or c.

A different scheme for solving equation (16), based on heuristically guided search techniques, and which proved to be computationally more successful was given by Camerimi, Fratta and Maffioli [4].

It is simple to show that in general a positive gap g exists between the value of the bound $V(P_T(\lambda^*))$ and the optimum TSP solution $V(T^N)$. However, g is often less than 1% of the value of $V(T^N)$.

Asymmetric TSP

Consider the case when G is a directed graph and let $\hat{\Phi}$ be a hamiltonian circuit which is the solution to the TSP. Again let v be a distinguished vertex on $\hat{\Phi}$. If an arc (i, v) incident at v is removed from $\hat{\Phi}$, the remaining graph is a directed path - with root at vertex v - passing through all the vertices of G. Clearly, if T_v is the directed shortest spanning tree (shortest spanning arborescence)with root at v, then the cost of T_v plus the cost of $\hat{\Phi}$ the shortest arc (i,v) - say (i_1,v) - is a lower bound on the cost of $\hat{\Phi}$.

Let the graph composed of the arcs in T_v and arc (i_1,v) be called G_v. The problem whose optimum solution is the graph G_v is:

$$P_{DT}\begin{cases} \text{Min:} & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\,\xi_{ij} \\ \text{s.t.} & \sum_{i=1}^{n} \xi_{ij} = 1 \qquad \forall j \in X \\ & \sum_{i\in X_k}\sum_{j\in\overline{X}_k} \xi_{ij} \geq 1 \qquad \forall X_k \subset X \end{cases}$$

Let TSP_a be the asymmetric TSP composed of (1), (2), (3), (5) and (4a). It is then clear that problem P_{DT} above is a relaxation of TSP_a where constraints (2) have been dropped.

Once more, a better bound than $V(P_{DT})$ on $V(TSP_a)$ can be derived by a lagrangean relaxation of constraints (2) thus producing the problem:

$$P_{DT}(\lambda) \begin{cases} \text{Min:} & \sum_{i=1}^{n} \sum_{j=1}^{n} (c_{ij} + \lambda_i)\, \xi_{ij} - \sum_{i=1}^{n} \lambda_i \\ \text{s.t.} & \text{constraints (3), (5) and 4(a)} \end{cases}$$

The value:

$$V(P_{DT}(\lambda^*)) = \max_{\lambda} [V(P_{DT}(\lambda))] \qquad (17)$$

is then a lower bound on $V(TSP_a)$.

For a given λ, $P_{DT}(\lambda)$ is essentially the problem of finding the minimum cost directed spanning tree rooted a v of the graph G with modified costs c_{ij}. This problem can be solved by a simple polynomial bounded algorithm as shown by Edmonds [13] and Fulkerson [14].

The problem of solving equation (17) is similar to the problem of solving equation (16) and one of the previously mentioned techniques can be used to derive the penalties λ^* which are the solution to (17).

(iii) Bounds from the shortest n-path

Consider a path with n arcs starting from some given vertex v of the graph G and finishing at some given vertex w. Such a path is called an n-path from v to w. It is clear that the solution of the TSP is an n-path (circuit) from v back to v, where each vertex of G appears exactly once on this path. Clearly, the value of the shortest n-path from v back to v (where a vertex can appear an arbitrary number of times on the path) is a lower bound to the TSP.

The computation of the shortest n-path from v back to v is a

simple path problem which can be solved by a dynamic programming recursion similar to that used by Bellman [2] for finding shortest paths in graphs. Thus, let $f_k(i)$ denote the shortest k-path from vertex v to vertex i.

Starting from:

$$f_1(i) = c_{1i} \qquad i \neq v,\ i \in X$$

the recursion is:

$$f_k(i) = \min_{\substack{j \neq i,v \\ j \in X}} [f_{k-1}(j) + c_{ji}] \qquad \begin{matrix} i \neq v,\ i \in X \\ k=2,\ldots,n-1 \end{matrix} \tag{18}$$

and

$$f_n(v) = \min_{\substack{j \neq v \\ j \in X}} [f_{n-1}(j) + c_{jv}]$$

The n-path corresponding to $f_n(v)$ can be found by recording for each $f_k(i)$ the value of j which produced the minimum in expression (18). Obviously, if the n-path passes through each vertex exactly once it is a solution to the TSP. If not, then $f_n(v)$ is a lower bound on the value of the TSP.

It is easy to see that if a penalty λ_i is associated with each vertex i and the costs c_{ij} are transformed by $c'_{ij} = c_{ij} + \lambda_i + \lambda_j$, then the cost of any hamiltonian circuit in G is increased by the same constant amount $2 \sum_i \lambda_i$, but n-paths that are not hamiltonian circuits are penalised differently. Let $f_n(v)$ be recomputed with the modified costs c'_{ij} and let this n-path pass through vertex i d_i times, then:

$$w(\lambda) = f_n(v) + 2 \sum_i (d_i - 1)\, \lambda_i$$

is also a valid bound to the TSP. We therefore, wish to choose that

λ^*, which corresponds to the maximum of the expression:

$$w(\lambda^*) = \max\ [w(\lambda)] \qquad (19)$$

and use $w(\lambda^*)$ as a bound for the TSP. Subgradient optimisation is one possible procedure for solving equation (19).

In the path corresponding to $f_n(v)$, the same vertex (or arc) can appear more than once or not at all. It is not possible to constrain the path so that a vertex does not appear more than once on the path, but one type of vertex repetition, namely when the r^{th} and $(r+2)^{nd}$ vertices on the path correspond to the same vertex of G - for some value of r - can be excluded by a slight modification to recursion (18). If $f'_n(v)$ is the shortest n-path from v to v with the above vertex repetition excluded, a bound $w'(\lambda^*)$ can be computed in a way similar to that for $w(\lambda)$ but with $f_n(v)$ replaced with $f'_n(v)$.

Computational results reported in [20] indicate that bound $w'(\lambda^*)$ is slightly worse than bound $V(P_T(\lambda^*))$ obtained from the SST for the symmetric case, and about the same as bound $V(P_{DT}(\lambda^*))$ obtained from the directed SST for the asymmetric case. A great disadvantage of bound $w'(\lambda)$ is that $O(n^3)$ operations are required to compute the shortest n-path of a graph (for a given root vertex v) as compared with $O(n^2)$ operations for the SST and $O(n^{2.5})$ for the AP. (Note, however, that in the last case only solution of the AP is required). On the other hand, an advantage of $w'(\lambda^*)$ is that it can be easily modified to include additional constraints which exist in problems closely related to the TSP such as the vehicle routing problem and some machine scheduling problems.

2.2 Linear Programming Based Methods

The methods described above only allow integer values of the variables to occur during the search for a solution. A number of

methods have been developed in which the variables are allowed to take fractional values [11, 15, 24, 25, 26] and are based on linear programming. These methods have mainly been used for solving problems in non-directed graphs. The number of variables and constraints increases considerably for directed graphs. The integer programming formulation on which these methods have been based is T_a^N and T_b^N.

Problem T_a^N

$$\text{minimise} \sum_{j>1}^{n} \sum_{i=1}^{n} c_{ij}\, \xi_{ij} \tag{6}$$

such that

$$\sum_{\substack{x_i \in X_1 \\ x_j \in \overline{X}_1}} \xi_{ij} = 2 \tag{7}$$

for all partitions $(X_1, \overline{X}_1)$ of the set of vertices X.

$$\sum_{\substack{x_i \in X_k \\ x_j \in \overline{X}_k}} \xi_{ij} \geq 2 \tag{8a}$$

for all partitions $(X_k, \overline{X}_k)$ of the set of vertices X, where $|X_k| = k$ and $2 \leq k \leq \left\lfloor \frac{n}{2} \right\rfloor$

$$\xi_{ij} = 0,1 \tag{5}$$

The LP relaxation of this problem has constraint (5) relaxed to:

$$0 \leq \xi_{ij} \leq 1 \tag{20}$$

Call this problem T_a^L. T_a^L is not an easy problem to solve as there are of the order of 2^{n-1} constraints (8a). Clearly in order to solve T^L the constraints (8a)must be relaxed, call this relaxed problem T_k^L, and only the 'binding constraints' (8a) included in the LP. The LP

based methods that have been developed differ mainly in the methods used to find these binding constraints. In particular at which stage of the solution process these constraints are found.

The first method using LP to solve the TSP was developed by Dantzig, Fulkerson and Johnson [11]. In this approach T_k^L is first solved and an integer solution obtained by adding extra constraints. The binding constraints (8a) are generated by inspection of the graph corresponding to the LP solution.

Miliotis [25] using an approach similar to that of Martin [24] first obtains an integer solution to T_k^L by using branch and bound. If an integer solution is obtained at any node in the search tree then the binding constraints (8a) are added and the search for a new integer solution continued. In a later paper [26] Miliotis uses Gomory cuts to obtain integrality then adding binding constraints. This method was more successful than the tree search approach.

Grotschel [15] developed a method based on T_b^N. Binding constraints (8b) are generated by plotting the LP solution on a map and observing which constraints (8b) are violated. This can only be done for Euclidean problems. Integrality is imposed by cutting planes, these are chosen to be facets of the TSP polytope. In particular, they are chosen from a class of constraints known as the comb inequalities [10, 16]. The inequalities (which are only defined on non-directed graphs) are similar to the inequalities introduced by Edmonds [12] for the linear characterisation of the matching polytope. They eliminate fractional vertices from their intersections with constraints of type (8b). The integrality constraints were also chosen visually.

3. Method D for Solving the Travelling Salesman Problem

In the last chapter branch and bound algorithm C is described for the solution to problem P^u. In this section we describe how a similar algorithm can be used to solve the travelling salesman problem.

If we compare the formulation of the non-directed TSP T_a^N with the integer programming formulation of P^2, it can be seen that they are very similar. It is possible to create problem P^2 from T_a^N by replacing the equality degree constraint in T_a^N by a greater than or equal to degree constraint. Let T_a^L (say) be the LP relaxation of T_a^N, clearly this corresponds to P_L^u for problem P^u. Thus it is possible to solve the travelling salesman problem by substituting T_a^L for P_L^u in procedure H described in the last chapter and then substitute this new procedure H' (say) into algorithm C to produce algorithm C'. Due to the relative simplicity of the TSP when compared with P^u it is possible to develop methods for improving this procedure.

3.1 Variable Reduction Techniques

There are two problems associated with the integer programming formulation of the TSP. These are as follows:

(i) The TSP contains a large number of constraints (i.e. has 2^{n-1} for a symmetric problem).

(ii) It contains a large number of variables (for a complete symmetric graph this is $\frac{n(n-1)}{2}$).

Problem (i) is overcome in algorithm C' in which it is rarely necessary to include more than n additional constraints and usually much less than n.

Problem (ii) is not overcome by algorithm C'. In this section a method for reducing the number of variables in the TSP is described.

The technique is in fact very simple and can be described as follows:

Suppose that t_ℓ is a lower bound on the value of TSP and $\bar{c}_{ij}$ are the reduced costs associated with each arc corresponding to t_ℓ. Let t_u be an upper bound on the TSP, then the set A_ℓ of arcs can be

removed from the formulation of the TSP where,

$$A_\ell = \{(x_i, x_j) \mid t_\ell + \bar{c}_{ij} > t_u\} \qquad (21)$$

In order for the above variable reduction technique to be effective it is necessary to obtain values of t_ℓ and t_u that are close to the optimal value t_{opt} (say). It has been found in practice (see section 4.1) that if these bounds differ by less than 5% then up to 85% of arcs can be removed. The three-optimal method of Lin [21] was used to obtain t_u. This is an algorithm which has been found to give excellent results often within 1% of optimal [22].

A new bound developed by Balas and Christofides was used to obtain t_ℓ and $\{\bar{c}_{ij}\}$, which is described in the Appendix.

3.2 Use of the Shortest Spanning Tree Algorithm for Variable Reduction

In the last section it was shown how, given an upper bound on the TSP and a lower bound with corresponding reduced costs, the number of variables in an integer programming formulation can be reduced a priori. It was suggested that a bound based on the solution to the assignment problem be used as a lower bound. However, it is possible to use an alternative lower bound, either in place of or as well as the assignment bound. If a different lower bound is used in addition to the assignment bound this may result in further variable reduction. In this section we describe the use of the shortest spanning tree algorithm as a lower bound.

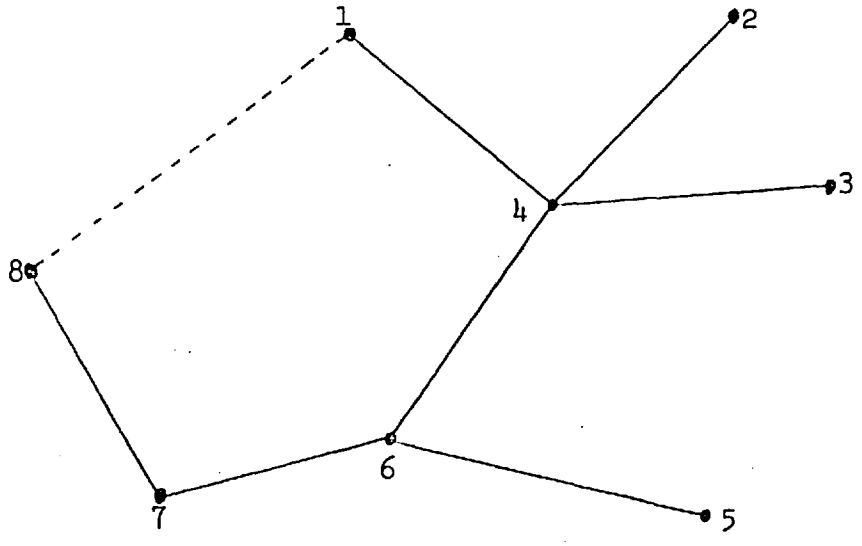
The method described in section 2.1.2, part (ii), shows how to find a good lower bound for the TSP. However, it does not provide us with any reduced costs which are an essential part of the variable reduction technique. Reduced costs for the SST bound can be calculated by considering the SST corresponding to this bound. Let A^T be the set of arcs in this SST, if $\bar{c}_{ij}$ are the reduced costs corresponding to the

lower bound, then clearly:

$$\bar{c}_{ij} = 0 \; ; \qquad \forall \; (x_i,x_j) \in A^T \qquad (22)$$

We now consider the arcs $(x_i,x_j) \notin A^T$.

Figure 1 - Illustration of how to calculate reduced costs for the SST



Consider Figure 1, the straight lines show the arcs in the SST corresponding to the lower bound. (Note that this SST has been found using the modified costs c'_{ij} where $c'_{ij} = c_{ij} + \lambda_i + \lambda_j$, λ_i being the penalty attached to vertex x_i). The dotted line shows an arc $(x_1,x_8) \notin A^T$. The addition of (x_1,x_8) to the graph in Figure 1 generates a circuit θ_{18} (say) = $(x_1,x_4,x_6,x_7,x_8,x_1)$. Let

$$\Delta_{18} = \max \{c'_{ij} \mid (x_i,x_j) \in \theta, \; (x_i,x_j) \neq (x_1,x_8)\} \; .$$

Clearly the reduced cost for (x_1,x_8) is given by,

$$\bar{c}_{18} = c'_{18} - \Delta_{18} \; .$$

In general for an arc $(x_i,x_j) \notin A^T$ the reduced cost is calculated as follows:

Let θ_{ij} be the circuit formed by adding (x_i,x_j) to the SST corresponding to the lower bound. Define

$$\Delta_{ij} = \{\max \{c'_{\ell m} \mid (x_\ell,x_m) \in \theta_{ij}, \quad (x_\ell,x_m) \neq (x_i,x_j)\}$$

Then

$$\bar{c}_{ij} = c'_{ij} - \Delta_{ij} \tag{23}$$

Expressions (22) and (23) above are used to evaluate the reduced costs from the lower bound obtained from the SST. This lower bound in conjunction with the corresponding reduced costs, can be used to remove variables from the integer programming formulation of the TSP.

3.3 Dynamic Variable Reduction Procedure

The variable reduction techniques described in 3.1 and 3.2 are used once only to remove variables from the TSP formulation. This considerably reduces the size of the problem. However, it is possible to apply a further variable reduction procedure which uses a combination of heuristics and LP reduced costs to remove variables from the problem. The procedure checks during the search for a solution whether the variables should be removed. We first describe a heuristic for removing additional variables from the problem.

(i) Heuristic for reducing the number of variables in the TSP

Let ϕ_u be the circuit corresponding to t_u (an upper bound on the TSP), and N_{max} be a specified number of variables. A heuristic (T) for choosing a set of arcs A^H to include in the integer programming formulation for the TSP is as follows:

Heuristic H (to choose initial LP core problem)

(1) Set $A^H = \phi_u$

(2) If $|A^H| \geq N_{max}$ stop. Otherwise continue.

(3) Let $(x_i,x_j)^+$ have a reduced cost $\bar{c}^+_{ij}$ (say from the lagrangean ascent on the solution to the assignment problem) such that:

$$\bar{c}^+_{ij} = \min_{\substack{(x_i,x_j)\in A \\ (x_i,x_j)\notin A^H}} (\bar{c}_{ij})$$

(4) Set $A^H = A^H \cup \{(x_i,x_j)^+\}$ and go to (2).

At the termination of H, A^H has N_{max} arcs which contain at least one feasible solution to the TSP.

(ii) Variable reduction using LP reduced costs

Let V be the value of the solution to the LP relaxation LP^H (say) of the TSP using the variables in A^H. Let $A^R = (A - A_\ell)$ (where A_ℓ is the set of arcs that have been eliminated a priori - see equation (21)). Call the reduced cost of the variables in A^R corresponding to V, $\bar{v}_{ij}$ (note: some of the variables in A^R are in A^H, in which case their reduced costs are given by the solution to LP^H. The reduced costs of the remaining variables are calculated using simple post optimality techniques). Clearly

(1) if $\bar{v}_{ij} \geq 0$, $\forall$ $(i,j) \in A^R$ then V is the value of the optimal solution to the LP relaxation of the TSP over A^R (LP^R, say). It is possible to remove all variables in A^R for which

$$V + \bar{v}_{ij} > t_u .$$

(2) If $\bar{v}_{ij} < 0$, $\forall$ $(i,j) \in A^-$ (say), the variables in A^- can be added to the LP tableau, and the LP resolved. This process can be repeated until all reduced costs are positive. However, the lower bound V obtained from the solution to LP^R can be improved by the addition of Gomory cuts. The process of adding the Gormory cuts is best combined with that of adding variables with negative reduced cost. This is done in the following procedure K, which dynamically removes variables from A^R.

Procedure K (to reduce variables in the TSP formulation)

Initialisation

1.0 Let $LP^C = LP^H$, $A^C = A^R$, KOUNT = 0

Solve current LP

2.0 Solve LP^C. Let V and $\bar{v}_{ij}$ be the value of the optimal solution and the reduced costs of the variables in A^C (corresponding to V) respectively, and let A^- be the set of variables in A^C such that:

$$\bar{v}_{ij} < 0$$

2.1 If the solution to LP^C is all integer set $t_u = \min(V, t_u)$ and go to 4.0; else continue.

LP solution is fractional

3.0 If $A^- \neq \phi$ go to 3.4

LP solution is optimal

3.1 Let A^O be the set of variables in A^C such that:

3.2

$$V + \bar{v}_{ij} > t_u$$

3.2 Set $A^C = A^C - A^O$, and remove all columns from LP^H which represent variables in A^O.

3.3 If $(|A^O| / |A^C|) \times 100\% < \varepsilon\%$ (a present percentage), set KOUNT=KOUNT+1. If KOUNT $\geq$ M (a present integer) stop.

Add Gomory Cut

3.4 Add a Gomory cut (derived from LP^C) and all the variables in A^- into LP^C. Call the new problem LP^C, and go to 2.0.

LP solution to LP^C is integer optimal

4.0 If $A^- = \phi$ stop (optimal solution has been found).

4.1 Add all variables in A^- to LP^C. Call the new problem LP^C, and

go to 2.0.

Two possible conditions occur at the termination of procedure K, these are:

(1) The optimal solution to the TSP has been found (at step 4.0).

(2) The last Gomory cut is the M^{th} one that caused less than $\in$% of the variables in A^C to be removed from the problem (at step 3.3).

In case 1 the process of finding a solution to the TSP is complete. In case 2 the variables in A^C are added into the LP and a branch and bound used to solve the problem as described in the next section.

3.4 Algorithm D

In the preceding sections 3 procedures have been described for removing variables a priori from the fomulation of TSP. In the first two procedures an upper bound t_u and a lower bound t_ℓ (with associated reduced costs) are calculated. Variables with reduced costs in excess of the difference between t_u and t_ℓ are eliminated. In the third procedure (K) a subproblem to the complete TSP is solved. The solution to this problem is used to further reduce the number of variables and also to improve the upper bound t_u. In some cases the problem is solved by procedure K. We now describe algorithm D as follows:

Algorithm D:

1. Generate a lower bound t_ℓ with associated reduced costs for the TSP over G.
2. Generate an upper bound t_u for the TSP over G.
3. Eliminate all variables from the problem with reduced cost greater than $t_u - t_\ell$.
4. Use heuristic T to form a reduced problem, and use procedure K on this problem in order to further remove variables.
5. If procedure K produces the optimal solution stop, otherwise

formulate the reduced LP relaxation of the TSP and call the problem T^R.

6. Insert T^R in place of P_L^U in procedure L (Chapter 5) and use algorithm C (with u=2) to solve the TSP.

4. Computational Results

In the following sections we describe tests performed on algorithm D and also on the bounding procedure of Balas and Christofides to find how successful it is in eliminating variables from the TSP. All programs in this section were written in FORTRAN and tested on the CDC 7600 computer. The results show that algorithm D is an effective algorithm for solving the TSP. The computational results for this algorithm show that it is competitive with other methods for the symmetric TSP.

4.1 The Bound on the TSP

In this section we describe tests performed on the bounding procedure of Balas and Chrstofides. The bound was tested on a large number of cost matrices with non-symmetric random, symmetric random and euclidean costs. For each matrix the lower bound B_{AP} and the reduced costs $\{\bar{c}_{ij}\}$ were calculated. The percentage of variables that are removed if an upper bound ℓ_u is Y percent greater than B_{AP} were then calculated for Y=0%, 1%, 2%, 3%, 4%, 5%. 10% and 20%. This was done a number of times for between 10 and 100 vertices, and an average taken for each number of vertices n. A graph of the percentage of variables remaining - (R say) - against n was plotted for each value of Y.

The results of these tests for non-symmetric random, symmetric random and euclidean cost matrices are shown in Figures 2(a), 2(b) and 2(c) respectively.

The random cost matrices were generated by sampling from a uniform distribution between 1 and 100. The euclidean cost matrices were generated by randomly placing points on a 100 by 100 square.

From the results it can be seen that for random problems even if $t_\ell (=B_{AP})$ and t_u differ by as much as 20% then over 50% of the variables are eliminated from the TSP. For differences of less than 5% (which would normally be expected) more than 85% of the variables are eliminated. For euclidean costs less variables are eliminated. However, for a difference of less than 5% (this is normally the type of figure that has been encountered in practice) over 50% of the variables are eliminated.

4.2 Tests on Algorithm D

Algorithm D has been tested on both symmetric random and euclidean complete cost matrices. The costs for these matrices have been generated as described above. The table of results for algorithm D is given in Tables 1(a) and 1(b). Table 1(a) is for symmetric random cost matrices, Table 1(b) is for euclidean cost matrices.

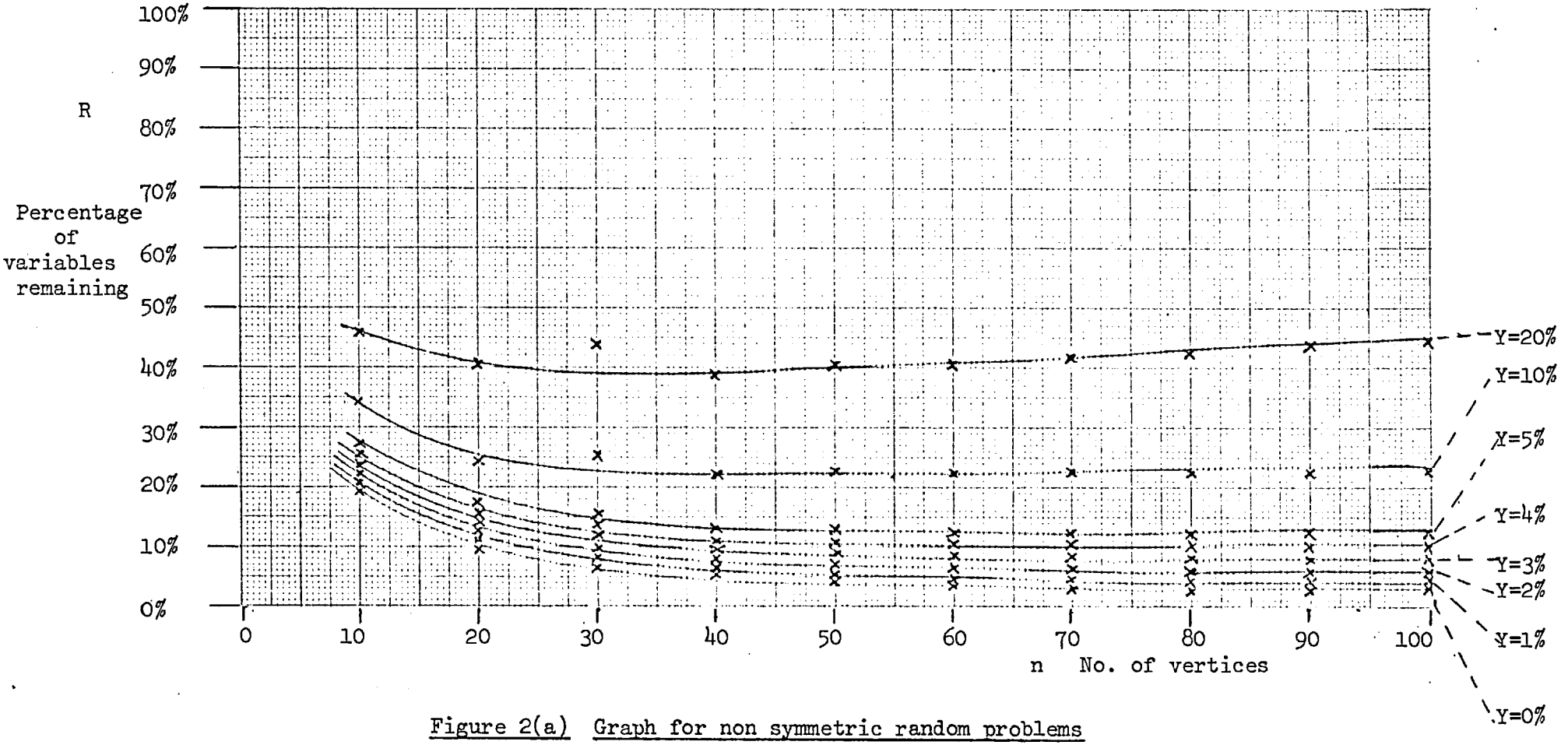


Figure 2(a) Graph for non symmetric random problems

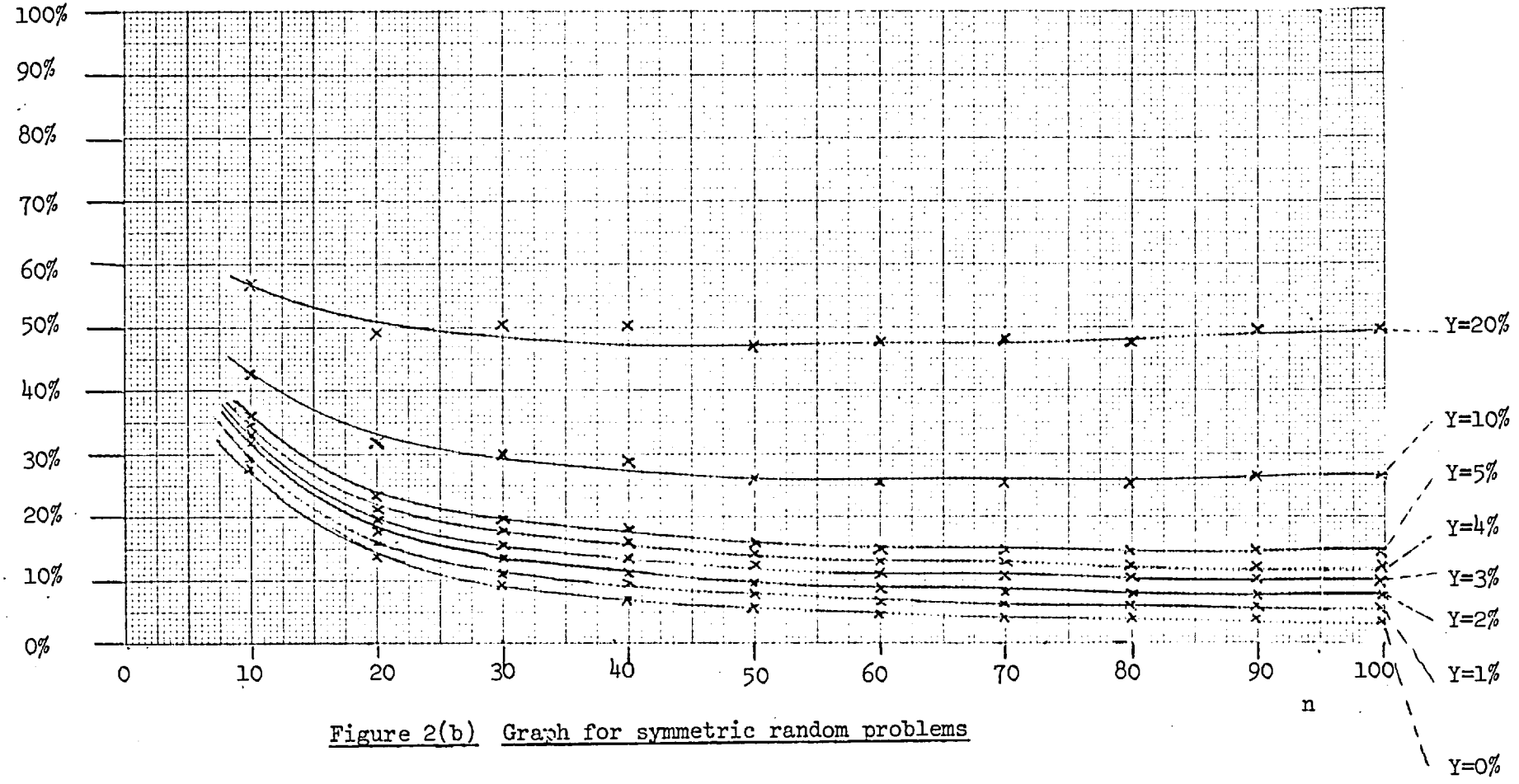


Figure 2(b) Graph for symmetric random problems

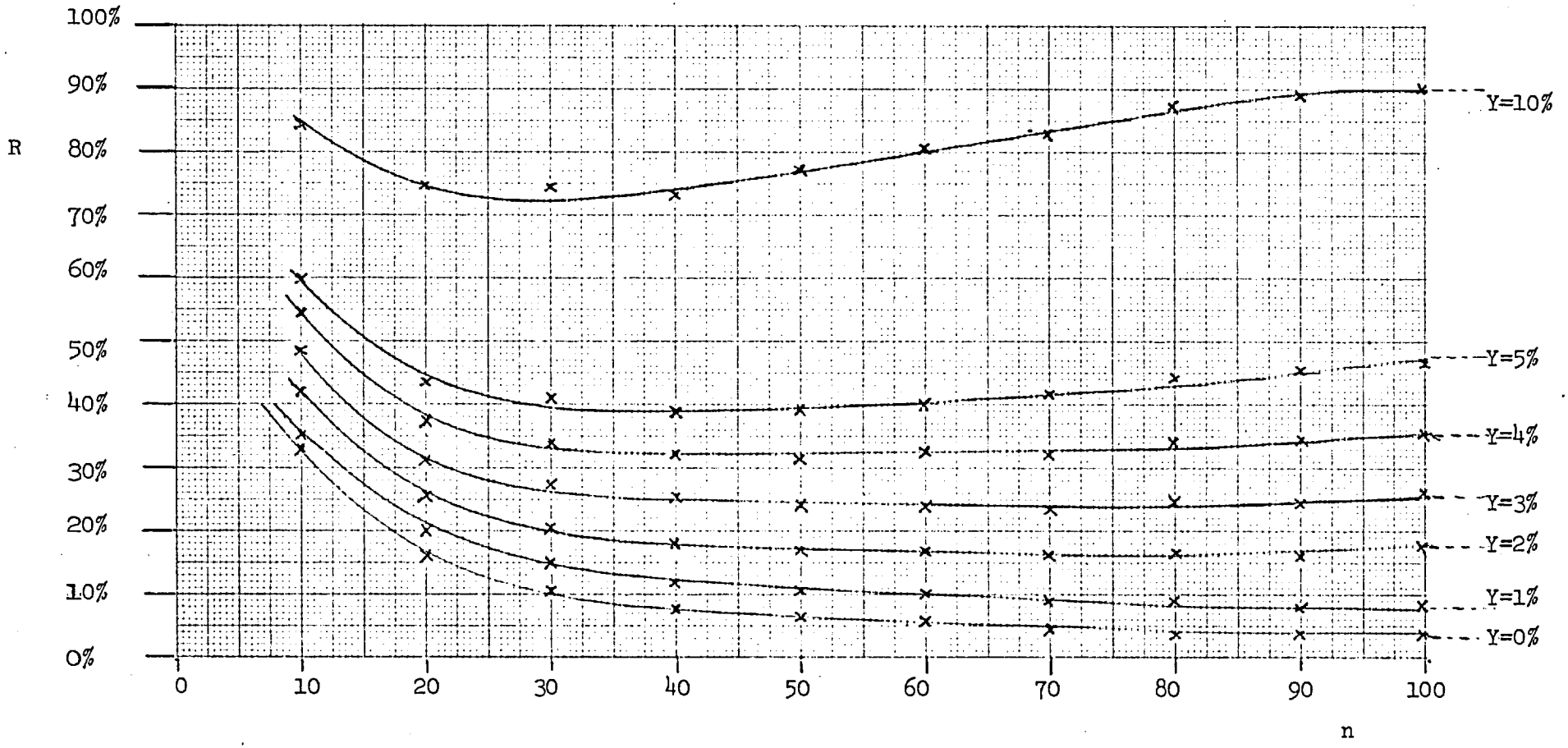


Figure 2(c) Graph for euclidean problems

No. of vertices	No. of problems solved	Average no. of nodes in tree	Average time (7600 sec)	Max. time	Min. time
10	25	1	0.15	0.16	0.07
20	18	2.7	0.30	0.47	0.10
30	22	3.2	1.03	3.77	.23
40	9	5.4	3.45	6.13	1.90
50	14	8	10.57	19.22	1.97
60	5	10	30.12	60.34	5.41
70	2	15	84.89	135.98	33.80

Table 1(a) - Performance of Algorithm D on Random Symmetric Post Matrices

No. of vertices	No. of problems solved	Average no. of nodes in tree	Average time (7600 sec)	Max. time	Min. time
10	24	1.2	0.16	0.25	0.08
20	20	3	0.20	.39	.12
30	14	4.5	1.16	3.14	.91
40	17	10	5.93	24.95	1.86
50	9	15	13.07	23.59	3.12
60	4	19	52.93	91.85	6.32
70	3	23	105.88	200.10	25.14

Table 1(b) - Performance of Algorithm D on Euclidean Cost Matrices

Appendix

Assignment Based Lower Bound to the TSP

Consider the assignment problem P_A defined by expressions (1) to (3) and (5). This is a relaxation of problem TSP_a where constraints 4(a) or 4(b) or 4(c) have been dropped. Balas and Christofides [1] considered the introduction in a lagrangean fashion of some of the violated constraints from 4(a) and some others from 4(c). Let λ_t be the multiplier associated with the t^{th} constraint in 4(a) which is not satisfied. The problem then becomes:

$$P_A(\lambda) \begin{cases} \text{Min: } \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, \xi_{ij} - \sum_t \lambda_t \sum_{i \in X^t} \sum_{j \in \bar{X}^t} \xi_{ij} + \sum_t \lambda_t \\ \text{s.t. constraints (2), (3) and (5)} \end{cases}$$

In contrast to the previously introduced relaxations $P_T(\lambda)$ and $P_{DT}(\lambda)$ used for the SST bound, the number of multipliers λ_t in the objective function of $P_A(\lambda)$ can increase exponentially, rather than linearly, with n. Thus, if h is the number of circuits formed by the AP solution, then there are $2^{h-1}-1$ possible sets X^t for which constraint 4(a) could be violated; once for cut $(X^t, \bar{X}^t)$ and once for cut $(\bar{X}^t, X^t)$, thus producing a maximum total of 2^h-2 violated constraints of type 4(a). Since h could be as high as $\frac{n}{2}$, the number of violated constraints is exponential in n. Balas and Christofides [1] considered a sequential procedure for choosing the constraints to enter into the objective function by means of a lagrange multiplier, and gave an approximate non-iterative method for determining these multipliers. The procedure is as follows:

Let $\left[\bar{c}_{ij}\right]$ be the reduced cost matrix after the solution to the AP. Form the graph $G_0 = (X, A_0)$ where $A_0 = \{(i,j) \mid \bar{c}_{ij} = 0\}$. Choose any vertex $i \in X$ and form the set of vertices R(i) which can be reached from i via arcs of G_0. If R(i) = X choose another vertex from X and repeat. If R(i) = X $\forall i \in X$ stop, no more cuts will be

generated. If $R(i) \neq X$ then a cut $K_t = (X^t, \bar{X}^t)$ with $X^t = R(i)$ is generated which violates 4(a).

Once K_t is identified, an initial value of the corresponding multiplier λ_t is computed as:

$$\lambda_t^0 = \min_{(i,j) \in K_t} \left[\bar{c}_{ij}\right] \qquad (24)$$

and $\left[\bar{c}_{ij}\right]$ is updated by:

$$\bar{c}_{ij} = \bar{c}_{ij} - \lambda_t^0 \qquad \forall (i,j) \in K_t$$

$\bar{c}_{ij}$ unchanged otherwise.

Update G_0 to include the arcs for which $\bar{c}_{ij}$ has just become 0 and repeat until no more cuts can be generated. The number of cuts generated by this procedure is limited to be at the most 2(h-1).

At this stage it is no longer possible to increase $V(P_A(\lambda))$ by modifying the λ's and keep the modified costs $\bar{c}_{ij}$ non-negative. However, the AP solution for problem P_A is still optimal for $P_A(\lambda)$ and it is possible to identify constraints of the form 4(c) which are violated by this solution. Let the solution contain circuits Φ_p where Φ_p is the set of arcs of forming that circuit. The lagrangean relaxation of the corresponding constraints 4(c) then leads to the problem:

$$P_A(\lambda,\mu) \begin{cases} \text{Min: } \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\xi_{ij} - \sum_t \lambda_t \sum_{i \in X^t} \sum_{j \in \bar{X}^t} \xi_{ij} \\ \\ + \sum_p \mu_p \sum_{(i,j) \in \Phi_p} \xi_{ij} - \sum_p \mu_p |\Phi_p| + \sum_t \lambda_t + \sum_p \mu_p \\ \\ \text{s.t. constraints (2), (3) and (4) .} \end{cases}$$

If the λ's are assumed fixed at λ_t^0 as mentioned earlier, the problem of maximising $V(P(\lambda,\mu))$ over all μ can be considered in a

way similar to that used above to determine the λ, that is by setting μ_p to a value μ_p^0 which is the largest value of μ_p that will leave the AP solution to problem $P(\lambda,\mu)$ unchanged. This value μ_p^0 is simple to derive and can be computed in much the same way as the duals are computed for the AP.

With μ_p^0 determined, new dual variables u_i and v_j are computed for problem $P_A(\lambda^0,\mu_p^0)$ which can then be used to update the cost $\bar{c}_{ij}$ corresponding to problem $P_A(\lambda^0)$ by $\bar{c}_{ij} = \bar{c}_{ij} - u_i - v_j$.

Another circuit Φ_p in the AP solution is now identified, and another value of μ_p^0 determined. New dual variables u_i and v_j are computed for problem $P_A(\lambda^0, \lfloor \mu_{p_1}^0, \mu_{p_2}^0 \rfloor)$, where p_1 and p_2 are the indices of the first and second circuits Φ_p considered, and $\bar{c}_{ij}$ again updated, etc. until all circuits Φ_p in the AP solution have been considered.

At the end of this procedure, it is clear that, since the initial AP solution is still optimal, the quantity:

$$V(P_A) + \sum_t \lambda_t^0 + \sum_p \mu_p^0$$

is a lower bound to the TSP and has been found to be a reasonable approximation to:

$$\max_{\lambda,\mu} \lfloor V(P_A(\lambda,\mu)) \rfloor$$

Let B be an optimal solution to the assignment problem $P_A(\lambda^0,\mu^0)$. We know that the initial AP solution is optimal, but there may also be other alternative solutions. If B is a hamiltonian circuit, and if it satisfies the constraints for which $\lambda \neq 0$ and $\mu \neq 0$ as equalities, then B is an optimal solution to the TSP. If B is a hamiltonian circuit but some constraints with $\lambda,\mu \neq 0$ are not satisfied as equalities, then it may still be possible to modify the λ and μ and derive a better upper bound (and a better solution). Some techniques for modifying

λ and μ in such circumstances are given in [1].

If B is not a hamiltonian circuit, let us again define the graph $G_0 = (X,A_0)$, $A_0 = \{(i,j) \mid \bar{c}_{ij} = 0\}$, where $\bar{c}_{ij}$ are the reduced costs of problem $P_A(\lambda^0,\mu^0)$. Figure 3 shows an example of a possible G_0.

Let $\hat{\Phi}$ be the optimum solution to the TSP. $\hat{\Phi}$ defines a graph $G(\hat{\Phi})$ consisting of a single hamiltonian circuit. If a vertex v (say) is removed from $G(\hat{\Phi})$, the resulting graph $G_v(\hat{\Phi})$ is a hamiltonian path through the remaining (n-1) vertices and is, therefore, a unilaterally connected graph; that is for any two vertices i and j of $G_v(\hat{\Phi})$ there exists a path either from i to j or from j to i.

Let us now consider the graph G_0 with vertex v removed thus producing graph $G_0^v = (X^v,A_0^v)$. If G_0^v is not unilaterally connected, there must exist a pair of cuts K_v', K_v'' of G_0^v for which $A_0^v \cap K_v' = A_0^v \cap K_v'' = \phi$.

If we now define:

$$\pi_v = \min_{(i,j) \in K_v' \cup K_v''} [\bar{c}_{ij}]$$

and transform:

$$\bar{c}_{ij} = \bar{c}_{ij} - \pi_v \qquad \forall (i,j) \in K_v' \cup K_v''$$

$$\bar{c}_{ij} \text{ unchanged otherwise,}$$

a new graph G_0 can be defined which can again be tested for unilateral connectedness (after removing some other vertex v), and so on until G_0^v remains unilaterally connected after the removal of any v.

Let $W = \{v_1,v_2,\ldots,v_k\}$ be the set of vertices whose removal has led to G_0^v not being unilaterally connected during this procedure. It is then clear that the quantity:

$$B_{AP} = V(P_A) + \sum_t \lambda_t^0 + \sum_p \mu_p^0 + \sum_{v \in W} \pi_v$$

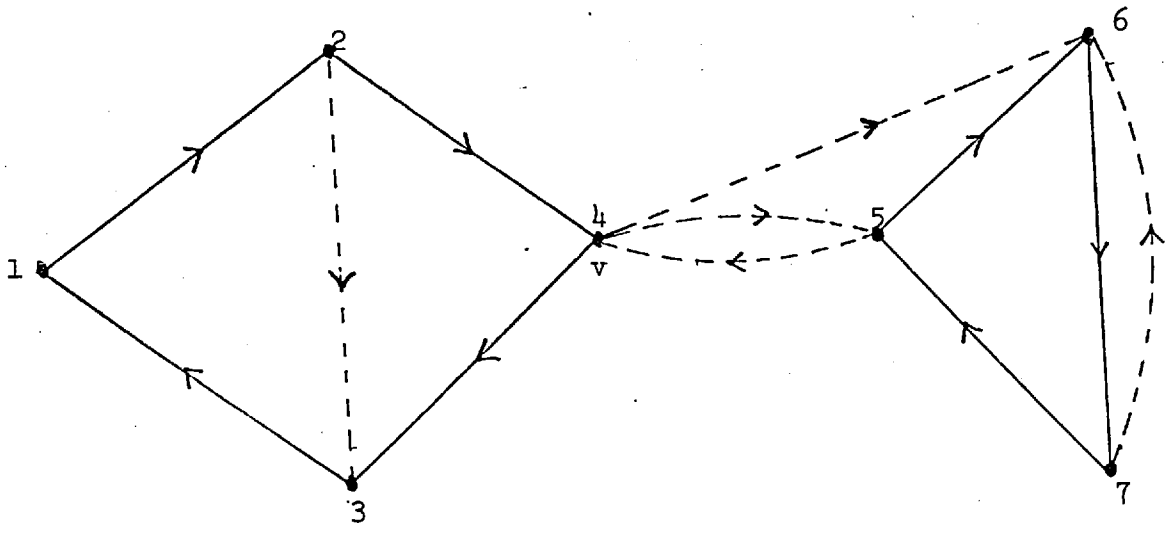


Figure 3(a) The 0-graph G_o

arcs in solution

other arcs

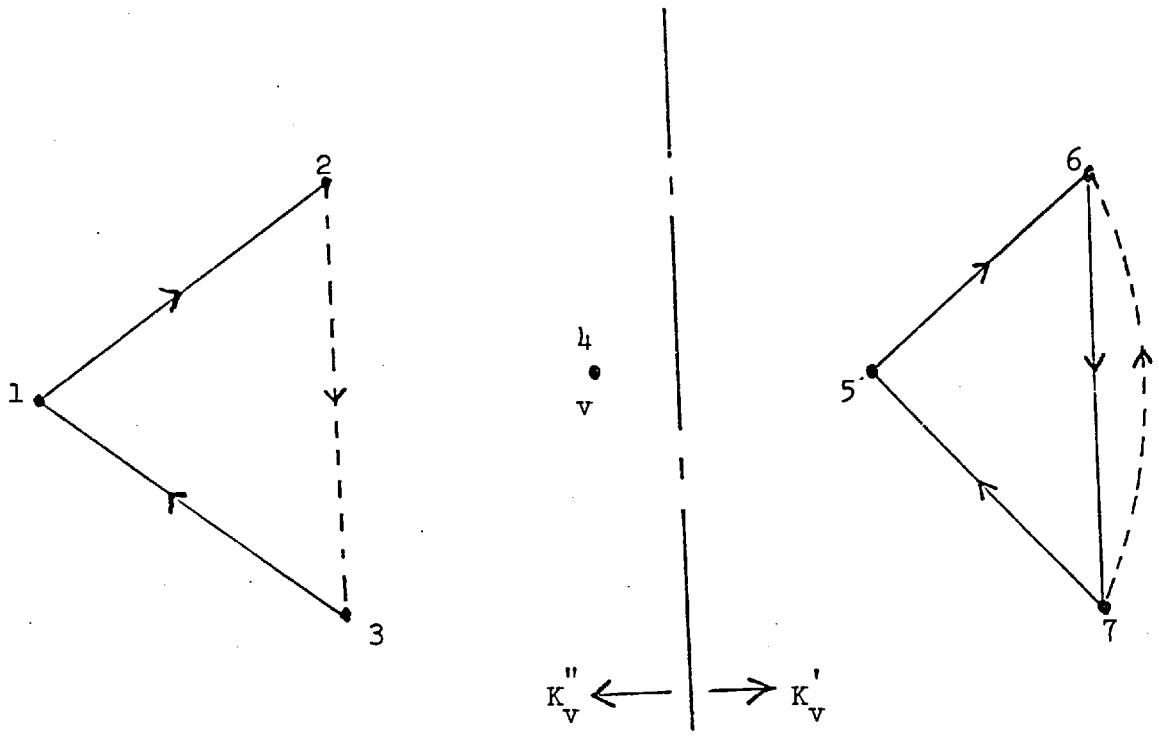


Figure 3(b) The graph G_o^v

is a lower bound to the TSP.

Some computational comments are in order here.*

The procedure can be applied to both symmetric and asymmetric problems but produces much better bounds for the latter type of problem.

For symmetric problems, the value of the assignment solution $V(P_A)$ is, on average, about 20% below $V(TSP_s)$, whereas B_{AP} is, on average, only 4% below $V(TSP_s)$.

---

* The computational results mentioned here depend, to some extent, on the way that the problems are generated, but the relative magnitudes of the bounds the conclusions drawn are, in general, correct.

REFERENCES

1. E. Balas and N. Christofides, "A new penalty method for the travelling salesman problem", presented at the 9th Math. Prog. Symposium, Budapest, 1976.

2. R. Bellman, "On a routing problem", Quart. J. Appl. Math., p.87, (1958).

3. M. Bellmore and J.C. Malone, "Pathology of travelling salesman subtour elimination algorithms", Op. Res., p. 278 (1971).

4. P. Camerini, L. Fratta and F. Maffioli, "Travelling salesman problem : Heuristically guided search and modified gradient techniques". Report of Instituto di Elettronica, Politecnico di Milano, (1975).

5. N. Christofides, "The shortest hamiltonian chain of a graph", J. SIAM (Applied Mathematics), p. 689, (1970).

6. N. Christofides, "Bounds for the travelling salesman problem", Ops. Res., p. 1044, (1972).

7. N. Christofides, Graph Theory - An Algorithmic Approach, Academic Press, London, (1975).

8. N. Christofides and S. Eilon, "Algorithms for large scale TSPs", Opl. Res. Quart., p. 511 (1972).

9. N. Christofides, "Worst case analysis of a new heuristic for the travelling sleasman problem", Math. Prog. (to appear).

10. V. Chvátal, "Edmonds polytopes and weakly hamiltonian graphs", Math. Prog., p. 29 (1973).

11. G. Dantzig, D.R. Fulkerson and S. Johnson, "Solution of a large scale travelling salesman problem", Ops. Res., p. 393 (1954).

12. J. Edmonds, "Maximum matchings and a polyhedron with (0,1) vertices", Journal of Research, National Bureau of Standards, p. 125 (1965).

13. J. Edmonds, "Optimum branchings", J. of Res., Nat. Bureau of Stand., p. 233 (1967).

14. D.R. Fulkerson, "Packing routed directed cuts in a weighted directed graph, Math. Prog., p. 1 (1974).

15. M. Grötschel, "An optimal tour through 120 cities in Germany", Report 7770, University of Bonn, (1977).

16. M. Grötschel and M.W. Padberg, "Linear characterisation of the symmetric travelling salesman polytope", Report 7417, University of Bonn, (1974).

17. K.H. Hansen and J. Krarup, "Improvements of the Held-Karp algorithm for the symmetric travelling salesman problem", Math. Prog., p. 87 (1974).

18. M. Held and R. Karp, "The travelling salesman problem and minimum spanning trees", Ops. Res., p. 1138 (1970).

19. M. Held and R. Karp, "The travelling salesman problem and minimum spanning trees II", Math. Prog., p. 6, (1971).

20. D. Houck, J.-C. Picard, M. Queyranne and R.R. Vemuganti, "The travelling salesman problem and shortest n-paths", University of Maryland (1977).

21. S. Lin, "Computer solution of the TSP", Bell System Tech. J., p. 2245 (1965).

22. S. Lin and B.W. Kernighan, "An effective heuristic algorithm for the travelling salesman problem", Ops. Res., p. 498, (1973).

23. J.D. Little, K.G. Murty, D.W. Sweeney and C. Karel, "An algorithm for the travelling salesman problem", Ops. Res., pp. 972-989, 11, (1963).

24. G.T. Martin, "Solving the travelling salesman problem by integer linear programming", CIER, New York (1966).

25. P. Miliotis, "Integer programming approaches to the travelling salesman problem", Math. Prog., p. 367 (1976).

26. P. Miliotis, "Using cutting planes to solve the symmetric travelling salesman problem", Math. Prog., pp. 177-188, 5, (1978).

27. M.W. Padberg and S. Hong, "On the symmetric travelling salesman problem : A computational study", presented at Symp. on Discrete Opt., Banff, (1977).

28. D. Shapiro, "Algorithms for the solution of optimal cost travelling salesman problem", Sc.D. Thesis, Washington University, St. Louis (1966).

CHAPTER 7

CONCLUSIONS

A number of tree search algorithms have been developed in this thesis. These are as follows:

- Algorithm for solving the constrainted two dimensional stock cutting problem.
- Algorithms A, B and C for the solution of a variety of network synthesis problems.
- Algorithm D for the solution of the travelling salesman problem.

We summarise the results obtained for these algorithms in the following.

1. Constrained Two Dimensional Stock Cutting Problem

We have shown that constrained two dimensional stock cutting problems of practical size can be solved. Problems with up to 20 different types of pieces to be cut have been solved. These problems are complex as is shown by the examples given in Chapter 3. It is not possible to give an upper bound on the size of stock rectangle that can be cut using this algorithm, since this is a function of the sizes of the pieces to be cut and their dimensions. The use of normalised cuts also aides the solution of problems with large stock rectangles.

2. Algorithm A for the Solution of Network Synthesis Problem S^u

A 1-matching code has been written for the purpose of developing algorithm A. This code has been tested in conjunction with the graph transformation suggested by Edmonds to solve the u-matching problem.

Bounds to the network synthesis problem S^u are derived from lagrangean relaxations using the u-matching code above together with a subgradient optimisation procedure for maximising the lagrangean function.

This bound has been embedded into algorithm A and problems of type S^u with up to 20 vertices have been solved for u=4. The 4-

matching has been shown to be an excellent lower bound on the solution to this problem.

3. Algorithm B for the Solution of Network Synthesis Problem Q^u

Algorithm B has been tested on a large number of network synthesis problems of type Q^u for u=4. Problems with up to 20 vertices have been solved. In order to develop algorithm B the problem of finding the largest number of disjoint hamiltonian circuits of a graph has been investigated. Two procedures for solving this problem have been described. One has been embedded effectively in algorithm B.

4. Algorithm C for the Solution to Network Synthesis Problem P^u

Problem P^u has been formulated as an integer programming problem for the development of algorithm C. The formulation contains a very large number of constraints which make the problem difficult to solve. It has been shown that by the use of network flow methods most of these constraints can be removed. Problems of up to 50 vertices have been solved.

5. Algorithm D for the Solution of the Travelling Salesman Problem

The integer programming formulation of the TSP is similar in structure to that for problem P^u. A network flow technique was also adopted in algorithm D to remove constraints. In addition - because the TSP is well structured - techniques were developed to remove large numbers of variables from the formualtion. Problems up to 70 vertices have been solved.