

UNIVERSITY OF LONDON  
IMPERIAL COLLEGE OF SCIENCE AND TECHNOLOGY  
DEPARTMENT OF MANAGEMENT SCIENCE

ALGORITHMS FOR THE SET COVERING PROBLEM

by

ANGELA MARGARET HEY MA, MMATH, FRGS

A thesis submitted for the  
Degree of Doctor of Philosophy  
and  
Diploma of Imperial College

JULY 1980

## ABSTRACT

Solution methods for the set covering problem, SCP, are the subject of this thesis. This problem is widely encountered, notably in operational research, computer science and electrical engineering. A survey of applications and algorithms is given in the first chapter. Heuristic algorithms that obtain upper and lower bounds on the optimal solution value are given in Chapter 2. The SCP can be formulated as an integer program and one of the more successful approaches to this type of problem is Lagrangean relaxation embedded in a branch and bound (tree search) strategy. Chapter 3 illustrates techniques for efficiently increasing lower bounds obtained from Lagrangean relaxations. Lower bounds to the SCP are derived using network flow and graph theory in Chapters 4 and 5. Chapter 6 discusses decomposition and state space relaxations for obtaining lower bounds to the SCP. Branching strategies are considered in Chapter 7. The implementation of an algorithm for the SCP using the graph covering relaxations is given in Chapter 8. Conclusions, together with ideas for future research, are given in the final chapter.

ACKNOWLEDGEMENTS

Firstly I should like to thank my supervisor Dr N. Christofides for many stimulating discussions, encouragement and support throughout the project. I am grateful to Dr T. Coleman of Argonne National Laboratory, Illinois for helpful comments on Chapter 3. Dr U. Derigs and Mr G. Kazakidis of the University of Cologne not only supplied me with their matching code but allowed me full use of their facilities for one week. I am indebted to them for their assistance and for explaining their programs to me. I should like to thank Prof. E. Balas of Carnegie-Mellon University, Mr C. Krabek of Control Data and Dr S. Korman (formerly of Imperial College) for supplying test problems.

Above all many thanks must go to Miss K. Shrimanker for the excellent typing of this thesis. I am also grateful to Miss Suzy Brown for typing the contents, index and some tables.

The financial support of the Science Research Council and the Department of Management Science is gratefully acknowledged. Appendix 5 is included by kind permission of Bell Laboratories, New Jersey. Finally I should like to thank the many others with whom I have had discussions.

<u>CONTENTS</u>	Page <sup>(iv)</sup>
Title page	(i)
Abstract	(ii)
Acknowledgements	(iii)
Contents	(iv)
Figures, Diagrams and Tables	(xiii)
Index to Procedures	(xvi)
Notation	(xviii)

CHAPTER 1                      INTRODUCTION AND LITERATURE SURVEY

1.1	Definitions	1
1.2	Relaxations	2
1.3	Literature Survey	6
1.3.1.	Applications	7
1.3.2.	Problems related to the SCP and problems used in its solution	8
1.3.2.1.	Well-solved cases of the SCP	8
1.3.2.2.	Problems of which the SCP is a special case	13
1.3.2.3.	Relationship between the SPP and the SCP	16
1.3.2.4.	Network Flow Problems	17
1.3.3.	Algorithms for the SCP	18
1.3.3.1.	Preliminary reductions	19
1.3.3.2.	Sorting the constraint matrix	21
1.3.3.3.	Algorithms for lower bounds to the SCP	22
1.3.3.4.	Algorithms for upper bounds to the SCP	23
1.3.3.5.	Branching strategies for the SCP	23
1.3.3.6.	Cutting plane strategies for the SCP	25

1.3.4. Theoretical Results for the SCP	26
1.3.4.1. Complexity results	26
1.3.4.2. The set covering polyhedron	27
1.3.4.3. The structure of the constraint matrix	28
1.3.4.4. Duality	29
1.3.5. Data structures	30

CHAPTER 2HEURISTICS

2.1. Introduction	31
2.2. Outline of the Heuristic Methods	32
2.3. Additional Methods for Computing Upper Bounds	40
2.4. Reasons for Failure of the Heuristics to Solve the SCP	41
2.5. Computational Results	42

CHAPTER 3LAGRANGEAN RELAXATION

3.1. Introduction	50
3.2. Implementation of Lagrangean Relaxation	52
3.3. Calculating the Search Direction, $\lambda$	55
3.4. Calculating the Stepsize, $\sigma$	61
3.4.1. Introduction	61
3.4.2. Computing $\sigma^k$ Using a Target Value	62
3.4.3. Computing $\sigma^k$ Using "Near Alternative" Solutions	63
3.4.4. Other Methods of Computing $\sigma^k$	64
3.5. Computational Results	66
3.5.1. Case Study	66
3.5.2. Comparison of the methods on different problems	75

<u>CHAPTER 4</u>	<u>NETWORK FLOW RELAXATIONS OF THE SCP</u>	(vi) Page
4.1.	Introduction	78
4.2.	Network Flow Relaxation, NF1	78
4.1.1.	Formulation	78
4.2.2.	Changing the costs $d_{ij}$ on the Network G1	81
4.2.3.	Further improvements	83
4.2.4.	Summary of the Algorithm	85
4.3.	Network Flow Relaxation, NF2	89
4.3.1.	Construction of the Network, G2, from SCP	89
4.3.2.	Formulation of the problem and calculation of costs	90
4.4.	Computational Results	93
4.4.1.	Explanation of Table of Results	93
4.4.2.	Implementation of the Algorithm	94
<u>CHAPTER 5</u>	<u>GRAPH COVERING RELAXATIONS OF THE SCP</u>	
5.1.	Introduction	96
5.2.	The Graph Covering Problem, GCP	96
5.3.	Graph Covering Relaxation 1, GCR1, a row relaxation of the SCP	98
5.3.1.	Description of the relaxation	98
5.3.2.	Quality of the bound	99
5.3.3.	Calculation of $\lambda^*$	99
5.3.4.	Partitioning the constraints	100
5.3.5.	Changing the partition of $A'$	102
5.4.	Graph Covering Relaxation 2, GCR2, a column relaxation	104
5.4.1.	Description of the relaxation	104

	(vii) Page
5.4.2. Quality of the Bound	107
5.4.3. Calculating the Costs	107
5.5. Further Improvements to the Graph Covering Relaxations	108
5.5.1. Ensuring the costs of the relaxed problem are non-negative in GCR1.	108
5.5.2. Ensuring the costs of the relaxed problem are non-negative in GCR2.	110
5.5.3. Changing costs of arcs in a GCP to retain the same optimal solutions	111
5.5.4. Using the graph covering solution in consecutive iterations of the subgradient optimization procedure	113
5.6. Combining the two relaxations GCR1 and GCR2	114
5.7. Computational Results	122
5.7.1. Case study	122
5.7.2. Comparison between GCR1, GCR2 and a Combination of the Two Relaxations	123
5.7.3. Comparison between the Graph Covering Relaxation, Heuristics and Linear Programming	124
5.7.3.1. Korman's Problems	124
5.7.3.2. Four Problems of Salkin and Koncal	128
5.7.3.3. Results for Graph Covering Problems	131
5.7.4. Using the Heuristic, Graph Covering and LP Bounds in a Tree Search	133
5.7.5. Conclusions	133

CHAPTER 6      LOWER BOUNDS TO THE SCP USING DECOMPOSITION  
AND STATE SPACE RELAXATION

	Page
6.1. Introduction	134
6.2. The Decomposition Method for obtaining a Lower Bound to the SCP	134
6.2.1. Definition	134
6.2.2. Calculating the costs initially	136
6.2.3. Updating the costs	136
6.2.4. Using integer costs $d_t$	137
6.2.5. Reduced costs for the SCP	138
6.2.6. Recursive tree search	139
6.2.7. Sorting the constraint matrix initially	140
6.2.8. Description of the decomposition algorithm	141
6.3. A lower bound to the SCP from State Space Relaxation	143
6.3.1. Definition	143
6.3.2. State Space Relaxation 1, SSR1	143
6.3.2.1. Definition	143
6.3.2.2. Reduced Costs	144
6.3.2.3. Improving the bound using subadditivity	144
6.3.2.4. Comparison with other relaxations	146
6.3.3. State Space Relaxation 2, SSR2	146
6.3.4. Other State Space Relaxations and Extensions	146
6.4. Solving a class of SCP's	147
6.4.1. Introduction	147
6.4.2. Defining the Relaxation	148
6.4.3. Changing the costs	149



6.5. Computational Results	150
6.5.1. Case Study	150
6.5.2. Comparison between the Heuristic Bound and the Decomposition Relaxation	151
6.5.3. The State Space Relaxation, SSRI, Bound compared with the Decomposition and Heuristic Bounds	158
6.5.4. Conclusions	158

## CHAPTER 7      BRANCHING STRATEGIES FOR THE SCP

7.1. Introduction	161
7.2. Tactical Problems - choosing the branching variable	162
7.3. Branching on rows for the set covering problem	165
7.3.1. Description of Branching strategy	165
7.3.2. Branching on Rows when the Constraint Matrix is Sorted into Blocks	166
7.3.2.1. Sorting the Matrix	166
7.3.2.2. The Branching Strategy	166
7.3.2.3. Removal of Rows	167
7.3.2.4. Removal of Columns or Blocks	167
7.3.2.5. Removing Coefficients in Constraints	167
7.3.2.6. Solving the problem after assigning blocks to all rows	168
7.3.2.7. Example	168
7.3.3. Branching on Rows when the Constraint Matrix is stored as a list of Non - Zero rows by Column	170
7.3.3.1. Description of the forward step of the branching strategy	170
7.3.3.2. A depth first implementation	171
7.3.3.3. A best bound implementation	174
7.3.4. Improving the Branching on Rows Method	177
7.4. Computational Results for the Depth First and Best Bound Tree Search on Rows	
7.4.1. Case study	180
7.4.2. Test problems	183

<u>CHAPTER 8</u>	<u>IMPLEMENTATION OF AN ALGORITHM FOR SOLVING THE SCP USING GRAPH COVERING RELAXATIONS</u>	Page
8.1.	Introduction	137
8.2.	Design of a FORTRAN program	137
8.3.	Data Structures for the Graph Covering Algorithms	139
8.3.1.	The Set Covering Problem	139
8.3.2.	Lagrangian Relaxation	190
8.3.2.1.	Graph Covering Relaxation, GCR1, the Row Relaxation	190
8.3.2.2.	Graph Covering Relaxation, GCR2, the Column Relaxation	191
8.3.3.	Branching Strategies	192
8.3.3.1.	Depth First Search on rows	192
8.3.3.2.	Best Bound Search on rows	193
8.3.4.	Total Storage Required	194
8.4.	Solving the Graph Covering Problem	195
8.4.1.	The Graph Covering Algorithm	195
8.4.2.	Converting an Algorithm for Solving the Minimum Perfect Matching Problem to a Graph Covering Problem	196
8.4.2.1.	Introduction	196
8.4.2.2.	Outline of the Matching Algorithm (Edmond's Algorithm)	197
8.4.2.3.	Modifying Edmond's Algorithm	198
8.4.3.	Use of Reduced Costs to Reduce Problem Size	203
8.4.4.	Start Procedures for the Graph Covering Algorithms	205
8.5.	Parameters of the Program	206

	Page
<u>CHAPTER 9</u>	
<u>CONCLUSIONS</u>	
9.1 Summary	207
9.2 Extensions and Ideas for Future Research	208
9.2.1 Extensions of the Graph Covering Algorithm	208
9.2.2 Aggregating Constraints	209
9.2.3 Extensions to 0-1 Integer Programs	209
9.2.4 Improvement of the State Space Relaxations	210
9.2.5 Duality	210
9.2.6 Methods for Improving the Code	210
References	212
Journal Abbreviations	230
Appendices	232

<u>Appendix 1</u>	Page
Analysis of Preliminary Reduction Strategies	233
<u>Appendix 2</u>	
Index of terms used	235
<u>Appendix 3</u>	
An Example	239
A3.1 Introduction	239
A3.2 The Heuristic Lower Bounds, Chapter 2	239
A3.3 The Network Flow Lower Bound	241
A3.4 The Second Network Flow Lower Bound, v (NF2)	246
A3.5 The First Graph Covering Relaxation, GCR1, Chapter 5	248
A3.6 The Second Graph Covering Relaxation, GCR2, Chapter 5	249
A3.7 A Second Example to Illustrate a Combination of the Two Graph Covering Relaxations, Chapter 5	249
A3.8 Lower Bounds from Decomposition, Chapter 6	254
A3.9 Lower Bounds from State Space Relaxations, SSRI	255
A3.10 Branching Strategies for the SCP, Chapter 7	256
A3.11 Implementation, Chapter 8	258
<u>Appendix 4</u>	
Explanation of the Test Problems Used	260
<u>Appendix 5</u>	
Language used in the PROCEDURES	262

FIGURES, DIAGRAMS and TABLES

	Page
Fig. 1.1 An Example of a Search Tree for a Branch and Bound Procedure	5
Fig. 1.2 Graph $G_1$ in which a Shortest Path Solves SCP1	11
Table 2.1 Quality of Bounds obtained using Heuristics	44
Table 2.2 Performance of Heuristics when Incorporated in a Tree Search	46
Table 2.3 Lower Bound as a Percentage of Upper Bound	48
Table 2.4 Comparison of Heuristic Bounds with LP Bound at Root Node	48
Fig. 3.1 A Flowchart of PROCEDURE & SUBPROBLEM to solve the Lagrangean Relaxation of a problem $P$	51
Fig. 3.2 Level Sets of $L(\lambda)$ for example 3 showing Zigzagging path	57
Fig. 3.3 Non-Zero Indices of $x$ for example. To show Behaviour Near Subgradient Optimum	65
Table 3.4 To show the Effect of Varying $\beta$ when Implementing the Subgradient Ascent Procedure of Camerini et al	67
Table 3.5 Bound Values for 3 Subgradient Optimization Methods	68
Table 3.6 Best Bound Values and Tree Search Information	71
Table 3.7 Best Bound Values at Root Node for 4 Methods	72
Fig. 3.8 Comparison of Bound Values Against Iteration Number for 3 Different Subgradient Optimization Methods	73
Fig. 3.9 Comparison of Bound Values Against Computing Time for 3 Different Subgradient Optimization Methods	74
Table 3.10 Comparison of 4 Methods for Subgradient Optimization embedded in a Tree Search	76
Table 4.1 Network Flow Graph, $G_1$ , for example NF1	80
Table 4.2 Lower Bounds from the Network Flow Relaxations	92

	<u>Page</u>
Fig. 5.1 A Flowchart of PROCEDURE 13 GRAPHBOUND to Compute Lower Bounds to the SCP from Graph Covering Relaxations	115
Table 5.2 Graph Covering Lower Bounds for a 30x60 problem, density 0.15, to show variation with stepsize parameter $\delta$ , and number of 1's per column, KCOL	121
Table 5.3 Number of Graph Covering Subproblems, Tree Search Nodes and Computing Time to show Variation with Stepsize Parameter, $\delta$ , and number of 1's per column, KCOL, for 30x60 SCP	125
Table 5.4 A Comparison between the Relaxations, GCR1, GCR2 and a Combination of these two Relaxations	126
Table 5.5 A Comparison between the Heuristic, Graph Covering and Linear Programming Lower Bounds for Korman's Test Problems	127
Table 5.6 A Comparison between the Heuristic, Graph Covering and Linear Programming Lower Bounds for Test Problems of Salikin and Koncal	129
Table 5.7 Computational Results for Graph Covering Problems	130
Table 5.8 Using the Graph Covering, Heuristic and LP Bounds in a Tree Search	132
Table 6.1 Bound Values for the 30x60 example Using the Decomposition Relaxation	152
Table 6.2 Comparison of the Decomposition Bound and the Heuristic Bound	154
Table 6.3 Computing times for Decomposition Bound and the Heuristic Bound	155
Table 6.4 Decomposition Bound as Percentage of Optimal Solutions	156
Table 6.5 Decomposition Bounds for Different Partitions of $A$	157
Table 6.6 Comparison between Bounds from Decomposition, SSRI and Heuristics	

	Page	
Fig. 7.1	Amount of Tree Searched by Strategy 1	164
Fig. 7.2	Amount of Tree Searched by Strategy 2	164
Fig. 7.3	Amount of Tree Searched by Strategy 3	165
Fig. 7.4	Constraint Matrix $A$ sorted into blocks	166
Fig. 7.5	Tree Search for Example	
Fig. 7.6	The Branch and Bound Tree for the Best Bound Strategy for a 30x60 SCP of Density 0.15	181
Fig. 7.7	The Branch and Bound Tree for the Depth First Strategy for a 30x60 SCP of Density 0.15	182
Table 7.8	The Number of Nodes generated by the Depth First and Best Bound Tree Searches using Branching on Rows	184
Table 7.9	The Time taken to find the Optimal solution by the Depth First and Best Bound Searches	185
Fig. 8.1	Design of a FORTRAN computer program for the SCP	188
Fig. 8.2	An Augmenting Path	197
Fig. 8.3	Formation of a Pseudo - Vertex	197
Fig. 8.4	Case 1. Vertex $v_*$ is Matched in the Tree	200
Fig. 8.5	Case 2. Vertex $v_*$ is not Matched	200
Fig. 8.6	A Graph $G$ in which an Optimal Matching Corresponds to an optimal Cover in $G'$ .	202
Fig. A3.1	Graph for Network Flow Relaxation NF1	242
Fig. A3.2	Graph for Example, NF2	245
Fig. A3.3	Graph $G$ , The Complement of the Row Intersection Graph	247
Fig. A3.4	Graph Covering Problem for GCRI	247
Fig. A3.5	A Depth First Tree Search Branching on Rows for the SCP	257

INDEX TO PROCEDURES

The algorithms tested in this thesis are outlined in the following procedures. A description of the language used in the procedures is given in Appendix 5.

<u>Procedure number</u>	<u>Title</u>	<u>Description</u>	<u>Page</u>
1	-INITIAL BOUNDS	Computes upper and lower bounds to the SCP using heuristics	33
2	LP BOUND	Computes a lower bound from a feasible solution to DLP using an extension of Erlenkotter's method for the facility location problem	36
3	HEURISTICS	Combines Procedures 1 and 2	38
4	RELAX	Forms a Lagrangean Relaxation of an Integer Program	not explicitly described
5	SOLVELR	Solves a Lagrangean Relaxation	"
6	FEATEST	Tests whether or not a solution to a relaxed problem is feasible for the original IP	"
7	COSTCHANGE	Changes the costs in a Lagrangean relaxation	"
8	SUBPROBLEM	Describes the basic steps in solving a Lagrangean Relaxation of an Integer Program using subgradient optimization	53



<u>Procedure Number</u>	<u>Title</u>	<u>Description</u>	<u>Page</u>
9	PROJECT	Computes a search direction for optimizing $L(\lambda)$ using projection methods	59
10	NETFLO 1	Solves the network flow relaxation NF1 of the SCP	86
11	PARTITION	Partitions the constraints of the SCP to give a Lagrangean relaxation	101
12	COSTPLUS	Ensures all costs are non-negative in a Lagrangean relaxation in which rows of the SCP are relaxed	109
13	GRAPHBOUND	Computes lower bounds to the SCP from graph covering relaxations	117
14	DECOMPOSITION BOUND	Computes lower bounds to the SCP by partitioning the problem and solving smaller problems	140
15	DEPTHFIRST SEARCH	A Depth First Tree Search on rows for the SCP	172
16	BESTBOUND SEARCH	A Best Bound Tree Search on Rows for the SCP	176

$A$  In a table of results denotes a randomly generated constraint matrix

$A$  SCP constraint matrix

$A_\ell$  Submatrix of  $A$  consisting of some rows of  $A$

$A(p,q)$  A cyclic matrix with  $P$  non-zero entries per column and  $q$  zero entries

$a^i$   $i$ th row of  $A$

$a_j$   $j$ th column of  $A$

$B$  Matrix used for relaxed constraints (Ch.3)

$B$  Matrix used for blossom constraints (Ch.5)

$B_k$   $\{j | a_{kj} = 1 \text{ and } a_{\ell j} = 0 \text{ for all } \ell < k\}$   $k$ th block of constraint matrix

$B_k^g(i)$  Set of variables in block  $B_k$  with a 1 in row  $i$  that have not been fixed equal to 0 or 1

BOCH Logical variable in Procedure 3

BLP Bank location problem:

$$\max_{x,y} \left( \sum_i \sum_j d_{ij} x_{ij} - \sum_j c_j y_j \mid \sum_j x_{ij} = 1 \text{ for all } i, \right. \\ \left. 1 \leq \sum_j y_j \leq k, x, y \in \{0,1\} \right)$$

$c$  Cost vector for the SCP

$d$	Costs for a relaxation of the SCP in which columns are split
DLP	Dual linear program to LP: $\max[1^T u \mid A^T u \leq c, u \geq 0]$
DLPB	Dual linear program to LPB: $\max_w \left[ \sum_{i=1}^m w_i + \sum_{p \in P} p^r_p \mid A^T w + B^T \leq c, w, \geq 0 \right]$
DSCP	Integer programming dual of the SCP: $\max_F [F(1) \mid F(Ax) \leq c, F \text{ subadditive}, F(0) = 0]$
DYSCP	Dynamic Set Covering Problem: $\min_x \left[ \sum_k \sum_j x_{jk} c_{jk} \mid \sum_j a_{ijk} x_{jk} \geq 1 \text{ for all } i, j, x_{jk} \in \{0,1\} \right]$
$d_{ij}$	Cost of arc $(i,j)$ in network flow relaxation NF1( $d$ ) (Ch.4)
$d_t$	Cost of $t$ th variable of GCR2 (Ch.5), Cost of $t$ th variable in DEC( $d$ ) (Ch.6)
$E$	Arc set of a graph $G(V,E)$
$E_i$	Set of vertices in a graph incident with vertex $v_i$
$f_{ij}$	Cost of flow in arc $(i,j)$ for network flow problem
$G$	Graph $G(V,E)$
$G1$	Graph for network flow relaxation NF1
$G2$	Graph for network flow relaxation, NF2
$G(V,E)$	Graph with vertex set $V$ , arc set $E$

- GCP Graph Covering Problem:  $\min_x [cx | Ax \geq \underline{1}, x \in \{0,1\}^n]$   $A$  has at most 2 non-zero entries per column].
- GCR1( $\lambda$ ) Graph Covering Relaxation 1 obtained by relaxing rows of the SCP.
- GCR2( $d$ ) Graph Covering Relaxation 2 obtained by relaxing columns of the SCP.
- GLR( $F$ ) Generalised Lagrangean Relaxation:  
 $\min_x [c^T x - F(Ax - b) | Bx \geq d, x \text{ integer}]$
- GSCP Generalised Set Covering Problem:  $\min_x [c^T x | Ax \geq \underline{1}, x \in \{0,1\}^n]$  where  $a_{ij} = 0, 1$  or  $-1$ .
- $h_j$  Number of non-zero entries in column  $j$  of the SCP:  

$$h_j = \sum_{i=1}^m a_{ij}$$
- $I$  Index set of rows under consideration.
- $i$  Subscript corresponding to rows (constraints) of the SCP.
- $I_m$  The  $m \times m$  identity matrix.
- IP Integer program:  $\min_x [cx | Ax \geq b, Bx \geq d, x \text{ integer}]$
- IP1 Integer program to partition constraints:  
 $\max [1^T y | A^T y \leq \underline{2}, y \in \{0,1\}]$
- $J$  Index set of columns under consideration.
- $J'$  Index SCP of columns of the SCP that have been split to give variables of unequal value.

- $j$  Subscript corresponding to columns (subsets) of the SCP.
- $j_0$  Current variable under consideration.
- $K$  A feasible solution the the SCP or GCP.
- $K^*$  An optimal collection of subsets for the SCP or arcs in an optimal solution for the GCP (Ch.5).
- $k$  Iteration counter, or subscript.
- KCOL Number of non-zero entries in any column in a row relaxation of the SCP.
- KMAX Maximum number of iterations allowed.
- KP Knapsack problem:  $\min_x [cx \mid ax \geq b, x \geq 0]$
- $l_{ik}$  Lower bound on flow in arc  $(i,k)$ .
- $L$  List of non-zero elements of  $x$ .
- $L(\lambda)$  Optimal objective function of LR( $\lambda$ ):  $c^T \bar{x} - \lambda(A\bar{x} - b)$ .
- LP Linear programming relaxation of the SCP:  
 $\min_x [c^T x \mid Ax \geq 1, x \geq 0]$
- LPB Linear program equivalent to SCP:  
 $\min_x [c^T x \mid Ax \geq \underline{1}, Bx, r, 1 \geq x \geq 0]$
- $L(\lambda)$  Optimal objective function of LR( $\lambda$ ).
- LR Problem of finding the best multipliers  $\lambda$  for LR( $\lambda$ ):  
 $\max_{\lambda} L(\lambda)$ .

- LR( $\lambda$ ) Lagrangean relaxation:  $\min_x [c^T x - \lambda(Ax - b) \mid Bx \geq d, x \text{ integer}]$
- LR1( $\lambda$ ) Lagrangean relaxation equivalent to the LP relaxation that gives the network flow relaxation NF1( $d$ ).
- $M$  Index set of rows (constraints, elements) for the SCP.
- $m$  Number of rows (constraints, elements) for the SCP.
- $M_j$   $\{i \mid a_{ij} = 1\}$ ,  $M_j \leq M$ .
- $N$  Index set of columns (variables, subsets) for the SCP also used for a matrix in Procedure 10.
- $n$  Number of columns (variables, subsets) for the SCP.
- $N_i$   $\{j \mid a_{ij} = 1\}$ ,  $N_i \leq N$ .
- N( $f$ ) Network flow problem equivalent to the LP relaxation.
- NF Network flow problem:  

$$\min_{\xi} \left[ \sum_k \sum_i c_{ik} \xi_{ik} \mid \sum_k \xi_{ik} - \sum_k \xi_{ik} = u(i), u \geq \xi \geq \ell \right]$$
- NF1 Network flow relaxation 1:  

$$\min_{\xi} \left[ \sum_{i=1}^m \sum_{j \in N_i} d_{ij} \xi_{ij} \mid \sum_{j \in N_i} \xi_{ij} \geq 1, \xi_{ij} \geq 0 \right]$$
- $P$  Set of odd subsets of vertices of a graph.
- $P_i$  Set of odd subsets of vertices that contain vertex  $i$ .
- $P(w)$  Reformulation of the problem, LR:  

$$\max_w [w \leq c^T x(\epsilon) + \lambda^T (Bx(t))]$$
- $P_j$  number of variables derived from column  $j$  equal to 1.

- $R$  Set of relaxed constraints.
- $\bar{R}$  Set of subgradients in the projection (Ch.3).  
Set of constraints not relaxed (Ch.5).
- $r$  Vector of slack variables (Ch.2).  
Right hand side for blossom constraints (Ch.5).
- $S$  Subset of constraints or vertices.
- $\bar{S}$  Set of subgradients to be considered for the projection.
- $s$  Vector of reduced costs.  
Also source vertex for network flow relaxations (Ch.4).
- SCP Set covering problem:  $\min_x [cx \mid Ax \geq \mathbf{1}, x \in \{0,1\}^n]$
- SCPD( $d$ ) A definition of the SCP that defines the decomposition relaxation (Ch.6).
- SCPG( $d$ ) A definition of the SCP that defines relaxation GCR2 (Ch.5).
- SPP Set partitioning problem:  $\min_x [cx \mid Ax = \mathbf{1}, x \in \{0,1\}^n]$
- $T$  Superscript  $T$  denotes transpose of a matrix.
- $T_j$  Index set of arcs derived from column  $j$ th relaxation GCR2.
- $T(S)$  Set of arcs with at least one end in set of vertices  $S$ .
- $u$  Problem variable for DLP.

$u_{ik}$  Upper bound on flow in arc  $(i,k)$ .

UIP Unconstrained Integer Program:

$$\min_x [c^T x + \sum_i k_i (a^i x - b_i)^2 | x \in \{0,1\}]$$

UPLP Uncapacitated Plant Location Problem:

$$\min \left\{ \sum_i \sum_j d_{ij} x_{ij} + \sum_j c_j y_j \mid \sum_j x_{ij} \geq 1 \text{ for all } i, \right. \\ \left. x_{ij} \leq y_j, x, y \in \{0,1\} \right\}$$

$V$  Vertex set of graph  $G(V,E)$ .

$v_i$  Vertex of graph  $G(V,E)$ .

$v(P)$  The optimal solution value of problem  $P$ .

$w$  Problem variable in §3.3 used for projection method (Ch.3).  
Dual variable corresponding to constraint  $a^i x \geq 1$  in GCP.

$x_j$  Problem variable  $\xi$ ,  $j=1,2,\dots,n$ .

$x^*$  Optimal solution to the SCP.

$\bar{x}$  Prime cover for the SCP.

$z$  Objective function value for the SCP:  $z = c^T x$

$z^*$  Optimal objective function value.

$z_l$  Lower bound to the SCP.

$z_u$  Upper bound to the SCP.



$\alpha$	Constant used in subgradient optimization
$\beta$	Parameter for Camerini et al's method (Ch.3)
$\beta_t$	Column derived from column $a_j$ of the SCP for $t \in T_j$
$\gamma$	Subgradient of the Lagrangean Function
$\Delta$	Amount by which a variable is changed
$\delta$	Parameter used to calculate steplength
$\epsilon$	Small positive number used as a tolerance
$\epsilon_j$	Arc in a graph
$\zeta$	Dual Variable for blossom constraint (Ch. 5)
$\theta$	Parameter for Camerini et al's method (Ch. 3)
$\kappa_j$	Number of arcs in the GCP derived from column $a_j$ in the SCP
$\lambda$	Lagrange multiplier
$\nu$	Search direction for subgradient optimization (Ch. 3)
$\pi$	Penalty for changing costs in the relaxed problem
$\xi$	Flow in an arc for a network flow problem
$\rho$	Density of the SCP
$\sigma$	Steplength for subgradient optimization (Ch. 3)
$\omega$	Dual variable for vertex in the GCP
$\underline{1}$	Vector with all components equal to 1
$+$	Superscript + gives the value of a variable at the next iteration
$\vee$	Logical inclusive 'or'
$\wedge$	Logical 'and'
$\geq$	Greater than or equal to
$\leq$	Less than or equal to

$ S $	Cardinality of a set $S$
$\subseteq$	Set inclusion. $A \subseteq B$ means $A$ is contained in $B$ .
$\cap$	Set intersection. $A \cap B$ means the elements in both $A$ and $B$ .
$\cup$	Set Union. $A \cup B$ means the elements in either $A$ or $B$ .

CHAPTER 1

INTRODUCTION AND LITERATURE SURVEY

1.1 Definitions

The set covering problem, SCP, is the integer program:

$$\text{SCP} \quad \min_x \{c^T x \mid Ax \geq \underline{1}, x_j \in \{0,1\}, j=1,2,\dots,n\}$$

where  $A$  is an  $m \times n$  matrix with  $a_{ij}$  equal to 0 or 1 and  $\underline{1}$  is an  $m$ -dimensional vector of 1's. It is so called because each column of the constraint matrix  $A$  represents a subset of a set,  $M$  say, of  $m$  elements with  $a_{ij} = 1$  if and only if the  $i$ th element of  $M$  belongs to the  $j$ th subset,  $M_j$ . The cost of subset  $M_j$  is  $c_j$ . The optimal solution,  $x^*$ , to the SCP gives a minimum cost collection of subsets,  $K^*$ , in which each element of  $M$  occurs at least once. A subset,  $M_j$ , is in  $K^*$  if and only if  $x_j^* = 1$ . Let  $N = \{1,2,\dots,n\}$  be the index set of subsets of  $M$  and let  $N_i = \{j \mid a_{ij} = 1\}$ . The  $j$ th column of  $A$  will be denoted by  $a_j$  and the  $i$ th row by  $a^i$ .

Closely related to the SCP is the set partitioning problem, SPP, in which the inequality constraints are replaced by equality constraints.

The graph covering problem, GCP, is a special case of the SCP in which each column has at most two non-zero entries.

A graph,  $G = G(V,E)$ , consists of a vertex set  $V$  and an edge (arc) set  $E$ . If there are  $m$  vertices denoted by  $v_i, i=1,2,\dots,m$ , and  $n$  arcs denoted by  $\epsilon_j, j=1,2,\dots,n$ , let  $A$  be the vertex-arc incidence matrix. Thus if arc  $\epsilon_j$  connects  $v_i$  and  $v_k$  then  $a_{ij} = a_{kj} = 1$  and  $a_{\ell j} = 0$  for  $\ell \neq i,k$ . Associated with each arc  $\epsilon_j$  let there be a cost  $c_j$ . The

resulting SCP is the problem, GCP, of finding a minimum cost set of arcs such that each vertex is incident with at least one arc in the set. A path of length  $p$  is a sequence of vertices  $v_{i^0}, v_{i^1}, \dots, v_{i^p}$  and a circuit is a path  $v_{i^0}, v_{i^1}, \dots, v_{i^k} \neq v_{i^l}$  for  $k \neq l$ . A tree is a connected graph with no circuits.

## 1.2 Relaxations

A relaxation of an SCP is an easier problem such as a graph covering, network flow or linear program whose feasible region contains the feasible region of the SCP. Thus solving the relaxation gives a lower bound to the SCP.

The linear programming relaxation of the SCP, LP, is obtained by replacing  $x_j \in \{0, 1\}$  by  $1 \geq x_j \geq 0$  in the SCP. The dual linear program, DLP, is then:

$$\text{DLP} \quad \max_u \{c^T u \mid A^T u \leq c, u \geq 0\}$$

Another relaxation of the SCP gives a knapsack problem, KP:

$$\text{KP} \quad \min \{c^T x \mid \sum_{i=1}^m \lambda_i \sum_{j=1}^n a_{ij} x_j \geq \sum_{i=1}^m \lambda_i : x_j \in \{0, 1\}, j=1, 2, \dots, n\}$$

where the weights  $\lambda_i \geq 0$  are given and the weighted constraints of the SCP are added together to give a single constraint.

Given an integer program, IP,

$$\text{IP} \quad \min_x \{c^T x \mid Ax \geq b, Bx \geq d, x \text{ integer}\}$$

$\alpha$  Lagrangean relaxation [G8],  $\text{LR}(\lambda)$ , is the problem:

$$\text{LR}(\lambda) \quad \min_x \{c^T x - \lambda^T (Bx - d) \mid Ax \geq b, x \text{ integer}\}$$

The constraints  $Ax \geq \bar{b}$  are chosen so as to give an easily solved problem  $LR(\lambda)$  such as a network flow, minimum spanning tree or shortest path problem. If  $L(\lambda)$  is the optimal objective function value of  $LR(\lambda)$  this gives a lower bound to the optimal solution value of IP. If a solution,  $\bar{x}$ , to the relaxation  $LR(\lambda)$  satisfies  $\lambda^T (B\bar{x} - d) = 0$  and is feasible for IP then  $\bar{x}$  is an optimal solution to IP.

The best lower bound obtainable from such a Lagrangean relaxation is given by  $L(\lambda^*)$  where  $\lambda^*$  is an optimal solution to the problem LR below:

$$LR \left[ L(\lambda^*) = \max_{\lambda \geq 0} L(\lambda) \right]$$

One method of solving LR is to use subgradient optimization, in which, for a given value of  $\lambda$ ,  $LR(\lambda)$  is solved. If the optimal solution to  $LR(\lambda)$ ,  $\bar{x}$ , is not optimal for IP then a subgradient  $\gamma$  is given by  $\gamma = d - B\bar{x}$ . The Lagrange multiplier  $\lambda$  is then updated by  $\lambda + \lambda + \sigma\gamma$  for a positive constant  $\sigma$ . More generally a subgradient,  $\gamma$ , of a function  $f: R^p \rightarrow R$  at a point  $\lambda \in R^p$ , is a vector for which  $f(\lambda + \sigma d) - f(\lambda) \leq \sigma \gamma^T d$  for all  $d \in R^p$  and sufficiently small  $\sigma$ ,  $\sigma > 0$ .

A network flow problem [F2] is used to describe the problem of finding a minimum cost feasible flow in a network in which each arc has an upper and lower bound on the flow and a cost. This is formulated as the integer program, NF,

$$\begin{array}{l}
 \text{NF} \left[ \begin{array}{l}
 \min_{\xi} \sum_{ki} c_{ik} \xi_{ik} \\
 \text{subject to} \\
 \sum_k \xi_{ik} - \sum_k \xi_{ki} = v(i) \quad \text{for all vertices } i \\
 u_{ik} \geq \xi_{ik} \geq l_{ik} \quad \text{for all arcs } (i,k)
 \end{array} \right.
 \end{array}$$

where  $v(i) = v$  if  $i$  is the source vertex,  $v(i) = -v$  if  $i$  is the sink vertex and  $v(i) = 0$  otherwise;

$u_{ik}$  is an upper bound on the amount of flow in arc  $(i,k)$

$l_{ik}$  is a lower bound on the flow in arc  $(i,k)$ .

A feasible solution to the SCP is one that satisfies the constraints and is known as a cover. A prime (minimal) cover,  $\bar{x}$ , is a cover for which no element  $\bar{x}_j$  that is set equal to 1 can be set equal to 0 without violating a constraint. An optimal solution to the SCP is a prime cover if all the costs are positive. The optimal solution of a problem  $P$  will be marked by \* and the optimal objective function value by  $v(P)$ . An optimal solution of the SCP will be denoted by  $x^*$  and  $z^* = c^T x^* = v(\text{SCP})$ . An upper bound to the SCP will be denoted by  $z_u$  and a lower bound by  $z_l$ .

A successful technique in solving integer programs is branch and bound [B3] (tree search). The given SCP corresponds to the root node of the tree. Subsequent SCP's are generated by fixing variables in the original problem and these give rise to successor nodes. The bounds can be generated by solving a relaxation of the SCP at each node. *A node is fathomed if one of the following conditions holds:-*


- (i) The lower bound exceeds an upper bound,  $z_u$ , to the SCP
- (ii) The relaxed problem is infeasible
- (iii) The solution to the relaxed problem corresponds to a feasible solution to the SCP and satisfies the complementary slackness conditions.


No successor nodes are generated from a fathomed node, otherwise variables are fixed and further subproblems are generated until all the nodes have been fathomed. An active node in a tree search is

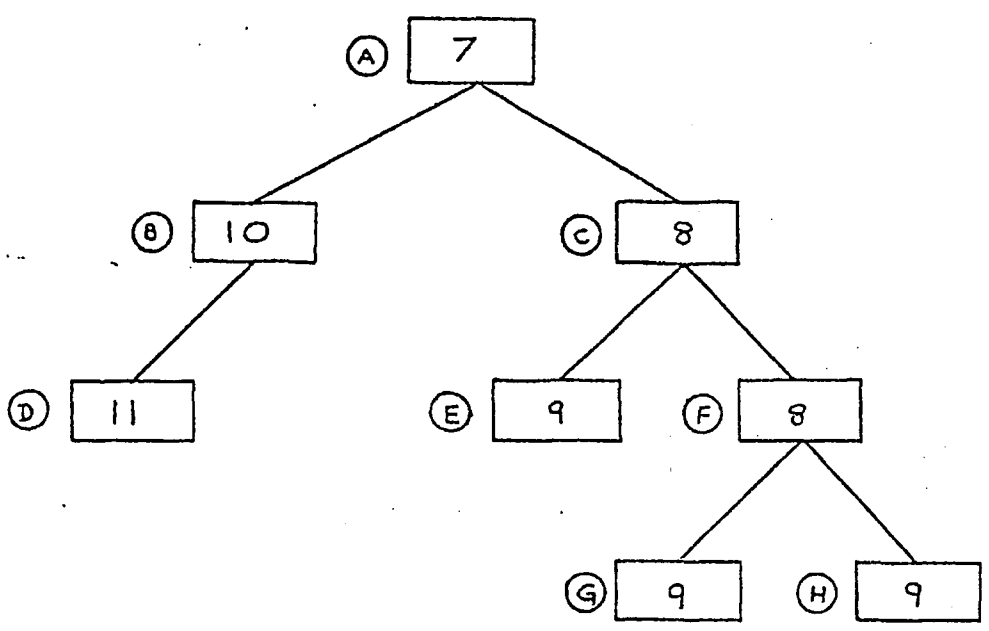
Figure 1.1

An Example of a Search Tree for a Branch and Bound Procedure

An upper bound,  $z_u$ , is 11

Nodes are labelled 

The lower bound at a node,  $z_l$ , is shown as 



A depth-first tree search would search the nodes in the order

A, B, D, C, E, F, G, H

A breadth-first tree search would search the nodes in the order

A, B, C, D, E, F, G, H

A best bound search would search the nodes in the order

A, C, F, E, G, H, B, D

Node C is the father of E

Node F is the brother of E

Node D is fathomed since  $z_l = z_u$

Nodes E, G and H are active

Node F is branched

one which is not fathomed and from which no branching has taken place. A branched node is one from which branching has taken place and which has not been fathomed. A father node of a node,  $P_u$ , in a search tree is the node immediately above  $P_u$  in the tree and a brother node of  $P_u$  is one which has been generated from the same father node as that of  $P_u$ . A depth-first tree search explores recursively a successor node until a node is fathomed. The algorithm then backtracks until a node from which a successor node can be generated is found. A breadth-first tree search explores all the successor nodes of a single node and then takes the first successor node and explores all its successors, the second successor node and all its successors and thus continues until all nodes are fathomed. The best bound search chooses the next node from which to branch as the one with the lowest bound. Other heuristics can be used for the strategic problem of choosing the next node from which to branch. Tree searches are shown in Fig.1.1.

### 1.3 Literature Survey

Applications of the SCP are listed in the first of four parts to this section. A discussion of problems that are related to the SCP follows. Solution techniques for these problems are often applicable to the SCP. Algorithms for the SCP are reviewed briefly in the third part and the final part outlines some of the many theoretical results that have been obtained for the SCP. Surveys of the SCP are given in Garfinkel and Nemhauser [G3], Gondran [G16] and Christofides and Korman [C8]. Many of the practical problems given in the survey paper for the SPP by Balas and Padberg [B8] can also be solved using the SCP.



### 1.3.1 Applications

One of the first applications of the SCP was to airline crew scheduling problems [A2, B1, M5, R3, B26]. The columns of the SCP represent sequences of flight legs and the rows represent crews. The optimal solution to the SCP then gives an optimal allocation of crews to flight legs. More general personnel scheduling problems that have been solved by SCP's are given in [T3]. The SCP has also been used to allocate buses to schedules [G5, S8].

Location of emergency facilities can be analysed using SCP's [B22, B25, R1, T7, T8, W1]. In these problems each row of the SCP represents a district in a town and each column a possible location for an emergency facility such as an ambulance or fire station. A location problem using the cardinality constrained set covering problem in which the number variables that can be set equal to 1 is constrained is given in [C6].

The SCP is also used in routing problems. For a delivery problem each route can be represented by a column of the SCP and each destination by a row. Associated with each route is a cost (distance) and the SCP solution gives a set of routes of minimum cost (distance) that visits each destination. This is described in Pierce [P4].

The SCP has been used for circuit and switching theory in electrical engineering and for minimising boolean expressions [B32, G12, Q1, R4, S9]. Other network and graph theoretic problems, such as the vertex colouring and minimum dominating set in a graph problems can be solved using the SCP [B18, B19, B24].

Other problems to which the SCP can be applied are those of data storage and information retrieval [C13, D1]. For an information

retrieval problem each variable  $x_j$  can represent a library and  $a_{ij} = 1$  if and only if the information indexed by  $i$  can be found in the  $j$ th library. The cost of visiting the  $j$ th library is given by  $c_j$  and an optimal solution  $x^*$  to the SCP has  $x_j^* = 1$  if and only if the  $j$ th library is used in a minimum cost set of libraries needed to access all the information.

In production planning the SCP can be used for scheduling problems such as a simple assembly line balancing analysis [S4]. Decision theory can also be modelled using SCP's [K3].

### 1.3.2 Problems related to the SCP and problems used in its solution

#### 1.3.2.1 Well-solved cases of the SCP

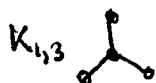
The SCP belongs to a class of problems that is known as NP- complete [G2]. This means that it can be solved by a nondeterministic algorithm in polynomial time or that the depth of a binary search tree is polynomial [A1, H9]. No polynomial algorithm is known that can solve the SCP. However, in the case when there are at most two 1's per column of the constraint matrix, the SCP is a graph covering problem. Algorithms for this problem are based on the matching algorithm of Edmonds [E1, E2]. This algorithm maintains a primal feasible and dual feasible solution to the equivalent LP problem and aims to satisfy complementary slackness.

Edmonds' algorithm is  $O(n^4)$  and by using improved data structures  $O(n^{2.8})$  [G1, L2] and  $O(n^{2.5})$  [L2] implementation can be obtained. For an efficient algorithm to solve successive matching problems on the same graph, with slightly different costs for each problem, sensitivity analysis can be useful; for example if the costs are changed only on arcs incident to one particular vertex. This procedure is described in a primal

algorithm for the matching problem in [C16]. Further details on sensitivity analysis for the case when only two costs on arcs are changed are given in [W2], where the matching problem is used to solve a Lagrangean relaxation of the SPP. An algorithm for the graph covering problem is given in White and Gillenson [W5] which starts with a set of arcs that cover all the vertices in a graph and removes arcs until an optimal solution is obtained.

Unlike the GCP the node covering problem, NCP, of finding a set of vertices in a graph of minimum weight such that each arc is incident with at least one vertex in the set is not well solved except for special classes of graph. These are firstly chordal graphs [G6] i.e. graphs in which there are no circuits of length greater than three without chords (a chord is an arc whose ends are both vertices in the circuit). Secondly, node covering problems on circle graphs can be solved by a polynomial algorithm [G7]. A circle graph is defined by letting each vertex represent a chord in a given circle. If two chords intersect then the corresponding vertices are linked by an arc. Claw-free graphs and interval graphs are other classes of graphs for which the node covering problem can be solved optimally [G2]. A claw-free graph is one without a subgraph that is a claw. A claw is the <sup>\*</sup>bipartite graph,  $K_{1,3}$ . An interval graph is formed by letting each interval between two numbers on the real line be represented by a vertex and connecting two vertices if their corresponding intervals have a non-empty intersection. Lastly if the graph is a line graph the node covering problem can be well solved [H2]. A line graph  $L$  of a graph  $G$  is derived by letting each arc of  $G$  represent a vertex of  $L$  and two vertices are joined in  $L$  if and only if the corresponding arcs meet each other in  $G$ .

# A bipartite graph has the vertices partitioned into two sets,  $V_1$  and  $V_2$ . Each arc joins a vertex in  $V_1$  to a vertex in  $V_2$ . Hence there are no odd circuits.



The SCP is an easily solved problem whenever the non-zero entries of  $A$  occur in consecutive rows as the example SCP1 below shows. [B15]

Example 1

$$\begin{array}{l}
 \text{SCP}_1 \left[ \begin{array}{l}
 \min_x \quad 4x_1 + 5x_2 + 3x_3 + 6x_4 + 2x_5 \\
 \text{Subject to} \quad x_1 \quad \quad \quad + x_3 \quad \quad \quad \geq 1 \\
 \quad \quad \quad x_1 + x_2 + x_3 \quad \quad \quad + x_5 \geq 1 \\
 \quad \quad \quad x_1 + x_2 \quad \quad \quad + x_4 + x_5 \geq 1 \\
 \quad \quad \quad \quad \quad x_2 \quad \quad \quad + x_4 \quad \quad \quad \geq 1 \\
 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad x_j \in \{0,1\} \quad j=1,2,\dots,4
 \end{array} \right.
 \end{array}$$

Dynamic programming can be used to solve the problem by first defining an  $m$ -dimensional vector  $\beta^k$  that has 1's in the first  $k$  components and 0's elsewhere. Define  $\beta_i^k \setminus a_j$  for a column  $a_j$  of the SCP constraint matrix by  $\beta_i^k \setminus a_{ij} = \max(0, \beta_i^k - a_{ij})$ . Let function  $F_k$  be defined on vectors  $\beta^1, \beta^2, \dots, \beta^k$  by:

$$F_0(0)$$

$$F_k(\beta^k) = \min_{j \in N_k} [F_{k-1}(\beta^k \setminus a_j) + c_j]$$

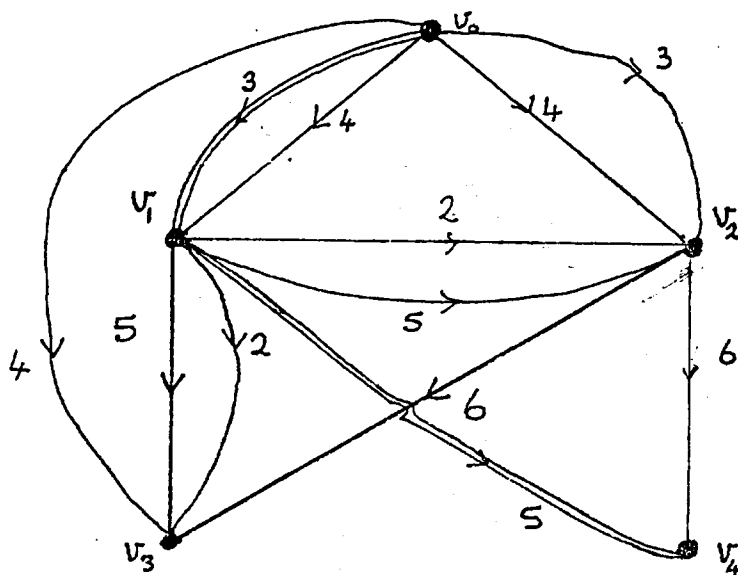
for  $k = 1, 2, \dots, m$ .

The SCP optimal solution value is  $F_m(\beta^m)$ . Unlike dynamic programming when applied to the general SCP this problem does not need excessive storage because only vectors of type  $\beta^k$  are used in the above recursion equation. This problem can also be thought of as a shortest path problem on a graph. An initial vertex  $v_0$  is defined plus  $m$  vertices  $v_1, v_2, \dots, v_m$ , one for each row of the SCP. Each column  $a_j$  of the SCP is represented by  $h_j$  arcs, where  $h_j$  is the number of 1's in the column. If the first non-zero entry occurs in row  $i_1$  and the last in row  $i_2$  then arcs  $(v_{i_1-1}, v_\ell)$  are constructed for  $\ell = i_1, i_1+1, \dots, i_2$ .

Each arc has cost  $c_j$ . The SCP solution is given by the shortest path from  $v_0$  to  $v_m$ . The graph  $G_1$  for the example is shown in Figure 1.2. The shortest path from  $v_0$  to  $v_4$  is given by the path  $(v_0, v_1, v_4)$  and has cost 8.

Figure 1.2

Graph  $G_1$  in Which a Shortest Path Solves SCP1



Each arc is labelled with the distance between endpoints.  $(v_0, v_1, v_4)$  is the shortest path from  $v_0$  to  $v_4$ .

The case of cyclic matrices can be solved by a rounding argument. A cyclic matrix,  $A(p,q)$ , has  $p$  1's per column and  $q$  0's. The  $j$ 'th column has 1's in rows  $j, j+1, \dots, j+p-1 \pmod{p+q}$  and 0's elsewhere. As an example  $A(3,2)$  is shown below.

Example 2

A cyclic matrix  $A(3,2)$

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

In general the LP solution to the unicost SCP with constraint matrix  $A(p,q)$  is given by setting each  $x_j$  equal to  $1/p$  and since there are  $p+q$  columns the optimal solution value is equal to  $(1 + q/p)$ . Suppose  $p+q = k p + r$  where  $p > r \geq 0$  and  $k$  and  $r$  are integers. Then the lower bound to an optimal solution of the SCP is  $k$  if  $r=0$  and  $k+1$  if  $r > 0$  since it must be integer. A feasible solution to the SCP can be obtained by setting  $x_j = 1$  for  $j=1, p+1, \dots, tp+1$  where  $t$  is the largest integer for which  $tp + 1 \leq p+q$ . Hence if  $r=0$ ,  $t=k-1$  giving  $k$  non-zero components of  $x$  with total cost  $k$  and the SCP solution equals the LP solution. If  $r > 0$  then  $t=k$  and the cost of the SCP solution is  $k+1$  and again equal to the lower bound. The cost of the unicost SCP with constraint matrix  $A(p,q)$  is thus  $1 + \lceil q/p \rceil$  where  $\lceil y \rceil$  denotes

the least integer greater than or equal to  $y$ . This type of constraint matrix occurs frequently in scheduling problems and the determinant of the matrix can be very large which means that traditional cutting planes derived from LP, such as Gomory cuts, are unlikely to solve the problem quickly.

Details of how the rows can be manipulated using graph theory to give a related network flow problem are given in [B14, B15].

### 1.3.2.2 Problems of which the SCP is a special case

The SCP is a special case of the following uncapacitated plant location problem, UPLP:

$$\text{UPLP} \left[ \begin{array}{l} \min_{x,y} \sum_{j=1}^n \sum_{i=1}^m d_{ij} x_{ij} + \sum_{j=1}^n c_j y_j \\ \text{Subject to} \quad \sum_{j=1}^n x_{ij} \geq 1 \quad \text{for } i=1,2,\dots,m \\ \quad \quad \quad x_{ij} \leq y_j \quad \text{for all } i,j \\ \quad \quad \quad x_{ij}, y_j \in \{0,1\} \quad \text{for all } i,j \end{array} \right.$$

This can be transformed into an SCP by setting  $d_{ij} = \infty$  if  $a_{ij} = 0$  and  $d_{ij} = 0$  if  $a_{ij} = 1$ . The coefficients of  $y_j$  in the objective function are the costs of the SCP. The dual heuristic procedures for this problem given by Erlenkotter [E4] can be applied to the SCP as shown in Chapter 2 to give lower bounds. This problem is similar to the bank float location problem, BLP, given by Cornuejols, Fisher and Nemhauser [C14] as:

$$\begin{array}{l}
 \text{BLP} \left[ \begin{array}{l}
 \max_{x,y} \sum_{i=1}^m \sum_{j=1}^n d_{ij} x_j - \sum_{j=1}^n c_j y_j \\
 \text{Subject to} \quad \sum_{j=1}^n x_{ij} = 1 \quad \text{for } i = 1, 2, \dots, m \\
 1 \leq \sum_{j=1}^n y_j \leq K \\
 0 \leq x_{ij} \leq y_j \leq 1 \quad i \in I, j \in J \\
 x_{ij}, y_j \text{ integral}
 \end{array} \right.
 \end{array}$$

The worst case analysis for this problem is applicable to the SCP [C15].

The generalised set covering problem, GSCP, is an integer program:

$$\text{GSCP} \quad \min_x \{c^T x \mid Ax \geq \underline{1}, x_j \in \{0, 1\}, j=1, 2, \dots, n\}$$

in which the constraint matrix  $A$  can have entries equal to 0, 1 and -1. It can be shown that any integer program is equivalent to a GSCP [G17]. *It is assumed that the integer variables have upper and lower bounds.*

A stronger result, given by Zorychta [Z3], is that any 0-1 integer programming problem can be converted to an SCP. Since an integer program can easily be transformed to a 0-1 integer program this means that any integer program is equivalent to an SCP. Zorychta shows that an  $m \times n$  0-1 IP can be converted to an SCP with at most  $\frac{n(n-1)}{2}$  constraints and  $\frac{n(n+1)}{2}$  variables in  $O(mn^2)$  steps.

Firstly any <sup>bounded</sup> IP can be transformed to one in which all the costs, constraint coefficients and right-hand sides are non-negative. Additional variables may have to be added at this stage. Let  $m$  and  $n$  be the number of rows and columns, respectively in the resulting IP.

The IP:

$$\text{IP} \quad \min_x \{c^T x \mid Ax = b, x_j \in \{0, 1\}\}$$



is then equivalent to the following unconstrained integer program,

UIP:

$$\text{UIP} \quad \min_x \{c^T x + \sum_{i=1}^m k_i (a_i^T x - b_i)^2 \mid x_j \in \{0,1\}\}$$

for sufficiently large  $k_i$ . Replacing  $x_i x_j$  by a single variable  $x_{ij}$ , adding the constraints:

$$x_{ij} \geq x_i + x_j - 1$$

and transforming the objective function of UIP to remove redundant information gives an SCP with constraint matrix of the form (S,I) where S has at most 2 non-zero entries in each row and I is an identity matrix of appropriate dimension.

Other variations on the SCP include the dynamic SCP, DYSCP and a stochastic SCP [H1].

The former is the following SCP with special structure:

$$\text{DYSCP} \quad \min \left\{ \sum_{k=1}^r \sum_{j=1}^n x_{jk} c_{jk} \mid \sum_{j=1}^n a_{ijk} x_{jk} \geq 1; i=1, \dots, m; k=0, 1, \dots, r, x_{jk} \in \{0,1\} \right\}$$

The subscript  $k$  refers to time periods and for a location problem

$x_{jk} = 1$  if a facility is located at site  $j$  in time  $k$ . The term:

$$\sum_{j=1}^n \sum_{k=0}^r u_{jk} (x_{jk} + x_{j,k-1} - 2x_{jk}x_{j,k-1})$$

can be added to the objective function where  $u_{jk}$  is a phase-in/phase-out cost for facility  $j$  in time period  $k$ .

The stochastic SCP differs from the SCP in that a variable  $x_j$  is chosen to be in the cover with probability  $p_j$ . The objective is to find a sequential procedure for selecting the variables  $x_j$  which minimises the expected cost among all selection programs. In general this is a hard problem to solve and therefore is only useful for

constraint matrices with special structure.

The SCP is a particular case of a general IP problem for which heuristics are often useful as shown in the survey by Zanakis [Z1]. In the next chapter heuristics are given which compute upper and lower bounds to the SCP and can be used to reduce problem size. Other techniques that are useful for 0-1 programming problems are branch and bound, Lagrangean relaxation and logical tests. These are discussed fully in subsequent chapters.

#### 1.3.2.3 Relationship between the SPP and the SCP

The only difference between the SPP and the SCP is that the former has equality constraints and the latter inequality constraints. The question of which is easier to solve has been considered by several authors, e.g. [B8]. The answer depends on the solution technique used because to find a feasible solution to SCP is trivial, but not so for the SPP. In a branching strategy for the SPP fixing a variable equal to 1 excludes many more variables than in the SCP and therefore an algorithm based purely on branching would generate fewer nodes for an SPP than for an SCP. On the other hand a lower bound to the SCP based on a dual feasible solution to LP would also be a lower bound to SPP. To improve the lower bound for SPP may require extra work by considering negative values of the dual variables that are not feasible for DLP. Hence an algorithm based on lower bounds from the dual linear program may be easier for the SCP than for the SPP. Any SPP can be transformed to an SCP without changing the constraint matrix [B8]. This can be done by adding ~~a vector~~ *a vector of slack variables*  $y$  to the SPP with suitable large cost  $\theta$ , then SPP is equivalent to:

$$\min_{x,y} [-cx + \theta \mathbf{1}^T y \mid Ax - y = \mathbf{1}, y \geq 0, x_j \in \{0,1\}, j=1,2,\dots,n]$$

Substituting for  $y$  and letting  $c' = \theta \mathbf{1}^T A - c$  gives the SCP:

$$\min_x [c'x - \theta m \mid Ax \geq \mathbf{1}, x_j \in \{0,1\}, j=1,2,\dots,n]$$

To transform an SCP into an SPP the constraint matrix  $A$  must be modified [K4]. Each column  $a_j$  is represented by  $|a_j|$  columns each of cost  $c_j$ . The  $r$ th column derived from column  $a_j$  is equal to column  $a_j$  with the first  $(r-1)$  non-zero elements set equal to 0. The modified constraint matrix then has  $\sum_{j=1}^n |a_j|$  columns which is equal to the number of non-zero entries in the original constraint matrix. The problem size can be reduced by the dominance tests of §1.3.3.1. The SPP resulting from these additional columns is *not* equivalent to the SCP from which it was derived, *but an algorithm used to solve this SPP [K4] can be adapted to solve the SCP.*

#### 1.3.2.4 Network flow problems

In this thesis two network flow relaxations of the SCP are given. Both graphs have been described before but have not been used in Lagrangean relaxations of the SCP as in Chapter 4. The first graph, in which vertices represent both rows and columns of the SCP and arcs represent non-zero elements of  $A$ , was first used by Swissair for airline crew scheduling and the costs were assigned to arcs by a somewhat arbitrary procedure [A2]. A variation of this graph in which columns of the SCP with less than 3 non-zero elements were treated differently from other columns is given by Glover and Mulvey [G15]. They solved an integer programming problem with 300 rows and 460 columns with a simple depth first search in 10 seconds (CDC 6600) which had proved to be intractable by using linear programming for a lower bound. The network formulation had 2860 arcs

and 780 nodes. The success of this method, which can only give a bound as good as that obtained from linear programming, can be attributed to the efficient data structures in the network flow routines of Glover et al [G14] that were used. The simplex pivots on these networks then have a simple graph theoretic interpretation because the basis is represented as a tree. Efficient labelling schemes enable simplex pivots to be made by traversing a tree [G14, G15].

The second network in which the vertices represent rows of the SCP and paths represent the columns was used in a relaxation for the unicost SPP by Nemhauser et al [N1]. Again the bound is only as good as that obtained from LP. A more theoretical interpretation of the second network was given by Fulkerson [F3, F5] in conjunction with blocking and antiblocking theory. The main purpose of this was to give a round-off result similar to that for cyclic matrices that could be used in a bound for the SPP. The network was also used by Tind [T4] again in conjunction with antiblocking theory to get a lower bound for the SPP. It has also been used in electrical engineering for finding a set of paths in a circuit that start at a source and end at a sink and pass through all vertices, [H0].

### 1.3.3 Algorithms for the SCP

All practical algorithms for the SCP have three components. The first is preprocessing in which elementary tests are used to remove some of the variables and constraints. The second is to find upper and lower bounds to the problem. The third is to close the gap between the bounds by either generating more problems as in branch and bound methods or reducing the size of the feasible region as in cutting

plane methods. The preprocessing strategies for the SCP are well known and comments concerning their practicability are given here. There then follows a survey of methods for lower and upper bounds to the SCP.

### 1.3.3.1 Preliminary reductions

Before using an algorithm for the SCP a number of elementary reductions can be made.

#### (a) Negative Costs

If a cost  $c_j$  is negative then  $x_j$  equals 1 in any optimal solution of the SCP and all rows covered by  $a_j$  can be removed. In a practical SCP it is unlikely that any of the costs are negative, however if the SCP is being used as a sub-problem of some other problem as in Lagrangean relaxation it is likely that negative costs will occur and thus this reduction test is useful.

#### (b) Single 1 in a Row

If any row,  $i$ , of the constraint matrix has just one non-zero element,  $a_{ij(i)}$  say, then  $x_{j(i)}$  must equal 1 in any feasible solution to the SCP. Although this is unlikely to be useful initially in a branch and bound procedure, further down the search tree subproblems do occur with single 1's in a row and it is here that this reduction is most useful.

#### (c) Row Dominance

If  $a_{i_1j} \geq a_{i_2j}$  for any two rows  $i_1$  and  $i_2$  all  $j$  then row  $i_1$  can be removed as any  $x$  satisfying the constraint  $a^{i_2}x \geq 1$  will also satisfy  $a^{i_1}x \geq 1$ . In a randomly generated problem in which the coefficients  $a_{ij}$  are independent random variables with  $p$  a fixed probability that

$a_{ij} = 1$  the probability (given any two rows) that one dominates another is:

$$* \quad P(\rho, n) \equiv 2(1 - \rho + \rho^2)^n - (1 - 2\rho + 2\rho^2)^n$$

For small  $\rho$ , equal to 0.05 say, this is very small. This reduction is therefore only likely to be useful when  $n$  is very small or  $\rho$  is larger. The probability can be increased and  $n$  reduced by considering the test only on a block of columns as in a branching strategy given in Chapter 7.

#### (d) Column Dominance

For a subset  $S$  of columns and a single column  $j_0$  if  $\sum_{j \in S} c_j \leq c_{j_0}$  and  $\sum_{i \in S} a_{ij} \geq a_{ij_0}$  for all  $i$  then column  $j_0$  can be removed from the problem. This test means that rows covered by column  $j_0$  can be covered for no greater cost by columns in  $S$ . For a unicast problem that is randomly generated as in (c) the probability that one column dominates another is  $P(\rho, m)$  which is small for sparse matrices. Thus these tests are only useful if  $m$  is small or the constraint matrix is dense for random problems.

#### (e) Reduced Costs

If a dual feasible solution to LP,  $u$ , is available then the reduced cost of column  $j$  is given by:

$$\bar{s}_j \equiv c_j - u^T a_j$$

If  $z_u$  is a known upper bound on the solution of the SCP then  $z_l = \underline{1}^T u$  is a lower bound and if

$$\bar{s}_j \geq z_u - \underline{1}^T u$$

then  $x_j = 0$  in any solution with value less than  $z_u$ .

If  $\bar{z}$  is the cost of a known feasible solution to an SCP with integer costs then  $z_u = \bar{z} - 1 + \epsilon$  for any  $\epsilon > 0$  is a suitable upper bound. In

\* See Appendix 1.

practical problems reduced costs are extremely useful. They are not useful in problems for which the cost of a column  $c_j$  is equal to a scalar,  $\theta$ , say, times the number of 1's in the column (i.e.  $c_j = \theta \sum_{i=1}^m a_{ij}$ ). Then a dual feasible solution is  $u_i = \theta$  for all  $i$  and all reduced costs equal 0. In this case if the corresponding SPP has a feasible solution this will be optimal for the SCP. If the constraint matrix has some structure then the column dominance tests may work well as all that is required, given a column  $j_0$ , is to find a subset of columns,  $S$ , for which  $\sum_{j \in S} a_{ij} = a_{ij_0}$  for all  $i$  and then column  $j_0$  can be removed.

### 1.3.3.2 Sorting the constraint matrix

Both the rows and columns of the constraint matrix can be presorted. The algorithms given in [G4, K4, M1, P5] for set partitioning and set covering sort the columns of the constraint matrix  $A$  into blocks  $B_i$ ,  $i=1,2,\dots,m$ . A block  $B_i$  is a set of columns whose first non-zero entry occurs in row  $i$ , that is  $a_{i'j} = 0$  for  $i' < i$  and  $a_{ij} = 1$  for all  $j \in B_i$ . The advantage of this approach for the set partitioning problem is that if  $x_i$  is fixed equal to 1 in a branch and bound scheme then all blocks  $B_i$  for which  $a_{ij} = 1$  can be removed from the problem. Experiments on row permutations have been carried out for the SPP in [M4] and [B7] and it generally seems better to try and cover rows with the least number of elements at the top of a branch and bound tree.

In practice the procedure that generates the SCP constraint matrix from a scheduling problem say, can usually be designed to produce a matrix in block form as shown in §7.3.2.1.

### 1.3.3.3 Algorithms for lower bounds to the SCP

The LP solution was one of the first lower bounds to be used in a tree search method for the SCP [L5]. A problem with the LP is that it cannot handle large sparse problems because of degeneracy and this means that many simplex iterations are required to reach optimality. An improvement is to solve DLP, the dual of LP, for which a sub-optimal solution gives a bound to the SCP [S2]. Although faster than LP, DLP is slow to solve large problems.

Another relaxation is the knapsack relaxation in which the constraints of the SCP are added together to form a single constraint [B7, P5]. This gives a very weak relaxation and it is usually possible to get an improvement by premultiplying the constraints by positive scalars before adding them together.

Heuristics for finding feasible solutions to DLP are given by Balas [B5] which are discussed together with other heuristics in Chapter 2.

A group theoretic relaxation which depends on the size of determinants of square submatrices of  $A$  is given in [G16], but group theory has not been very successful in the solution of large scale 0-1 programs.

Lagrangean relaxation [G8] has been used for a wide variety of combinatorial programs such as the travelling salesman [H4], facility location [G9] and cluster analysis [M6] problems.

Etcheberry [E5] used this approach for the SCP in which some of the constraints were relaxed and the remainder had no variables in common. This relaxation was also used in the disjunctive cut approach of [B7]. Subproblems of the SCP that are network flow and graph covering problems are solved by Lagrangean relaxation in this thesis.



#### 1.3.3.4 Algorithms for upper bounds to the SCP

Upper bounds to the SCP can be found from any feasible solution and heuristics have been studied and analysed by Johnson [J1], Chvatal [C11] and Ho [H7]. One of the easiest ways to find a prime cover is to use the "least-cost-per-constraint-satisfied" to choose columns of the SCP that are in the solution. Hence if  $\frac{c_j}{h_j} = \min \frac{c_j}{h_j}$  where  $h_j = \sum_{i=1}^m a_{ij}$  then  $x_j$  is set equal to 1 and all rows covered by  $x_j$  are removed. The procedure is repeated for the remaining rows and columns until all the rows are covered. The resulting cover is then reduced to a prime cover. If instead of  $\frac{c_j}{h_j}$  the minimum of any function  $f(c_j, h_j)$  is chosen as the criterion for fixing  $x_j = 1$  then the upper bound  $z_u$  satisfies  $z_u \leq z^* \frac{d}{J}$  where  $d = \max_{j \in N} |M_j|$  and this bound can be attained.

An  $r$ -optimal method for an upper bound is given by Roth [R2]. This method means that if any subset of  $r$  columns is deleted from the problem no better solution can be obtained by covering the remaining rows with another subset of up to  $r$  columns. For  $r = 1$  this would mean that if  $x_j = 1$  in a feasible solution to the SCP then setting  $x_j = 0$  and setting another variable,  $x_k$ , say, equal to 1 would not yield an improvement in the bound. This type of method is also used by Baker et al [B1] for solving large airline crew scheduling problems.

#### 1.3.3.5 Branching strategies for the SCP

Any branching strategy can be divided into two parts, the tactical problem of deciding how to fix variables at a node to generate sub-nodes and the strategic problem of finding which node to expand next.

The easiest approach to the tactical problem is to alternately fix  $x_j = 0$  and 1 [B3]. This does not work very well for most problems. The algorithm of Pierce and Lasky [P5] uses a depth first binary search but the block structure of the constraint matrix means that additional variables can be removed. In an algorithm for the SPP Marsten [M1] branches on blocks of variables instead of a single variable. Only one variable in each block can be fixed equal to 1. Blocks of variables can be removed at each node of the search tree instead of fixing a single variable  $x_j$  equal to 0 as in a conventional tree search.

Etcheberry [B5] uses logical combinations of rows to branch. Suppose  $a^i x \geq 1$  and  $a^{i'} x \geq 1$  are two constraints of the SCP with some variables occurring in both constraints. These can be used to divide the SCP into two problems. In the first problem the sum of variables that occur in both constraints must be greater than or equal to 1. In the second problem their sum must equal to 0. As an example consider the constraints:

$$\begin{aligned} x_1 + x_2 + x_3 &\geq 1 \\ \text{and } x_1 + x_3 + x_4 &\geq 1 \end{aligned}$$

Then either  $x_1 + x_3 \geq 1$

$$\text{or } x_1 = x_3 = 0 \quad \text{implying that } x_2 \geq 1 \text{ and } x_4 \geq 1$$

Marsten's algorithm for the SPP generates a subset of the tree search nodes generated by the above method when the latter is implemented as a depth first strategy.

A theoretical survey of branching strategies for IP is given by Ibaraki [I1]. As expected the depth first search is less likely to work well than say a heuristic search where the next node to branch from is chosen by a rule such as the node having the best bound. Breu and Burdet [B31] give a computational survey of branching

strategies and a computational comparison between depth-first and breadth-first methods for the SCP is given in [T6]. Dominance tests to eliminate nodes are given in [K4].

#### 1.3.3.6 Cutting plane strategies for the SCP

Traditional cutting plane strategies for integer programming have been employed by Salkin and Koncal [S2] who used Gomory cuts [G3] derived from the tableau for DLP. However the large number of cuts generated makes this method unsuitable for large problems.

An algorithm that was able to solve some very large problems (up to 150 rows and 7000 columns) and yet failed on other smaller ones and was therefore not robust was developed by Martin [M2]. This was a cutting plane algorithm based on Gomory cuts with additional steps to try and enforce integrality of the simplex tableau. Other cutting plane approaches based on disjunctive cuts have been proposed by Balas [B5] and Lev and Soyster gave a similar method which uses an LP relaxation [L6].

Disjunctive cuts are generated by considering a set  $S$  of reduced costs for which  $\sum_{j \in S} s_j \geq z_u - z_l$  where  $z_l$  is the lower bound corresponding to the dual variables used to give  $s_j$ . It can be shown that at least one  $x_j, j \in S$ , must be 0 in any solution to the SCP of value less than  $z_u$ . Cuts can then be generated that are SCP type constraints. <sup>Details are given in [B5]</sup> These cuts are superior to the cuts generated by Bellmore and Ratliff [B19] in an earlier cutting plane method and preliminary computational tests indicate that they are capable of solving sparse problems of up to 200 rows and 1000 columns more efficiently than previous methods.

Strengthening inequalities in 0-1 integer programs has been studied by Zemel [Z2] but this method seems unlikely to be useful in a practical algorithm for the SCP.

### 1.3.4 Theoretical Results for the SCP

#### 1.3.4.1 Complexity results

The SCP is an NP-complete problem which means that it is unknown if it can be solved by a deterministic algorithm in polynomial time. It can be solved by a non-deterministic algorithm in polynomial time. A non-deterministic algorithm can be solved by a tree search in which the depth of the tree is a polynomial in the dimension of the problem. Further details are given in Aho, Hopcroft and Ullman [A1] (Chapter 10), Horowitz and Sahni [H9] (Chapter 11) and Garey and Johnson [G2].

Q1 A question can be posed: Is there a polynomial time algorithm that gives a solution  $z_u$  to the SCP such that  $(z_u - z^*)/z^* \leq \epsilon$ , for a given  $\epsilon > 0$ ?

No fixed value of  $\epsilon$  is known for the SCP. For the unicost SCP Johnson [J1] shows that the heuristic of the previous section gives  $z_u \leq (1 + \log(k))z^*$  where  $k$  is the maximum number of 1's in a column of the SCP constraint matrix.

Q2 A second question arises: Is there a polynomial time algorithm that gives a solution  $z_u$  to the SCP such that  $(z_u - z^*)/z^* \leq \epsilon$  most of the time?

Karp [K2] proposes a tree search algorithm for the unicost randomly generated SCP in which unpromising nodes are discarded and the number

of nodes expanded is  $O(n)$  and which gives  $z_u \leq (1 + \epsilon)z^*$  almost always. Probabilistic search methods have also been studied by Gimpel [G12] who shows that for sufficiently large random unicost set covering problems randomly picking columns to cover each uncovered row until all the rows are covered solves the SCP almost always within  $1 + \epsilon$ . These results say little about algorithms for SCP's with specific costs or problems less than a given size, but Karp's result helps to explain why a tree search method which has exponential worst case behaviour can very often produce an optimal solution quickly even if it cannot be proved so. Further, complexity results are given in Karp [K2] and Sahni and Gonzales [S1].

#### 1.3.4.2 The set covering polyhedron

The convex hull of all feasible solutions to the set covering constraints  $(Ax \geq \underline{1}, x_j \in (0,1))$  defines a polyhedron,  $P$ . Facets of  $P$  are linear inequalities that are satisfied by exactly  $d$  \* affinely independent points  $x \in P$  where  $d$  is the dimension of  $P$ . Facets uniquely define the convex hull of the feasible region of SCP unlike cutting planes which do not necessarily intersect the convex hull. For an example of ~~facets~~ for the SPP see [P3]. The reason that the study of facets is useful is that they can be used to generate cuts and thus reduce the gap between an upper and lower bound to the SCP.

Fulkerson [F3] shows that if the row dominance tests of §1.3.3.1 are used to remove all redundant rows then for the set covering polytope the remaining inequalities  $Ax \geq \underline{1}$  define all facets of the form  $\pi x \geq 1$  where  $\pi$  is a 0-1 vector. A complete characterisation of all

\*  $r$  vectors  $x_1, \dots, x_r$  are affinely independent if the vectors  $x_2 - x_1, \dots, x_r - x_1$  are linearly independent or alternatively if every vector  $y$  can be written in at most one way in the form:

$$y = \sum_{k=1}^r \lambda_k x_k \quad \text{where} \quad \sum_{k=1}^r \lambda_k = 1.$$

the facets of an integer program is only known for a few special cases. One of these is the GCP.

The constraints that must be added to a linear programming relaxation of the GCP are the following:

$$\sum_{j \in T(s)} x_j \geq (|s| + 1)/2$$

where  $s$  is a set of vertices with odd cardinality and  $T(s)$  is the index set of arcs that have at least one end in  $s$ , for all such  $s$ .

These are the facets that are used in the graph covering relaxations.

In principle it would be possible to solve an SCP by considering a feasible solution as a vertex of the set covering polyhedron and then finding all adjacent vertices, that is other prime covers, and showing that none of them have lower objective value than the given feasible solution. In practice only polyhedra with a few vertices can be analysed this way and therefore this is not a practical approach. Codes for adjacency in polyhedra are given by Von Hohenbalken [V1] and a mathematical analysis of adjacency is given by Hausmann and Korte [H3].

#### 1.3.4.3 The structure of the constraint matrix

The question of when the solution to the LP relaxation of the SCP has an integer solution has been studied extensively, but no necessary and sufficient conditions exist. The most well known sufficient condition is unimodularity. The constraint matrix  $A$  is unimodular

if all square submatrices have determinant equal to 0, 1 or -1. In this case the LP has an integer solution [G3]. A balanced 0-1 matrix is one for which no submatrix of odd size has row and column sum equal to 2. If the matrix  $A$  is balanced then the LP solves the SCP [P2]. An earlier result which is less strong was given in the form of a tree structure associated with the constraint matrix by Meir and Moon [M3]. Cyclic matrices can be transformed by using trees to give a network flow constraint matrix [B14].

#### 1.3.4.4 Duality

Analogous to linear programming duality, a duality for integer programming can be developed using subadditive functions [W6].

A function  $f$  is subadditive if  $f(a) + f(b) \geq f(a+b)$ . Thus, one can define the dual of the SCP as:

$$\text{DSCP} \left[ \begin{array}{ll} \max_F F(\underline{1}) & (\underline{1} \text{ is an } n\text{-dimensional vector} \\ & \text{of 1's}) \\ \text{subject to } F(Ax) \leq c & \\ & F \text{ subadditive} \end{array} \right.$$

Thus, for LP duality  $F(\cdot)$  is the function  $u^T(\cdot)$  for  $u \geq 0$ . For the problem IP given a subadditive function  $F$  with  $F(0) = 0$ , a generalised Lagrangean relaxation,  $\text{GLR}(F)$ , is given:

$$\text{GLR}(F) \left[ \begin{array}{ll} \text{GL}(F) = \min_x c^T x - F(Ax-b) & \\ \text{subject to } Bx \geq d & \\ & x \text{ integer} \end{array} \right.$$

Then analogous to linear programming duality, where  $v(\text{LP}) = v(\text{DLP})$ ,  $v(\text{IP}) = v(\text{GLR})$  where GLR is the problem:

$$\text{GLR} \left\{ \begin{array}{l} \max \quad \text{GL}(F) \\ F \text{ subadditive} \\ F(0) = 0 \end{array} \right.$$

The optimal solution to GLR cannot be found easily without using branch and bound, cutting planes, dynamic programming or any other technique of integer programming. However state space relaxation could be used to get lower bounds as used in Chapter 6. Wolsey [W6] uses dynamic programming on a large network to generate subadditive functions for the SCP.

### 1.3.5 Data Structures

As can be seen by the results for network flow problems, efficient data structuring is essential for fast algorithms. This is even more important for combinatorial problems where subproblems have to be solved many times. It is also useful to have a data structure which allows transition from one subnode of the tree search to another. Data structures are explained in [A1, H8]. Algorithms for the SCP which store the non-zero entries of the constraint matrix in bits are those of Garfinkel and Nemhauser [G4], Pierce and Lasky [P5] and Korman [K4]. Storage in lists is used by Marsten [M1], Mevert [M4] and gives faster computation times. Chapter 8 describes the data structures used in the algorithms here. A survey of sparse matrix techniques is given by Duff and Reid [D4].

Data structuring for the graph covering relaxations follows those given in Gabov [G1], Derigs [D3] and Even and Kariv [E6] for the matching problem.



## CHAPTER 2

### HEURISTICS FOR UPPER AND LOWER BOUNDS TO THE SCP

#### 2.1 Introduction

Heuristic algorithms are not guaranteed to solve the SCP optimally but they can be used to get both upper and lower bounds on the solution cheaply and quickly. A survey of three heuristic methods for a class of IP's proposed by Senju and Toyoda [S6], Kochenberger, McCarl and Wyman [K3], and Hillier [H5] is given by Zanakis [Z1]. The IP's have inequality constraints and non-negative coefficients and thus there is no problem finding a feasible solution. This study concludes that for large problems there is little to choose between the three methods in terms of *bound quality*. For small problems Hillier's method was more accurate, but it was unsuitable for large problems because it required an excessive amount of storage. In terms of speed the Senju-Toyoda method was the fastest. Other heuristics for both upper and lower bounds are given by Balas [B5] and Balas and Ho [B7]. The SCP is a special case of the uncapacitated facility location problem. A successful heuristic for this problem devised by Erlenkotter [E4] can also be applied to the SCP to improve the lower bound.

The heuristics used in this chapter obtain a lower bound,  $z_{\ell}$ , to the SCP from a feasible solution,  $u$ , to DLP which means that  $z_{\ell}$  can never be greater than  $v(LP)$ . An initial value of  $u$  is obtained from an adaptation of Senju-Toyoda's heuristic which calculates the least cost per constraint satisfied and is summarised in Procedure 1 in §2.2. The lower bound is improved by testing the linear programming

complementary slackness conditions for  $u$  and a feasible solution,  $x$ , to the SCP. If they are satisfied then  $x$  is an optimal solution to the SCP. This is an application of Erlenkotter's method and is described in Procedure 2 in 2.2. Reduced costs are associated with any dual feasible solution,  $u$ , to the LP relaxation and these can be used to remove variables. The heuristics are also used to obtain initial costs for the graph covering and network flow relaxations.

Besides giving lower bounds to the SCP Procedures 1 and 2 also give an upper bound. The computational results of §2.5 show that this bound was often above the optimal solution to the SCP and thus further methods of obtaining upper bounds are discussed in §2.3.

## 2.2 Outline of the Heuristic Methods

The first heuristic, described in Procedure 1 below, initially sets  $u = 0$  as a dual feasible solution to the LP which implies that the associated reduced costs,  $s$ , are equal to the costs  $c$ . A column  $j_0$  for which the reduced cost per constraint satisfied is least is chosen and dual variables are fixed for all rows covered by this column. These rows are then removed and the procedure is repeated until no rows are left. By setting  $x_{j_0} = 1$  for each such column chosen a feasible solution to the SCP is obtained which satisfies  $s_{j_0} x_{j_0} = 0$ . This solution may be improved by reducing  $x$  to a prime cover. (A description of the language used in the Procedures is given in Appendix 5). An example is given in Appendix 3.



4. Calculate Dual Variables in Rows Covered by Column  $j_0$

For  $i \in M_{j_0} \cap I$

$$u_i := u_i + \Delta$$

Increase dual variable

$$I := I \setminus \{i\}$$

Remove row  $i$  from further consideration

For  $j \in N_i \cap J$

$$s_j := s_j - \Delta$$

Decrease reduced costs

$$h_j := h_j - 1$$

Decrease column sums

If  $h_j = 0$

then  $J := J \setminus \{j\}$

If  $I \neq \emptyset$

then goto 2.

5. Reduce  $x$  to a Prime Cover

For  $j \in L$

If  $\sum_{i \in M_j} a_{ij} \geq 2$  for all  $i \in M_j$

Remove column  $j$  from the cover if all rows it covers are overcovered

Set  $L := L \setminus \{j\}$

6. Calculate Upper and Lower Bounds

$$z_u := \sum_{j \in L} c_j$$

Calculate upper bound

$$z_l := \sum_{i=1}^m u_i$$

Calculate lower bound

7. Calculate  $x$

$$x_j := 1 \quad \text{for } j \in L$$

$$x_j := 0 \quad \text{for } j \notin L$$

### 8. Test for Feasibility to the SCP

If  $\sum_{j \in L} a_{ij} x_j = 0$  for any  $i$  Set  $u_i = \infty$  and  $z_l = z_l + \infty$

If  $z_l > z_u$   
then the problem is infeasible.

PROCEDURE 2 describes the second heuristic which checks that the primal and dual feasible solutions,  $x$  and  $u$ , obtained by PROCEDURE 1 satisfy the optimality conditions for linear programming, i.e.:

#### a. Primal Feasibility

$$Ax \geq \underline{1} \quad (2.1)$$

$$\underline{1} \geq x \geq 0$$

#### b. Dual Feasibility

$$A^T u \leq c \quad (2.2)$$

$$u \geq 0$$

#### c. Complementary Slackness

$$u^T (Ax - \underline{1}) = 0 \quad (2.3)$$

$$x^T (c - A^T u) = 0 \quad (2.4)$$

Constraints (2.1) and (2.2) are satisfied by  $x$  and  $u$ . Also  $x$  was chosen to satisfy (2.4). PROCEDURE 2 adjusts  $u$  if constraints (2.3) are not satisfied. This is done by choosing a constraint  $i$  for which  $u_i (Ax - \underline{1})_i \neq 0$  and reducing  $u_i$  to 0. This alters the reduced costs and an attempt is made to increase dual variables  $u_i$  for which  $(Ax - \underline{1})_i = 0$ . If it is possible to increase the lower bound by these adjustments then constraints (2.4) may be violated and another vector  $x$  may have to be chosen. The adjustments to  $u$  are made so that it is always possible to find a vector  $x$  satisfying 2.4. If (2.1), (2.2), (2.3) and (2.4) are all satisfied then the SCP is solved. The method is given in PROCEDURE 2, below:



3. Check  $\min_{j \in N_i} s_j = 0$  for Each Row  $i$

If  $\Delta = \min_{j \in N_i} s_j \neq 0$

then  $u_i := u_i + \Delta$   
 $s_j := s_j - \Delta$  for all  $j \in N_i$

4. Check Complementary Slackness for Columns  $j$  for which  $s_j x_j = 0$

For  $j \in L$

If  $s_j x_j \neq 0$

then  $L := L \setminus \{j\}$  Remove column  $j$  from cover if  $s_j x_j \neq 0$   
 $r := r - a_j$

5. Cover Exposed Rows

For  $i = 1$  to  $m$

If  $r_i = 0$

then  $L := L \cup \{j_0\}$  Find a column to cover exposed row  $i$

$r := r + a_{j_0}$  Adjust slack variables

where  $s_{j_0} = \min_{j \in N_i} s_j$

6. Reduce  $x$  to a Prime Cover

For  $j \in L$

If, for all  $i \in M_j$ ,  $r_i \geq 2$  Remove  $x_j$  from cover if all the rows  
it covers are overcovered

then  $L := L \setminus \{j\}$

$r := r - a_j$

7. Calculate Upper and Lower Bounds

$$z_u := \sum_{j \in L} c_j$$

Calculate upper bound

$$z_l := \sum_{i=1}^m u_i$$

Calculate lower bound

The heuristic of PROCEDURE 2 is repeated until no change in either the upper or lower bound is obtained. The results are summarised in PROCEDURE 3, at the end of which reduced costs are used to remove variables.

PROCEDURE 3 HEURISTICS (SCP,  $z_u, z_l, \bar{x}, u$ )

COMPUTE UPPER AND LOWER BOUNDS TO THE SCP AND REDUCE THE PROBLEM SIZE USING REDUCED COSTS

Input: SCP	The set covering problem
Output: SCP	The set covering problem, possibly reduced in size
$z_u, z_l$	Upper and lower bounds to the SCP
$\bar{x}$	Feasible solution corresponding to best upper bound
$u$	Feasible solution to DLP

1. Initialise Variables

BDCH: = .FALSE.	BDCH is a logical variable which is .TRUE. when the lower bound has changed
KMAX	Maximum number of iterations allowed
$k : = 0$	Set iteration counter to 0
$z_u : = \infty$	Initialise bounds
$z_l : = 0$	



## 2. Calculate Upper and Lower Bounds

INITIAL BOUNDS (SCP,  $z_u$ ,  $z_l$ ,  $x$ ,  $u$ )

Use Procedure 1 to get initial bounds

If  $z_l \geq z_u$   
then if  $z_l = \infty$  then the problem is *infeasible*  
else  $z_u$  is the optimal solution to the SCP  
else goto 3.

## 3. Calculate Improved Bounds, Iteration $k$

$k := 1$

Use PROCEDURE 2 to improve bounds  
 and replace old bounds by new  
 bounds if they have improved

If  $k > KMAX$  then STOP

else LPBD(SCP,  $z_u'$ ,  $z_l'$ ,  $x'$ ,  $u'$ )

If  $z_u' < z_u$   
then  $z_u := z_u'$   
 $x := x'$

If  $z_l' > z_l$   
then  $z_l := z_l'$   
 $u := u'$

If BDCH = .TRUE.  
then repeat 3  
else goto 4.

## 4. Use Reduced Costs to Reduce Problem Size

For  $j = 1,$

If  $s_j \geq z_u - z_l$

then remove  $x_j$  from the SCP

$z_u$  and  $z_l$  are upper and lower bounds  
 STOP.

The computational results for these methods are given in §2.5.

### 2.3 Additional Methods for Computing Upper Bounds

The method used in Procedure 1 of calculating an initial solution  $x$  was to choose columns for which  $s_j/h_j$  was the least and then reduce this to a prime cover. Instead of using  $s_j/h_j$  three other functions were used, i.e.:

- (i)  $c_j/h_j$
- (ii)  $c_j/\log(h_j)$
- (iii)  $c_j/h_j \log(h_j)$

where  $h_j$  is updated as rows are covered and  $\log$  equals the logarithm to base 2 if  $h_j \geq 1$  and 1 otherwise. If  $h_j$  is not updated at each iteration and remains constant then the bounds obtained are not very good thus the reason for updating  $h$  at each iteration. For large problems using (i), (ii) or (iii) often gave a better bound than the one obtained at the end of Procedure 3. In [B7] Balas and Ho found that (i) and (ii) were the most useful functions on large randomly generated problems in that they gave the best upper bounds most often.

Another approach was to take a feasible solution to the SCP and delete a variable  $x_j$  and replace it by another variable. This approach produced many solutions of the same value as the original feasible solution but only on small problems did it produce a solution that was better than one obtained by one of the preceding methods. This method of calculating an upper bound generalises to the  $r$ -optimal method which was first used by Lin [L7] for the travelling salesman problem and later by Roth for the SCP [R2]. Table 2.1 presents the upper bound calculations for several SCP's using the above methods.

## 2.4 Reasons for Failure of the Heuristics to Solve the SCP

The value of the lower bound obtained from the heuristics is never greater than that obtained from the LP relaxation as the former bound is obtained from a dual feasible solution to the LP. The heuristic method, like a complementary pivot algorithm [L4a] for the LP, maintains both primal and dual feasible solutions but fails to reach the LP optimum because the primal variables are restricted to take integer values and also unlike a complementary pivot algorithm the heuristics always maintain  $s_j x_j = 0$  for all  $j$ . Thus even in cases where the LP solves the SCP the heuristics may not do so.

One reason that the heuristics can fail to solve the SCP is because of an odd circuit in a graph covering relaxation of the SCP. The graph covering relaxations are defined in Chapter 5. The 0-graph of a graph is a subgraph which consists of all the arcs with reduced cost equal to 0. Suppose that the following SCP, SCP2, which is also a graph covering problem, has dual feasible solution and corresponding reduced costs equal to 0 in Columns 1 and 2, i.e.  $s_1 = s_2 = 0$ . Then if SCP2 is given by:

$$\text{SCP2} \left[ \begin{array}{l} \min c_1 x_1 + c_2 x_2 + c_3 x_3 \\ \text{Subject to } x_1 + x_2 \geq 1 \\ \quad \quad \quad x_2 + x_3 \geq 1 \\ \quad \quad \quad x_1 + x_3 \geq 1 \\ \quad \quad \quad \quad \quad x_j \in \{0,1\} \quad j=1,2,3 \end{array} \right.$$

a prime cover generated by Procedure 1 would be  $x_1 = x_2 = 1, x_3 = 0$ . If  $u_1 > 0$  then complementary slackness is not satisfied. If  $s_3 = \Delta$  and  $\Delta < u_1$  then Procedure 2 sets  $u_2 \leftarrow u_2 + \Delta$  and  $u_1 \leftarrow u_1 - \Delta$ . Hence the lower bound is unchanged. However  $s_1 x_1$  is no longer equal to 0 hence the prime cover is changed to  $x = (0,1,1)$  which has a cost less

than that of  $x = (1, 1, 0)$  if  $\Delta > 0$ . If  $\Delta > u$ , then  $u$  becomes equal to  $(0, u_2 + u_1, u_3 + \min(\Delta - u_1, u_1))$ . This means that the lower bound is equal to  $u_1 + u_2 + u_3 + \min(\Delta - u_1, u_1)$ . Since the last term is positive the lower bound has increased. If neither the upper nor the lower bound change for this problem then  $\Delta = 0$  and if for simplicity it is assumed that  $c_1 = c_2 = c_3 = 2$  and  $u_1 = u_2 = u_3 = 1$  the bound can be improved by using a cut  $x_1 + x_2 + x_3 \geq 2$ . This corresponds to an odd circuit in the graph with vertex-arc incidence matrix equal to the constraint matrix above. The above analysis can be extended to larger constraint matrices but as it is lengthy will not be considered here. Example SCP2 suggests that the vertex weights obtained by the heuristics will give columns of the SCP with reduced costs equal to 0 which in graph covering relaxations of the SCP will give odd circuits.

## 2.5 Computational Results

Most sparse SCP's with less than 20 rows and less than 100 columns can be solved optimally using heuristics without entering a tree search and thus all the test problems have at least 30 rows. The results, showing percentage differences between upper and lower bounds at the root node of a branch and bound tree, are given in Table 2.1. Table 2.2 shows how the heuristic procedure performs in a best bound tree search.

Columns (1) to (3) of Table 2.1 give the number of rows,  $m$ , number of columns,  $n$ , and density,  $\rho$ , of the SCP. The density is the ratio of non-zero entries to  $m \times n$  the total number of entries in the constraint matrix.

Column (4) gives the structure of the constraint matrix. Type A means that there is a fixed probability  $\rho$  that any element  $a_{i,j} = 1$

Type B has a density that varies uniformly from  $p/2$  in row 1 to  $3p/2$  in row  $m$ .

The cost structure is given in Column (5) and indicates that  $c_j = 1$  for all  $j$ . An  $\times$  indicates that  $c_j = 2 \sum_{i=1}^m a_{ij} + 5$ , but if  $c_j$  exceeds 15 it is reduced by 10. The costs  $c_j$  were not set equal to  $\sum_{i=1}^m a_{ij}$  because then a dual feasible solution is  $u_i = 1$  for all  $i$  and all reduced costs equal 0. For this type of problem if the corresponding SPP is feasible then it gives an optimal solution to the SCP. If it is not feasible then any method which gives a lower bound derived from dual feasible solutions to the DLP will fail. Hence this cost structure was not used.

Column (6) gives the best upper bound known for the problem,  $z_u$ . If  $z_u$  is optimal then it is marked by \*. The bound was found by one of the methods mentioned earlier in this chapter or by tree search.

Column (7) gives the percentage by which the upper bound obtained at the end of Procedure 1 exceeds  $z_u$ . As can be seen it can sometimes be 50% above the best known solution.

Column (8) gives the percentage by which best upper bound at the root node of the search tree exceeds  $z_u$ . The next column gives the method by which it was calculated. PROC3 means that the best upper bound was calculated using heuristics and for small problems this was often optimal. The upper bound was calculated by Procedure 3 and then by using:

- (i)  $c_j/h_j$
- (ii)  $c_j/\log(h_j)$
- (iii)  $c_j/h_j \log(h_j)$

TABLE 2.1 QUALITY OF BOUNDS OBTAINED USING HEURISTIC

Problem No	Size m (1)	n (2)	Density ρ (3)	Type (4)	Costs (5)	Best known Solution $z_u$ (6)	Upper Bound After Proc1 % above (6) (7)	Upper Bound At Root Node % above (6) (8)	Method of Calculating (8) (9)	Lower Bound After Proc1 % of (6) (10)	Lower Bound After Proc3 % of (6) (11)	Number of Iterations to get $z_t$ (12)	Final Gap Between $z_u$ and $z_t$ (13)	Time CPU sec
1	30	60	0.15	A	X	56*	5	0	PROC3	68	88	4	11	0.03
2	30	100	0.15	A	X	50*	24	4	PROC3	73	85	10	14	0.06
3	30	200	0.15	A	X	46*	37	11	C/H	76	76	1	14	0.03
4	30	300	0.15	A	X	44*	59	11	C/H	80	80	1	20	0.04
5	30	400	0.15	A	X	44	43	11	C/H	80	80	1	20	0.05
6	30	500	0.15	A	X	44	43	11	C/H	80	80	1	20	0.06
7	30	600	0.15	A	X	44	43	11	C/H	80	80	1	20	0.07
8	30	700	0.15	A	X	44	43	11	C/H	80	80	1	20	0.08
9	30	800	0.15	A	X	42	50	0	C/H	83	83	1	17	0.09
10	30	900	0.15	A	X	42	50	0	C/H	83	83	1	17	0.09
11	50	500	0.04	A	U	14*	21	7	PROC3	73	84	16	14	0.30
12	50	500	0.15	A	X	76	29	0	C/L(H)	77	77	1	22	0.98
13	60	400	0.11	A	U	4*	0	0	PROC3	82	82	1	0	0.01
14	60	400	0.05	A	U	14	21	0	C/H	72	80	15	14	0.44
15	60	400	0.05	B	U	14	14	7	C/H	72	83	13	14	0.30
16	60	500	0.08	A	X	93	35	0	C/H	75	75	1	25	0.29
17	60	500	0.08	B	X	97	23	5	C/L(H)	72	72	1	29	0.09
18	60	800	0.04	A	U	14	14	0	C/H	73	82	12	14	0.49
19	100	1000	0.02	A	U	29	0	0	PROC3	67	76	17	24	1.22
20	150	800	0.02	A	X	314	8	0.1	C/H	65	85	6	14	0.66
21	150	800	0.02	B	X	355	8	0	C/H	66	88	15	12	1.06
22	150	800	0.02	A	U	35	22	3	C/H	69	80	17	20	2.00
23	160	1000	0.02	A	U	36	6	0	C/H	67	74	16	25	2.65
24	200	1000	0.02	A	U	40	5	0	C/H	64	76	16	22	4.11
25	200	1000	0.02	B	U	42	19	0	C/H	63	70	17	21	4.06
Average										73.6	80.3	7.5	17	

\* Optimal solution

in that order as explained in §2.3. "C/H" in Column (9) means that method (i) above gave the best upper bound and "C/L(H)" means that (ii) gave the best upper bound.

Column (10) gives the lower bound at the end of Procedure 1 as a percentage of  $z_u$ . The lower bound at the end of Procedure 3 as a percentage of  $z_u$  is given in Column (11).

The number of iterations of Procedure 3 is given in Column (12). Lastly Column 13 gives the CPU time required to calculate the bounds excluding data input time. The FTN FORTRAN optimizing compiler was used under the SCOPE 2.1 operating system on the CDC 7600 at the University of London.

From Table 2.1 for the 25 problems tested only 5, which tended to be the smaller ones, had the best upper bound for the root node of the search tree generated by Procedure 3. Thus it is important to use other methods. The number of variables removed by reduced costs was 17 out of 60 for Problem 1, which had 30 rows. However for larger problems no variables were removed. Thus reduced costs are most useful at the nodes in the search tree of a branch and bound procedure that are not near the root. The average ratio of the lower bound at the end of Procedure 1 to the best solution known was 74% and this percentage varied between 63% and 83% for the problems studied. Using Procedure 3 increased this ratio on average to 80%. Problems 2 to 10 were generated by increasing the number of columns by 100 each time. As can be seen the upper bound was the same for Problems 4 to 8 and Problem 4 has less than half the columns of Problem 8. Also the lower bound was the same for Problems 4 to 10. Thus it would be advantageous in a randomly generated problem to calculate the lower bound from a subset of the columns only and then

TABLE 2.2 PERFORMANCE OF HEURISTICS WHEN INCORPORATED IN TREE SEARCH

Problem Number	PROBLEM					Optimal Solution Value (6)	Number of Tree Search Nodes (7)	Number of Variables Removed by reduced Costs at the root of the tree (8)	Time CPU Sec (9)
	Size		Density	Type	Costs				
	M (1)	N (2)	$\rho$ (3)	(4)	(5)				
1	30	60	0.15	A	X	56	20	23	0.11
2	30	100	0.15	A	X	52	48	19	0.37
3	30	200	0.15	A	X	46	502	2	4.99
13	60	400	0.11	A	U	4	1	Not tested*	0.38
26	50	100	0.06	A	X	132	8	57	0.09
27	50	100	0.09	A	X	76	5	74	0.06
28	50	100	0.08	A	U	10	6	-	0.17
29	50	100	0.06	A	U	14	3	67	0.08
30	50	100	0.12	A	U	8	124	-	1.60
31	40	100	0.08	A	X	97	33	2	0.27
32	35	100	0.13	A	X	58	50	-	0.47
33	40	100	0.15	A	X	69	50	-	1.05
34	65	100	0.15	A	X	84	50	-	0.86
35	50	100	0.08	A	X	111	50	-	0.59
36	50	100	0.10	A	X	90	50	-	1.45
37	50	100	0.10	A	U	10	50	-	1.87
38	50	100	0.12	A	X	83	50	-	1.48
39	50	100	0.12	A	U	8	50	-	1.31
40	50	100	0.14	A	U	8	50	-	2.20
41	50	100	0.15	A	X	90	50	-	2.30

\*Not tested because lower bound  $< (\text{upper bound} - 1 + \epsilon)$  and all costs were equal to 1



test the remaining columns to see if the reduced costs were negative. Any columns with negative reduced costs could be added to the problem from which the lower bound is calculated. This would be similar to bringing elements into the basis in the simplest method. Analogous to removing elements from the basis, columns with large positive reduced costs could be disregarded for the purpose of calculating bounds. An initial set of columns could be chosen by picking the first 100, say, and then all other columns for which  $c_j/h_j$  was less than a certain value. It is not necessarily true that denser problems are ~~harder~~ to solve than sparse ones as Problems 13 and 14 show. In this case Problem 13 - with more than twice as many elements as Problem 14 and the same dimensions and cost structure is solved optimally whereas there is a gap of 14% between the best upper and best lower bounds for Problem 14.

Table 2.2 shows how the heuristics performed in a best bound tree search procedure. The branching rules are given in Chapter 7 and are using branching on rows. The first 10 problems in this table could all be solved optimally. The second set of 10 problems could not be.

Columns (1) to (6) give the same information as in Table 2.1. For the problems that could not be solved the best bound after 50 tree search nodes is given. Column (7) gives the number of tree search nodes examined and Column (8) gives the number of variables removed either by reduced costs or by the 'single 1 in a row test'.

The maximum size of problem solved was a unicast problem of size 60 x 400 and density 0.11. Problems with less than 4 1's per column and less than 100 columns were easily solved, often with less than

TABLE 2.3

LOWER BOUND AS PERCENTAGE OF UPPER BOUND

Problem Number	Size			Lower Bound at Root Node as Percentage of Best Upper Bound	Lower Bound after 50 Tree Search Nodes as Percentage of Best Upper Bound	Best Upper Bound Value $z_u$
	$m$	$n$	$t$			
	(1)	(2)	(3)			
32	35	100	0.13	84	98	58
33	40	100	0.15	79	97	69
34	45	100	0.15	77	85	84
35	50	100	0.08	89	95	111
36	50	100	0.10	78	91	90
37	50	100	0.10	78	87	10
38	50	100	0.12	82	94	83
39	50	100	9.12	78	85	8
40	50	100	0.14	71	88	8
41	50	100	0.15	78	85	90
Average				79.4	90.5	

TABLE 2.4

COMPARISON OF LP BOUNDS WITH HEURISTIC BOUNDS AT ROOT NODE

Problem Number	PROBLEM					HEURISTIC LOWER BOUND		LP BOUND
	Size		Density	Type	Costs	Bound as %	Time	Time
	$m$	$n$	$\rho$				CPU Sec	CPU Sec
	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)
42	50	100	0.10	A	X	97	0.14	0.29
43	50	300	0.10	A	X	98	0.08	0.89
44	100	400	0.05	A	X	98	0.21	3.40
45	110	300	0.20	A	X	98	10.3	17.70
46	120	400	0.04	A	X	96	1.3	5.02

50 tree search nodes being examined. In Problems 26 and 27 half the variables were removed by reduced costs and subsequently fixing variables by the 'single 1 in a row' test reduced the problem size further by removing rows.

Table 2.3 shows for the problems that were not solved the lower bound as a percentage of the best solution known at the root of the search tree and the least lower bound at an active node as a percentage of the best solution known after 50 nodes had been searched. The results show that there is least improvement in the bound for the larger denser problems as expected. The average value of the lower bound as a percentage of best known solution at the root of the tree was 79.4% and after 50 iterations 90.5%. A comparison of LP solution values and heuristic values at the root node of the search tree is given in Table 2.4. As can be seen from these results the heuristic gives a good approximation to the LP bound, within 5% in most cases in reasonable time. The LP solution was found by solving DLP using the Land and Powell FORTRAN code [L1]. Further results comparing the APEX III linear programming package with the heuristics on standard test problems are given in Chapter 5.

## CHAPTER 3

### LAGRANGEAN RELAXATION

#### 3.1 Introduction

The Lagrangean relaxation,  $LR(\lambda)$ , of IP [G8] is defined in §1.2 as:

$$LR(\lambda) \left[ \begin{array}{l} L(\lambda) = \min_x L(\lambda, x) = \min_x \left[ c^T x - \lambda^T (Bx - d) \right] \\ \text{subject to} \quad Ax \geq b \\ \quad \quad \quad x \text{ integer} \end{array} \right.$$

A lower bound to  $v(\text{IP})$ , the optimal solution value of an integer program, is given by the optimal solution value,  $L(\lambda)$ , for any  $\lambda \geq 0$ . The best such lower bound is given by  $L(\lambda^*)$  where  $\lambda^*$  is the optimal solution to the problem LR:

$$LR \left[ \begin{array}{l} \max L(\lambda) \\ \lambda \geq 0 \end{array} \right. \quad (3.1)$$

Computing  $\lambda^*$  exactly is not easy and in practice a suboptimal value of  $\lambda$  is often used. An initial value of  $\lambda$  is chosen and this is updated recursively, as described in §3.2, by the formula:

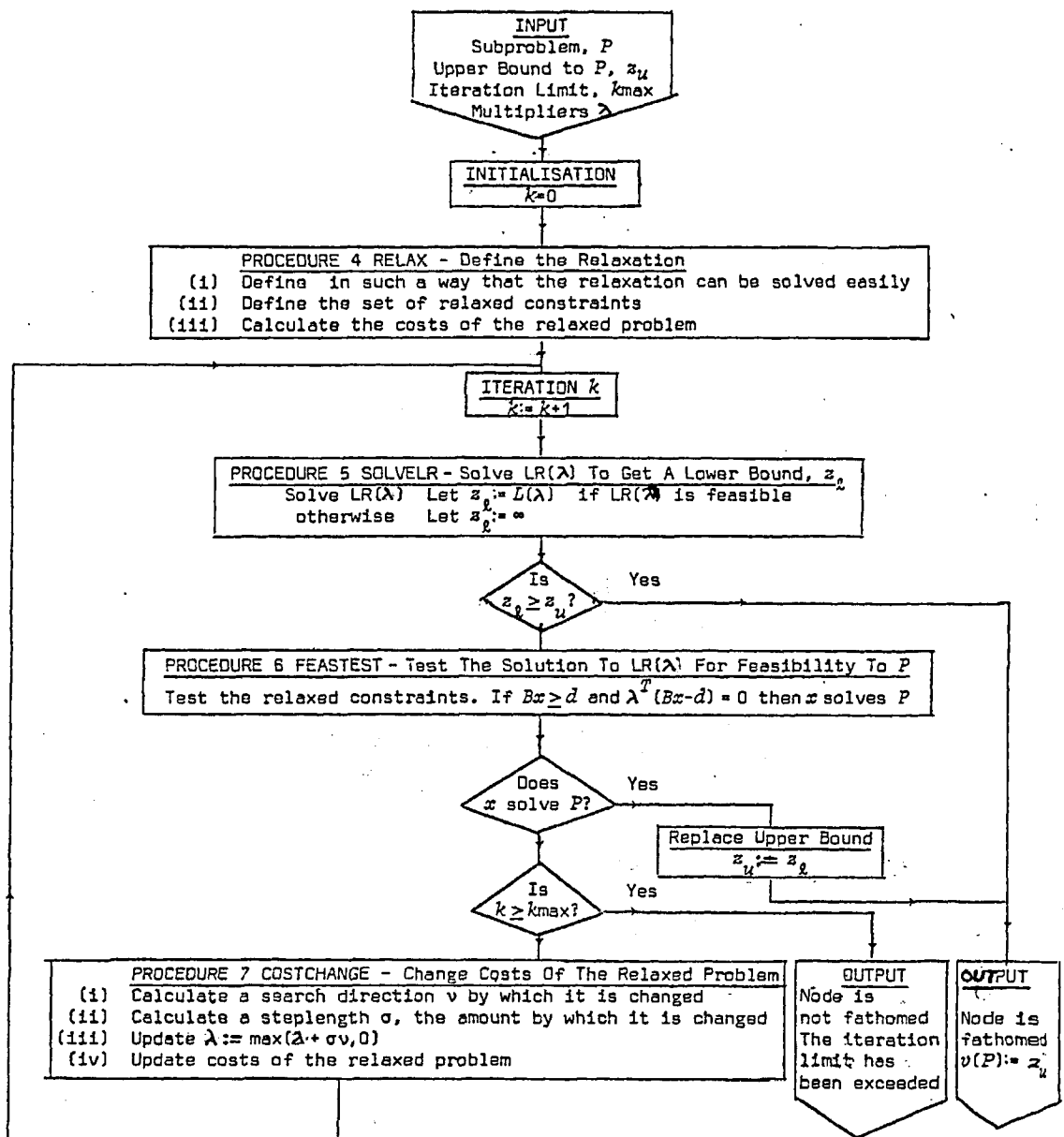
$$\lambda^+ = \max(\lambda + \sigma \nu, 0) \quad (3.2)$$

where  $\sigma$  is a positive scalar steplength and  $\nu$  a search direction.

The computation of these two variables is discussed in §3.3 and §3.4 and results of different methods are considered in §3.5 for a relaxation of the SCP given by Etcheberry [E5].

FIGURE 3.1

A FLOWCHART OF PROCEDURE 8 SUBPROBLEM  
TO SOLVE THE LAGRANGEAN RELAXATION OF A PROBLEM  $P$



### 3.2 Implementation of Lagrangean Relaxation

Initially  $\lambda$  takes the value  $\lambda^0$  which can be obtained by setting  $\lambda^0$  equal to  $u$ , a dual feasible solution of the LP relaxation of IP obtained from the heuristics of Chapter 2. Subgradient optimization is used to obtain  $\lambda^k$  for  $k \geq 1$ . Iteration  $k$ ,  $k \geq 0$ , starts by solving  $LR(\lambda^k)$  and obtaining a solution,  $x^k$  say. If  $Bx^k \geq d$  and  $\lambda^k (Bx^k - d) = 0$  then  $x^k$  is optimal for IP. Otherwise the search direction  $v$  and steplength  $\alpha$  are computed from  $x^k$  and  $\lambda^{k+1}$  is updated by equation (3.2) above. SCP's of up to 30 rows and 100 variables can usually be solved optimally by Lagrangean relaxation without the need for a branch and bound procedure, because the lower bound is in fact the optimal solution.

For most larger problems it is necessary to use Lagrangean relaxation to obtain lower bounds in a branch and bound procedure. To solve a subproblem,  $P$ , of the SCP at a node of a branch and bound tree there are four distinct stages. Firstly the constraints of  $P$  must be partitioned so that the relaxed problem can be solved easily using PROCEDURE 4 RELAXATION. Then PROCEDURE 5 SOLVER is used to solve the problem  $LR(\lambda)$  and its solution is tested for optimality to the IP using PROCEDURE 6 FEASTEST. Either the solution solves IP in which case the node of the search tree is fathomed or it does not. If  $LR(\lambda)$  exceeds the best known optimal solution to the IP or  $LR(\lambda)$  is infeasible the node is fathomed otherwise the fourth stage, PROCEDURE 7 COSTCHANGE, is executed in which the multipliers  $\lambda$  and hence the costs of the relaxed problem are changed. PROCEDURES 5, 6 and 7 together make one iteration of the subgradient optimization phase in an algorithm for the SCP. A flowchart of the subgradient optimization algorithm, described as PROCEDURE 8 SUBPROBLEM below, is given in Figure 3.1.

PROCEDURE 8 SUBPROBLEM ( $P, z_u, z_l, k_{\max}, \lambda$ )

SOLVE A SUBPROBLEM OF THE SCP AT A NODE OF THE BRANCH AND BOUND TREE USING LAGRANGEAN RELAXATION

Input:  $P$  a subproblem of the SCP  
 $\lambda$  (optionally) feasible dual variables for an LP relaxation of  $P$  from which multipliers  $\lambda$  can be defined  
 $z_u$  an upper bound to the SCP  
 $z_l$  a lower bound to the SCP equal to  $\sum_{i=1}^m \lambda_i$   
 $k_{\max}$  maximum number of iterations allowed

Output: A solution to  $P$  or an indication that  $P$  has not been solved.  
 $z_l$  a lower bound to the SCP from Lagrangean relaxation.

1. Initialise Variables

Set the iteration counter  $k := 0$

2. Define the Lagrangean Relaxation

PROCEDURE 4 RELAX Relax the constraints

3. Iteration  $k$

$k := k + 1$  Update the iteration counter

4. Subgradient Optimization

PROCEDURE 5 SOLVELR Solve the Lagrangean relaxation

Set lower bound  $z_l := v(\text{LR}(\lambda))$

If  $z_l \geq z_u$  goto 6.

## PROCEDURE 6 FEATEST

If the solution to  $LR(\lambda), x$ ,  
is feasible for  $P$  and  
 $\lambda^T(Bx - d) = 0$  goto 5.

Test the solution to  $LR(\lambda)$  for  
feasibility to  $P$  and complementary  
slackness conditions.

If  $k \geq k_{\max}$  goto 7.

Test if the iteration limit has  
been exceeded.

Call PROCEDURE 7 COSTCHANGE  
to change the multipliers  $\lambda$   
and the costs of the relaxed  
problem. Goto 3.

5. Replace the Upper Bound to the SCP

Set  $z_u := z_l$

6. P Has Been Solved

Exit with the solution  $z_u$  to  $P$

7. P Has Not Been Solved

The iteration limit has been exceeded. Exit.

PROCEDURE 4 RELAX considers whether or not to represent  $P$  by the data structures used for the original SCP. It is not always necessary to store the constraints  $Bx \geq d$  explicitly as shown in Chapter 4 for the two network flow relaxations, NF1, NF2, and in Chapter 5 for the second graph covering relaxation, GCR2. Data structures are discussed in more detail in Chapter 8. For each relaxation the PROCEDURES SOLVELR, FEATEST and COSTCHANGE are described in Chapters 4 and 5.



### 3.3 Calculating the Search Direction, v

Three ways of choosing the search direction  $v$  which is used to update  $\lambda$  in maximizing  $L(\lambda)$  are described here. If  $LR(\lambda)$  had the same solution  $x$  for all values of  $\lambda$  then methods one and three would be equivalent to the steepest ascent method [B15a].

The first choice of  $v^k$ , the search direction at iteration  $k$  of PROCEDURE 8, is to set:

$$v^{k+1} = d - Bx^k \quad (3.3)$$

This method uses only one solution,  $x^k$ , to  $LR(\lambda^k)$  and is widely used because it is quick to compute.

The second method [C15a] also uses only single solutions  $x^k$  to  $LR(\lambda^k)$  but includes information from previous iterations to compute  $\lambda^k$ . Let the initial search direction be  $v^0$ , with  $v^0 = 0$ . Then to update  $v$  the following recursion is used:

$$v^{k+1} = (d - Bx^k) + \theta^k v^k \quad (3.4)$$

where  $\theta^k$  is a positive scalar. Two methods of choosing  $\theta^k$  were compared. The first was to set  $\theta^k$  equal to a constant between 0 and 1. The second choice of  $\theta^k$  used by Camerini et al [C1] is to set:

$$\theta^k = \frac{-\beta v^k (d - Bx^k)}{\|v^k\|^2} \quad \text{if } v^k \cdot (d - Bx^k) < 0$$

$$= 0 \quad \text{otherwise}$$

where  $\beta$  is a constant,  $0 < \beta < 2$ . If  $\beta = 1$  then  $v^{k+1}$  is orthogonal to  $v^k$  and the method resembles the conjugate direction method for optimizing quadratic functions. In [C1]  $\beta = 1.5$  was found to be a suitable value for the travelling salesman problem, but here no such conclusions for the SCP could be drawn.

Thirdly  $v$  can be chosen by considering more than one solution to  $LR(\lambda^k)$  and then combining the resulting subgradients. Ideally a direction  $\lambda$  that follows the lines of discontinuity of  $L(\lambda)$ , formed by alternative solutions to  $LR(\lambda)$ , should be chosen as it is likely that at  $\lambda^*$   $L(\lambda)$  is non-differentiable. This can be explained by considering the optimality conditions for  $L(\lambda)$  [B17].

Alternative solutions to  $LR(\lambda)$  are denoted by  $x(t)$  where

$Q = Q(\lambda) = \{t \mid L(\lambda) = L(\lambda, x(t))\}$ . Then  $\lambda^*$  is an optimal solution to LR if there exist scalars  $\pi^t$  such that:

$$\sum_{t \in Q} \pi^t = 1 \quad \text{and} \quad \pi^t \geq 0 \quad (3.5)$$

$$\lambda^* \sum_{t \in Q} \pi^t \gamma^t = 0 \quad (3.6)$$

$$\sum_{t \in Q} \pi^t \gamma^t \geq 0 \quad (3.7)$$

$$\text{where } \gamma^t = d - Bx(t) \quad (3.8)$$

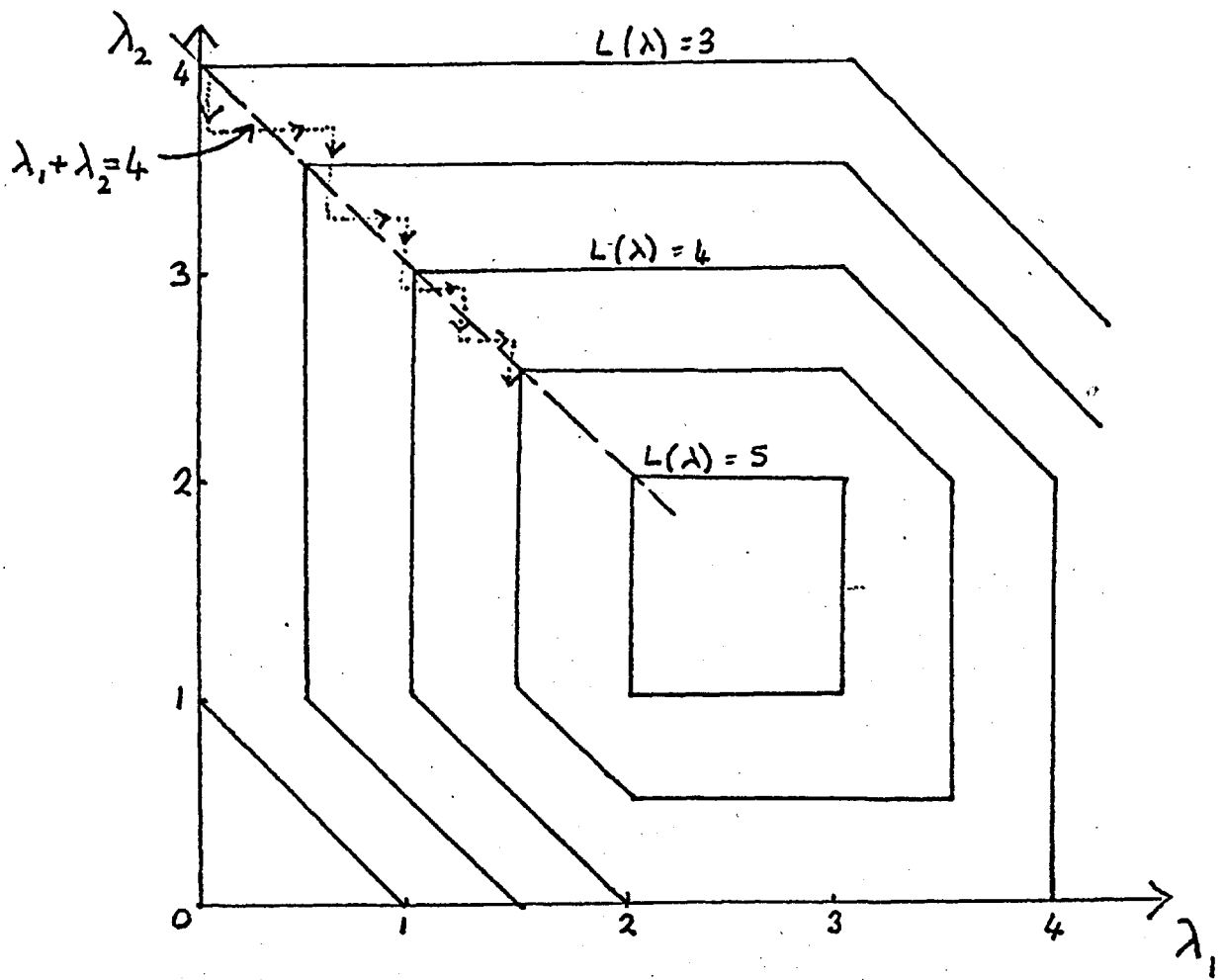
Thus for each non-zero component of  $\lambda^*$  there is a convex combination of subgradients with corresponding components equal to 0. At  $\lambda^*$  very often there is more than one solution to  $LR(\lambda^*)$ .

Example 3,  $LR(\lambda)$ , below is a Lagrangean relaxation of an SCP (denoted SCP3) and contours of the function  $L(\lambda)$  are plotted in Figure 3.2.

$$\text{SCP3} \left[ \begin{array}{ll} \min_x & 3x_1 + 2x_2 + 4x_3 + x_4 + x_5 \\ \text{subject to} & x_1 + x_3 \geq 1 \\ & x_2 + x_3 \geq 1 \\ & x_4 + x_5 \geq 1 \\ & x_1 + x_2 + x_5 \geq 1 \\ & x_1 + x_3 + x_4 \geq 1 \\ & x_j = 0, 1 \quad j = 1, 2, \dots, n \end{array} \right.$$

FIGURE 3.2

Level Sets of  $L(\lambda)$  for Example 3 Showing Zigzagging Path



...> zigzagging path

Relaxing the first two constraints the Lagrangean relaxation,  $LR(\lambda)$ , is:

$$LR(\lambda) \left[ \begin{array}{l} L(\lambda) = \min_x (3-\lambda_1)x_1 + (2-\lambda_2)x_2 + (4-\lambda_1-\lambda_2)x_3 + x_4 + x_5 + \lambda_1 + \lambda_2 \\ \text{subject to} \\ \begin{array}{rcl} & & x_4 + x_5 \geq 1 \\ x_1 + x_2 & & + x_5 \geq 1 \\ x_1 & + x_3 + x_4 & \geq 1 \end{array} \\ \\ x_j = 0,1 \quad j = 1,2,\dots,n \end{array} \right.$$

The optimal solution is given by  $2 \leq \lambda_1^* \leq 3$  and  $1 \leq \lambda_2^* \leq 2$  giving  $L(\lambda^*) = 5$  with  $x^* = (0,0,1,0,1)$ . A line of non-differentiability of  $L(\lambda)$  is given by the line  $\lambda_1 + \lambda_2 = 4$ .

Suppose that an initial vector of multipliers is given by  $\lambda^0 = (0,4)$ ; then using a single subgradient leads to the zigzagging search path shown. An alternative strategy from this starting position that eliminates zigzagging is to search directly along the line  $\lambda_1 + \lambda_2 = 4$ . The discontinuities arise at  $\lambda = (0,4)$  because  $LR(\lambda)$  has three solutions.

$$x(1) = (0,1,0,1,0)$$

$$x(2) = (0,1,1,1,0)$$

$$x(3) = (0,1,1,0,1)$$

The given zigzagging path gives relaxations  $LR(\lambda^k)$  that oscillate between having  $x(1)$  and  $x(2)$  or  $x(3)$  as solutions.

When alternative solutions to  $LR(\lambda)$  are present the computation of  $v$  is better understood by reformulating LR using an additional variable,  $w$ , to give  $P(w)$ :

$$P(w) \left[ \begin{array}{l} \max w \\ w \leq c^T x(t) + \lambda^T (d - Bx(t)) \quad \text{for all } t \in Q(\lambda) \\ \lambda_i \geq 0 \quad i \in R \quad \text{where } R \text{ is the set of relaxed} \\ \quad \quad \quad \text{constraints} \end{array} \right. \quad (3.9)$$

Let the set of relaxed constraints  $R$  equal  $\{1,2,\dots,r\}$  then in the absence of constraints (3.9) an ascent direction for maximizing  $w$  in  $P(w)$  is the vector  $v^0 = (0,0,\dots,0,1)$  where the first  $r$  components correspond to the vector  $\lambda$  and the last component to  $w$ . Projection methods can be used to incorporate constraints (3.9) as was successfully used for the minmax problem in [C3]. The aim is to project  $v^0$  so that  $\lambda$  is changed in such a way that  $w$  increases. Rosen [R1a] suggests how a matrix  $P^\ell$  can be computed iteratively which enables a search direction  $v^\ell = P^\ell v^0$  to be computed. This is described in PROCEDURE 9 below.

#### PROCEDURE 9 PROJECT

##### CALCULATE SEARCH DIRECTION, $v$

##### 1. Initialisation

Calculate the alternative solutions  $x(t)$  to LR( $\lambda$ )

$$\gamma^t = d - Bx(t)$$

To each vector  $\gamma^t$  add a component corresponding to  $w$  equal to  $-1$  to give the vector  $\bar{\gamma}^t$

$\bar{R} = \phi$  be the set of subgradients in the projection

$$\bar{S} = Q(\lambda)$$

$v^0 = (0,0,\dots,0,1)$  be an  $(r+1)$  dimensional vector

$P^0 = I_{r+1}$  be the  $(r+1)$  identity matrix

$\ell = 0$  be the iteration counter

##### 2. Find A Subgradient

If  $\bar{S} = \phi$  then goto 4.

else  $l := l+1$       update iteration counter

Let  $t'$  be the  $t$  that minimizes

$$\frac{\left( v^{l-1} \right)^T \bar{\gamma}^t}{\|v^{l-1}\| \|\bar{\gamma}^t\|}$$

If  $v^{l-1} \bar{\gamma}^{t'} \geq 0$     then goto 4.

### 3. Add $\bar{\gamma}^{t'}$ To The Projection

$$\bar{R} := \bar{R} \cup \{t'\} \quad \text{and} \quad \bar{S} := \bar{S} \setminus \{t'\}$$

$$q := P^{l-1} \bar{\gamma}^{t'}$$

$$q := q / \|q\|$$

$$P^l := P^{l-1} - qq^T$$

$$v^l := P^{l-1} v^0$$

### 4. Check The Signs Of The Lagrange Multipliers Corresponding To $\bar{\gamma}^t$ For $t \in \bar{R}$

Since  $v = \sum \pi^t \bar{\gamma}^t$  for some  $\pi \geq 0$ ,  $\pi^t$  is approximately  $\frac{v^T \bar{\gamma}^t}{\|\bar{\gamma}^t\|^2}$

If  $(v^l)^T \bar{\gamma}^t \geq 0$  for all  $t \in \bar{R}$     goto 6.

### 5. Drop Subgradients From The Projection

If  $(v^l)^T \bar{\gamma}^{t''} < 0$  for some  $t''$  remove  $\bar{\gamma}^{t''}$  from the projection

$$\bar{R} := \bar{R} \setminus \{t''\} \quad \text{and} \quad \bar{S} := \bar{S} \cup \{t''\}$$

Let  $N$  be the matrix with  $\bar{\gamma}^t$  for  $t \in \bar{R}$

$$P^l := I - N(N^T N)^{-1} N^T$$

$$v^l := P v^0$$

Goto 4.

## 6. Search Direction $v$ Is Obtained

If  $\|y^l\| < \epsilon$  then exit  $\lambda^*$  has been found.

else  $v := v^l$  exit with search direction  $v$ .

In practice if, for some relaxed constraint,  $y_i^t$  is equal to a constant,  $\alpha$  say, for all  $t \in Q(\lambda)$  then this component can be dropped from  $\bar{y}^t$  as the corresponding component of  $v$  will also equal  $\alpha$ .

Only components of  $y$  which take different values need be considered thus reducing the size of  $P$ .

The first method of computing  $v$  is quick and effective in the absence of alternative solutions to  $LR(\lambda)$ . The second method requires the storage of an additional vector of dimension at most  $m$  and very few additional computations. The third method requires more computation and storage but enables an ascent direction to be found that gives an increase in  $L(\lambda)$  in the presence of multiple solutions. Section §3.5 gives a computational analysis of these methods.

### 3.4 Calculating the Step size, $\sigma$

#### 3.4.1 Introduction

After computing a direction  $v$  by one of the above three methods a steplength  $\sigma$  must be chosen. In theory [H4a] any stepsize  $\sigma^k$  satisfying

$$\lim_{k \rightarrow \infty} \sum_{k=1}^n \sigma^k = \infty$$

$$\lim_{k \rightarrow \infty} \sigma^k = 0$$

has the property that if  $v^k$  is a subgradient of  $L(\lambda^*)$  then setting  $\lambda^{k+1} := \lambda^k + \sigma^k v^k$  ensures that  $\lambda^k$  converges to an optimal value  $\lambda^*$

as  $k \rightarrow \infty$ . The difficulty is that  $k$  may be very large before an optimal value of  $\lambda$  is found.

One method of computing  $\sigma^k$ , which was used by Held and Karp [H4] for the travelling salesman problem is to set

$$\sigma^k = \alpha(z_u^{LR} - L(\lambda^k)) / \|v^k\|^2 \quad (3.9)$$

where  $\alpha$  is an a priori constant,  $0 < \alpha < 2$ ,  $z_u^{LR}$  is an upper bound on  $L(\lambda^*)$  and  $\|v\|^2 = \sum_{i=1}^m v_i^2$  for any  $m$ -dimensional vector  $v$ .

### 3.4.2 Computing $\sigma^k$ Using a Target Value

In practice using a fixed value of  $z_u^{LR}$  did not lead to a rapid increase in the lower bound  $L(\lambda)$ . Instead using a variable  $z_T$  which can be adjusted depending on the value of  $L(\lambda)$  is preferable. Initially  $z_T$  is set to a value slightly greater than  $L(\lambda^0)$ .

If  $z_\ell$  is much less than  $z_T$  then it is unlikely that the lower bound will increase much and thus  $z_T$  must be reduced. As  $z_T$  is not a true upper bound on  $z_\ell$  it may be exceeded by  $z_\ell$  in which case  $z_T$  must be increased. In practice if  $z_\ell$  exceeds  $z_T - \epsilon$  for a positive constant  $\epsilon$  then  $z_T$  is increased. It is necessary to use  $\epsilon$  for otherwise the algorithm may stop with a suboptimal value of  $\lambda$  satisfying  $L(\lambda) = z_T$ . The adjustments to  $z_T$  were made using an a priori chosen constant  $\delta$ .

The following rules were found by experiment to give suitable values of  $z_T$ :

$$\text{If } z_T > z_\ell + 2\delta \quad \text{set } z_T = z_\ell + 1.5\delta \quad (3.10)$$

$$\text{If } z_T < z_\ell + \delta/40 \quad \text{set } z_T = z_\ell + \delta/10 \quad (3.11)$$

$$\text{If } z_T > z_u \quad \text{set } z_T = z_u + \delta \quad (3.12)$$



where  $z_u$  is an upper bound on  $v(\text{SCP})$ . In the third rule (3.12) using an overestimate of the upper bound usually gives a better increase in the lower bound than if the exact upper bound is used. The initial value of  $z_u$  can also be used to scale  $\delta$ .

### 3.4.3 Computing $\sigma^k$ Using "Near-Alternative" Solutions

The projection method is most effective when  $LR(\lambda)$  has more than one solution. Using  $\lambda^0$  equal to a dual feasible solution of the LP relaxation of IP usually gives a relaxed problem that has several solutions. At subsequent iterations an attempt is made to make  $LR(\lambda + \sigma v)$  have several solutions. This is done by calculating values of  $x, x'$  say, for which  $L(\lambda, x') - L(\lambda)$  is small, where  $L(\lambda, x') = c^T x' - \lambda^T (Bx' - d)$ . Suppose  $x$  is an optimal solution to  $LR(\lambda)$  then the aim is to find by how much one can change  $\lambda$  in the direction of  $v$  without the optimal solution,  $x$ , changing. Suppose that it changes to  $x'$  when  $\sigma$  equals  $\bar{\sigma}$ . Then  $LR(\lambda + \bar{\sigma}v)$  has optimal solutions  $x'$  and  $x$ . Therefore:

$$L(\lambda + \bar{\sigma}v, x) = L(\lambda + \bar{\sigma}v, x')$$

$$\text{or } c^T x - (\lambda + \bar{\sigma}v)^T (Bx - d) = c^T x' - (\lambda + \bar{\sigma}v)^T (Bx' - d)$$

$$\bar{\sigma} v^T (Bx' - Bx) = (c^T - \lambda^T B)(x' - x)$$

thus if  $v^T (Bx' - Bx) \neq 0$

$$\bar{\sigma} = \frac{(c^T - \lambda^T B)(x' - x)}{v^T (Bx' - Bx)} \quad (3.13)$$

otherwise  $\bar{\sigma} = \infty$

Thus one way of computing  $\sigma$  would be to calculate a set  $Q'(\lambda)$  of 'near-alternative' solutions to  $LR(\lambda)$ , that is solutions for which

$L(\lambda, x')$  -  $L(\lambda)$  is small. Each 'near-alternative' solution would give a value of  $\sigma$  by (3.13). These could then be ranked, assuming  $\sigma \geq 0$ , as:

$$\sigma(r_1) \leq \sigma(r_2) \leq \dots$$

where  $r_K$  denotes the  $K^{\text{th}}$  'near-alternative' solution.

In general since  $L(\lambda)$  is concave one expects  $L(\lambda + \sigma v)$  to increase as  $\sigma$  increases from 0 and then decrease. Thus  $\sigma$  should be set equal to the first  $\sigma(r_K)$  for which  $L(\lambda + \sigma(r_K)v) \geq L(\lambda + \sigma(r_{K+1})v)$ . A heuristic for deciding when  $r_K$  has been found would be to find the first 'near-alternative' solution for which:

$$v^T(d - Bx(r_K)) < 0$$

If  $v^T(d - Bx(r_K)) \geq 0$  for all 'near-alternative' solutions then  $\sigma$  could be chosen using the first method suggested. In practice it was found that  $L(\lambda + \sigma v)$  varied considerably with very small changes in  $\sigma$  and the above method tended to give values of  $\sigma$  that were too large. Another reason why this method did not often give any increase in the bound value is that in solving  $LR(\lambda)$  the solution  $x$  may have been an optimal solution to  $LR(\lambda^*)$  even though  $\lambda$  was not an optimal multiplier.

#### 3.4.4 Other Methods of Computing $\sigma^k$

The third method of computing  $\sigma$  was to set  $\sigma$  equal to the minimum of  $\sigma^1$  and  $\sigma^2$  where  $\sigma^1$  is calculated from (3.9) and  $\sigma^2$  is calculated by (3.13). If  $L(\lambda + \sigma v)$  is very much less than  $L(\lambda)$ , say less than  $L(\lambda) - 0.1\delta$  (where  $\delta$  is as before), then  $\sigma$  is halved until  $L(\lambda + \sigma v) \geq L(\lambda) - 0.1\delta$ . Very often this gave a good increase in the lower bound.

FIGURE 3.3

Non-Zero Indices Of  $x$  For Example, To Show Behaviour Near Subgradient Optimum

ITERATION NUMBER $k$	BOUND VALUE $L(u)$	Index of $x, j$																														
		1	2	3	4	5	6	8	9	11	12	13	14	15	16	17	19	20	21	22	23	26	27	28	29	30	31	35	36	37	41	
2	35.93		o	o			o				o	o				o						o		o	o		o	o	o		o	
3	42.93		o				o				o	o				o							o		o	o		o		o	o	
4	46.54	o		o	o			o	o	o	o	o	o	o	o	o	o	o	o	o	o			o	o	o						
5	48.50				o						o			o								o										
6	48.98										o			o								o				o					o	
7	49.05						o				o											o		o		o	o				o	
31	48.82				o						o	o				o						o		o	o			o		o	o	
32	49.57				o																		o		o	o						
33	50.04										o	o		o								o			o						o	
34	50.06			o																		o	o			o					o	
35	50.12										o			o																	o	
36	50.14				o																	o	o	o		o					o	o
37	50.27										o			o								o		o								
38	50.32													o								o	o								o	

o means  $x_j = 1$  in solution to  $LR(\lambda)$  at iteration  $k$ .

LP solution = 51.0

A fourth alternative which was not tested would have been to use either exactly or approximately a cubic linesearch as in [C1].

### 3.5 Computational Results

#### 3.5.1 Case Study

Firstly Etcheberry's relaxation will be used to examine in detail how the different ascent methods performed on one particular problem. This relaxation relaxes constraints of the SCP until there is at most 1 non-zero entry per column for the constraints of  $LR(\lambda)$ . This problem had 30 rows, 60 columns and density 0.15. It was randomly generated with a fixed probability of 0.15 that  $a_{ij}$  was equal to 1. The costs  $c_j$  were set equal to  $2 \sum_{i=1}^m a_{ij} + 5$ , but if  $c_j$  exceeded 15 it was reduced by 10.

The zigzagging between solutions to  $LR(\lambda^k)$  for successive iterations as  $\lambda^k$  approaches  $\lambda^*$  is illustrated in Figure 3.3. It shows indices  $j$  for which  $x_j = 1$  for iterations 2 to 7 and 31 to 38 of the subgradient optimization. Iterations 2 to 7 were chosen as the bound value was least for these. It is noted that there are 30 different values of  $j$  for which  $x_j = 1$  and also that if  $x_j = 1$  for more than one iteration these iterations are likely to be consecutive. For example  $x_{17} = 1$  in iterations 2, 3 and 4 and  $x_{15} = 1$  in iterations 4, 5 and 6. For iteration 38 the bound value was greatest and in the 7 preceding iterations it is seen that there are 15 different values of  $j$  for which  $x_j = 1$ . Further if  $x_j = 1$  at one iteration it may be alternately equal to 0 and 1 at subsequent iterations. For example  $x_{12} = 1$  only at iterations 31, 33, 35 and 37 and  $x_{26} = 1$  at iterations 31, 34, 36 and 38. This illustrates the zigzagging between solutions that is

TABLE 3.4

TO SHOW THE EFFECT OF VARYING  $\beta$  WHEN IMPLEMENTING THE SUBGRADIENT ASCENT PROCEDURE OF CAMERINI et al

$\beta$	$\delta = 1.5$		$\delta = 2.0$		$\delta = 2.5$		$\delta = 3.0$	
	ITERATION NUMBER	BOUND VALUE	ITERATION NUMBER	BOUND VALUE	ITERATION NUMBER	BOUND VALUE	ITERATION NUMBER	BOUND VALUE
1.5	25	49.66	25	49.71	25	49.43	25	48.76
	47	<span style="border: 1px solid black;">50.20</span>	36	50.01	38	50.05	50	49.99
2.0	25	49.44	25	49.14	25	47.85	25	47.09
	38	50.00	50	<span style="border: 1px solid black;">50.26</span>	40	49.75	50	<span style="border: 1px solid black;">50.02</span>
2.5	25	49.48	25	49.09	25	48.19	25	47.01
	50	50.06	48	50.20	50	<span style="border: 1px solid black;">50.20</span>	50	49.98

gives best bound value for given  $\delta$ .

TABLE 3.5  
BOUND VALUES FOR 3 SUBGRADIENT OPTIMIZATION METHODS

ITERATION NUMBER	BOUND VALUE		
	SINGLE SUBGRADIENT (i)	CAMERINI et al (ii)	#PROJECTION METHOD (iii)
1	49.27	49.27	49.27
2	35.93	35.93	42.96*
3	44.91	33.74	48.55
4	47.38	34.68	47.27
5	47.44	39.27	47.55
6	47.15	40.14	42.79
7	47.62	41.00	47.96
8	47.72	41.70	47.61
9	46.65	42.43	47.40
10	49.04	43.20	48.52
11	48.44	43.96	48.27
12	49.33	44.72	47.66
13	49.07	45.45	48.22
14	49.39	45.83	47.37
15	49.65	45.93	48.54
16	49.81	46.36	48.64
17	49.97	46.42	48.45
18	38.09	46.84	48.92
19	45.30	46.91	48.30
20	48.07	47.33	49.08
21	46.72	47.39	43.43*
22	48.32	47.72	48.03
23	48.50	47.83	48.55
24	48.05	47.87	47.83
25	48.12	48.19	48.30
26	48.65	48.26	47.74
27	48.53	48.33	47.31
28	48.74	48.64	48.35
29	47.47	48.76	48.40
30	48.40	48.81	49.00
31	48.52	49.00	49.30
32	47.94	49.19	49.58*
33	48.86	49.26	49.71*
34	48.25	49.40	50.20*
35	49.39	49.52	50.20*
36	48.76	49.60	50.20*
37	48.99	49.67	50.20*
38	48.55	49.71	50.20*
39	48.78	49.73	50.22*
40	49.24	49.76	50.22*
41	48.80	49.78	50.22*
42	49.31	49.88	50.22*
43	49.16	49.95	50.24*
44	49.63	49.98	50.25*
45	49.20	50.01	50.25*
46	48.92	50.11	50.20*
47	48.99	50.13	49.72*
48	48.62	50.13	49.34
49	49.19	50.15	49.45
50	49.30	50.20	49.30

Steplength Parameter

$$\delta = 2.5$$

Problem

30 x 60 density 0.15

The starting value of  $\lambda$  was obtained using the heuristics of Chapter 2.

#For method (iii) it was too expensive to use the projection method at each iteration of sub-gradient optimization. Therefore it was only used at iterations marked \*. The other iterations used method (ii).

\* means that bound has been obtained using projection method

characteristic of subgradient optimization near an optimal solution to  $LR$ . In other problems it is often found that the values of  $j$  for which  $x_j = 1$  for values of  $z_\ell$  near the best bound obtainable from this Lagrangean relaxation are precisely the values of  $j$  for which  $x_j$  is non-zero in an optimal solution to the LP relaxation. This is because the value of an optimal LP solution is equal to the best bound obtainable from this relaxation and any optimal LP solution can be written as a convex combination of all the possible 0-1 solutions to the best Lagrangean relaxation,  $LR(\lambda^*)$ . The frequency with which variables occur as solutions to a Lagrangean relaxation can be used to determine branching variables in a tree search.

Of the different methods of calculating  $v$  first Camerini et al's method [C1] is examined. Although the choice of  $\beta$  used in (3.6) affected the value of the lower bound no firm conclusions as to the best value of  $\beta$  could be obtained. Table 3.4 shows the best value of the lower bound in the first 50 iterations and the iteration at which it occurred for different values of  $\beta$  and  $\delta$ . It also gives the bound value after 25 iterations. The best bound value of 50.26 with this method was given by  $\delta = 2.0$  and  $\beta = 2.0$  at iteration 50. The LP solution gives a bound value of 51.0 and thus is the best theoretically obtainable lower bound value for this relaxation. The bound values were also calculated using a fixed value of  $\theta$  in (3.5) and were slightly worse than those derived by varying  $\theta$  as in (3.6). This was therefore not analysed further.

The bound values,  $L(\lambda)$ , for the three methods of computing the ascent direction  $v$  for maximising  $LR(\lambda)$  are given in Table 3.5 for  $\delta = 2.5$ . They are plotted against iteration number for the first 50 iterations in Table 3.6. The ascent pattern was found to be similar for other values of  $\delta$ . The simple subgradient procedure produced a very erratic

variation in the bound. The method of Camerini et al ascended slowly at first, but then was very steady and reached a higher bound value than that obtained by using a single subgradient. The computational effort of Camerini et al's method was slightly greater than that required by the simple method. The extra storage required was one vector of minimum dimension equal to the number of relaxed constraints, MREL, and in any case less than  $m$ . The projection method required more storage,  $O((MREL)^2)$ , and as it was more expensive to use than the other two methods was only used when the bound exceeded its previously best known value. Iterations for which it was used are marked by \* in Table 3.5. At the other iterations Camerini et al's method was used. The main problem was that the search direction generated was extremely sensitive to small changes in the stepsize  $\sigma$  and for the results recorded here the stepsize was calculated for all iterations by the method used by Held and Karp given in (3.10) as this was then the same for all three ascent methods. The projection method had the feature that it reached a relatively high value of the bound at an earlier iteration than by the other two methods. Since the simple subgradient method was used in (iii) for the ascent direction when the projection method was not used, i.e., when the bound had not reached a better value than in previous iterations, the bound value was erratic for these iterations.

Thus the projection method and Camerini et al's method were combined to produce further improvements. The computation times for the three methods plotted here are:

- (i) 0.75
- (ii) 0.88
- (iii) 1.34

where the times are CDC 6500 sec. under the NOS BE operating system and the MNF5 Fortran compiler at Imperial College.



TABLE 3.6

BEST BOUND VALUES AND TREE SEARCH INFORMATION

METHOD	ROOT NODE			TREE SEARCH		
	BEST BOUND VALUE	ITERATION NUMBER	TIME CPU sec	NUMBER OF NODES	NUMBER OF SUBGRADIENT ITERATIONS	TOTAL TIME CPU sec
(i) Single subgradient	50.38	150	1.63	8	364	3.2
(iia) Camerini et al $\beta = 1.5$	50.82	144	1.66	9	647	5.3
(iib) $\beta = 2.0$	50.80	123	1.56	10	472	4.3
(iii) Projection method	50.70	130	3.17	6	415	12.7
(iva) $\theta$ fixed at 0.25	50.53	112	1.31	7	516	4.2
(ivb) $\theta$ fixed at 0.5	50.82	116	1.33	10	697	5.4
(ivc) $\theta$ fixed at 0.75	50.81	148	1.61	13	1088	8.3

Time CDC 6500  
MNF5 compiler

TABLE 3.7

BEST BOUND VALUES AT ROOT NODE FOR 4 METHODS

ITERATION NUMBER	(i) Single Subgradient		(iia) Camerini et al $\beta = 1.5$		(iii) Projection Method		(ivb) Fixed $\theta = 0.5$	
	Bound	Time	Bound	Time	Bound	Time	Bound	Time
10	49.03	0.5	49.64	0.5	49.81	0.6	49.33	0.5
20	49.90	0.6	50.24	0.6	50.12	0.9	50.08	0.6
30	50.24	0.7	50.27	0.7	50.27	0.9	50.25	0.7
40			50.38	0.8	50.32	1.1		
50					50.38	1.3	50.40	0.8
60	50.26	0.9	50.53	1.0	50.40	1.4		
70	50.29	1.0						
80	50.30	1.1	50.55	1.1	50.48	1.7	50.62	1.1
90	50.33	1.1	50.68	1.2	50.55	1.9	50.67	1.1
100	50.34	1.2						
110	50.35	1.3					50.75	1.3
120	50.37	1.4			50.68	2.4	50.82	1.3
130					50.70	3.1		
140	50.38	1.6						
150	50.38	1.6	50.82	1.7				

Time CDC 6500  
MNF5 compiler

FIGURE 3.8

Comparison of Bound Values Against Iteration Number for 3 Different Subgradient Optimization Methods

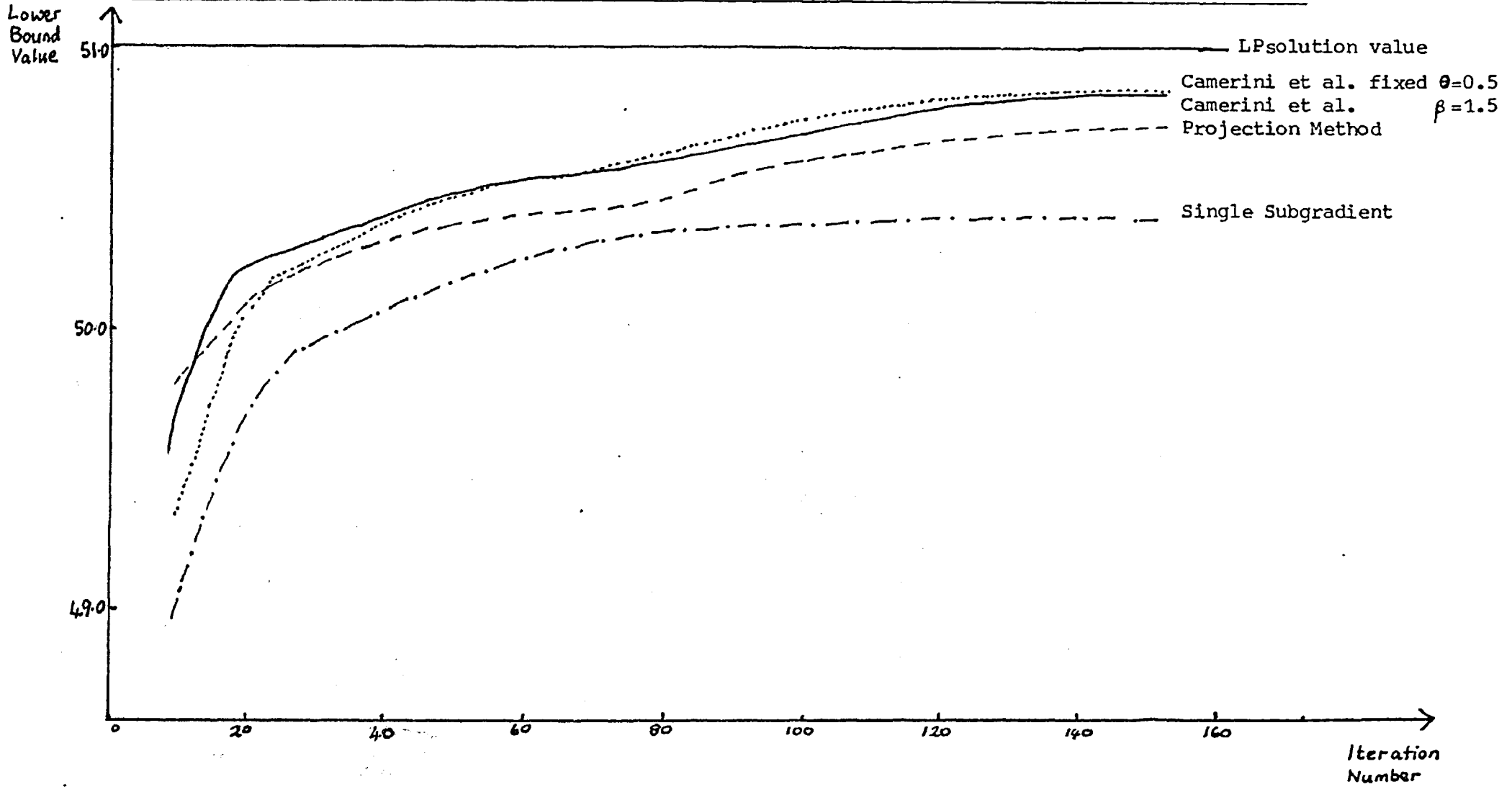
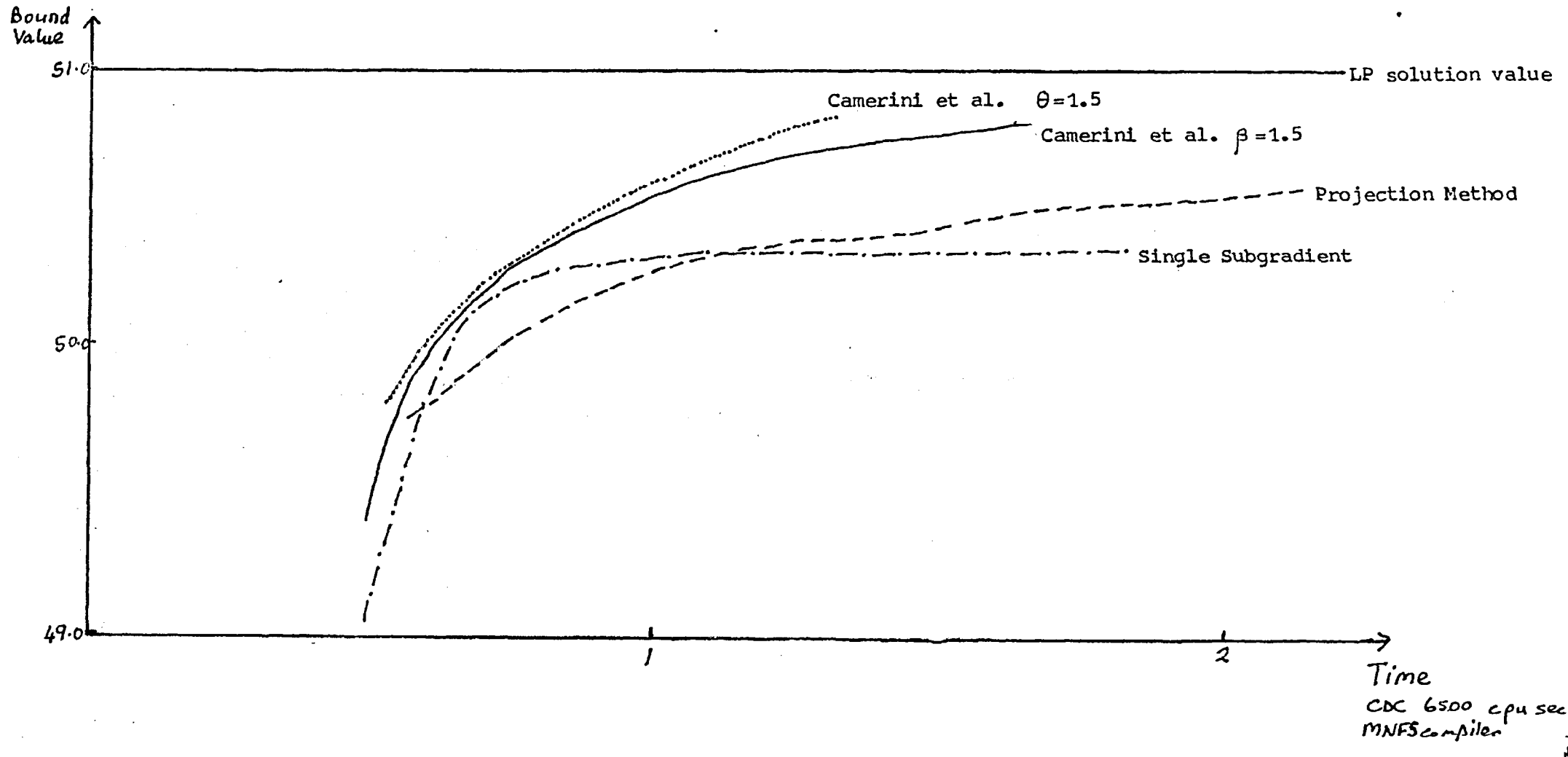


FIGURE 3.9

Comparison Of Bound Values Against Computing Time For 3 Different Subgradient Optimization Methods



As the bound value was still increasing at 50 iterations the iteration limit was increased to 150 iterations, beyond which no improvement was obtained. With this new limit and  $\delta = 1.5$  the three methods were compared in a tree search procedure.

For method (ii) two tests (iia) with  $\beta = 1.5$  and (iib)  $\beta = 2.0$  were made. The projection method used method (ii) at iterations where the bound had not increased and a linesearch based on all the methods described in (3.3). In addition using  $\theta$  fixed at 0.25, 0.50 and 0.75 gave tests (iva), (ivb) and (ivc). The best bound values at the root node of the search tree, the iteration number at which it was reached and the time is shown in Table 3.7 together with total computation times and number of tree search nodes used.

Table 3.7 gives the best bound values at the root node for methods (i), (iia), (iii) and (ivb) and the results are plotted against iteration number in Figure 3.8 and against computing time in Figure 3.9. The subgradient optimization gave at best a 3% improvement in the bound at the end of the heuristic procedures.

There was little difference at the root node between methods (ii) and (ivb) in terms of bound value or time. The single subgradient method gave a worse bound than the other methods at the node, but for this size problem had the fastest overall computing time.

### 3.5.2 Comparison of the Methods on Different Problems

Table 3.10 gives 5 problems which were solved by the above methods using the same best bound tree search as in Chapter 2. The first 4 columns describe the problems as in Chapter 2. The first column for each method gives the increase in the bound obtained from Lagrangean relaxation over that obtained by the heuristics as a

TABLE 3.10

COMPARISON OF 4 METHODS FOR SUBGRADIENT OPTIMIZATION EMBEDDED IN A TREE SEARCH

PROBLEM					METHOD											
$\delta = 1.5$ For All Problems					SINGLE SUBGRADIENT (i)			CAMERINI et al $\beta = 1.5$ (iia)			PROJECTION METHOD (iii)			FIXED $\theta = 0.5$ (ivb)		
No	m	n	p	Cost Type	% Increase in Bound	No of Nodes	Total Time	% Increase in Bound	No of Nodes	Total Time	% Increase in Bound	No of Nodes	Total Time	% Increase in Bound	No of Nodes	Total Time
47	20	80	0.2	X	5	3	1.13	7	3	1.15	7	1	2.13	1	8	1.35
48	30	80	0.15	X	3	11	6.74	4	16	8.8	3	10	16.6	1	111	33.6
2	30	100	0.15	X	3	37	15.83	0	22	14.05	3	38	23.0	1	750	21.
49	36	80	0.17	X	2	13	4.83	3	16	4.66	1	13	8.3	0	72	21.53
50	40	300	0.15	X	0	4	16.5	5	2	22.0	7	4	60.6	3	10	26.0

percentage of the heuristic bound. As can be seen method (ii) was the most robust and on further larger problems not listed here tended to generate fewer nodes than by (i). However for the size of problem tested here there is little to choose between methods (i) and (ii). Although there was little difference between (ii) and (iv) for the example problem, the latter generated many more tree search nodes and is therefore not recommended. The projection method whilst not generating too many nodes was costly to implement at each node and there were storage problems on larger examples. The advantage of using method (ii) to overcome the zigzagging is demonstrated here as is the importance of using a good target value by adjusting  $\delta$ . Thus using method (ii) with either a robust way of choosing  $\delta$  or a line-search procedure leads to an effective implementation of Lagrangean relaxation.

## CHAPTER 4

### NETWORK FLOW RELAXATIONS OF THE SCP

#### 4.1 Introduction

Network flow problems can be solved using conventional linear programming methods, but by exploiting their structure more efficient algorithms result which easily solve problems of up to 500,000 arcs and 1000's of vertices [B12, G15]. Two different network flow relaxations of the SCP are described in §§4.2 and 4.3 and in both cases the lower bound obtained from the network flow solution is bounded above by  $v(LP)$ . The conclusions of Section §4.4, where computational results are presented, are that the first relaxation produces a bound very close to the LP bound in a reasonable time but that the second relaxation requires too much storage to be useful. Unlike the integer programming problems of [G15] the SCP has its own structure which can be more efficiently exploited than the structure of the derived network flow problems.

#### 4.2 Network Flow Relaxation, NF1

##### 4.2.1 Formulation

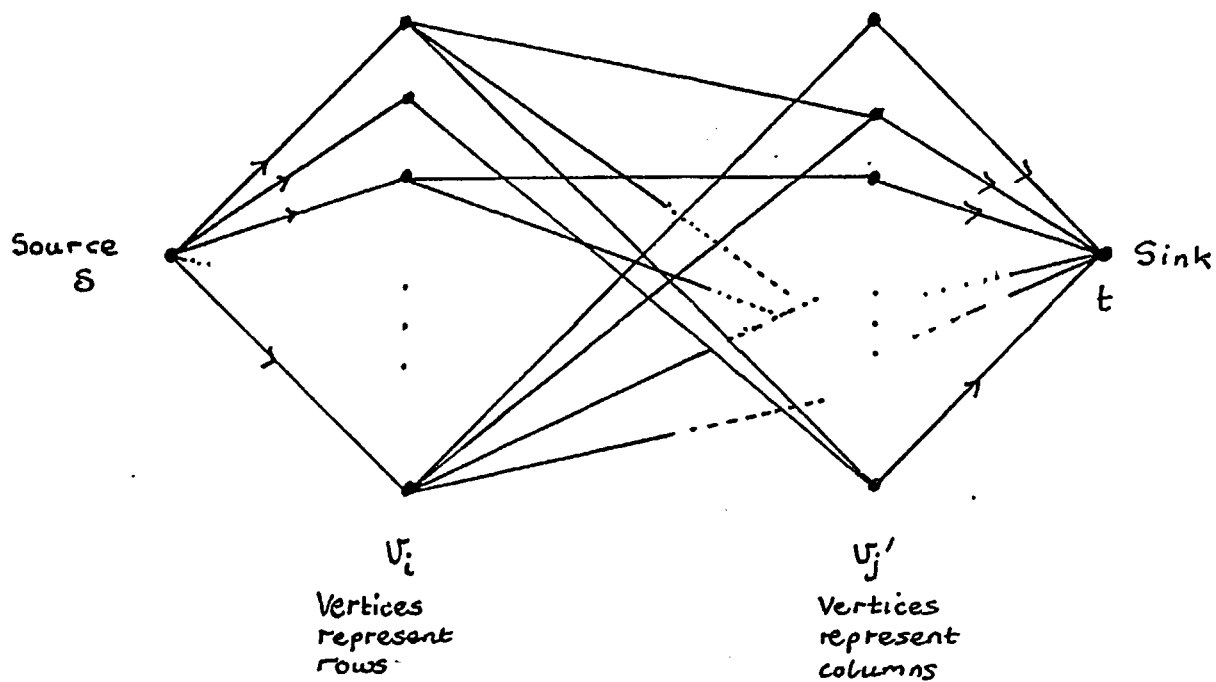
Network flow relaxation NF1, is derived from the LP relaxation of the SCP by replacing a single variable  $x_j$  with a set of variable  $\xi_{ij}$  whenever  $a_{ij} = 1$ . For each  $j$  there are constraints that the variables  $\xi_{ij}$  must take equal values for all  $i \in M_j$ . This gives the problem  $N(f)$  which is clearly equivalent to the problem LP:





FIGURE 4.1

Network Flow Graph,  $G_1$ , For Example NF1



Arcs	$(s, v_i)$	$(v_i, v'_j)$	$(v'_j, t)$
Upper Bounds	$ N_i $	1	$n_j =  M_j $
Lower Bounds	1	0	0
Costs	0	$d_{ij}$	0

connected to a sink vertex  $t$ . Arcs  $(s, v_i)$  have a lower bound of 1 on the flow. There is an arc  $(v_i, v'_j)$  if  $a_{ij} = 1$  with cost  $d_{ij}$  in which the flow must be between 0 and 1. All other arcs have zero cost.

A minimum cost flow in this network can be found by setting  $\xi_{ij} = 1$  for all arcs satisfying  $d_{ij} = \min_{j \in N_i} d_{ij}$  and then setting  $\xi_{ij} = 1$  for any remaining arcs for which  $d_{ij}$  is negative.

Problem NF1( $d$ ) always has an integer solution. This is feasible (and thus optimal) for the SCP providing constraints (4.2) are satisfied.

#### 4.2.2 Changing the Costs $d_{ij}$ on the Network G1

If the solution is not feasible for the SCP then the costs  $d_{ij}$  must be changed. In the last chapter the multipliers  $\lambda_{ij}$  were changed using subgradient optimization. Here it is more efficient to change  $d_{ij}$  directly. From (4.4) it follows that changing  $\lambda_{ij}$  by  $\gamma_{ij}$  such that  $\sum_{i \in M_j} \gamma_{ij} = 0$  is equivalent to changing  $\lambda'_{ij}$  by another variable,  $\pi_{ij}$  say, such that  $\sum_{i \in M_j} \pi_{ij} = 0$ . By (4.5) this is equivalent to changing  $d_{ij}$  directly. At iteration  $k \geq 0$  let  $d = d^k$  and  $\pi = \pi^k$ , then

$$d^{k+1} = d^k + \pi^k \quad (4.5)$$

Initially  $d^0_{ij}$  can be set to  $u_i + s_j/h_j$  where the variables  $u_i$  are dual variables computed as in Chapter 2 and  $s_j$  are associated reduced costs. It is clear that  $\sum_{i \in N_j} d^0_{ij} = c_j$  and  $\pi^0$  can be set to 0.

The algorithm aims to find  $d^*$  where:

$$v(\text{NF1}(d^*)) = \max_d [v(\text{NF1}(d)) \mid \sum_{i \in M_j} d_{ij} = c_j] \quad (4.6)$$

Then  $v(\text{NF1}(d^*)) = v(\text{LP})$  and  $d_{ij}^*$  can be obtained from an optimal solution  $u^*$  to DLP by setting

$$d_{ij}^* = u_i^* + s_j^*/h_j$$

where  $s_j^*$  are the corresponding reduced costs.

To increase the bound  $v(\text{NF1}(d^k))$  using subgradient optimization at each iteration the set of penalties  $\pi$  is computed as follows.

Let  $J'$  be the index set of columns  $j$  for which equation (4.2) is not satisfied. If  $J'$  is empty (i.e. equation (4.2) is satisfied for all values  $j$ ) then the solution to  $\text{NF1}(d)$  is feasible and therefore optimal for SCP. Assuming  $J'$  is not empty then for every  $j \in J'$  let  $p_j$  be the number of  $\xi_{ij}$  which are equal to 1 and  $\bar{p}_j$  be the number of  $\xi_{ij}$  which are equal to 0. The penalties  $\pi_{ij}$  are then calculated as:

$$\text{For } j \in J' \quad \pi_{ij} = \frac{\alpha \bar{p}_j (z_u^F - z_l^F)}{h_j ||w||^2} \quad \text{if } \xi_{ij} = 1 \quad (4.7)$$

$$\pi_{ij} = \frac{-\alpha p_j (z_u^F - z_l^F)}{h_j ||w||^2} \quad \text{if } \xi_{ij} = 0$$

$$\text{For } j \notin J' \quad \pi_{ij} = 0$$

where  $||w||^2 = \sum_{i,j} (\xi_{ij} - p_j/h_j)^2 = \sum_j (p_j \bar{p}_j / h_j^2)$ , by definition of  $p_j$ ,  $0 < \alpha < 2$  is an a priori chosen constraint and  $z_u^F$  and  $z_l^F$  are upper and lower bounds on  $v(\text{NF1}(d^*))$ .

The penalties  $\pi_{ij}^k$  calculated at iteration  $k$  from the solution to  $\text{NF1}(d^k)$  are used to derive  $d^{k+1}$  using equation (4.5). The iterations continue until either a solution  $\xi$  is found which satisfies constraints (4.2) giving an optimal solution to LP (which incidentally happens to be integer) or the maximum number of permitted iterations is reached.

This describes the procedures RELAX, SOLVELR, FEATEST and COSCHANGE<sup>T</sup> for relaxation NF1.

#### 4.2.3 Further Improvements

Further improvements can be made. Firstly resetting any negative cost  $d_{pq}$  to a non-negative value cannot decrease the bound. In many cases the bound may actually improve after all the negative costs have been removed. This is given in Lemma 4.2.3.1 below:

##### 4.2.3.1 Lemma

If  $d_{pq} < 0$  for some  $p, q$  then  $d_{pq}$  can be reset to a non-negative value without decreasing the bound.

It is sufficient to consider a single cost  $d_{pq} < 0$  and set this to a non-negative value. The procedure can then be repeated for all negative costs until  $d_{pq} \geq 0$  for all  $p, q$ .

##### Proof

Suppose  $d_{pq} < 0$  for some  $p, q$  then since  $\sum_{i=1}^m d_{iq} = c_q$  and  $c_q$  is assumed positive there exists  $d_{p'q}$  say for which  $d_{p'q} > 0$ .

Setting  $d'_{ij} = d_{ij}$  for  $(i, j) \neq (p, q)$  or  $(p', q)$

and  $d'_{pq} = \min[d_{pq} + d_{p'q}, 0]$

$$d'_{p'q} = \max[d_{p'q} + d_{pq}, 0]$$

means  $v(\text{NF1}(d')) \geq v(\text{NF1}(d))$ .

It follows that if  $\xi$  is an optimal solution to  $\text{NF1}(d)$  then there is an optimal solution  $\xi'$  to  $\text{NF1}(d')$  such that in only one component  $\xi'$

differs from  $\xi, \xi_{p'q}$ . The change in bound value is then  $v(\text{NF1}(d')) - v(\text{NF1}(d))$  which equals:

$$d'_{pq} - d_{pq} + d'_{p'q} \xi'_{p'q} - d_{p'q} \xi_{p'q}$$

If  $\xi_{p'q} = 1$  then since  $d'_{p'q} < d_{p'q}$   $\xi'_{p'q} = 1$  and since the sum of the changed costs remains unchanged the change in bound is equal to 0.

If  $\xi_{p'q} = 0$  then since  $d_{pq} < 0$  the change in bound is positive and therefore the bound increases. Hence  $d_{pq}$  can be increased without decreasing the bound. This process which is finite is repeated until all the costs in a column are non-negative.  $\square$

A second improvement that can be made is to reduce a priori the number of variables in the SCP using reduced costs,  $\bar{c}_{ij}$ . These are given by:

$$\bar{c}_{ij} = d_{ij} - \max[0, \min_{\ell \in N_i} d_{i\ell}]$$

Then, if  $\bar{c}_{ij} \geq z_u - v(\text{NF1}(d))$  this implies that  $\xi_{ij} = 0$  and hence  $x_j = 0$  in any optimal solution to the SCP of value less than  $z_u$ , where  $z_u$  is an upper bound on  $v(\text{SCP})$ .

Thirdly, a problem arises in solving  $\text{NF1}(d)$  when there is more than one variable  $\xi_{ij}$  for a given value of  $i$  that can be set equal to 1. This can arise when there is more than one value of  $j(i)$  satisfying  $d_{ij(i)} = \min_{j \in N_i} d_{ij}$ , or it can happen when  $d_{ij} = 0$ . In both of these cases the following strategy which first fixes those variables that can be chosen uniquely and then fixes the remainder, can be used.

First set  $\xi_{ij} = 1$  if  $d_{ij} < 0$ ,

then set  $\xi_{ij} = 1$  if there is a unique  $j$  such that  $d_{ij} = \min_{\ell \in N_i} [d_{i\ell}]$

then set  $\xi_{ij} = 1$  where column  $j$  has the highest proportion of

variables  $\xi_{kj}$  already set equal to 1 from  $j \in J(i)$  where

$J(i) = \{j | d_{ij} = \min_{l \in M_i} d_{il}\}$ . The problem of multiple solutions in

NF( $d$ ) corresponds to degeneracy in LP.

Fourthly, attempting to satisfy the LP optimality conditions can improve the bound. This is done by assuming all costs  $d_{ij}$  are non-negative and setting  $u_i = \min_{j \in N_i} d_{ij}$ . Clearly this gives a feasible solution to DLP from which reduced costs  $s_j$  can be calculated.

To derive an optimal solution to LP from DLP it is first necessary to consider the set of columns,  $S$ , for which the reduced costs are zero. Thus  $S = \{j | s_j = 0\}$ . If  $u_i$  represents an optimal solution to DLP then there is a vector  $\mu$  with  $1 \geq \mu_j \geq 0$  and  $u_i (\sum_{j \in S} \mu_j a_{ij} - 1) = 0$ . The vector  $\mu$  can be found by iterations similar to phase 1 of the Simplex Method [H6]. If there is no  $\mu$  satisfying these constraints then  $u$  is not optimal for DLP. If  $\mu$  is found, then setting  $x_j = \mu_j$ ,  $j \in S$  and  $x_j = 0$  otherwise, gives a solution to the LP. From the formulation of NF1 it follows that  $v(\text{NF1}(d)) \leq v(\text{LP})$ .

Lastly, a feasible solution,  $x$ , to the SCP can be obtained from a solution,  $\xi$ , to NF1 by letting  $x_j = 1$  whenever there is an  $i$  for which  $\xi_{ij} = 1$  and then reducing this to a prime cover.

#### 4.2.4 Summary of the Algorithm

An example is given in Appendix 3 and the algorithm is summarised in PROCEDURE 10 NETFLO1 below:

PROCEDURE 10 NETFLO1 (SCP,  $z_u, z_l, x, u, \xi$ )

SOLVE RELAXATION NF1 OF THE SCP

Input:	SCP	Set covering problem
	$u$	Dual feasible solution
	$z_u$	Upper bound to the SCP
Output:	$u$	Dual feasible solution to the SCP
	$\xi$	Network flow solution
	$z_u, z_l$	Upper and lower bounds to the SCP
	$x$	Feasible solution to the SCP

1. Initialise Variables

$k_{max}$	Set iteration limit
$k := 0$	Set iteration counter
BIG	BIG is a large number

2. RELAX . Define the Relaxation.

For  $j = 1$  to  $n$

$h_j := \sum_{i=1}^m a_{ij}$  Calculate column sums and

$s_j := c_j - \sum_{i=1}^m a_{ij} u_i$  reduced costs

For  $j = 1$  to  $n$

For  $i \in M_j$

$d_{ij} := u_i + s_j/h_j$  Calculate cost of arc  $(i,j)$  for NF1(d)

3. SOLVERR. Solve the Relaxation at Iteration  $k$

$k := k + 1$

If  $k > k_{max}$  then goto 8.

$z_l := 0$



For  $i = 1$  to  $m$   
 $d'_i := \text{BIG}$   
 $j'_i := 0$   
For  $j \in N_i$   
If  $d_{ij} < d'_i$   
then  $d'_i := d_{ij}$       Calculate minimum cost of an arc  
 $j'_i := j$       incident to vertex  $i$   
If  $d_{ij} < 0$       Set flow equal to 1 in arcs with  
then  $\xi_{ij} := 1$       negative cost  
 $p_j := p_j + 1$   
 $z_l := z_l + d_{ij}$   
If  $d'_i \geq 0$   
then  $\xi_{ij} := 1$       Set flow equal to 1 in arc of  
 $p_j := p_j + 1$       minimum cost if this cost is  
 $z_l := z_l + d_{ij}$       non-negative

4. FEATEST . Test the Network Flow Solution for Feasibility to the SCP

If  $z_l \geq z_u$   
then goto 7.  
else  $w := 0$   
 $J' := \emptyset$   
For  $j = 1$  to  $n$   
If  $p_j \neq 0$  or  $p_j \neq h_j$   
then  $w := w + p_j(1-p_j)/h_j^2$       Arcs derived from column  $j$   
 $J' := J' \cup \{j\}$       are not feasible for the SCP  
If  $w = 0$  then goto 6.  
If  $k \geq k_{\max}$  goto 8.

5. COSTCHANGE . Change Costs in the Network

For  $j \in J'$

$$\gamma_j := \alpha p_j (z_u - z_l) / h_j w$$

For  $i \in N_j$

$$\begin{aligned} \text{If } \xi_{ij} = 1 \text{ then } d_{ij} &:= d_{ij} = \gamma_j (1 - p_j) / p_j \\ \text{else } d_{ij} &:= d_{ij} - \gamma_j \end{aligned}$$

Goto 3.

6. Network Flow Solution is Feasible for the SCP

$$z_u := z_l$$

For  $j = 1$  to  $n$

$$\begin{aligned} \text{If } p_j = h_j; \text{ then } x_j &:= 1 \\ \text{else } x_j &:= 0 \end{aligned}$$

7. Stop With an Optimal Solution

The upper bound  $z_u$  is optimal for the SCP.

8. Try to Improve the Bound

Readjust costs to non-negative values.

For  $i = 1$  to  $m$

$$u_i := \min_{j \in N_i} d_{ij}$$

For  $j = 1$  to  $n$

$$s_j := c_j - \sum_{i=1}^m u_i a_{ij} \quad \text{Calculate reduced costs}$$

Use heuristics to improve dual

feasible solution and try to

improve the upper bound.

Stop with lower bound  $z_l$

upper bound  $z_u$

In the implementation of PROCEDURE 10 it is not necessary to store the variables  $\xi_{ij}$  as separate variables. If the non-zero columns of the SCP are listed by row in the vector  $ITJ$  then if  $\xi_{ij} = 1$  the corresponding element of  $ITJ$  can be set equal to  $-j$ . An additional array for  $d_{ij}$  is required which makes the storage for this method greater than that required by Etcheberry's relaxation.

#### 4.3 Network Flow Relaxation, NF2

A second Lagrangean relaxation of the SCP involves a minimum cost flow problem, NF2, in the graph G2 - described below - and can be used to give a lower bound to the SCP. As in the previous relaxation and ascent procedure is used to maximise the Lagrangean function and thus increase the bound further.

##### 4.3.1 Construction of the Network, G2, from SCP

Each row  $i$  of SCP is represented by two vertices  $v_i$  and  $v'_i$  and an arc  $(v_i, v'_i)$  in which the flow must be at least 1. Each column  $j$  is represented by a path in the graph G2. For each column  $j$ , whenever  $a_{ij}$  and  $a_{kj}$  are two consecutive non-zero entries (i.e. when  $a_{lj} = 0$ ,  $i < l < k$ ) with  $i < k$  an arc  $(v'_i, v_k)^j$  with cost  $d_{ik}^j$  is constructed.

A source vertex  $s$  is added and an arc  $(s, v_i)^j$  for  $i$  the first non-zero entry in each column  $j$  (i.e.  $i = \min[l | a_{lj} = 1]$ ) with flow  $\xi_{si}^j$  and cost  $d_{si}^j$ .

Likewise there is a sink vertex  $t$  with flow  $\xi_{it}^j$  in arc  $(v_i, t)^j$  whenever  $i$  is the last non-zero entry in column  $j$  and the cost of this arc is  $d_{it}^j$ . The set of pairs  $(i, k)$  derived from column  $j$  together

with  $(s, i')$ ,  $(i'', t)$  (where  $i'$  is the first one in column  $j$  and  $i''$  the last 1 in column  $j$ ) form the set  $T_j$ . The flows  $\xi_{si}^j$ ,  $\xi_{ik}^j$  and  $\xi_{it}^j$  are constrained to lie between 0 and 1. The costs  $d_{si}^j$ ,  $d_{ik}^j$  and  $d_{it}^j$  are chosen so that  $\sum_{(\alpha, \beta) \in T_j} d_{\alpha\beta}^j = c_j$ . Appendix 3 gives an example.

The Graph G2 has been used by Nemhauser et al [N1] to find a lower bound to the set partitioning problem (SCP with equality constraints) in the case of equal costs. The problem was then to decompose G2 into a minimum number of chains. (A chain is a directed path or isolated vertex).

The network flow problem NF2 can alternatively be thought of as that of finding a set of paths of minimum total cost so that each path starts at a source vertex  $s$ . It ends at a sink vertex  $t$  and every other vertex in the graph lies on at least one path.

#### 4.3.2 Formulation of the Problem and Calculation of Costs

The problem of finding the minimum cost flow in G2 can be stated as follows:

$$\text{NF2}(d) \left[ \begin{array}{l} \min \sum_{j=1}^n \sum_{(i,k) \in T_j} d_{ik}^j \xi_{ik}^j \\ \text{subject to} \quad \sum_j \xi_{si}^j + \sum_{k,j} \xi_{ki}^j = \sum_{k,j} \xi_{ik}^j + \sum_j \xi_{it}^j \\ \sum_j \xi_{si}^j + \sum_{k,j} \xi_{ki}^j \geq 1 \\ 0 \leq \xi_{ik}^j \leq 1 \quad (i,k) \in T_j, j=1, \dots, n \end{array} \right. \quad (4.8)$$

If constraints (4.9) below are added to NF2( $d$ ) and  $\xi_{ik}^j$  is restricted to take integer values the resulting problem is equivalent to SCP.

$$\xi_{ik}^j = \frac{\sum_{(\alpha, \beta) \in T_j} \xi_{\alpha\beta}^j}{(h_j + 1)} \quad (4.9)$$

Constraints (4.9) can be incorporated in the objective function of NF2(d) to give a Lagrangean Relaxation:

$$\min \sum_{j=1}^n \sum_{(i,k) \in T_j} d_{ik}^j \xi_{ik}^j + \lambda_{ik}^j \left[ \frac{\xi_{ik}^j - \sum_{\alpha, \beta} \xi_{\alpha\beta}^j}{(h_j + 1)} \right] \quad (\alpha, \beta) \in T_j$$

subject to

$$\sum_j \xi_{si}^j + \sum_{k,j} \xi_{ki}^j = \sum_{k,j} \xi_{ik}^j + \sum_j \xi_{it}^j$$

$$\sum_j \xi_{si}^j + \sum_{k,j} \xi_{ki}^j \geq 1$$

$$0 \leq \xi_{ik}^j \leq 1 \quad \begin{array}{l} (i,k) \in T_j \\ j=1,2,\dots,n \end{array}$$

As in the previous relaxation, one can substitute penalties  $\pi_{ik}^j$  for  $\lambda_{ik}^j - \left[ \sum_{\alpha, \beta} \lambda_{\alpha\beta}^j / (h_j + 1) \right]$  and the costs  $d_{ik}^j$  can be computed recursively.

In this case let  $p_j$  be the number of variables  $\xi_{ik}^j$ ;  $(i,k) \in T_j$ , which are set equal to 1 and  $\bar{p}_j = h_j + 1 - p_j$ . Denoting by  $J'$  the set of columns  $j$  for which  $\xi_{ik}^j$  does not satisfy (4.9); the penalties  $\pi_{ik}^j$ ,  $j \in J'$ , are:

$$\pi_{ik}^j = \begin{array}{ll} \alpha \bar{p}_j h_j \frac{(z_u^N - z_l^N)}{\|w\|^2} & \text{if } \xi_{ik}^j = 1 \\ -\alpha p_j h_j \frac{(z_u^N - z_l^N)}{\|w\|^2} & \text{if } \xi_{ik}^j = 0 \end{array} \quad (4.10)$$

Where  $z_u^N, z_l^N$  are upper and lower bounds on  $v(\text{NF2}(d^*))$  and  $\|w\|^2 =$

$\sum_j p_j \bar{p}_j$ . For all  $j \in J'$ ,  $\pi_{ik} = 0$ .

The costs  $d_{ik}^j$  are then updated to  $d_{ik}^j + \pi_{ik}^j$  as in Section 4.2.

TABLE 4.2

## LOWER BOUNDS FROM THE NETWORK FLOW RELAXATIONS

PROBLEM					HEURISTIC BOUND		NF1 BOUND			NF2 BOUND			LP BOUND		
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
No	m	n	density	cost	% deviation from LP	time	% deviation from LP	No. Itns	time	% deviation from LP	No. Itns	time	v(LP)	No. Itns	time
51	40	350	.20	U	4.4	0.7	3.2	200	2.4	-	-	-	3.44	128	2.6
52	40	500	.20	F	2.4	1.4	2.3	22	1.8	2.2	10	25.0	47.79	124	3.9
53	40	500	.25	F	4.6	2.0	4.4	11	2.8	-	-	-	49.74	146	5.4
54	50	300	.30	F	1.5	3.4	1.3	28	4.9	-	-	-	75.38	181	5.4
55	50	1000	.13	H	0.0	2.8	0.0	-	-	-	-	-	50.00	748	12.9
56	50	1000	.15	U	4.3	2.5	3.6	30	3.1	-	-	-	3.96	154	9.3
57	60	100	.05	U	3.6	0.6	2.7	150	0.8	0.7	52	48.0	15.27	127	0.8
58	60	200	.15	H	0.0	1.3	0.0	-	-	-	-	-	60.00	841	13.5
59	60	300	.18	F	2.7	3.0	2.2	41	10.4	-	-	-	79.37	206	5.5
15	60	400	.05	U	2.8	0.7	1.2	100	2.0	1.0	27	115.5	10.37	151	2.8
60	60	400	.10	U	3.8	0.8	2.0	200	3.2	3.7	10	90	6.13	184	4.4
61	60	400	.20	U	5.4	0.9	4.2	100	5.1	5.1	4	90.5	3.55	222	7.2
62	60	700	.05	U	5.8	0.8	3.9	100	1.5	4.2	10	94.2	9.76	184	5.3
63	60	1200	.05	U	3.2	1.8	3.0	100	4.5	-	-	-	9.02	217	9.5
64	70	400	.05	U	4.7	0.9	3.7	100	1.4	-	-	-	11.53	227	4.9
65	75	300	.16	F	2.2	4.2	1.9	38	5.9	-	-	-	107.35	178	5.4
66	80	300	.20	U	3.9	0.9	2.5	100	7.1	3.9	1	75.0	3.95	228	7.7
67	80	400	.20	U	2.1	7.2	2.1	11	7.3	-	-	-	120.08	290	12.7
68	80	1000	.18	H	0.0	5.9	0.0	-	-	-	-	-	80.0	>2000	54.0
69	110	300	.20	F	1.6	10.3	1.4	21	13.0	-	-	-	190.71	345	17.7
Average for problems with cost:															
(i)				F	2.5	4.1	2.3	27	6.5	2.2	10	25.0	-	197	7.2
(ii)				U	3.7	1.6	2.9	92	3.5	2.3	17	85.5	-	178	6.1
(iii)				H	0.0	3.3	0.0	-	-	-	-	-	-	>1196	26.8
Average for all problems															
					3.0	2.6	2.3	67	4.4	2.9	16	60.4	-	> 333	9.5

#### 4.4 Computational Results

##### 4.4.1 Explanation of Results, Table 4.2

All the problems were randomly generated. The algorithms were written in FORTRAN and tested on the CDC 7600. Columns 2 and 3 give  $m$  and  $n$  the number of rows and columns, respectively, of the SCP. The density of the SCP (Column 4) is the ratio of non-zero entries to the total number of entries in the constraint matrix  $A$ .

The problems tested belonged to one of the following three classes, denoted by  $U$ ,  $H$  and  $F$  in Column 5:

- (i)  $U$  these are unicast problems with  $c_j = 1$  for all  $j$ .
- (ii)  $H$  the cost of each column is equal to the number of 1's in each column,  $c_j = h_j$  for all  $j$ .
- (iii)  $F$  this is a combined fixed cost and variable cost,  $c_j = ah_j + b$ , where  $a$  and  $b$  are positive constants. ( $a=2$ ,  $b=5$ ).

The bounds given in Table 4.2 are expressed as a percentage deviation from the LP bound. The heuristics given in Chapter 2 almost always obtained a solution within 4% of the LP solution value. This was improved by NF1 by an average of 1%. The network flow algorithm used for NF2 was a straightforward implementation of the Out-of-Kilter algorithm [F2] and proved to be very slow for a bound calculation. The LP code used was that of Land and Powell [L1], and problem DLP (rather than the primal problem) was solved because there were fewer rows than columns, and this was much faster in most cases. The initial costs for algorithms NF1 and NF2 were derived from the dual variables  $u$  and associated reduced costs  $s$  found by the algorithms

of Chapter 2. For NF1 and NF2 an iteration consists of solving a particular relaxation and then changing the costs using the subgradients derived from the solutions.

#### 4.4.2 Implementation of the Algorithms

The heuristic bound was calculated by first using PROCEDURE 3 Heuristics until there was no further increase in the lower bound up to a maximum of 5 iterations. Very often the upper bound on SCP decreased during these calculations.

For NF1 it was found that using a fixed upper bound  $z_u^F$  in (10) did not always give a good lower bound at the end of the subgradient optimization phase. Therefore  $z_u^F$  (an estimate of  $z_u^F$ ) was used which was not necessarily a true upper bound on  $v(\text{NF1})$ . This was computed as:

$$z_u^F = 1.08 \times \text{heuristic lower bound} \dots$$

In all the examples tested this would have overestimated  $v(\text{LP})$  slightly and - as others, [H4a], have noted - an overestimate of the upper bound is often more successful than using the upper bound itself.

Initial costs  $d_{ik}$  for NF1 were derived from the solution  $u$  to DLP after using algorithm 2. Had this not been the case it would have taken NF1 considerably longer to reach the same value of the lower bound as shown in Table 4.2. However tests showed that when the maximum number of allowable iterations was very large, say 700, then very often the solution of NF1 after using the worse starting position was better than that obtained after using the heuristic solutions. As the number of computations required to achieve this better bound



was so large it was not practical to use NF1 without using the heuristics first. The bound obtained from NF1 is on average within 2.5% of  $v(LP)$ . The subgradient iterations were stopped if the bound did not increase for 10 iterations. On the size of problems tested the bound never found a feasible solution to SCP which is hardly surprising considering that none of the LP solutions was integer.

As can be seen from Table 4.2 an excessive amount of computing time was required to solve NF2 and in each case the algorithm terminated because the time limit had been reached. This was despite the fact that the network flow solution was saved as the starting flow at the next iteration. The main reason for the slow execution time of the algorithm was because auxiliary storage had to be used to accommodate all the network flow information.

## CHAPTER 5

### GRAPH COVERING RELAXATIONS OF THE SCP

#### 5.1 Introduction

Graph covering problems are used in two different Lagrangean relaxations of the SCP to obtain lower bounds. In each case the optimal lower bound value for the graph covering relaxation is greater than that obtained from the LP. The two relaxations can be combined to give further improvements in the bound. Computational tests which compare the two relaxations and then combine the two are presented.

#### 5.2 The Graph Covering Problem, GCP

The graph covering problem, GCP, of finding a minimum cost set of arcs that cover all the vertices of a graph was defined in §1.2. It can be represented as an SCP with at most two 1's in each column. (Any problem with a single 1 in a column can be transformed into one with two 1's in a column by the addition of an extra row which is considered to be covered). In this case the constraint matrix  $A$  of the SCP is the vertex-arc incidence matrix of a graph  $G = \mathcal{G}(V, E)$  where  $V$  is the set of vertices and  $E$  is the set of arcs of  $G$ . Thus, the rows of  $A$  represent vertices of  $G$  and the columns represent arcs. Suppose the  $i$ th row of  $A$  is represented by a vertex  $v_i$ . If the  $j$ th column of  $A$  has non-zero elements in rows  $i$  and  $k$  then it is represented by an arc  $\epsilon_j = (v_i, v_k)$  of cost  $c_j$ . A cover,  $K$ , is a set of arcs which meets every vertex of  $G$ . The graph covering

problem is solved by finding a cover  $K^*$  of minimum cost. The corresponding solution,  $x^*$ , to the SCP has  $x_j^* = 1$  if the  $j$ th arc is in  $K^*$  and  $x_j^* = 0$  otherwise.

The GCP can be formulated as a linear program, LPB:

$$\text{LPB} \left[ \begin{array}{ll} \min & c^T x \\ & \text{subject to } Ax \geq \underline{1} \\ & Bx \geq r \\ & 1 \geq x_j \geq 0 \end{array} \right. \quad \begin{array}{l} (5.1) \\ (5.2) \\ j=1,2,\dots,n \end{array}$$

where  $c_j$  is the cost of arc  $\epsilon_j$  and  $A$  is the vertex-arc incidence matrix. It is assumed  $c_j \geq 0$  otherwise  $x_j^* = 1$  and vertices covered by arc  $\epsilon_j$  can be removed. Constraints (5.2) are added to the LP relaxation of the GCP and are exponential in number. They restrict every set of vertices of odd cardinality,  $2r-1$  say, to be covered by at least  $r$  arcs. These constraints, known as blossom constraints, need not be stored explicitly as the graph covering algorithm detects them when certain odd circuits arise in the graph  $G$ . The implementation of the graph covering algorithm is described in §8.3.

The linear programming dual of LPB is DLPB:

$$\text{DLPB} \left[ \begin{array}{ll} \max & \sum_{i=1}^m w_i + \sum_{p \in \mathcal{P}} r_p \zeta_p \\ & \text{subject to } A^T w + B^T \zeta \leq c \\ & w, \zeta \geq 0 \end{array} \right. \quad (5.3)$$

where  $\mathcal{P}$  is the set of odd cardinality subsets of  $V$ .

An approximate solution to the GCP can be obtained from any feasible solution  $(w, \zeta)$  to DLPB. Using the heuristics of Chapter 2 to get  $w$  and setting  $\zeta = 0$  gives a lower bound. The reduced cost of arc  $\epsilon_j$

is given by:

$$s_j = c_j - \sum_{i=1}^m a_{ij} w_i - \sum_{p \in P} b_{pj} \zeta_p$$

and if  $z_u^G$  is an upper bound to the GCP then if  $s_j \geq z_u^G - z_l^G$  then  $x_j = 0$  in any feasible solution the the GCP of lower cost than  $z_u$  where  $z_l^G$  is the lower bound to the GCP corresponding to  $(w, \zeta)$ . For

convenience  $u_i$  will be used to denote  $w_i + \sum_{p \in P_i} \zeta_p$  where  $P_i$  is the set of odd cardinality subsets containing vertex  $v_i$ . Then

$\bar{s}_j = c_j - \sum_{i=1}^m a_{ij} u_i$  is a lower bound on the reduced cost  $s_j$  and hence if  $\bar{s}_j \geq z_u^G - z_l^G$  then column  $a_j$  can be removed.

### 5.3 Graph Covering Relaxation 1, GCR1, A Row Relaxation of the SCP

#### 5.3.1 Description of the Relaxation

The constraint matrix  $A$  of the SCP is partitioned into two sets of rows  $R$ , the relaxed constraints, and  $\bar{R} \equiv M \sim R$ , the graph covering constraints. The rows in  $\bar{R}$  have at most 2 non-zero entries in each column. Thus if:

$$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \quad (5.4)$$

where  $A_1$  is made up of rows of  $A$  indexed by  $R$  and  $A_2$  of rows indexed by  $\bar{R}$  the Lagrangean relaxation,  $GCR1(\lambda)$  is a graph covering problem:

$$GCR1(\lambda) \left[ \begin{array}{l} \min_x \quad c^T x - \lambda(A_1 x - \underline{1}) \\ \text{subject to} \quad A_2 x \geq \underline{1} \\ \quad \quad \quad x_j \in \{0,1\} \quad j=1,2,\dots,n \end{array} \right. \quad (5.5)$$

For  $\lambda \geq 0$ ,  $v(GCR1(\lambda))$  is a lower bound to the SCP and the optimal value of this bound for all  $\lambda \geq 0$ ,  $v(GCR1(\lambda^*))$  say, gives a bound no less than  $v(LP)$  as proved below in Lemma 5.3.2. Let the graph be  $G_1$ .

### 5.3.2 Quality of the Bound

#### LEMMA

#### To prove

The best lower bound to the SCP obtained from GCR1 is at least as good as that obtained from the LP relaxation, i.e.

$$v(\text{GCR1}(\lambda^*)) \geq v(\text{LP})$$

#### Proof

Let  $u^*$  be an optimal solution to DLP and for relaxed rows in  $R$  let

$$\bar{\lambda}_i = u_i^*. \text{ Since } \lambda^* \text{ is optimal and } \bar{\lambda}_i \geq 0$$

$$v(\text{GCR1}(\lambda^*)) \geq v(\text{GCR1}(\bar{\lambda})) \quad (5.6)$$

Let LP1 be the LP relaxation of GCR1 then

$$v(\text{GCR1}(\bar{\lambda})) \geq v(\text{LP1}(\bar{\lambda})) \quad (5.7)$$

From LP duality theory it follows that

$$v(\text{LP1}(\bar{\lambda})) = v(\text{LP}) \quad (5.8)$$

Hence from (5.6), (5.7), (5.8) the result follows. □

### 5.3.3 Calculation of $\lambda^*$

The calculation of the Lagrange multipliers is the same as in Chapter 3 for Etcheberry's relaxation of the SCP. Initially the multipliers,  $\lambda^0$ , for GCR1 are obtained by setting  $\lambda_i^0 = u_i$  for  $i \in R$  where  $u_i$  is a dual feasible solution to DLP obtained using heuristics. Subgradient optimization is used to obtain subsequent values of the multipliers.

At iteration  $k$ ,  $k \geq 0$ ,

$$\lambda^{k+1} = \left[ \max \left\{ 0, \frac{\lambda^k + \alpha(z_u^G - z_x^G)\gamma^k}{\|\gamma^k\|^2} \right\} \right] \quad (5.9)$$

where  $\alpha$  is a constant,  $0 < \alpha < 2$

$x^k$  is a solution to  $\text{GCR1}(\lambda^k)$

$$\gamma^k = (A_1 x^{k-1})$$

$z_u^G$  is an upper bound on  $v(\text{GCR1}(\lambda^*))$   
 and  $z_l^G = v(\text{GCR1}(\lambda^k))$ . As described in §3.3,  $z_u^G$  can be replaced  
 by a 'target value'  $z_T^G$  and the method of Camerini et al can be used  
 to make the bound increase more rapidly.

#### 5.3.4 Partitioning the Constraints

If a graph is bipartite then the GCP can be solved optimally using  
 the LP relaxation. Hence the solution to the GCP can only be greater  
 than that obtained from LP in non-bipartite graphs. To ensure  
 that the matrix  $A$  is partitioned so that  $A_2$  gives a non-bipartite  
 graph is not easy and in any case this does not guarantee that the  
 bound obtained from graph covering is better than that obtained from  
 the LP. A practical way of partitioning  $A$  is to choose  $A_2$  with the  
 maximum possible number of rows. This can be done by solving the  
 integer program, IP1:

$$\text{IP1} \left[ \begin{array}{l} \max_y \sum_{i=1}^m y_i \\ \text{subject to } A^T y \leq \underline{z} \\ y_i \in \{0,1\} \quad i = 1,2,\dots,m \end{array} \right. \quad (5.10)$$

If  $y_i = 1$  in an optimal solution to IP1 then row  $i$  is a graph covering  
 constraint, otherwise constraint  $i$  is relaxed. In practice IP1 is a  
 large problem to solve and therefore it is solved heuristically by  
 choosing the row  $i_1$  with as few 1's as possible in it. Row  $i_2$  with  
 with the next fewest number of 1's in it is then chosen. Other rows  
 covered by columns  $j$  that have  $a_{i_1 j} = a_{i_2 j} = 1$  are removed. Row  $i_3$   
 has the least number of 1's in it in the remaining problem. Rows  
 covered by columns  $j$  that have  $a_{i_1 j} = a_{i_3 j} = 1$  or  $a_{i_2 j} = a_{i_3 j} = 1$  are  
 removed. The procedure is repeated until all the rows have either



## 2. Select $i_k$ th Row to be in Constraints

Let row  $i_k$  satisfy

$$\sum_{j \in J} a_{i_k j} = \min_{i \in S} \left( \sum_{j \in J} a_{ij} \right)$$

Find row with the least number of 1's in it

Set  $S := S - \{i_k\}$

Remove row  $i_k$  from further consideration

$$\bar{R} := \bar{R} \cup \{i_k\}$$

## 3. Find Relaxed Rows

For  $j \in J$

if  $\sum_{i \in \bar{R}} a_{ij} = \text{KCOL}$

then set  $J := J - \{j\}$   
 set  $R := R \cup (M_j - \bar{R})$   
 set  $S := S - (M_j - \bar{R})$

else next  $j$ .

If  $S \neq \emptyset$  goto 2.

## 4. Exit

$R$  is set of relaxed constraints

$\bar{R}$  = constraints for relaxed problem

(if  $\text{KCOL} = 2$ ,  $\bar{R}$  gives the graph covering constraints)

### 5.3.5 Changing the Partition of $A$

If the constraints of  $A$  are partitioned into a set of relaxed constraints  $R$  and a set of graph covering constraints  $\bar{R}$  and the bound  $v$  ( $\text{GCR1}(\lambda^*)$ ) has been obtained as described earlier, it may still be possible to improve this bound by using a different



partition of rows  $A$ . The aim is to find a partition in which the penalty  $-\lambda(A_1\hat{x} - \underline{1})$  incurred in the Lagrangean relaxation  $GCR1(\lambda)$  is as small as possible. Let  $\hat{x}$  be the solution to  $GCR1(\lambda^*)$ . If  $A_1\hat{x} \geq \underline{1}$  and  $\lambda^*(A_1\hat{x} - \underline{1}) = 0$ , then  $\hat{x}$  is optimal for the SCP and the procedure can terminate. If  $\lambda^*(A_1\hat{x} - \underline{1}) \neq 0$  and  $A_1\hat{x} \geq \underline{1}$  then a new feasible (possibly better) solution to SCP can be found by reducing  $\lambda_i^*$  to 0 for a constraint  $i \in R$  for which  $\lambda_i^*(A_1\hat{x} - \underline{1})_i > 0$ . Suppose now that this is not the case and  $\hat{x}$  is not feasible for the SCP. Let  $R_1 \subseteq R$  be the set of constraints that are not satisfied by  $\hat{x}$ . The penalty for not satisfying a constraint  $i$  is  $\lambda_i^*$ . Let  $i^*$  be the constraint for which this penalty is greatest, that is,  $\lambda_{i^*} = \max_{i \in R_1} (\lambda_i^*)$ . Constraint  $i^*$  is then removed from  $A_1$  and added to  $A_2$ . This means that some columns in  $A_2$  will now have more than 2 non-zero entries. Let  $J^+$  denote the index set of these columns.

Some constraints of  $A_2$  must now be relaxed so that the resulting problem is a graph covering problem. A heuristic estimate of the penalty incurred for relaxing a constraint of  $A_2$  can be made by considering its associated dual variable  $\lambda_i$  which is available after solving  $GCR1(\lambda^*)$ . Constraints for which  $\lambda_i$  is smallest are relaxed until  $A_2$  has at most 2 1's in each column. It may then be possible to add further constraints from  $A_1$  to  $A_2$  so that  $A_2$  still gives a graph covering problem.

Rather than just considering the dual variables  $w_i$ , one can also take into account the blossom constraints. A penalty term  $u_i = w_i + \sum_{P \in \mathcal{P}_i} \zeta_P$  can be defined where  $\mathcal{P}_i$  is the set of odd subsets of vertices containing vertex  $v_i$ . Let  $\bar{R}$  be the set of constraints that are rows of  $A_2$  and let  $i^*$  be chosen as before. Then a subset  $R_0 \subseteq \bar{R}$  of constraints that must be relaxed can be chosen so that the penalty term  $\sum_{i \in R_0} u_i$  is small.

The process of changing the partition  $\begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$  is termed a "rotation".

After the rotation, let  $R^+$  denote the index set of relaxed constraints.

Then multipliers  $\lambda^+$  for  $i \in R^+$  can be chosen by setting:

$$\lambda_i^+ = \lambda_i \quad \text{for} \quad i \in R^+ \cap R$$

$$\lambda_i^+ = u_i \quad \text{for} \quad i \in R^+ \cap \bar{R}$$

The dual variables,  $w_i^+$ , for the graph covering problem can be found by setting:

$$w_i^+ = \lambda_i \quad \text{for} \quad i \in \bar{R}^+ \cap R$$

$$w_i^+ = w_i \quad \text{for} \quad i \in \bar{R}^+ \cap \bar{R}$$

where  $\bar{R}^+$  is the set of constraints in  $A_2$  after the rotation. Then  $w$  can be checked for dual feasibility.

#### 5.4 Graph Covering Relaxation, GCR2, A Column Relaxation

##### 5.4.1 Description of the Relaxation

Whereas the last relaxation, GCR1, was derived from the SCP as originally defined in Chapter 1 this relaxation GCR2 relies on an alternative formulation. Each column of the original SCP,  $a_j$ , is split into a set of 0-1 vectors  $\beta_t$ ,  $t \in T_j$ , such that  $a_j = \sum_{t \in T_j} \beta_t$ .

Each column  $\beta$  has at most 2 non-zero entries. For example the vector  $a_j$  defined below equals  $\beta_1 + \beta_2 + \beta_3$  and  $T_j = \{1,2,3\}$ .

Row No.	$a_j$	=	$\beta_1$	+	$\beta_2$	+	$\beta_3$
1	1		1		0		0
2	1		0		1		0
3	0		0		0		0
4	1	=	1	+	0	+	0
5	1		0		1		0
6	0		0		0		0
7	1		0		0		1

The matrix with columns  $\beta_t$  is the vertex-arc incidence matrix of a graph  $G_2$ .

The SCP is then defined as  $SCPG(d)$ :

$$\begin{array}{l}
 \text{SCPG}(d) \left[ \begin{array}{l}
 \min_y \quad \sum_{j=1}^n \sum_{t \in T_j} d_t y_t \\
 \text{subject to} \quad \sum_{j=1}^n \sum_{t \in T_j} \beta_t y_t \geq 1 \\
 y_t = \left( \sum_{l \in T_j} y_l \right) / R_j \\
 y_t \in \{0,1\}
 \end{array} \right. \quad \begin{array}{l}
 (5.11) \\
 (5.12) \\
 (5.13)
 \end{array}
 \end{array}$$

where  $\sum_{t \in T_j} d_t = c_j$  [5.14]

$$R_j = \left\lceil \left( \sum_{i=1}^m a_{ij} / 2 \right) \right\rceil$$

and  $\lceil * \rceil$  is the least integer greater than or equal to  $*$ . This relaxation is similar to the network flow problem  $NF1(d)$  except that the constraints (5.11) have at most 2 non-zero coefficients for each variable  $y_t$  instead of at most 1 non-zero coefficient.

Associating a Lagrange multiplier,  $\lambda$ , with each constraint (5.12) gives the relaxation LR2( $\lambda$ ):

$$\text{LR2}(\lambda) \left[ \begin{array}{l} \min_y \sum_{j=1}^n \sum_{t \in T_j} [d_t y_t + \lambda_t (y_t - (\sum_{\ell \in T_j} y_\ell / \kappa_j))] \\ \text{subject to constraints (5.11) and (5.13).} \end{array} \right.$$

Let GCR2( $d$ ) be the problem SCPG( $d$ ) with constraints (5.12) omitted.

Defining  $\pi_t = \lambda_t - (\sum_{\ell \in T_j} \lambda_\ell / \kappa_j)$  for  $t \in T_j$ , the Lagrangean relaxation LR2( $\lambda$ ) is simply GCR2( $d + \pi$ ).

This suggests computing  $d$  recursively instead of changing the multipliers  $\lambda$  directly as in the first graph covering relaxation.

At iteration  $k$ , let  $d$  and  $\pi$  be given by  $d^k$  and  $\pi^k$  with  $k = 0$  initially. Then

$$d^{k+1} = d^k + \pi^k \quad k = 0, 1, 2, \dots \quad (5.15)$$

If  $\pi$  is chosen so that  $\sum_{t \in T_j} \pi_t = 0$  at each iteration, then constraint (5.14) is always satisfied by the costs  $d^{k+1}$ . Calculation of  $\pi$  is described in §5.4.3. This means that the total cost of variables  $y$  derived from the  $j$ th column of the SCP is the cost of the column,  $c_j$ . Section 5.4.3 gives an algorithm for computing  $d^*$  for which

$$v(\text{GCR2}(d^*)) = \max_d v(\text{GCR2}(d)) \quad (5.16)$$

$$\sum_{t \in T_j} d_t = c_j$$

### 5.4.2 Quality of the Bound

Since the optimal solution  $u^*$  to DLP is feasible for GCR2 then  $v(\text{GCR2}) \geq v(\text{LP})$ . Hence this bound is better than that obtained from the LP relaxation of the SCP.

### 5.4.3 Calculating the Costs

Initially one would like to split a column  $a_j$  into columns  $\beta_t : t \in T_j$ , so that the resulting GCP gives as high a lower bound as possible; but in general it is not easy to see how this should be done.

Therefore assume  $a_j$  is split arbitrarily. The initial costs  $d_t^0$  can be found by first calculating a feasible solution  $u$  to DLP. If the reduced costs are  $s_j = c_j - \sum_{i=1}^m u_i a_{ij}$ , then:

$$d_t^0 = u_{i'} + u_{i''} + 2s_j/h_j, \quad \text{if column } \beta_t \text{ has 2 non-zero entries in rows } i' \text{ and } i''$$

$$\text{and } d_t^0 = u_i + s_j/h_j, \quad \text{if column } \beta_t \text{ has a single non-zero entry in row } i.$$

The method used to improve the lower bound and compute  $d^*$  is an ascent method based on subgradient optimization. Initially  $\pi^0$  is zero and the solution  $y^0$  to  $\text{GCR2}(d^0)$  is found by solving the graph covering problem. The costs  $d_t$ ,  $t \in T_j$ , are only altered if they do not satisfy (5.12). Let the values of  $j$  for which this is true comprise a set  $J'$ , i.e.  $J' = \{j | y_t \neq \sum_{t \in T_j} y_t / k_j\}$ .

If  $J' = \emptyset$ , the algorithm can terminate with the optimal solution to the SCP. This can be obtained by letting  $x_j = y_t$  for some  $t \in T_j$ . If  $J' \neq \emptyset$  the penalties  $\pi_t$ ,  $t \in T_j$  for all  $j \in J'$ , must be calculated. First, for  $j \in J'$ , let  $p_j$  be the number of variables  $y_t$ ,  $t \in T_j$ , which are set equal to 1 and let  $\bar{p}_j = k_j - p_j$ .

The penalties  $\pi_t$  are then defined by:

$$\pi_t = \frac{\alpha \bar{p}_j}{\kappa_j} \frac{z_u^G - z_l^G}{\|w\|^2}, \quad \text{if } y_t = 1 \quad (5.17)$$

$$\pi_t = \frac{-\alpha p_j}{\kappa_j} \frac{z_u^G - z_l^G}{\|w\|^2}, \quad \text{if } y_t = 0 \quad (5.18)$$

Where  $z_u^G, z_l^G$  are upper and lower bounds on  $v(\text{GC2}(d^*))$ ,  $\alpha$  is an a priori determined constant with  $0 < \alpha \leq 2$  and  $\|w\|^2 = \sum_{j \in J} (p_j \bar{p}_j / \kappa_j^2)$ . The costs are then updated as in (5.15).

## 5.5 Further Improvements to the Graph Covering Relaxations

### 5.5.1 Ensuring the Costs of the Relaxed Problem Are Non-Negative in GCR1

If an arc,  $\epsilon_j$ , in  $G1$  has a cost  $c'_j < 0$  then (assuming the original SCP had positive costs) it is always possible to increase the cost to be non-negative without decreasing the value of the graph covering solution.

Suppose  $c'_j = c_j - \sum_{i \in R} \lambda_i a_{ij}$  then since  $c_j > 0$  there must be a positive multiplier,  $\lambda_\ell$  say. Let  $\Delta = \min[-c'_j, \lambda_\ell]$  and set  $\lambda_\ell$  to  $\lambda_\ell - \Delta$  thus increasing  $c'_j$  to  $c'_j + \Delta$ . Let  $\hat{x}$  be an optimal solution to the GCP corresponding to GCR1 with costs  $c'$  and  $\bar{x}$  be an optimal solution to the GCP after the costs are changed. By the optimality of  $\hat{x}$ , the feasibility of  $\bar{x}$  for the GCP and the observation that since  $c'_j + \Delta \leq 0$  it can be assumed that  $\bar{x}_j = 1$  (implying that  $\sum_{j=1}^n \alpha_{\ell j} \bar{x}_j \geq 1$ ) it follows that:

$$\begin{aligned}
c\bar{x} - \sum_{i \in R} \lambda_i a^i \bar{x} + \sum_{i \in R} \lambda_i + \Delta \left( \sum_{j=1}^n a_{\ell_j} \bar{x}_j - 1 \right) \\
\geq c\hat{x} - \sum_{i \in R} \lambda_i a^i \hat{x} + \sum_{i \in R} \lambda_i + \Delta \left( \sum_{j=1}^n a_{\ell_j} \bar{x}_j - 1 \right) \\
\geq c\hat{x} - \sum_{i \in R} \lambda_i a^i \hat{x} + \sum_{i \in R} \lambda_i = v(\text{GCR1}(\lambda))
\end{aligned}$$

Thus the solution to the GCP is not decreased after changing the costs. For each negative cost  $c_j'$ ,  $\Delta$  can be calculated repeatedly and the above changes made until  $c_j' \geq 0$  for all arcs  $\epsilon_j$ . It may then be possible to increase some of the multipliers as in PROCEDURE 2 LPBOUND by setting  $\Delta' = \min_{j \in N_i} c_j'$  where constraint  $i$  is relaxed. Then  $\lambda_i$  is increased to  $\lambda_i + \Delta'$  and the costs  $c_j'$  are correspondingly changed. This cost changing method applies to all Lagrangean relaxations in which inequality constraints are relaxed and PROCEDURE 12 COSTPLUS summarises it below.

PROCEDURE 12 COSTPLUS (SCP, R,  $\lambda$ )

SETS ALL THE COSTS IN A LAGRANGEAN RELAXATION TO NON-NEGATIVE VALUES

Input:	SCP	The set covering problem
	R	Set of relaxed constraints
	$\lambda$	Lagrange multipliers
	$c'$	Costs of relaxed problem
	EPS	Tolerance
Output:	$\lambda$ and $c'$	Multipliers and costs for Lagrangean relaxation with $\lambda \geq 0$ , $c' \geq 0$

1. Find A Negative Cost

For  $j = 1, 2, \dots, n$

(1a) If  $c_j' \geq 0$  then next  $j$   
else goto 2.

2. Find a Positive Multiplier

Find  $i \in M_j \cap R$  for which  $\lambda_i > 0$

Set  $\Delta := \min(-c_j', \lambda_i)$

Set  $\lambda_i := \lambda_i - \Delta$

$c_k' := c_k' + \Delta$  for  $k \in N_i$

Goto 1a.

3. Try To Increase The Multipliers

For  $i \in R$

Set  $\Delta := \min c_j'$

If  $\Delta < \text{EPS}$  then next  $i$

else set  $\lambda_i := \lambda_i + \Delta$

$c_j' := c_j' - \Delta$  for  $j \in N_i$

next  $i$

4. The Required Costs Are  $c'$  And The Required Multipliers Are  $\lambda$

5.5.2 Ensuring the Costs of the Relaxed Problem Are Non-Negative in GCR2

As in the previous section the costs on arcs in  $G_2$  can be adjusted



to take non-negative values without decreasing the value of the bound. This can be done by finding an arc,  $\epsilon_t$ , of negative cost,  $c_t'$ , derived from column  $j$  of the SCP. Then there must be an arc  $\epsilon_\ell$ ,  $\ell \in T_j$  with  $c_\ell' > 0$  (since it is assumed that the costs of the original SCP are positive). If  $\Delta = \min[c_\ell', -c_t']$  then  $c_\ell'$  and  $c_t'$  are changed to  $c_\ell' - \Delta$  and  $c_t' + \Delta$  respectively. This can be repeated until all costs are *non-negative*.

### 5.5.3 Changing Costs of Arcs in a GCP to Retain the Same Optimal Solution

This section considers the general problem of how to change the costs of arcs in a graph,  $G$ , so that the optimal solution to a GCP is unchanged.

Let  $E_i$  be the set of arcs incident with vertex  $v_i$ . Let  $c_j'$  be the cost and  $s_j'$  be the reduced cost of arc  $\epsilon_j$ . As the vertex-arc incidence matrix from which  $G$  is derived may have columns with only one non-zero entry it is assumed that an extra vertex has been added to the graph so that all columns have exactly 2 non-zero entries. Further details on the use of an extra vertex are given in §8.3. A vertex  $v_i$  is said to be overcovered if it is covered by more than one arc in an optimal solution to the GCP. The 0-graph is the subgraph of  $G$  for which the reduced costs equal 0 at the end of the graph covering algorithm.

Changes that can be made to costs of arcs are firstly if an arc  $\epsilon_j$  is in an optimal cover the cost  $c_j'$  can be reduced and if it is not in a cover the cost  $c_j'$  can be increased.

The amount  $\Delta_j$  by which a cost  $c_j'$  can be increased for an arc  $\epsilon_j$  in

the optimal solution will now be computed. Suppose  $\epsilon_j$  is the arc  $(v_i, v_k)$ . Two cases must be considered.

Case 1 - One Vertex,  $v_k$  say, is either the Extra Vertex or Overcovered

In this case the cost of arc  $\epsilon_j$  can be increased until it equals the cost of some other arc incident to  $v_i$ . Hence  $\Delta_j = \min_{\substack{l \in E_i \\ l \neq j}} (c_l') - c_j'$

$c_j'$  can be increased by  $\Delta_j$  if  $\Delta_j > 0$

Case 2 - Vertices  $v_i$  and  $v_k$  are Not Overcovered

Assuming that reduced costs  $s_j'$  are available at the end of the graph covering algorithm for each arc then

$$\Delta_j = \min_{\substack{l \in E_i \\ l \neq j}} (s_l') + \min_{\substack{t \in E_k \\ t \neq j}} (s_t')$$

Notice that  $\Delta_j$  will equal 0 if the degree of both  $v_i$  and  $v_k$  is greater than one in the 0-graph. If the degree of a vertex,  $v_i$ , say, is one in the 0-graph then the dual variable  $u_i$  and the cost  $c_j'$  can be increased by  $\Delta_j$ .

The amount  $\Delta_j$  by which the cost of an arc  $\epsilon_j$  that is not in the optimal solution can be reduced will now be considered. This is equal to  $s_j'$ , the reduced cost of arc  $\epsilon_j$ .

Having determined for each arc the amount  $\Delta_j$  by which the cost of an arc can be decreased or increased without altering the optimal solution gives the amount by which the costs  $d_t$  in the relaxation GCR2 can be changed to leave the solution unchanged. For GCR2 suppose the amounts  $\Delta$  by which the costs  $d_t$ ,  $t \in T_j$ , for arcs in G2 derived from

columns  $a_j$  of the SCP have been calculated. Then if  $\bar{\Delta} = \min_{t \in T_j} [\Delta_t]$  this means that arcs that are not in the solution must have their costs decreased by  $\bar{\Delta}$  and arcs in the solution must have their costs increased by  $\bar{\Delta}$ . This is done for each column  $a_j$ . After each change the variables  $\Delta$  must be updated for arcs incident to vertices that correspond to rows covered by column  $a_j$ .

For relaxation GCRL the bound can be increased by reducing  $\lambda_i$ , if a relaxed constraint  $i$  is not satisfied, i.e.,  $\sum_{j=1}^n a_{ij}x_j = 0$ , and increasing  $\lambda_i$  if  $\sum_{j=1}^n a_{ij}x_j \geq 1$ . If the relaxed constraint is satisfied with equality then if  $\lambda_i$  is changed without altering the solution  $x$  no change in the bound results. For the case in which the relaxed constraint  $i$  is not satisfied, consider the effect of reducing  $\lambda_i$  by  $\hat{\Delta}$ . Now  $\hat{\Delta}$  must be less than or equal to  $\lambda_i$  to prevent  $\lambda_i$  from becoming negative. Suppose that the arc derived from column  $a_j$  of the original SCP is denoted by  $\epsilon_j$ . Then, if  $\epsilon_j \in K^*$  and  $a_{ij} = 1$ ,  $\hat{\Delta}$  must be no greater than  $\Delta_j$ , so that the solution to the GCP does not change. Hence if, for a relaxed constraint,  $\sum_{j=1}^n a_{ij}x_j = 0$   $\lambda_i$  can be reduced by:

$$\hat{\Delta} = \min(\lambda_i, \min_{\substack{a_{ij}=1 \\ \epsilon_j \in K^*}} \Delta_j)$$

#### 5.5.4 Using the Graph Covering Solution in Consecutive Iterations of the Subgradient Optimization Procedure

Retaining the graph covering solution from the solution to a Lagrangean relaxation of the SCP and using it as an input for the next iteration in the subgradient optimization phase was not possible because of the type of graph covering algorithm used. However it is possible to save the optimal dual variables for the vertices in the

GCP, i.e., the variables corresponding to constraints (5.1), from one iteration to the next. These are then checked for dual feasibility and increased as in PROCEDURE 2 LPBD. This gives a 0-graph of the graph G2 in which an initial matching can be found to start the graph covering algorithm.

### 5.6 Combining The Two Relaxations GCR1 And GCR2

The two graph covering relaxations can be combined. Firstly a subset  $R$  of the constraints of the SCP is relaxed until there is a maximum number of non-zero entries, KCOL say, in each column of the resulting SCP. For a given vector of Lagrange multipliers  $\lambda$  let this relaxation define the problem SCPR(KCOL,  $\lambda$ ). Then if KCOL = 2 the problem SCPR(2,  $\lambda$ ) is equivalent to GCR1( $\lambda$ ) and heuristics used to determine the relaxed constraints in the latter relaxation can be extended to obtain SCPR(KCOL,  $\lambda$ ). If KCOL exceeds 2 then the columns of SCPR(KCOL,  $\lambda$ ) must be split as in GCR2 to give a graph covering problem, GCP( $\lambda, d$ ), this is the second stage. To increase the bound in the graph covering problem the costs of arcs are changed until either the solution to GCP is feasible for the column splitting relaxation and hence an optimal solution to SCPR(KCOL,  $\lambda$ ) or an iteration limit is reached. In the latter case a feasible solution to SCPR can be found by taking the solution  $y$  to the GCP when the iteration limit is reached and setting  $x_j = 1$  whenever  $y_t = 1$  for  $t \in T_j$ . The solution  $x$  is a cover for SCPR which is then reduced to a prime cover. This prime cover is then used to change the penalties of the relaxed SCP constraints as in GCR1. This defines a problem SCPR with different costs to which the relaxation GCR2 is once more applied. The procedure is repeated until the bound increases no further then the relaxation can be changed using a rotation of constraints as in §5.3.5. The method is illustrated in the flowchart of Fig.5.1 and outlined in

A FLOWCHART OF PROCEDURE 13 GRAPHBOUND TO COMPUTE LOWER BOUNDS TO THE SCP FROM GRAPH COVERING

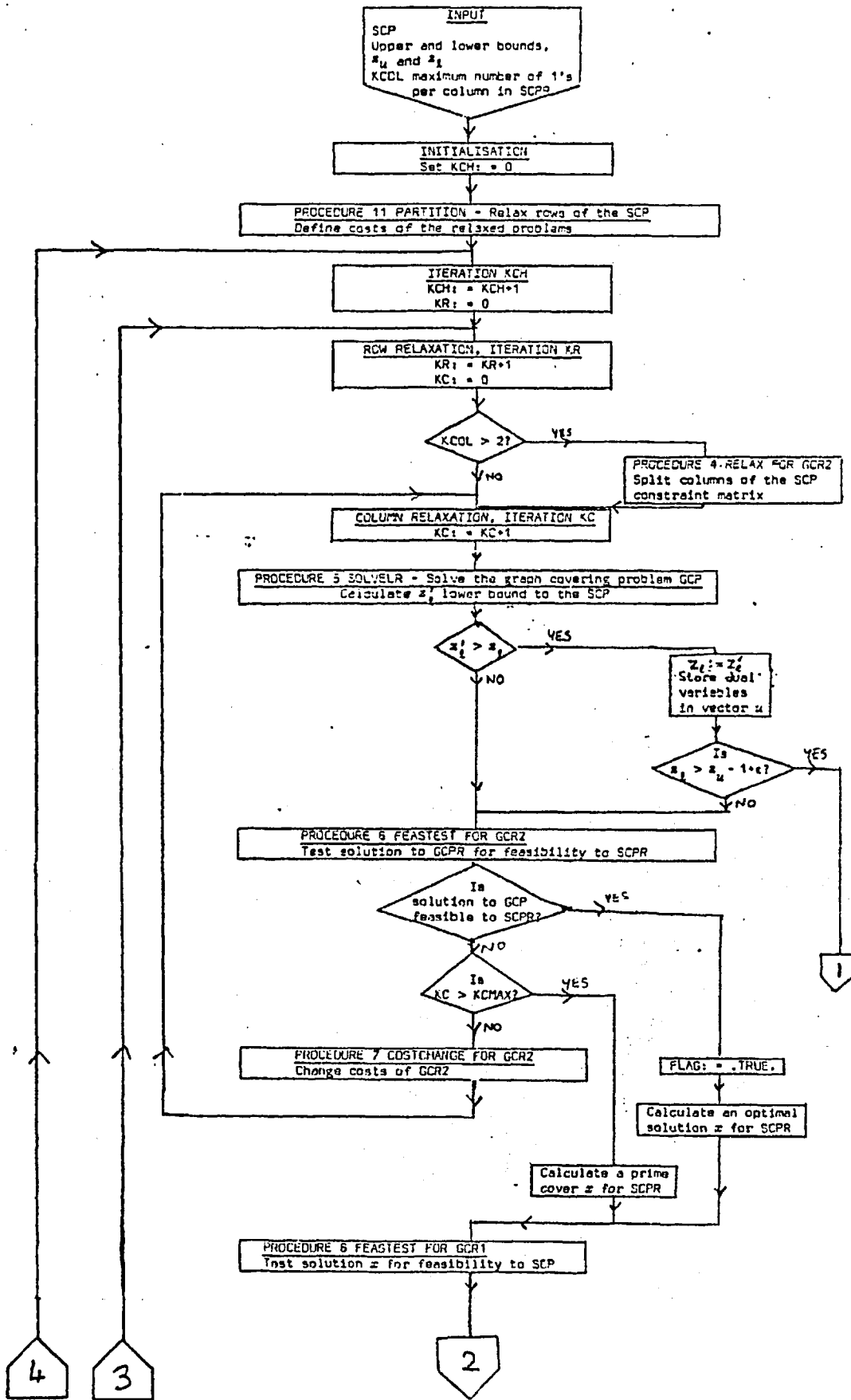
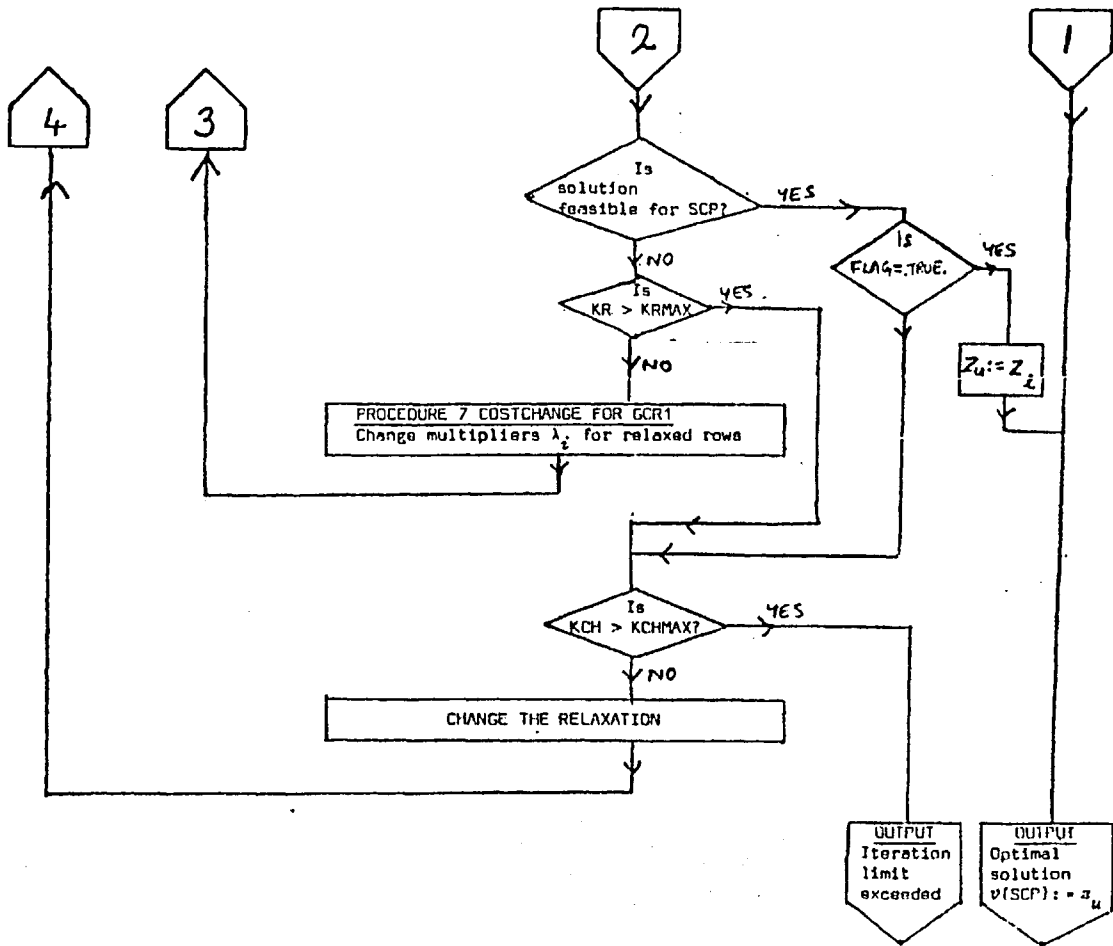


FIGURE 5.1 (cont.)

A Flowchart of Procedure 13



PROCEDURE 13 GRAPHBOUND(SCP,  $z_u$ ,  $z_l$ ,  $x$ ,  $u$ , KCOL,  $R_B$ ,  $L_B$ )

COMPUTE LOWER BOUNDS TO THE SCP FROM GRAPH COVERING

Input:	SCP	The set covering problem
	$z_u$	Upper bound to the SCP
	$u$	Dual feasible solution to the LP relaxation
	$z_l$	Lower bound = $\sum_{i=1}^m u_i$
	KCOL	Maximum number of 1's per column of constraint matrix in SCPR.
Output:	$z_l$	Graph covering lower bound
	$R_B$	Set of branching rows
	$L_B$	Set of branching variables
	$u$	Lagrange multipliers for relaxed constraints

1. Initialise Variables

$\epsilon$ : = positive tolerance

KR: = 0 iteration counter for row relaxation is set equal to 0

KRMAX: = maximum number of row relaxation iterations allowed

KC: = 0, iteration counter for column relaxation is set equal to 0

KCMAX: = maximum number of column relaxation iterations allowed

KCH: = 0, iteration counter for number of times relaxation can be changed

KCHMAX: = maximum number of times relaxation can be changed

FLAG: = .FALSE. 'FLAG' is set to the value .TRUE. if the graph covering problem solves SCPR.

PARTITION(SCP, KCOL, R)

Relax rows

 $R$  is the set of relaxed rows $\lambda_i := u_i$  for all  $i \in R$ Form relaxation SCPR and initialise  
Lagrange multipliers $c_j' := c_j - \sum_{i \in R} \lambda_i a_{ij}$ 

Calculate costs of relaxed problem

3.1 Constraints Partition Number, KCH $KCH := KCH + 1$ 

Update iteration counter

 $KR := 0$ 4.1 Row Relaxation Number, KR $KR := KR + 1$ 

Update iteration counter

 $KC := 0$ 4.2 Split ColumnsIf  $KCOL > 2$ 

Form relaxation GCR2 of SCPR

then split columns  
of the SCP  
constraint  
Matrix  $A$ 5.1 Column Relaxation Number, KC $KC := KC + 1$ 

Update iteration counter

5.2 Solve Graph Covering Problem $v(\text{GCP}) :=$  value of graph covering solution $z_l' := v(\text{GCP}) + \sum_{i \in R} \lambda_i$ 

Calculate lower bound to the SCP



If  $z'_\ell > z_\ell$   
then  $z_\ell := z'_\ell$  Update  $z_\ell$  as the best lower bound  
to the SCP  
 $u_i := w_i + \sum_{p \in P_i} \zeta_p$  Store information concerning graph  
for  $i \in M \cup \bar{R}$  covering dual variables  
if  $z_\ell \geq z_u - 1 + \epsilon$  Test if lower bound  
then goto 4.3 exceeds upper bound

### 5.3 Test Feasibility of Graph Covering Solution For SCPR

If graph covering solution  
is feasible for SCPR  
then FLAG: = .TRUE.  
 $x$ : = optimal solution to SCPR  
goto 4.3  
else if  $KC \geq KCMAX$   
then goto 5.5

### 5.4 Change Costs Of The Graph Covering Problem

Reset costs of graph covering problem as in §5.4.3 of the text

Goto 5.1

### 5.5 Calculate A Prime Cover For SCPR

$x$ : = prime cover for SCPR

### 4.3 Test The Solution $x$ For Feasibility To The SCP

If  $\lambda_i (a^i x - 1) = 0$  for all  $i \in R$   
then goto 4.4  
else if  $KR > KRMAX$   
then goto 3.2  
else change the  
multipliers  $\lambda_i, i \in R$  Update the Lagrange multipliers  
and costs  $c_j'$  for SCPR using subgradient  
goto 4.1 optimization



TABLE 5.2

Graph Covering Lower Bounds for A 30 x 60 Problem, Density 0.15, To Show Variation with Stepsize Parameter,  $\delta$ , And Number Of 1's Per Column, KCOL

$\delta$	0.5				1.0				2.0				2.5				3.5			
Column Number	1.1	1.2	1.3	1.4	2.1	2.2	2.3	2.4	3.1	3.2	3.3	3.4	4.1	4.2	4.3	4.4	5.1	5.2	5.3	5.4
KCOL=2																				
Best Bound, KCH, KR, KC	50.75	4	250		50.84	4	230		50.71	2	134		50.77	4	220		50.42	3	186	
Best Bound in 100 Iterations	50.48		100		50.83		100		50.65		99		50.12		36		50.12		36	
Best Bound in 200 Iterations	50.67		199		50.80		170		50.71		134		50.63		198		50.42		186	
KCOL=3																				
Best Bound, KCH, KR, KC	50.27	1	65	5293	50.09	0	8	627	50.06	0	1	1	50.06	0	1	1	50.06	0	1	1
Best Bound in 100 Iterations	50.06		1		50.06		1		50.06		1		50.06		1		50.06		1	
Best Bound in 200 Iterations	50.06		1		50.06		1		50.06		1		50.06		1		50.06		1	
KCOL=4																				
Best Bound, KCH, KR, KC	50.28	1	6	888	50.38	0	5	476	50.25	0	3	332	50.08	0	1	1	50.23	0	3	298
Best Bound in 100 Iterations	50.08		1		50.06		1		50.06		1		50.06		1		50.06		1	
Best Bound in 200 Iterations	50.07		178		50.06		1		50.06		1		50.06		1		50.06		1	
KCOL=5																				
Best Bound, KCH, KR, KC	50.54	1	13	1193	50.26	0	2	186	50.61	0	5	550	50.46	1	4	451	50.14	0	1	45
Best Bound in 100 Iterations	50.24		83		50.24		43		50.19		24		50.17		46		50.14		45	
Best Bound in 200 Iterations	50.24		83		50.26		185		50.21		135		50.17		46		50.14		45	
KCOL=10																				
Best Bound, KCH, KR, KC	50.37	0	2	188	50.50	2	2	200	50.57	3	3	299	50.52	1	1	100	50.53	1	1	161
Best Bound in 100 Iterations	50.33		82		50.39		74		50.41		49		50.52		100		50.45		51	
Best Bound in 200 Iterations	50.37		186		50.51		200		50.41		49		50.52		100		50.53		161	
KCOL=30																				
Best Bound, KCH, KR, KC	50.29	0	1	98	50.30	0	1	71	50.40	0	1	87	50.36	0	1	81	50.43	0	1	72
Best Bound in 100 Iterations	50.29		98		50.30		71		50.40		87		50.36		81		50.43		72	

CDC 6500 MNF 5 Compiler

For the best bound in  $k$  iterations column  $n.1$  gives the bound value and  $n.3$  the number of graph covering problems solved to get this lower bound

KCOL is the maximum number of 1's per column of the SCP after rows have been relaxed.

KCH is the number of times that the relaxation has been changed by a rotation.

KR is the number of times the multipliers have been changed on the relaxed rows.

KC is the number of times the costs of the GCP are changed for relaxed columns.

An iteration takes place each time a graph covering problem is solved when the best bound in  $k$  iterations is calculated.

## 5.7 Computational Results

### 5.7.1 Case Study

The same 30 x 60 example as used in Chapter 3 was used for the case study. The optimal solution is 56.0 and  $v(LP) = 51.0$ . Table 5.2 attempts to show variation in the bound value with variation in the stepsize parameter,  $\sigma$ , and KCOL, the maximum number of 1's per column in the constraints of SCPR. Columns n.1 (for  $n=1, \dots, 5$ ) give the lower bound value. Columns n.2 give the number of times the relaxation has been changed to get the bound. Columns n.3 give the number of row relaxation iterations and columns n.4 give the number of column relaxation iterations needed to get the lower bound. For each value of KCOL the best bound obtained after an unlimited number of iterations is obtained, the best bound after 100 row relaxation iterations and the best bound after 200 row relaxation iterations obtained was given (except for KCOL = 30). KCOL = 2 corresponds to the relaxation GCR1 and KCOL = 30 corresponds to GCR2. No definitive conclusions can be drawn from the results but when KCOL = 2 the best results were obtained. This could partly be due to the use of an anti-zigzagging strategy used in the subgradient ascent and also because changing the type of relaxation tends to make the bound decrease initially. Using small values of KCOL, equal to 3 or 4 say, was not particularly successful because it was difficult to find an optimal solution  $x$  to SCPR from the graph covering solution. This meant that in changing the multipliers  $\lambda_i$  in 4.3 of Procedure GRAPHBOUND an ascent direction for the Lagrangean relaxation was not always available.

Table 5.3 shows how the lower bound was used in a tree search. There was no possibility of solving the problem in a reasonable time using KCOL = 3 or 4 and the best times were given for the relaxation GCR1.

### 5.7.2 Comparison between GCR1, GCR2 and a Combination of the Two Relaxations

Six test problems are shown in Table 5.4 where the relaxations GCR1 and GCR2 are compared. Also shown are results for the two relaxations combined. Five of the problems were standard test problems. The first problem was randomly generated. Appendix 4 gives the source of the problems. Many more problems were tested, but they did not give a graph covering bound that was significantly better than that obtained using heuristics. On average these bounds were 0.2% better than the solution obtained from heuristics. The maximum increase over the heuristic bound was obtained by the problem SALK 13 where the GCR1 relaxation was 0.6% above the heuristic bound and the GCR2 relaxation was 0.7% above the heuristic bound. Table 5.4 shows results for the row relaxation, GCR1, in columns (v) to (ix). The bound value at the root node of a depth first tree search is given in column (v). The time taken to calculate this bound is given in column (vi). The number of graph covering problems solved in the tree search is given in column (viii). The number of tree search nodes is given in column (viii) and the total time taken for the tree search is given in column (ix). The same information for the column splitting relaxation, GCR2, is given in columns (x) to (xiv) and for the two relaxations combined in columns (xv) to (xix). Column (xx) gives the optimal solution.

The relaxation GCR2 gave graph covering problems with more vertices than GCR1 and thus it took longer to solve each graph covering problem. The number of tree search nodes generated by this relaxation was less than for the two relaxations combined or for the relaxation GCR1. One explanation for this is that the reduced cost tests removed more variables when they used the graph covering dual variables from the

larger graph given by GCR2 than when they were obtained from a smaller graph, GCR1.

For the graph covering relaxations combined the maximum number of 1's in a column of the problem SCPR obtained after relaxing the rows of GCR1 was determined according to the number of rows in the problem. It varied between 10 and 30.

### 5.7.3 Comparison Between the Graph Covering Relaxation, Heuristics and Linear Programming

#### 5.7.3.1 Korman's problems

Five problems of Korman [K4] were tested and Table 5.5 shows the lower bound values at the root node of a branch and bound tree for the heuristic, graph covering and LP relaxations. The times are in CDC 6500 seconds. The problems were all unicost SCP's with the rows of the SCP representing vertices of a graph and the columns representing \* cliques. The graph covering and heuristic bounds were almost identical in value and within 2% of the LP bound in 3 of the 4 problems solved. In each case all the bounds would have fathomed the root node had the optimal solution been available at the root of the tree. The solution times of the heuristic and LP methods were similar and the additional time spent to try and get an improved bound from graph covering was not computationally worthwhile.

Korman's program, a dynamic programming algorithm, was then compared with the best bound tree search, described in §7.3, using the graph

\* A clique is a subgraph that is a maximal complete graph: A clique has the property that every vertex in the clique is joined to every other vertex and no vertex can be added to the clique without destroying this property.

TABLE 5.3

Number Of Graph Covering Subproblems, Tree Search Nodes And Computing Time To Show  
Variation With Stepsize Parameter,  $\delta$ , And Number Of 1's Per Column, KCOL, for 30 x 60 SCP

Column Number KCOL	0.5			1.0			2.0			2.5			3.5		
	KGRAPH 1.1	KNODE 1.2	TIME 1.3	KGRAPH 2.1	KNODE 2.2	TIME 2.3	KGRAPH 3.1	KNODE 3.2	TIME 3.3	KGRAPH 4.1	KNODE 4.2	TIME 4.3	KGRAPH 5.1	KNODE 5.2	TIME 5.3
2	420	12	15.6	356	16	14.4	307	13	12.7	354	14	13.9	300	13	12.52
3	5293	*	*	5000	*	*	2521	*	*	1670	*	*	2215	*	*
4	2017	*	*	1641	*	*	1922	*	*	1959	*	*	1377	*	*
5	1193	*	*	972	*	*	1112	*	*	1036	*	*	808	*	*
10	361	12	40.1	301	16	35.6	444	*	*	301	16	35.6	343	*	*
30	300	9	33.2	300	16	33.6	400	17	37.7	400	17	38.4	300	12	32.8

KGRAPH is the number of graph covering subproblems solved

KNODE is the number of tree search nodes required to solve the SCP

TIME is the computation time on the CDC 6500 at Imperial College using the MNF5 FORTRAN compiler

\* means that the time limit was exceeded.

TABLE 5.4

A Comparison between the relaxations GCR1, GCR2 and a combination of these two Relaxations

PROBLEM				GCR1					GCR2					GCR1 combined with GCR2					Optimal Solution
No.	Size			Bound at Root Node		Tree Search			Bound at Root Node		Tree Search			Bound at Root Node		Tree Search			z*
	m	n	$\rho$	Value	Time	No. of GCP's	No. of Nodes	Total Time	Value	Time	No. of GCP's	No. of Nodes	Total Time	Value	Time	No. of GCP's	No. of Nodes	Total Time	
(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)	(xii)	(xiii)	(xiv)	(xv)	(xvi)	(xvii)	(xviii)	(xix)	(xx)
SALK8	30	80	.07	12.7	0.1	26	5	0.3	12.7	0.4	30	3	0.6	12.7	0.3	250	29	3.5	13
SALK9	30	90	.07	12.8	0.1	206	39	1.1	12.8	0.2	305	38	1.6	12.8	0.2	305	38	1.6	13
94	100	800	.02	455.7	1.3	130	27	3.2	455.3	2.9	161	7	8.9	455.2	11.7	788	17	20.4	461
SALK13	104	133	.04	1674.5	0.6	294	46	2.4	1675.2	0.9	140	13	3.5	1674.5	3.6	461	31	10.4	1678
LSSC16	200	1000	.02	428.7	1.5	198	44	5.1	428.0	5.7	-	-	*	428.0	16.3	-	-	*	429
LSSC17	200	1000	.02	510.3	4.3	-	-	*	510.2	9.2	-	-	*	510.1	3.7	-	-	*	512

\* means that the iteration limit of 30 CDC 7600 sec was exceeded

CDC 7600 seconds FTN compiler, OPT=2



TABLE 5.5

A Comparison between the Heuristic, Graph Covering and Linear Programming Lower Bounds for Korman's Test Problems

PROBLEM			Optimal Solution	HEURISTICS			GRAPH COVERING		LINEAR PROGRAMMING		TREE SEARCH with graph bound		KORMAN'S ALGORITHM	
No.	m	n	$z^*$	Upper Bound $z_u$	Lower Bound $z_l$	Time	Bound Value	Time	Bound Value	Time	No. of Nodes	Time	No. of Nodes	Time
(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)	(xii)	(xiii)	(xiv)	(xv)
K8	25	136	6	7	5.70	.2	5.70	0.8	5.80	0.3	3	6.4	3889	39.6
K9	25	136	5	6	4.90	.8	4.90	1.5	5.00	0.2	6	7.4	816	11.8
K10	30	139	7	7	6.50	.5	-	-	6.63	0.5	-	0.5	4318	51.6
K11	30	252	6	6	5.40	.3	-	-	6.00	0.5	-	0.3	4600	54.5
K15	35	564	≤7	7	5.49	.1	5.53	2.3	6.00	1.3	*	>60	* 76000	1800.0

CDC 6500 sec.

MNF5 Compiler

\* time limit

covering lower bounds. As two of the problems were solved by heuristics and as the other two terminated when the search found the optimal solution the tree search ended quickly. For all the problems the graph covering algorithm using heuristics before the graph covering phase was substantially faster than Korman's algorithm, in the case of the problem K11 180 times faster. The fifth problem of 35 rows and 564 columns required over  $\frac{1}{2}$  an hour of computer time for Korman's method and was still unsolved after 1 minute of the graph covering algorithm.

#### 5.7.3.2 Four test problems of Salkin and Koncal

The four test problems here each have density 2% and were used by Salkin and Koncal [S2]. Here the graph covering algorithm is compared with the CDC linear programming package APEX and Balas and Ho's method of disjunctive cuts. The results are shown in Table 5.6. Computing time is in CDC 7600 seconds with the FTN compiler except for column (xxi). Columns (i) to (iii) give problem number and size. In brackets in columns (ii) and (iii) is the number of rows and columns remaining after preliminary reduction tests. Column (iv) gives the optimal solution.

Lower bound values together with computation times are given in columns (vi)-(xi) for the heuristic, graph covering and APEX lower bounds. The time gives in column (ix) for the APEX lower bound is the time taken to solve the LP relaxation on the problem remaining after using reduced costs to eliminate some variables in the pre-processing stage. As can be seen the LP lower bound was greater than the other bounds in all problems except the first which was solved

TABLE 5.6

A Comparison between the Heuristic, Graph Covering and Linear Programming Lower Bounds for Four Test Problems of Salkin and Koncal

PROBLEM			Optimal Solution	LOWER BOUND VALUES AT THE ROOT NODE							COMPLETE TREE SEARCH								
No.	Size		z*	UPPER BOUND	LOWER BOUNDS														
(i)	m	n	(iv)	(v)	Heuristic		Graph Covering		LP (APEX)		Heuristic		Graph Covering		LP (APEX)		Balas & Ho		
	(ii)	(iii)			Bound	Time	Bound	Time	Bound	Time	No. of Nodes (xii)	Time (xiii)	No. of Nodes (xiv)	Time (xv)	No. of GCP's (xvi)	No. of Nodes (xvii)	Time (xviii)	No. of Cuts (xix)	Time (xx)
AHSC14	100	500	656	656	656.0	0.1	-	-	-	-	-	-	-	-	-	-	-	4	4.0
AHSC15	100 (73)	600 (128)	670	679	664.9	0.1	666.3	1.3	668.0	1.2	9	0.5	8	19.8	800	7	1.5	146	42.6
AHSC16	100 (98)	700 (222)	600	611	595.7	0.2	595.7	2.5	596.0	1.3	12	0.6	26	23.3	392	3	1.6	59	24.0
AHSC17	100 (100)	800 (227)	460	473	454.2	0.2	455.3	2.9	456.0	1.1	12	1.0	7	8.9	73	4	1.8	682	>300.0

All the times are CDC 7600 sec with the FTN compiler except for column (xx) where the times are for a DEC 20/50.

The DEC 20/50 is approximately 10 times slower than the CDC 7600.

For the Balas and Ho method the times are for solution of the problem at the root node using only cuts to raise the lower bound. When this method was used in a tree search, problem AHSC17 was solved in 92.24sec with 30 nodes and 362 cuts.

Table 5.7 Computational Results for Graph Covering Problems

PROBLEM			HEURISTIC BOUNDS			LINEAR PROGRAMMING			GRAPH COVERING	
No. of vertices		No. of arcs	Upper Bound	Lower Bound	Time	LP Bound	No. of Nodes	Time	Solution	Time
No. (i)	m (ii)	n (iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)
95	35	350	99	92.5	0.4	92.5	290	38.2	94	0.6
96	45	450	127	118.5	0.3	119.5	350	58.0	121	0.4
97	55	250	172	165.0	0.3	166.5	3	1.1	167	1.7
98	150	1000	466	428.0	0.5	430.0	7	5.3	431	2.6
99	200	1000	652	612.5	0.5	617.5	2	4.2	618	3.6

CDC 7600 sec.  
FTN compiler (OPT=2)

by heuristics and for which the LP solution was optimal. In problems (ii) and (iv) the graph covering bound was about 0.3% above the heuristic bound and was not as great as the LP bound.

Results for tree searches are given in columns (xii)-(xxi). Columns (xiii) and (xiv) give the number of tree search nodes generated and the time when heuristics are used to calculate the lower bound. Columns (xv) and (xvi) give the same information for the graph covering lower bounds.

#### 5.7.3.3 Results for graph covering problems

Five randomly generated graph covering problems were solved as SCP's using APEX linear programming package and the results are shown in Table 5.7. Bounds on the solution value were obtained using the heuristics of PROCEDURE 3, but upper bounds were also obtained using the methods of §2.3. The best upper bound is given in column (iv) and the lower bound in column (v). Column (vi) gives the time to calculate these bounds. The APEX linear programming code was used to solve the GCP's and the number of tree search nodes is given in column (viii). As can be seen the number of tree search nodes for the first two problems was approximately 300. The reason for such a large tree was that the APEX code took several branches before an upper bound was found. The graph covering solution is shown in column (x) and the time taken to compute it in (xi). For all the problems except problem 98 it was quicker to use the graph covering code than the APEX code. There was less difference in the computation times for the larger problems. Also the LP bound was very close to the optimal solution, differing by only 0.5 in the larger problems, whereas in the small problems it differed by 1.5. The heuristic upper bounds were on average 5% higher than the optimal solutions.

TABLE 5.8

Using the Graph Covering, Heuristic and LP Bounds in a Tree Search

PROBLEM				HEURISTICS			GRAPH COVERING			LINEAR PROGRAMMING			Optimal Solution
No.	m	n	p	Bound at Root Node	No. of Nodes	Total Time	Bound at Root Node	No. of Nodes	Total Time	Bound at Root Node	No. of Nodes	Total Time	z*
(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)	(xii)	(xiii)	(xiv)
LSSC1	200	2000	.02	250.2	42	4.2	250.5	-	*	251.2	-	unsolved after 10	253
LSSC9	200	2000	.02	277.3	25	2.3	277.5	19	41.7	279.0	1	1.7	279
LSSC16	200	1000	.02	427.1	10	2.1	428.0	44	15.3	429.0	1	4.9	429
LSSC20	200	1000	.02	512.0	1	0.8	-	-	-	512.0	1	4.8	512
LSSC21	200	1000	.02	555.75	15	3.2	555.81	>50	*	557.3	10	23.3	560
LSSC22	200	1000	.02	428.53	3	1.3	429.6	>50	*	430.0	1	19.1	430
SALK12	30	90	.04	12.5	1	0.1	-	-	-	12.7	11	1.2	13
SALK13	104	133	.04	1668.3	15	0.3	1674.5	46	2.4	1674.0	4	0.9	1678

CDC 7600 sec  
FTN compiler (OPT=2)

Thus using better methods to obtain the upper bounds would have been advantageous.

#### 5.7.4 Using the Heuristic, Graph Covering and LP Bounds in a Tree Search

As the previous tables of this chapter have shown the graph covering bounds were not very quick to compute. Hence when they were embedded in a tree search most of the test problems failed to be solved because of the time taken to find the bound. The results are shown in Table 5.8 and it is seen that the heuristics give the best algorithm for most of the test problems. The LP however gave a better bound than both the graph covering and heuristic solutions. It was also quicker at solving the problem, LSSC9, 200 x 2000 SCP of density 0.02.

#### 5.7.5 Conclusions

The bound calculated by the graph covering relaxations is expensive to compute and it is usually better to use the less good heuristic bound in a tree search. Only one problem, SALK 13 of Table 5.8 did the graph covering bound exceed the LP bound.

## CHAPTER 6

### LOWER BOUNDS TO THE SCP USING: DECOMPOSITION AND STATE SPACE RELAXATION

#### 6.1 Introduction

Two methods of finding lower bounds to the SCP are described in this chapter. A decomposition method, in which the SCP is divided into smaller SCP's whose solution values are summed to give a lower bound, is described first. Secondly a relaxation of a dynamic programming algorithm in which not all the state spaces are stored is used to give a lower bound to the SCP. More details on state space relaxations for the vehicle routing problem are given in Christofides et al [C7]. Both methods are illustrated on examples. The first method produces an excellent lower bound but is slow to compute. The second bound is more quickly computed but does not give a particularly high value. Extensions to the second method which improve the bound are also discussed.

#### 6.2 The Decomposition Method For Obtaining A Lower Bound To The SCP

##### 6.2.1 Definition

The constraints  $M$  of the SCP are partitioned into  $r$  disjoint subsets,  $R_1, R_2, \dots, R_r$ . Thus the constraint matrix (after suitable rearrangement of the rows if necessary) is given by:

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_r \end{pmatrix} \quad (6.1)$$



Let the non-zero columns of  $A_\ell$  be denoted by  $\bar{A}_\ell$ . Whenever the  $j$ th column of  $A_\ell$  is non-zero a variable  $y_t$  is defined. Let  $T_j$  be the index set of variables  $y_t$  derived from the  $j$ th column of the SCP. Let  $d_t$  be the cost of variable  $y_t$  such that:

$$\sum_{t \in T_j} d_t = c_j \quad (6.2)$$

For notational convenience let  $y_{(\ell)}$  and  $d_{(\ell)}$  be vectors of problem variables and costs corresponding to columns of  $\bar{A}_\ell$ . The SCP can then be reformulated as the problem SCPD( $d$ ):

$$\text{SCPD}(d) \left[ \begin{array}{l} \min_y \sum_{\ell=1}^r d_{(\ell)}^T y_{(\ell)} \\ \text{subject to } \bar{A}_1 y_{(1)} \geq \underline{1} \\ \qquad \qquad \bar{A}_2 y_{(2)} \geq \underline{1} \\ \qquad \qquad \qquad \vdots \\ \qquad \qquad \qquad \bar{A}_r y_{(r)} \geq \underline{1} \\ \\ y_t = \sum_{r \in T_j} y_r / |T_j| \quad (6.4) \\ y_t \in \{0,1\} \text{ for all } t \end{array} \right. \quad (6.3)$$

This formulation of the SCP has the same structure as those used to derive the problems NF1( $d$ ) and GCR2( $d$ ). Dropping constraints (6.4) from SCPD( $d$ ) gives the problem DEC( $d$ ) which is again a set covering problem. Define the problem  $\text{SCP}_\ell(d)$  as:

$$\text{SCP}_\ell(d) \left[ \begin{array}{l} \min_{y_{(\ell)}} d_{(\ell)}^T y_{(\ell)} \\ \text{Subject to } \bar{A}_\ell y_{(\ell)} \geq \underline{1} \\ \qquad \qquad \qquad y_t \in \{0,1\} \end{array} \right. \quad \begin{array}{l} \text{for all } \ell \quad (6.5) \\ \text{for all } t \end{array}$$

Problem  $DEC(\vec{d})$  is solved by solving each of the problems  $SCP_\ell(\vec{d})$  for  $\ell = 1, 2, \dots, r$ . Then  $v(DEC) = \sum_{\ell=1}^r v(SCP_\ell)$  is a lower bound to the SCP. Even if it is not possible to solve the problems  $SCP_\ell$  exactly lower bounds can be calculated which, added together, give a lower bound to  $v(DEC)$  and hence to the SCP.

As in the relaxations  $NF1(\vec{d})$  and  $GCR2(\vec{d})$  the aim is to divide the costs  $c$  so that optimal costs  $\vec{d}^*$  for the relaxation are found where:

$$v(DEC(\vec{d}^*)) = \max_{\vec{d}} v(DEC(\vec{d}))$$

$$\text{subject to } \sum_{t \in T_j} d_t = c_j \quad (6.6)$$

One way of calculating the costs  $d_t$  is to use subgradient optimization as described in the next two sections.

### 6.2.2 Calculating the Costs $d^0$ Initially

If  $u$  is a feasible solution to DLP and  $s$  is the vector of associated reduced costs then a cost  $d_t$  derived from column  $j$  and rows  $R_\ell$  of the original SCP can initially be defined as:

$$d_t^0 = s_j / |T_j| + \sum_{i \in R_\ell} u_i \quad (6.7)$$

This value guarantees that the bound obtained from the relaxation  $DEC(\vec{d})$  is at least as great as the heuristic lower bound,  $\sum_{i=1}^m u_i$ . Further if  $u^*$  is an optimal solution to DLP then  $v(DEC(\vec{d})) \geq v(DLP) = v(LP)$  thus giving a bound at least as good as that obtained from the LP relaxation.

### 6.2.3 Updating the Costs

The costs,  $\vec{d}$ , are updated as for  $NF1$  and  $GCR2$  at an iteration  $k$  of

the subgradient optimization for  $k \geq 0$  by first solving SCPD( $d^k$ ). The solution  $y^k$  is then tested for feasibility to the SCP, by checking that constraints (6.4) are satisfied. If  $y_t^k = 0$  for all  $t \in T_j$  then  $x_j = 0$  in the SCP and similarly if  $y_t^k = 1$  for all  $t \in T_j$  then  $x_j = 1$ . Otherwise let  $J' = \{j | (6.4) \text{ is not satisfied}\}$  and let  $p_j$  be the number of variables in  $T_j$  set equal to 1. At iteration  $k$  the penalties  $\pi_t$  are given by:

For  $j \in J'$  and  $t \in T_j$

$$\pi_t = \frac{\alpha(|T_j| - p_j)(z_u - z_l)}{|T_j| ||w||^2} \quad \text{if } y_t = 1$$

$$\pi_t = \frac{-\alpha p_j (z_u - z_l)}{|T_j| ||w||^2} \quad \text{if } y_t = 0$$

For  $j \notin J'$  and  $t \in T_j$

$$\pi_t = 0$$

where  $\alpha$  is an a priori chosen constant

$z_u$  is an upper bound to the SCP

$z_l$  is the lower bound,  $v(\text{DEC}(d^k))$ .

$||w||^2$  is equal to  $p_j \bar{p}_j / |T_j|^2$   
The costs  $d_t$  are updated at iteration  $k$  by:

$$d_t^{k+1} = d_t^k + \pi_t \quad (6.9)$$

The iterations terminate when either (i) the optimality conditions (6.4) are satisfied, (ii) the bound  $v(\text{DEC}) \geq z_u - 1 + \epsilon$  where  $z_u$  is an upper bound to the SCP,  $\epsilon$  is a tolerance and the costs  $c_j$  are integer or (iii) the bound has not increased for several iterations.

#### 6.2.4 Using Integer Costs $d_t$

If the costs  $d_t$  are restricted to be integral then the solution to

each subproblem  $SCP_\ell$  and hence to DEC must be integral. Therefore in any tree search to solve the subproblem  $SCP_\ell(d)$  rounding up non-integral lower bounds may accelerate the computation time. Instead of using heuristic solutions to DLP to calculate the initial costs,  $d^0$ , they are given integer values satisfying  $\sum_{t \in T_j} d_t = c_j$ . Changes to the costs must subsequently be made in integral amounts.

#### 6.2.5 Reduced Costs for the SCP

Both the network flow and graph covering relaxations can be described as linear programs from which reduced costs for the SCP are then derived. The decomposition relaxation yields combinatorial problems ( $SCP_\ell$ ), albeit smaller problems than the original SCP, and hence integer programming duality defined in §1.3.4.4 must be used to define reduced costs.

First let  $a_{j/\ell}$  be the  $j$ th column of  $A_\ell$ . Then let  $f_d(\cdot)$  be an optimal subadditive function analogous to optimal dual variables in linear programming. One choice for  $f_d(a_{j/\ell})$  is as the optimal solution to the  $SCP_\ell$  with right hand side replaced by  $a_{j/\ell}$ , i.e.

$$f_d(a_{j/\ell}) = \min_{y(\ell)} d_{(\ell)}^T y(\ell)$$

subject to  $\bar{A}_\ell y(\ell) \geq a_{j/\ell}$   
 $y_t \in \{0,1\}$

Then letting  $F_d(a_j) = \sum_{\ell=1}^r f_d(a_{j/\ell})$  the reduced cost of the  $j$ th column of the original SCP is given by :

$$s_j = c_j - F_d(a_j) \tag{6.10}$$

As with linear programming duality the  $j$ th column of the SCP can be

removed if:

$$s_j \geq z_u - 1 + \epsilon - v(\text{DEC}(d)) \quad (6.11)$$

Unless  $\text{SCP}_\ell(d)$  is solved by an LP relaxation, in which case  $f_d(a_{j/\ell})$  can be chosen as  $\sum_{i \in R_\ell} u_i^* a_{ij}$  (where  $u_i^*$  is the optimal LP dual variable for row  $i$ ), it is difficult to calculate  $f_d(a_{j/\ell})$  exactly. Instead an upper bound to  $f_d, \bar{f}_d$ , is calculated giving  $\bar{F}_d$ , an upper bound to  $F_d$ , by:

$$\bar{F}_d(a_j) = \sum_{\ell=1}^n \bar{f}_d(a_{j/\ell}) \quad (6.12)$$

then

$$\bar{s}_j = c_j - \bar{F}_d(a_j) \quad (6.13)$$

is a lower bound on the reduced cost  $s_j$ . Thus if (6.11) is true when  $s_j$  is replaced by  $\bar{s}_j$  the  $j$ th column of the original SCP can be removed.

Unfortunately no easy way of calculating  $\bar{f}$  was found. To remove variables using reduced costs it is sufficient to find a function  $\bar{f}$  that is an upper bound on an integer dual feasible subadditive function  $f'$  where  $f'(\underline{1}) = v(\text{DEC}(d))$ . Again no such function  $f'$  that could be calculated easily was found.

### 6.2.6 Recursive Tree Search

The decomposition method involves splitting the SCP into smaller SCP's,  $\text{SCP}_\ell(d)$ , that are solved using a tree search. The lower bound  $v(\text{DEC}(d))$  is in turn used in a tree search. Thus the tree search procedure is used at two levels in the algorithm presented in PROCEDURE 14 DECOMPOSITION BOUND below. There may be an advantage in decomposing  $\text{SCP}_\ell(d)$  as if it were the original SCP. Thus a whole

sequence of SCP's could be generated and tree searches used at a depth of up to a fixed number, say  $q$ , levels.

### 6.2.7 Sorting the Constraint Matrix $A$ Initially

If the constraint matrix  $A$  is almost in block diagonal form initially then fewer variables  $y_t$  will be generated and it is less likely that constraints (6.2) will be violated. Therefore a heuristic procedure should be used to sort the matrix into block diagonal form. Such a procedure has been developed by King and Nakornchai [K2a] which sorts the matrix into blocks by sorting columns as in §1.3.3.2. The procedure is then repeated by sorting the rows into blocks. The columns and rows are sorted alternately until no further improvement is made. The procedure tends to cluster the non-zero elements near the diagonal of the matrix.

### 6.2.8 Description of the Decomposition Algorithm

The procedure below describes how the decomposition algorithm is used to obtain a lower bound to the SCP.

PROCEDURE 14 DECOMPOSITION BOUND (SCP,  $z_u$ ,  $z_l$ ,  $u$ , NPART)

COMPUTE A LOWER BOUND TO THE SCP USING A DECOMPOSITION METHOD

Input:	SCP	The set covering problem
	$z_u$	An upper bound to the SCP
	NPART	The number of subproblems into which the SCP is to be divided
	$z_l, u$	Lower bound from heuristics and dual feasible solution

Output:  $z_\ell$                     A lower bound to the SCP  
            $z_u$                     Upper bound to the SCP

### 1. Initialise Variables

KMAX                    Maximum number of iterations allowed  
 $k := 0$                     Set iteration counter to 0  
 $\epsilon$                     Tolerance within which the solution  
                           must lie

### 2. Define The Relaxation

Partition the constraint matrix  $A$  into NPART submatrices. Calculate costs  $d_t$  for each problem  $SCP_\ell(d)$ .

### 3. Iteration $k$

$k := k+1$                     Update iteration counter  
  
If  $k > KMAX$  goto 8.  
 $\ell := 0$                      $\ell$  is index of the current subproblem  $SCP_\ell$   
 $L := \phi$                      $L$  gives solutions of relaxed problems  
 $z'_\ell := 0$                     Initialise bound value

### 4. Solve $SCP_\ell(d)$

4.1  $\ell := \ell+1$

If  $\ell > NPART$

then goto 5.

else solve  $SCP_\ell(d)$

$z'_\ell := z'_\ell + v(SCP_\ell(d))$  Calculate bound

$L := LU\{t | y_t = 1 \text{ in solution to } SCP_\ell(d)\}$

### 5. Test Solution Value

If  $z_l \geq z_u - 1 + \epsilon$

Test if lower bound exceeds upper bound

then goto 7.

else let  $J = \{j \mid \text{equation 6.4 of text is not satisfied}\}$

Test feasibility of solution for the SCP

if  $J = \emptyset$  then goto 6.

else change costs

$d_t$  for all  $t \in T_j$

and all  $j \in J$

goto 3.

### 6. Solution Is Feasible For The SCP

Set  $x_j = 1$

Whenever there is  $y_t = 1$  and  $t \in T_j$

Set  $z_u := z_l$

### 7. Exit With Optimal Solution $z_u$

Exit with  $z_u$  gives optimal solution to the SCP

### 8. Iteration Limit Exceeded

Exit with  $z_l$  a lower bound to the SCP

The computational results for the decomposition method are given alongside those for the state space relaxation in §6.5.



### 6.3 A Lower Bound To The SCP From State Space Relaxation

#### 6.3.1 Definition

In principle the SCP can be solved by dynamic programming, but this requires too much storage to be useful in practice. This section shows how the dynamic programming states can be mapped on to a smaller set of states. Instead of obtaining an optimal solution to the SCP solving dynamic programming recursions on the smaller set of states gives a lower bound to the SCP. This is known as state space relaxation and can be thought of as a generalisation of Lagrangean relaxation. In Lagrangean relaxation each constraint has a single multiplier  $\lambda_i$  and <sup>in state space relaxation</sup> a subset of constraints,  $S$ , is given a value

$$f(S) = \sum_{i \in S} \lambda_i.$$

State space relaxations enable functions  $f(S)$  which are non linear to be computed for a given state,  $S$ . For example  $f(S)$  could be the number of elements in  $S$ . To calculate the state space relaxation (SSR) bound a function  $g(S)$  is used to map the sets  $S$  onto a smaller set of sets. Suppose  $S$  represents a right-hand side vector  $b$  of the integer program, IP. At iteration  $k+1$  the lower bound is given by:

$$F_{k+1}(g(b)) = \min_{j \in N_{k+1}} [F_k(g(b - a_j)) + c_j] \quad (6.14)$$

where  $N_k = \{j | a_{kj} \neq 0\}$ ,  $F_0[g(b)] = 0$  for all  $b$  and  $a_{ij} \geq 0$

#### 6.3.2 State Space Relaxation 1, SSR1

##### 6.3.2.1 Definition

In the first state space relaxation, SSR1, the function  $g(b)$  for a 0-1 vector  $b$  is given by two values  $(\alpha, \beta)$  where  $\alpha$  is the number of

components equal to 1 in  $b$  and  $\beta$  is the index of the row containing the last 1. The lower bound to the SCP, obtained using dynamic programming recursion (6.14), is given by  $F_m(g(\underline{1}))$ .

### 6.3.2.2 Reduced costs

$F_m(b)$ , which will be used as an abbreviation for  $F_m(g(b))$ , is a subadditive integer dual feasible solution and thus the reduced cost of column  $a_j$  can be defined as:

$$s_j = c_j - F_m(a_j)$$

The reduced cost test is if

$$s_j \geq z_u - F_m(\underline{1}) \quad (6.15)$$

then  $x_j$  equals 0 in any solution better than  $z_u$ .

### 6.3.2.3 Improving the bound value using subadditivity

Since any subadditive non-decreasing function that satisfies the dual feasibility conditions can be used one has:

$$F(b-a_j) + F(a_j) \geq F(b)$$

$$\text{or } F(b-a_j) \geq F(b) - F(a_j) \quad (6.16)$$

Now because a relaxation is used condition (6.16) may be violated and hence it may be possible to improve  $F(b-a_j)$ .

An example for the 30 x 60 SCP used in previous chapters gave values for  $F$  as shown below:

The value of  $F_m(\alpha, \beta) = F_m(g(b))$

- $m = 30$  is the number of constraints in the SCP  
 $b$  is an  $m$ -dimensional binary vector  
 $\alpha$  is the number of 1's in the binary vector  $b$   
 $\beta$  is the index of the last non-zero component of  $b$

$\beta$	$\alpha$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	7																														
2	7	9																													
3	7	14	16																												
4	9	13	20	25																											
5	7	9	18	22	29																										
6	7	7	9	16	22	29																									
7	7	7	13	13	16	22	29																								
8	9	11	11	18	18	20	27	33																							
9	7	7	7	16	18	18	25	25	27																						
10	7	7	7	14	14	16	25	24	27	32																					
11	7	7	7	7	13	14	18	20	25	27	29																				
12	7	7	9	9	14	14	16	16	22	23	27	29																			
13	9	11	11	11	11	18	18	18	18	24	25	27	31																		
14	7	7	7	7	9	11	18	18	18	18	24	25	25	27																	
15	7	7	7	7	7	14	14	14	14	16	18	25	25	25	25																
16	7	7	7	7	9	14	14	14	14	16	21	21	21	23	25	27															
17	9	9	9	9	9	16	16	16	16	16	23	23	23	23	25	27	30														
18	9	9	9	9	11	16	16	16	18	20	23	23	23	25	27	30	30	32													
19	7	7	7	7	7	11	16	16	16	18	20	23	23	25	27	27	30	30	32												
20	7	7	7	7	7	14	14	14	14	14	18	23	23	23	25	27	30	30	32	34											
21	7	7	7	7	7	9	13	14	14	14	16	20	21	21	21	23	27	30	30	32	34										
22	7	7	7	7	7	9	14	14	14	14	16	18	21	21	21	23	25	28	28	28	30	32									
23	7	7	7	7	7	7	9	14	14	14	14	16	18	21	21	21	23	25	27	28	28	30	32								
24	9	9	9	9	9	9	9	11	16	16	16	16	18	23	23	23	25	27	30	30	30	32									
25	7	7	7	7	9	9	13	13	16	16	16	18	18	20	22	23	23	25	25	27	29	30	32	32	34						
26	7	7	7	7	7	7	9	14	14	14	16	16	18	18	20	23	23	23	25	25	27	30	30	32	32	34					
27	7	7	7	7	7	7	13	13	13	14	14	14	16	20	20	21	23	23	25	25	27	30	30	30	32	32	34				
28	7	7	7	7	7	7	9	14	14	14	14	14	16	20	20	21	21	21	23	25	27	28	30	30	32	32	34	36			
29	11	11	11	11	11	11	11	18	18	18	18	18	18	20	25	25	25	25	27	31	31	32	32	32	34	36	38	39			
30	7	7	7	7	7	7	11	13	14	14	16	16	16	18	20	20	21	25	25	25	27	31	31	32	32	32	34	36	38	39	

Notice that  $F(9,9) = 27$  and  $F(8,8) = 33$  and  $F(9,9) < F(8,8)$ . This means that the least cost of covering the first 9 rows of the SCP is less than that of covering the first 8 rows. Hence  $F(9,9)$  can be increased to 33. This kind of check can improve the bound further and can be generalised using (6.16).

#### 6.3.2.4 Comparison with other relaxations

If the parameter  $\beta$  is dropped then the state space relaxation gives the same result as solving the knapsack problem when all the constraints are added together. Hence the bound is at least as good as solving the knapsack relaxation.

#### 6.3.3 State Space Relaxation 2, SSR2

In this relaxation a third parameter,  $\gamma$ , is used to define  $g(S) = (\alpha, \beta, \gamma)$

This parameter is defined by assigning to the  $i$ th row of the SCP a small integer value  $u_i$ , say and then for a 0-1 vector  $b$ ,  $\gamma = \sum_{i=1}^m u_i b_i$ . Then  $\alpha$  and  $\beta$  are defined as before. This bound is then at least as good as that obtained from solving the weighted knapsack problem where the constraints of the SCP are added together after being multiplied by the weights  $u_i$ .

If the weights do not take integer values but instead take the LP optimal dual variables then  $\gamma$  can take real values instead of integer values and hence the number of values of  $g(S)$  may be very large.

If all these values can be stored and optimal values of the dual variables (weights),  $u_i$ , can be found then the bound is at least as good as that obtained from the LP relaxation. If there are too many values then  $g(S)$  must be redefined as  $(\alpha, \beta, \lfloor \gamma \rfloor)$  where  $\lfloor * \rfloor$  is the largest integer less than or equal to  $*$ . This means that the bound may be less than that from LP.

#### 6.3.4 Other State Space Relaxations and Extensions

Other possibilities for  $g(b)$  are to put  $g[b] = (\alpha, \beta, \gamma)$  where  $\alpha$  and  $\beta$  are as before and  $\gamma$  is the index of the row containing first non-zero entry of  $b$ . Thus  $g(01 0 11 100)^T = (4, 6, 2)$ .

For problems in which consecutive 1's occur for example  $b =$

$(0111\ 000\ 11111\ 0000)^T$  a function  $g(b) = (\alpha, \beta, \gamma)$  can be used where  $\beta$  is the index of the row containing the first non-zero entry,  $\gamma$  is the index of the last row of the first string of 1's and  $\alpha$  is the number of components of  $b$  equal to 1. This can be extended to  $g(b) = (\alpha, \beta_1, \gamma_1, \beta_2, \gamma_2, \dots, \beta_k, \gamma_k)$  where  $\beta_\ell$  is the index of the first row and  $\gamma_\ell$  is the index of the last row in the  $\ell$ th string of 1's. Thus for the vector  $\beta$  above and  $k=2$ ,  $g(b) = (8, 2, 4, 8, 12)$ .

Another possibility is to divide the matrix  $A$  of the SCP into  $r$  submatrices  $A_\ell$ ,  $\ell = 1, 2, \dots, r$ , as in §6.2.1 and set:

$$g(b) = (\alpha_1, \alpha_2, \dots, \alpha_r, \beta)$$

where  $\alpha_\ell$  is the number of components of  $b$  equal to 1 that are rows of  $A_\ell$  and  $\beta$  is the index of the row containing last non-zero entry of  $b$ .

## 6.4 Solving A Class Of SCP's

### 6.4.1 Introduction

The SCP's that are considered in this section have columns made up of strings of 1's. A string of 1's in a column  $j$  is a set of rows  $i_1, i_1+1, \dots, i_2$  with  $a_{i_1 j} = a_{(i_1+1)j} = \dots = a_{i_2 j} = 1$ . If  $i_1$  is not the first row then  $a_{(i_1-1)j} = 0$  and if  $i_2$  is not the last row then  $a_{(i_2+1)j} = 0$ . For the column:

$$a_j = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

there are two strings of 1's from rows 1 to 3 and 5 to 6.

When only one string of 1's occurs in each column the SCP can easily be solved as a shortest path problem as shown in §1.3.2.1. Strings of 1's occur frequently in vehicle scheduling and routing problems. Shephardson and Marsten [S8] solved problems with two strings of 1's in each column. Their method is generalised here. For a problem that can have any number of strings of 1's first a decomposition of each column is defined. This is in contrast to a decomposition of the entire matrix as given in §6.2. A lower bound is obtained from this relaxation which is solved as a shortest path problem. Costs of the relaxed problem are changed by subgradient optimization to improve the lower bound.

#### 6.4.2 Defining the Relaxation

Let each string of 1's in the constraint matrix be indexed by  $t$ . Let  $T_j$  be the index set of strings of 1's that occur in the  $j$ th column of the SCP. Let the variable,  $y_t$ , be associated with the  $t$ th string of 1's. The column  $a_j$  of the SCP is then split  $|T_j|$  into columns,  $\beta_t (t \in T_j)$ . If the  $t$ th string of 1's goes from rows  $i_1$  to  $i_2$  then column  $\beta_t$  has 1's in rows  $i_1$  to  $i_2$  and 0's elsewhere. Thus for the column  $a_j$  defined in the previous paragraph there are two strings of 1's so  $\beta_1$  and  $\beta_2$  are defined as follows:

$$a_j = \beta_1 + \beta_2$$

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

As in relaxation GCR2 and the decomposition relaxation the cost of column  $\beta_t$  is  $d_t$  and  $\sum_{t \in T_j} d_t = c_j$  :

$$\text{SCPP}(d) \left[ \begin{array}{l} \min_y \sum_{j=1}^n \sum_{t \in T_j} d_t y_t \\ \text{subject to } \sum_{j=1}^n \sum_{t \in T_j} \beta_t y_t \geq 1 \\ y_t = \sum_{s \in T_j} y_s / |T_j| \quad \text{for all } t \in T_j \\ y_t \in \{0,1\}, \text{ for all } t \quad \text{for } j=1, \dots, n \end{array} \right. \quad (6.17)$$

Constraints (6.17) can be relaxed to give the Lagrangean relaxation,

LRP( $\lambda$ ):

$$\text{LRP}(\lambda) \left[ \begin{array}{l} \min_y \sum_{j=1}^n \sum_{t \in T_j} \left( d_t y_t - \lambda_t \left\{ y_t - \left[ \sum_{s \in T_j} y_s / |T_j| \right] \right\} \right) \\ \sum_{j=1}^n \sum_{t \in T_j} \beta_t y_t \geq 1 \\ y_t \in \{0,1\} \quad \text{for all } t \end{array} \right.$$

where  $\sum_{t \in T_j} d_t = c_j$

As in the graph covering relaxation GCR2 the problem can be reformulated by letting:

$$\pi_t = \lambda_t - \sum_{s \in T_j} \lambda_s / |T_j| \quad \text{for } t \in T_j \quad (6.18)$$

#### 6.4.3 Changing the Costs

Subgradient optimization gives exactly the same formula for updating the costs  $d_t$  at iteration  $k$  as in §6.2.3. At the  $k$ th iteration the problem that is solved is the shortest path problem,  $\text{SPP}(d)$ .

$$\text{SPP}(d) \left[ \begin{array}{l} \min_y \sum_{j=1}^n \sum_{t \in T_j} d_t^k y_t \\ \sum_{j=1}^n \sum_{t \in T_j} \beta_t y_t \geq 1 \\ y_t \in \{0,1\} \end{array} \right.$$

SPP( $d$ ) naturally has an integer solution.

The aim, as before, is to find optimal costs  $d^*$  that satisfy:

$$v(\text{SPP}(d^*)) = \max_d v(\text{SPP}(d))$$

$$\text{subject to } \sum_{t \in T_j} d_t = c_j \quad \text{for all } j.$$

This gives a lower bound to the SCP that is at the best bound equal to the LP bound. The proof follows the proof that  $v(\text{NF1}(d^*)) = v(\text{LP})$  given in Chapter 4. Firstly it can be shown that if the costs  $d_t$  are negative they can be set to non-negative values,  $d'_t$  say, with no decrease in bound, so that:

$$v(\text{SPP}(d')) \geq v(\text{SPP}(d))$$

Then associated with the solution to SPP( $d'$ ) is a dual feasible solution which is dual feasible for DLP and hence

$$v(\text{SPP}(d')) \leq v(\text{DLP}) = v(\text{LP})$$

The LP bound can be attained by setting  $d_t$  equal to  $s_j + \beta_t^T u^*$  for  $u^*$  an optimal solution to DLP and  $s_j$  the corresponding reduced cost where  $t \in T_j$ . Hence the best bound obtainable from this relaxation is the same as that obtained from the LP. The advantage of this method is that shortest path problems can be solved more quickly than linear programs. Also each string of 1's can be stored as two figures, the first and last row of the string. This reduces the storage requirements for the SCP when the strings of 1's are long.

## 6.5 Computational Results

### 6.5.1 Case Study

For the same 30 x 60 example of density 0.15 as used in the other



chapters the state space relaxation bound was 39 as shown in 6.3.2.3. The CDC 6500 time under the MNF compiler was 6 sec. The value of the bound obtained from the decomposition relaxation when the SCP was partitioned into 4 subproblems was 55.11 and the corresponding time was 22 sec. on the CDC 6500. This bound value obtained after 29 iterations of Procedure DECOMPOSITION BOUND. Bound values are shown in Table 6.1. The computing times were slow partly because of the time taken by the depth first on rows tree search strategy. Using an anti-zigzagging strategy in the subgradient ascent and a better choice of steplength would improve computing time. The SCP solution was 56.0, LP bound was 51.0 and the knapsack bound (formed by adding the constraints) of the SCP together) was 29.0. Hence the relaxation SSR1 is an improvement over the knapsack bound. The decomposition bound quickly exceeded the LP bound, at the 3rd iteration of Procedure DECOMPOSITION BOUND.

#### 6.5.2 Comparison Between the Heuristic Bounds and the Decomposition Relaxation

Bounds from the decomposition relaxation are compared with heuristic bounds for five problems in Table 6.2. The first five columns give details of the problem as in previous chapters. For the subgradient optimization phase of the decomposition relaxation the parameter  $\delta$  was equal to 1.0. The optimal solution value to the SCP,  $z^*$ , is given in column (vi). The number of tree search nodes and computation time for a depth first tree search on rows using heuristic bounds used to find  $z^*$  is given in columns (vii) and (viii).

Lower bound information is given in columns (ix) to (xiv). The heuristic bound value at the root node of the search tree is given in column (ix) together with the computation time in column (x).

TABLE 6.1

Bound Values For The 30 x 60 Example  
Using The Decomposition Relaxation

Iteration Number	Bound Value
1	<u>49.44</u>
2	48.42
3	<u>51.58</u>
4	50.53
5	49.66
6	<u>52.39</u>
7	51.34
8	50.65
9	50.64
10	50.47
11	51.67
12	50.39
13	<u>53.05</u>
14	52.14
15	52.11
16	52.39
17	51.33
18	<u>53.35</u>
19	51.59
20	52.57
21	<u>54.63</u>
22	51.53
23	<u>55.11</u>

The LP bound is 51.0.

The SCP solution,  $z^*$ , is 56.0.

*The best bound available is underlined.*

Columns (xi) to (xiv) give the same results for the decomposition bound. The SCP constraint matrix  $A$  was divided into submatrices  $A_k$  which had at most 15 rows. The number of subgradient iterations at which the decomposition bound exceeded the heuristic bound by at least 1% of the heuristic bound is given in column (xi). Column (xiii) gives the first subgradient iteration at which the decomposition bound exceeds the heuristic bound by at least 2%. The times taken to calculate the bounds from decomposition are given in columns (xii) and (xiv). Average results for columns (xi) to (xiv) are also given.

Table 6.3 gives the computing times for the heuristic and decomposition bounds of Table 6.2 as a percentage of the total computation time (as given in column (viii) of Table 6.2).

For the two most sparse problems, numbered 71 to 72, the time taken to calculate a decomposition bound that was 1% higher than the heuristic bound was longer than the time taken to solve the SCP. However as Table 6.4 shows both of these problems were solved very quickly and the initial bound calculation using heuristics was approximately one third of the total computing time. For the most dense problem, number 74, calculating this first decomposition bound was less than 4% of the total time. For all the problems it was possible to get a bound 2% greater than the heuristic bound from the decomposition relaxation after an average of 17 subgradient iterations. However the computing time was long and even for the most dense problem was nearly a quarter of the total time.

Table 6.4 gives the best decomposition bound found in 30 subgradient iterations. Compared with the heuristic bound the value was high, but for 3 of the five problems the computation time was longer than that taken to solve the entire SCP.

TABLE 6.2

Comparison Of The Decomposition Bound And The Heuristic Bound

PROBLEM					OPTIMAL SOLUTION			HEURISTIC BOUND		DECOMPOSITION BOUND			
NO	SIZE			COST	$Z_a$	NODES	TIME	$Z_e$	TIME	BOUND EXCEEDS $Z_e$ by 1% of $Z_e$		BOUND EXCEEDS $Z_e$ by 2% of $Z_e$	
	(ii)	(iii)	(iv)							(v)	(vi)	(vii)	(viii)
70	40	150	0.08	X	92	616	36.1	86.0	0.8	10	9.3	28	22.7
71	50	100	0.06	X	146	3	1.12	141.9	0.4	4	1.5	15	5.2
72	50	100	0.08	U	11	13	3.29	9.9	1.1	23	16.3	23	16.3
73	50	100	0.10	U	10	176	21.21	7.9	1.9	1	4.2	10	19.5
74	50	100	0.15	X	81	1370	198.4	65.8	2.2	1	7.12	9	47.4
Average										8	7.7	17	22.2

TABLE 6.3

Computing Times for the Decomposition and Heuristic Bounds

PROBLEM					HEURISTIC BOUND	DECOMPOSITION BOUND	
No.	Size			Costs	Time to compute bound as % of total computing time. (Bound = $z_l$ ) (vi)	Time to compute bound as % of total computing time. (Bound = $1.01 \times z_l$ ) (vii)	Time to compute bound as % of total computing time. (Bound = $1.02 \times z_l$ ) (viii)
(i)	m (ii)	n (iii)	$\rho$ (iv)	(v)			
70	40	150	.08	X	2	26	63
71	50	100	.06	X	36	134	464
72	50	100	.08	U	33	495	494
73	50	100	.10	U	9	19	92
74	50	100	.15	X	1	4	24
Average					16	135	227

CDC 6500 sec.

MNF5 Compiler

TABLE 6.4

Decomposition Bound as a Percentage of the Optimal Solution

PROBLEM					HEURISTIC BOUND		DECOMPOSITION BOUND		Optimal Solution
No.	Size			Costs	Bound as % of $z^*$	Time as % of total time	Best Bound in 30 iterations as % of $z^*$	Time to calculate bound as % of total time	Time
(i)	$m$ (ii)	$n$ (iii)	$\rho$ (iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)
71	40	150	.08	X	93	2	97	63	36.1
72	50	100	.06	X	97	36	100	799	1.1
73	50	100	.08	U	90	33	92	494	3.2
74	50	100	.10	U	79	9	96	277	21.2
75	50	100	.15	X	81	1	98	61	198.4
Average					88	16	97	338	

CDC 6500 sec  
MNF5 Compiler

TABLE 6.5

Decomposition Bounds for Different Partitions of A

PROBLEM				HEURISTIC BOUND		DECOMPOSITION BOUND after 30 subgradient iterations					
No.	Size			Bound	Time	Max No. of Rows in $A_{\ell} = 15$		Max. No. of Rows $A_{\ell} = 10$		Max No. of Rows $A_{\ell} = 8$	
	$m$	$n$	$\rho$			Bound <sup>ℓ</sup>	Time	Bound <sup>ℓ</sup>	Time	Bound <sup>ℓ</sup>	Time
(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)	(xii)
75	30	100	.12	45.0	0.6	51.9	17.9	46.0	7.9	45.3	5.1
76	40	100	.06	122.6	0.2	125.2	6.0	124.8	4.0	124.1	2.5
77	40	100	.08	96.8	0.5	98.3	8.0	97.8	5.0	97.7	4.5
78	40	100	.10	68.8	0.7	71.5	16.3	70.9	7.0	70.0	3.5
71	50	100	.06	146.4	0.6	147.2	2.1	147.0	2.8	147.0	1.6

All the problems have randomly generated costs of type X as defined in Chapter 2.

CDC 6500 sec.

MNF5 Compiler.

Tests were carried out on a further set of five problems to ascertain the difference between bound values when the constraint matrix was partitioned into submatrices of different sizes.

The bound value when the maximum size of a submatrix took different values was calculated. If a problem had 50 rows and the maximum submatrix size was 15 then 3 matrices with 15 rows would be generated and one matrix of 5 rows. Generally if MMAX was the maximum number of rows allowed in a submatrix then an SCP with  $m$  rows was divided into  $\lceil m/\text{MMAX} \rceil$  submatrices each of MMAX rows except possibly the last which would have all the remaining rows in the original SCP. The bound value and times for MMAX = 15, 10 and 8 are shown in columns (viii) to (xii) of Table 6.5. When the matrix was partitioned into the smallest number of submatrices the bound was higher as expected, but also the computation time was higher. For problem 75 the optimal solution was 56. This problem was only split into 2 parts when MMAX was 15. Although the corresponding decomposition bound of 51.9 was considerably higher than the heuristic bound of 45.0 it was far from the optimum and took considerable time to compute.

### 6.5.3 The State Space Relaxation, SSR1, Bound Compared With the Decomposition and Heuristic Bounds

This bound is shown with the heuristic bound and the best decomposition bound in 30 iterations (when the maximum size of a submatrix was 15) in Table 6.6. In all cases the SSR1 bound was lower than the heuristic bound and in most cases slower to compute. The SSR1 bound was approximately 2/3 of the optimal solution.

### 6.5.4 Conclusions

The decomposition bound is too costly to be useful as implemented.



TABLE 6.6

Comparison between Bounds from Decomposition, State Space Relaxation 1 and Heuristics

PROBLEM					HEURISTIC BOUND		STATE SPACE RELAXATION BOUND		DECOMPOSITION BOUND (after 30 subgradient iterations)		Optimal Solution	
No.	Size				Bound (vi)	Time (vii)	Bound (viii)	Time (ix)	Bound (x)	Time (xi)	z* (xii)	Time (xiii)
(i)	(ii)	(iii)	(iv)	(v)								
75	30	150	.05	U	5.7	1.6	5	0.3	5.7	1.9	7	5.4
76	40	100	.15	X	51.0	1.5	50	0.7	54.8	102.2	64	-
70	40	150	.08	X	86.0	0.8	63	1.6	89.0	22.7	92	36.1
77	40	150	.10	U	6.4	2.1	6	0.9	6.6	42.0	8	10.5
78	50	100	.05	X	173.3	0.5	130	1.2	177.1	3.3	178	1.1
79	50	100	.06	X	141.9	0.4	100	1.1	145.5	9.0	146	1.1
71	50	100	.06	X	146.4	0.6	129	0.9	147.2	2.1	150	1.5
80	50	100	.09	U	92.9	0.9	69	1.4	94.6	4.8	98	1.7
73	50	100	.10	U	7.9	1.9	6	0.7	8.2	22.8	10	21.2
74	50	100	.15	X	65.8	2.2	62	3.2	68.5	121.0	81	198.4

CDC 6500 sec.

MNF5 Compiler

Improvements could be made to the tree search, but as both the SCP and the subproblems  $SCP_{\lambda}(d)$  were solved using the same tree search it was felt that this had little effect. However decomposition could be useful for specially structured problems that had an almost block diagonal constraint matrix. Also experiments could be made on choosing the partitions more carefully. As the bound value was quite high it is probably sufficient to get lower bounds to the subproblems  $SCP_{\lambda}(d)$  rather than solve them exactly. Also further tests using integrality of the original costs could be made as described in §6.2.4.

The state space relaxation, SSR1, bound was too low to be useful. However it may be improved by combining it with other parameters as described earlier in this chapter.

CHAPTER 7BRANCHING STRATEGIES FOR THE SCP7.1 Introduction

Branch and bound is one of the most successful approaches in solving combinatorial programming problems. As mentioned in Chapter 1 extensive studies have been made of branching strategies. Thus this chapter only briefly discusses methods applicable to the SCP.

First a simple binary tree search is used to illustrate the importance of choosing a branching variable. Three different methods of selecting the branching variable are presented and an example is shown.

The number of rows in an SCP is usually much less than the number of variables, hence three implementations that use branching on rows are described. Branching on rows is of interest for several reasons. Firstly if the constraint coefficient matrix of the SCP is stored by row as a list of non-zero columns then it is very easy to implement. Secondly the successful branching strategy of Marsten [M1] for the SPP used branching on rows. In the SPP, unlike the SCP, the fixing of a variable to 1 in constraint  $i$  means all other variables in the constraint must be fixed equal to 0. Thus at each node of the search tree several variables can be fixed. Thirdly the dynamic programming algorithm of Korman [K4] was based on branching on rows. In Korman's algorithm to transform an algorithm for the SPP to one for the SCP many extra columns had to be generated. The SCP can however be solved directly by branching on rows without generating extra columns, as described in §7.3. Thus although branching on rows has been used

previously for the SPP this is the first time that it has been used for the SCP. Fourthly fewer nodes are generated than in the simple binary search. This is of little importance as a good tree search need not attempt to find a lower bound at every node of the tree and could branch forward without calculating bounds at selected nodes to save computational time.

Topalian [T6] gives an extensive analysis of branching strategies for small SCP's.

## 7.2 Tactical Problems - Choosing The Branching Variable

The relaxation  $NF1(d)$  is used to find a bound when nodes are generated by choosing a variable  $x_j$  and fixing it first equal to 1 and then to 0. Of the many ways in which  $x_j$  can be chosen, three methods are given. In each case the variable  $x_j$  was picked from amongst the set,  $S$ , of variables that were not fixed equal to 0 or 1 by the solution to the relaxation  $NF1(d)$ . That is some variables derived from the  $j$ th column of the SCP were fixed equal to 0 and some fixed equal to 1 in the optimal solution to  $NF1(d)$ . There are usually several such variables  $x_j$  and it was to decide between these variables that the following three methods were used.

The first method is to calculate a lower bound on the amount of increase of the lower bound  $v(NF1(d))$  if a variable  $x_j$  is set equal to 0. The variable for which this penalty is maximised is then selected as the branching variable. For each variable  $x_j$ ,  $j \in S$ , a penalty  $\eta_j$  is computed as:

$$\eta_j = \sum_{\{i | \xi_{ij} = 1\}} ((\min_{j' \in N_i, j' \neq j} d_{ij'}) - d_{ij})$$

This means that if  $\xi_{ij} = 1$  then fixing  $\xi_{ij}$  equal to 0 (i.e. branching by setting  $x_j = 0$ ) will increase the bound by at least the difference between the second minimum cost,  $d_{ij}'' = \min_{\substack{j' \in N_i \\ j' \neq j}} d_{ij}'$ , and the minimum

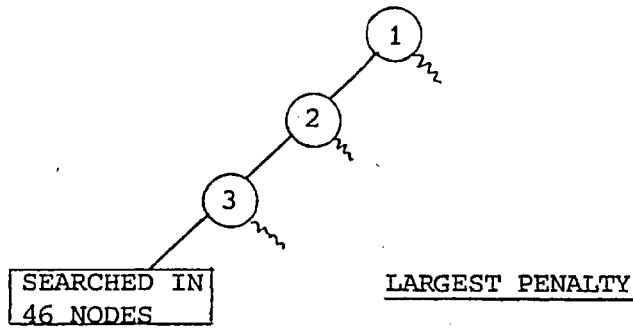
cost,  $d_{ij}$ , in row  $i$ . Thus  $\eta_j$  is a lower bound on the amount the lower bound to SCP will increase if  $x_j$  is set equal to 0. The choice of branching variable is that for which  $\eta_j$  is maximised.

Selecting the variable in  $S$  for which the reduced cost is least gives the second method. The aim is to incur a heavy penalty when this variable is set equal to 0. This may mean that another variable with higher reduced cost will then be set equal to 1. The reduced cost is already available at the end of the network flow problem and therefore it requires less computational effort to find the branching variable than the first method.

The third method is to select the variable in  $S$  with the largest reduced cost as the branching variable. The expectation is that the bound will increase as much as possible when the variable is set equal to 1. As this gives no indication of how the bound changes when the variable is set equal to 0 this is not a very good strategy.

For each of the above strategies Figures 1 to 3 show how much of the tree has been searched after 50 nodes have been generated for a  $60 \times 400$  randomly generated SCP of density 5% (problem number 15 of Tables 2.1 and 4.2). It is seen that strategy 1 is the most successful and strategy 3 the least. The number encircled is the index of each node.

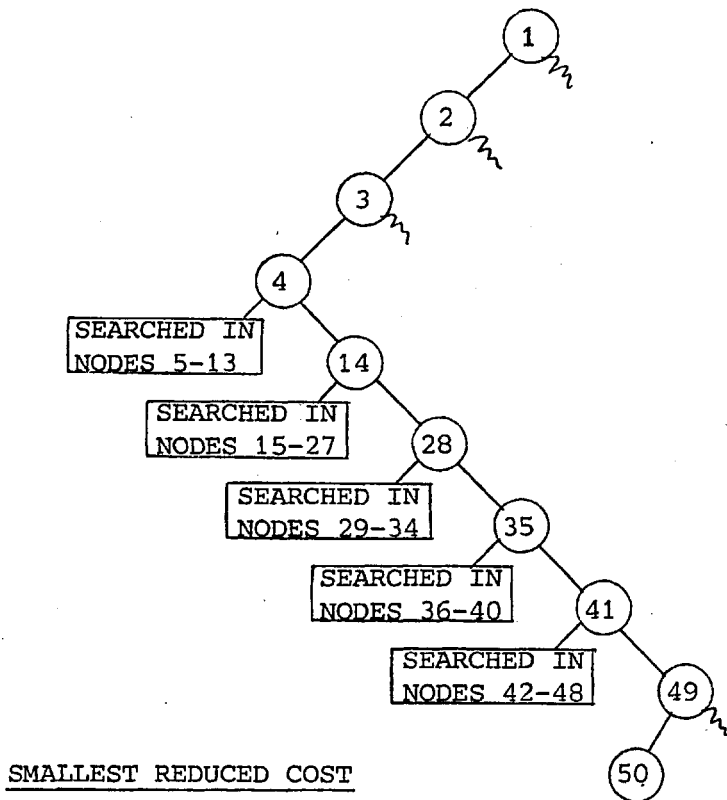
Fig.7.1 AMOUNT OF TREE SEARCHED BY STRATEGY 1



'SEARCHED' means that the nodes have been followed in a depth first tree search

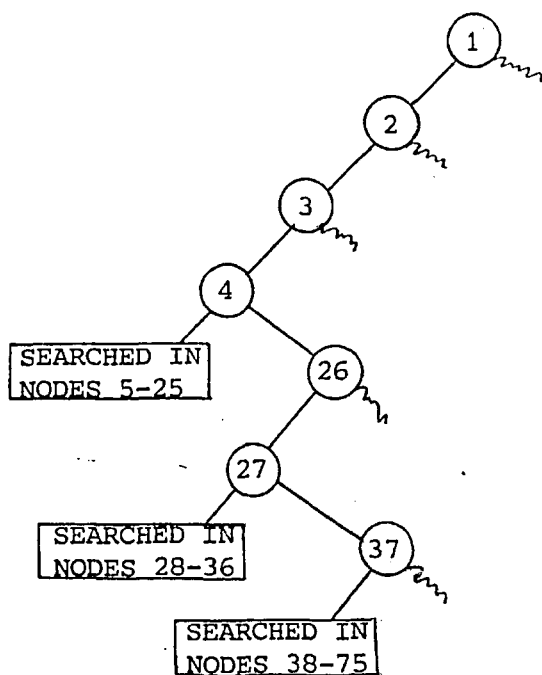
'z' means that this part of the tree is unformed

Fig.7.2 AMOUNT OF TREE SEARCHED BY STRATEGY 2



SMALLEST REDUCED COST

Fig.7.3 AMOUNT OF TREE SEARCHED BY STRATEGY 3



LARGEST REDUCED COST

7.3 Strategic Problems - Branching On Rows For The SCP

7.3.1 Description of Branching Strategy

Instead of branching on a single variable as in the computational example of section 7.2 it is possible to branch on rows or blocks. The extension of this method to the SCP will now be discussed.

This strategy was successfully used for the SPP by Marsten [M1].

Marsten sorted the constraint matrix into blocks and the extensions to the SCP are described in §7.3.2. For the SCP it is only advantageous to sort the constraint matrix into blocks when either it is stored only once as a list of non-zero rows by column or if decomposition techniques are to be used. Branching strategies for both implementations will be described.

### 7.3.2 Branching on Rows When the Constraint Matrix is Sorted into Blocks

#### 7.3.2.1 Sorting the matrix

The columns of the constraint matrix  $A$  are sorted into blocks,  $B_k$  ( $1 \leq k \leq m$ ) say, some of which may be empty. Block  $B_k$  is the set of columns of  $A$  that have the first non-zero entry in row  $k$ . The sorted matrix is shown below.

FIGURE 7.4

Constraint Matrix  $A$  Sorted Into Blocks

	$B_1$	$B_2$	$B_3$	$B_4$	...	$B_r$
$A$	11111	All entries are 0				
	All entries are 0 or 1					

#### 7.3.2.2 The branching strategy

If the matrix  $A$  is partitioned into blocks then, unlike the case of the SPP where at most one variable from each block can equal 1, it is possible in the SCP for more than one variable in a block to be set equal to 1. Thus the branching strategy will record only the block of highest index which covers each row. The  $m$ -dimensional vector  $g$  is used to record the block covering a row. Row  $i$  is covered by block  $k$  means that  $g(i) = k$  and  $g(i) = \infty$  if row  $i$  has not not been assigned to a block. In the procedure given by Marsten for the set-partitioning problem the blocks are assigned to rows in



increasing order of row index, which gives a depth-first strategy. The method presented here can be used for any strategy.

Let  $B_k^g(i)$  be the set of variables not fixed equal to 0 or 1 in block  $B_k$  which have a 1 in row  $i$ . Consider assigning row  $(r+1)$  to a block  $B_k$ , for  $1 \leq k \leq r$ , giving  $g(r+1)$  equal to  $k$ . First notice that if  $g(k) \neq k$  or  $\infty$  then it is impossible to assign row  $r+1$  to  $B_k$ . If  $g(k) = \infty$  then  $g(k)$  must immediately be set to  $k$ . If  $B_k^g(r+1) = 0$  then it is also impossible to set  $g(r+1) = k$ . Suppose that row  $r+1$  can be assigned to block  $B_k$  then it may be possible to reduce the size of problem to be solved by removing rows, columns or elements of the constraint matrix. This is described in the next three sections.

#### 7.3.2.3 Removal of rows

To remove rows, only columns of Block  $B_k$  need be considered. Recall  $B_k^g(r+1) = \{j | x_j \in B_k, a_{r+1,j} = 1\}$ . Any row  $t$ ,  $k \leq t \leq m$  and  $t \neq r+1$  for which  $a_{kj} = 1$  for all  $j \in B_k^g$  can be removed. This is the row dominance test of §1.3.3.1 applied to a block instead of a row.

#### 7.3.2.4 Removal of columns or blocks

Columns can be removed in two ways. First consider the case when  $B_k^g(r+1)$  consists of a single element,  $j_0$  say. Then  $x_{j_0}$  can be set equal to 1 and rows removed as in §7.3.2.3. Secondly if  $g(r+1) = k$  then  $x_j = 0$  for all  $j$  satisfying  $a_{kj} = 1$  and  $j \in B_{k'}$ ,  $k < k' \leq r+1$ . In particular all the variables in block  $B_{r+1}$  are removed.

#### 7.3.2.5 Removing coefficients of constraints

Not only can variables and rows be removed as in the preceding two sections, but if row  $r+1$  is assigned to block  $B_k$  then  $a_{r+1,j}$  can be

set equal to 0 for all  $j$  in blocks  $B_t$ ,  $1 < t < k-1$ . This means that the problems tend to become sparse as one proceeds down the search tree. For lower values of  $k$  not many coefficients will be set equal to 0, but on the other hand several blocks of variables can be removed. On the other hand for larger values of  $k$ , whereas few blocks will actually be eliminated, the subproblem is likely to be easily solved, because it is sparse.

#### 7.3.2.6 Solving the problem after assigning blocks to all rows

After blocks have been assigned to all the rows there may not be a unique solution to the SCP. At each stage of the branching a relaxation of SCP is used to find a lower bound. If the relaxation solves SCP or provides a lower bound which exceeds an upper bound then backtracking can take place. In the case of no solution to SCP after the branching strategy has been completed then smaller SCP's will have to be solved. For each block  $B_k$  which has not been completely determined and has been assigned to some row let  $\beta(k) = \{i | g(i) = k\}$  be the set of rows to which it has been assigned. Then solving the SCP:

$$\begin{aligned} \min \quad & \sum_{j \in B_k^g} c_j x_j \\ \text{subject to} \quad & \sum_j a_{ij} x_j \geq 1 \quad i \in \beta(k) \\ & x_j = 0 \text{ or } 1 \end{aligned}$$

will give the required solution.  $B_k^g$  is equal to  $\sum_{r=k}^m B_k^g(r)$ .

#### 7.3.2.7 Example

For the example all costs,  $c_j$ , equal 1 and it is assumed that the

matrix has been sorted. The rows are chosen in ascending order of index. The constraint matrix is shown below.

$A =$

Block	1		2			3				4				
Column	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Row														
1	1	1												
2			1	1	1									
3	1		1			1	1	1	1					
4		1		1	1			1		1	1	1	1	1
5	1		1			1							1	
6					1		1	1		1		1		
7		1				1					1			
8							1					1		1

An upper bound is given by  $x_1 = x_4 = x_7 = x_{11} = 1$  and has value  $z_u = 4$ .

The search tree is given in Figure 7.5.

The simple lower bound  $z_l$  is calculated as:

$$z_l = \sum_{k \in K} \min_{j \in B_k} c_j$$

where  $K^a$  is the index set of blocks that have been assigned to a row.

The cost of the fixed solution is denoted by  $z_F$ .

The following calculations are carried out for each node:

Root Node:

Node 1

$$g(1) = 1$$

Branch from Node 1:

Node 2

$$g(2) = 2. \quad \text{Note, since } B_1(2) = 0, g(2) \text{ cannot equal 1.}$$

Branch from Node 2:

Node 3

$$g(3) = 1. \quad \text{Set } x_1 = 1 \text{ and remove } x_3 \text{ and } B_3, z_F = 1. \\ \text{Rows 1, 3 and 5 are covered by } x_1.$$

Node 4

$g(3)=2$ . Set  $x_3=1$ , remove  $B_3$ ,  $z_F=1$ . Rows 2, 3 and 5 are covered by  $x_3$ .

Node 5

$g(3)=3$ ,  $z_L=3$

Branch from Node 3:Node 6

$g(4)=1$ , remove  $B_4$ , infeasible

Node 7

$g(4)=2$ , remove  $B_4$ , infeasible

Node 8

$g(4)=4$ ,  $z_L=3$

Branch from Node 4:Node 9

$g(4)=1$ , remove  $B_4$ , infeasible

Node 10

$g(4)=2$ , remove  $B_4$ , infeasible

Node 11

$g(4)=1$ ,  $z_L=3$

Branch from Node 5:Node 12

$g(4)=1$ , set  $x_1=1$ . Remove  $x_4, x_5, x_8, B_4$ .  
In addition single 1 in row test.  
Sets  $x_7=1$  and  $x_3=1$ . Upper bound = 3.

The tree search is completed as a new upper bound has been found.

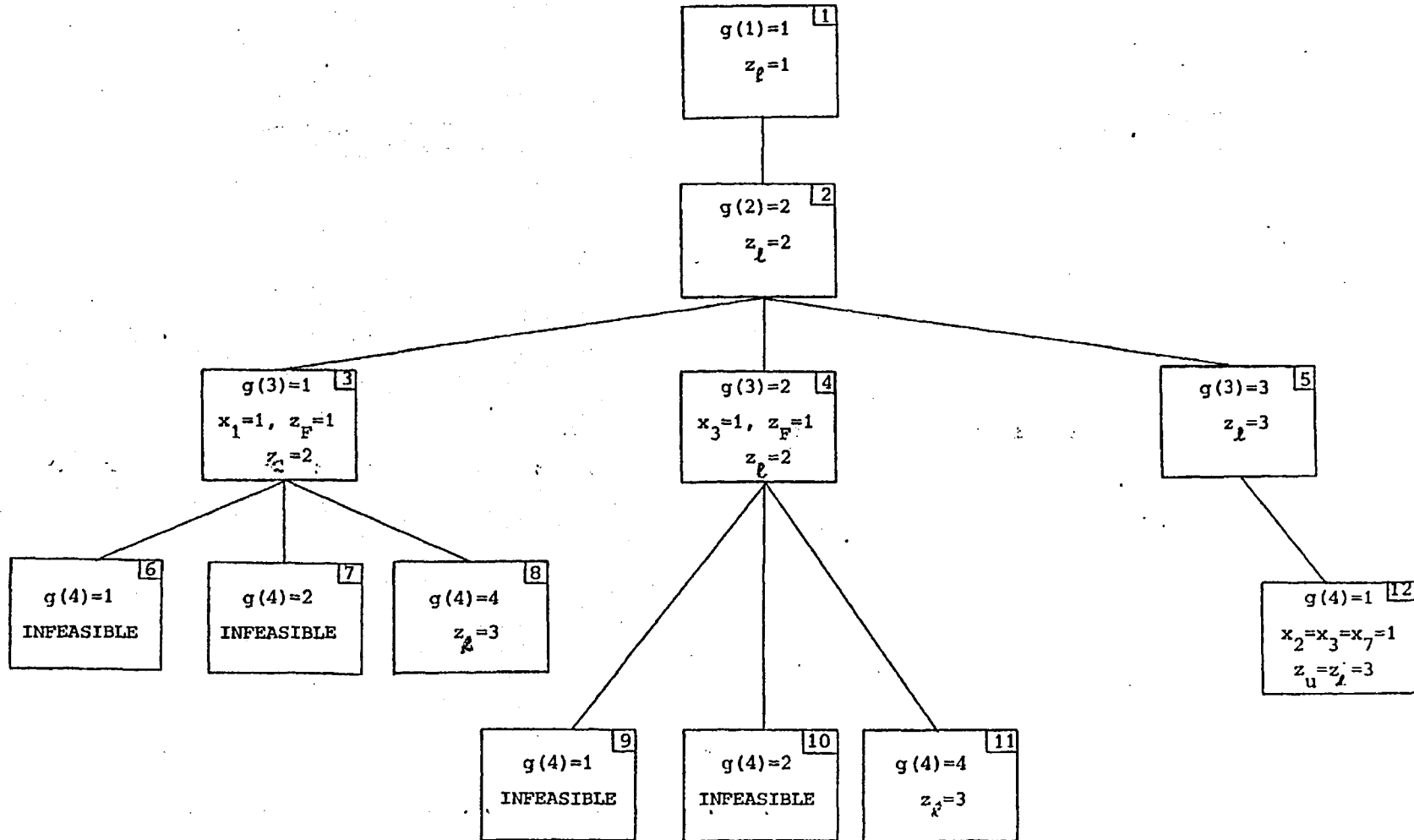
### 7.3.3 Branching on Rows When the Constraint Matrix is Stored as a List of Non-Zero rows by Column

#### 7.3.3.1 Description of the forward step of the branching strategy

The basic forward step to generate successor nodes is made by choosing a row  $i$  at a node of the search tree. Suppose  $N_i$ , the set of non-zero columns of the SCP contained in row  $i$ , is the set  $\{j_1, j_2, \dots, j_p\}$ .

This set is readily available if the constraint matrix is stored as a

Fig. 7-5 TREE SEARCH FOR EXAMPLE





LREP: =  $\phi$                       LREP is the set of variables fixed  
in the current search tree

IZ: =  $\phi$                             IZ is the cost of the variables fixed  
in the search tree

MR: =  $\phi$                             MR is the set of rows removed at the  
current node of the search tree

BIG: = big number

$\epsilon$ : = small number, tolerance

NODE: = 0                            NODE is the number of tree search nodes

NODEMAX: = maximum number of tree search nodes that can be examined

MRN: = vector of rows covered at each level by fixed solution

## 2. Calculate Lower Bound To The SCP

$z_l$ : = lower bound to the SCP

$u$ : = corresponding dual variables

$z_u$ : = upper bound to the SCP

If node is fathomed

then goto 4.

else goto 3.

## 3. Forward Step, Create A New Tree Search Node

NODE: = NODE + 1                    Increase the number of nodes examined

If (NODE > NODEMAX) goto 7.      Test for limit on number of nodes  
generated

LEV: = LEV + 1                      Increase the depth of the search tree

### 3.1 Calculate Reduced Costs

For  $j$ : = 1 to  $n$

if  $s_j \geq z_u - z_l - 1 + \epsilon$       Test the reduced costs

then  $c_j = c_j + \text{BIG}$





#### 4. Backward Step, Remove Branching Variable, JBR

Branching variable is  $j$  where  $j := \text{JBR}$

$\text{IZ} := \text{IZ} - c_j$                       Set  $x_j$  equal to 0

$\text{MR} := \text{MRN}(\text{LEV})$

goto 3.4.

#### 5. Decrease Level In Search Tree

$\text{LEV} := \text{LEV} - 1$

If  $\text{LEV} = 0$

then goto 6.

else  $\text{NREP} := \text{NRN}(\text{LEV})$                       Recalculate fixed rows and columns

$\text{MR} := \text{MRN}(\text{LEV})$                       at this level of the tree search

$\text{LREP} :=$  set of variables removed at level LEV

goto 4.

#### 6. Tree Search Completed Successfully

Stop with optimal solution. Tree search has been completed.

#### 7. Tree Search Terminated Because Too Many Nodes have Been Generated

Stop.

#### 7.3.3.3 A best bound implementation

The main reason for studying this approach to a branch and bound algorithm was so that dominance tests could be used to eliminate nodes.

The dominance tests consider two nodes in the search tree, NODE1 and

NODE2 say.  $\text{MRN}(\text{NODE})$  is the set of rows covered by the variables fixed

equal to 1 and  $IZ(NODE)$  is their cost. Then if  $IZ(NODE1) \geq IZ(NODE2)$  and  $MRN(NODE1) \subseteq MRN(NODE2)$   $NODE1$  is dominated by  $NODE2$  and can be removed from the problem because  $NODE2$  covers more rows for less cost than  $NODE1$ . Hence when a node,  $NODE$ , is generated the corresponding values of  $MRN$  and  $IZ$  were stored. Also an estimate of the lower bound value at the node,  $\hat{z}_\ell$ , is stored. This can be obtained by taking  $z_\ell$  the lower bound at the father node and the branching variable  $x_j$  that generated  $NODE$ . Then if  $u$ , corresponding to  $z_\ell$ , is the vector of dual variables, in the case of the heuristic lower bounds, and is the vector of vertex weights (including blossom weights), in the case of the graph covering relaxations, then  $\hat{z}_\ell = z_\ell + \max(0, c_j - \sum_{i=1}^m u_i a_{ij})$ . The node at which the next lower bound is calculated in a tree search strategy is the one for which  $z_\ell$  is least. A second reason for this approach is that if the algorithm terminates prematurely the least bound of unfathomed nodes remaining in the problem gives a lower bound on the solution to the SCP.

When a node is not fathomed all the successor nodes are generated by choosing a branching row  $i$  and storing information for all possible nodes that can be generated as in step 3 of PROCEDURE 15 DEPTH FIRST SEARCH.

The tree search strategy is described in PROCEDURE 16 BEST BOUND SEARCH below.

#### PROCEDURE 16 BEST BOUND SEARCH (SCP)

##### A BEST BOUND SEARCH ON ROWS FOR THE SCP

Input: SCP

The set covering problem

### 1. Initialise Variables

$z_F$  : = 0 cost of fixed variables at root node  
 MR : = 0 set of rows covered at root node  
 MRN : = MRN is vector of rows removed at current node of search tree  
 BIG : = large number,  $\epsilon$ : = small number, tolerance  
 NODE : = 0 index of current node being examined  
 NODEMAX: = maximum number of nodes that can be stored  
 LD : = list of nodes stored

### 2. Calculate Lower Bound To The SCP

$z_l$  : = lower bound to the SCP  
 $u$  : = corresponding dual variables  
 $z_u$  : = upper bound to the SCP  
If node is fathomed  
     then goto 5.  
     else goto 3.

### 3. Forward Step

#### 3.1 Calculating Branching Row

Find a row  $i$  on which to branch

#### 3.2 Generate Subnodes

NODEX: = NODE

For all  $j \in N_i$

if  $s_j \leq z_u - z_l - 1 + \epsilon$

    let NODE: = first free position in list LD

```

if NODE > NODEMAX
  then goto 6.
  else father node(NODE) := NODEX      Record father node
      bound estimate(NODE) :=  $z_\ell$     Calculate estimate of
                                          lower bound
      MRN(NODE) := MRU  $M_j$           Calculate rows covered
                                          by fixed variables
      IZ(NODE) :=  $z_F + c_j$          Calculate cost of fixed
                                          variables
      LD := LDU{j}
      Branching row(NODE) :=  $i$ 
      Branching variable(NODE) :=  $j$ 
                                          Perform dominance tests

      If NODE is dominated
      do not store node

      If NODE dominates other
      nodes in list LD remove
      them from LD

```

#### 4. Choose A New Node At Which To Calculate Bound

Let NODE be the node for which  
the lower bound estimate is least

$z_F$  := IZ(NODE)                      Retrieve cost of fixed variables

MR := MRN(NODE)                      Retrieve rows covered by fixed  
variables

Generate subproblem of SCP using  
branching row, branching variable  
and father node to set:

$c_j$  :=  $c_j$  + BIG for variables fixed  
equal to 0 or 1

Goto 2.

### 5. Backward Step

Remove NODE from list LD. Also if the father node of NODE has no successors left remove father node from LD. Repeat on the father node of the father node until no more nodes can be removed from LD.

Goto 4.

### 6. Storage Allocation Exceeded

Stop with tree search not completed. A lower bound on  $v(\text{SCP})$  is available from the least lower bound estimate of the active nodes.

### 7. Stop With The Optimal Solution

$z_u$  is an optimal solution to the SCP.

In practice dominance tests were only rarely useful. They were used to remove two or three nodes in problems of about 40 rows and 150 columns and density 0.15. Larger problems could not be solved by the tree search.

#### 7.3.4 Improving the Branching on Rows Method

Unless calculating a lower bound has a good chance of either fathoming a tree search node or removing variables, using reduced costs to prevent a large number of successor nodes being generated, it is probably quicker to branch forward and generate successor nodes than calculate the bound. Thus improvements to computing times could be made by not calculating a lower bound at each node.

A problem with the branching on rows method is that once a branching row has been chosen every single variable in the row is fixed to 1.

It may be that when some of the variables in a row, say those that are most likely to be in an optimal solution to the SCP, have been fixed equal to 0, that it is unnecessary to fix any of the remaining variables equal to 1. To implement this suppose, for row  $i$ ,  $N_i$  consists of variables  $\{j_1, j_2, \dots, j_p\}$ . Then setting:

$$x_{j_1} = 1$$

$$x_{j_1} = 0, \quad x_{j_2} = 1$$

$$x_{j_1} = 0, \quad \dots \quad x_{j_\ell} = 1 \quad \text{say}$$

$$\text{and } x_{j_1} = 0, \quad \dots \quad x_{j_\ell} = 0$$

would partition the SCP into  $\ell+1$  nodes instead of  $p$  as before where  $\ell$  is chosen so that  $\ell < p$ . So that the variables  $x_{j_1}, x_{j_2}, \dots, x_{j_\ell}$  are those likely to be in the optimal solution when a branching row  $i$  is chosen the set  $N_i$  can be sorted in ascending order of  $c_j/h_j$  where  $h_j$  is the sum of non-zero entries in column  $a_j$  that are in rows not covered by fixed variables.

#### 7.4 Computational Results For The Depth First and Best Bound Tree Searches On Rows

##### 7.4.1 Case Study

The same 30 x 60 problem used previously was studied in detail. The branch and bound trees for the depth first and best bound strategies are shown in Figs. 7.6 and 7.7. The branching variables and lower bound values are shown for each node. In both cases an estimate of the heuristic lower bound  $\hat{z}_\ell$  was available, thus it was not always necessary to use the heuristics to calculate the bound at a node. For this example the optimal solution is  $x_4 = x_{12} = x_{27} = x_{36} = x_{41} = x_{15} = 1$  and the remaining  $x_j = 0$ . The node at which this solution

FIGURE 7.6

The Branch and Bound Tree for the Best Bound Strategy For a 30 x 60 SCP of Density 0.15

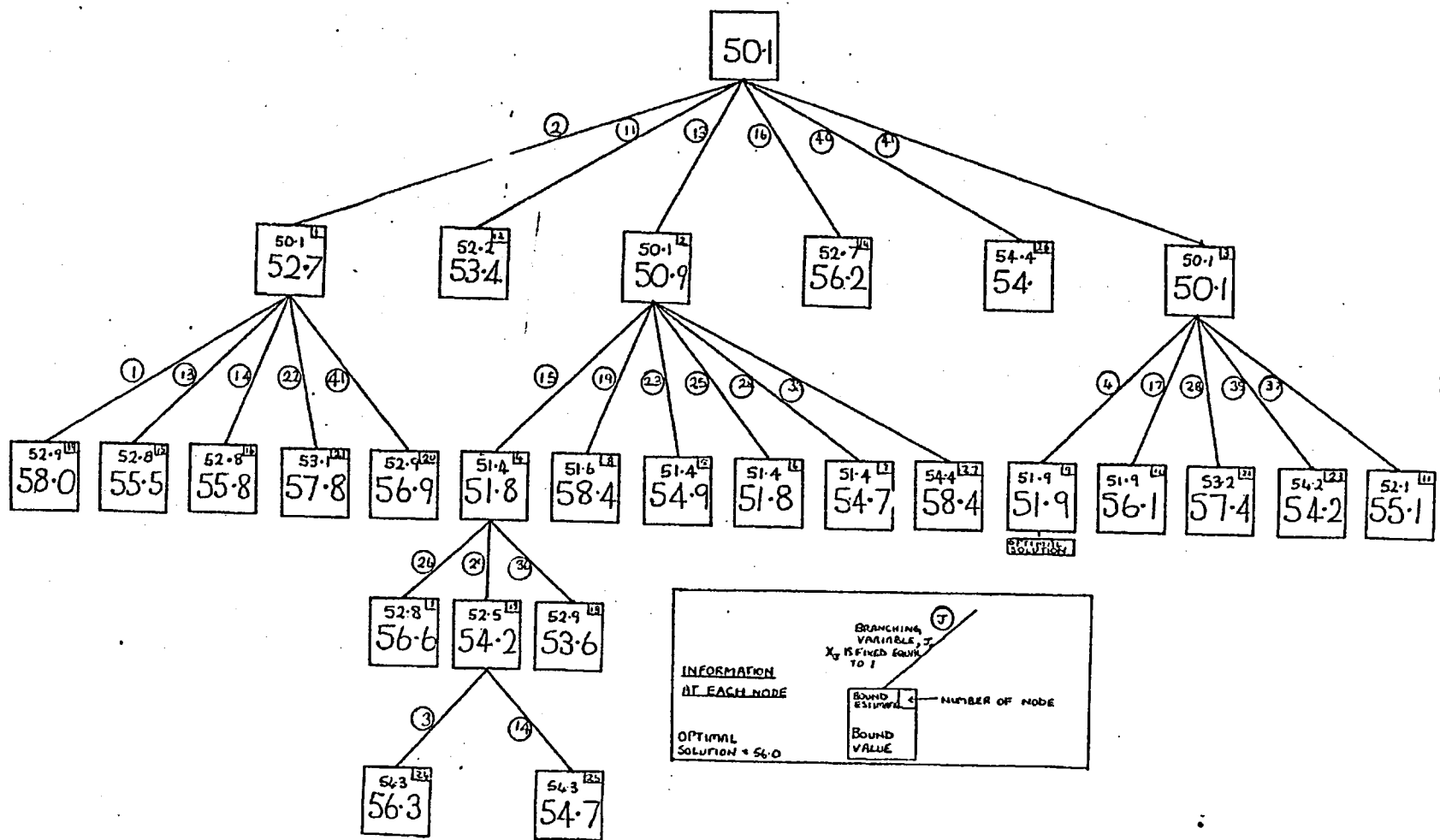
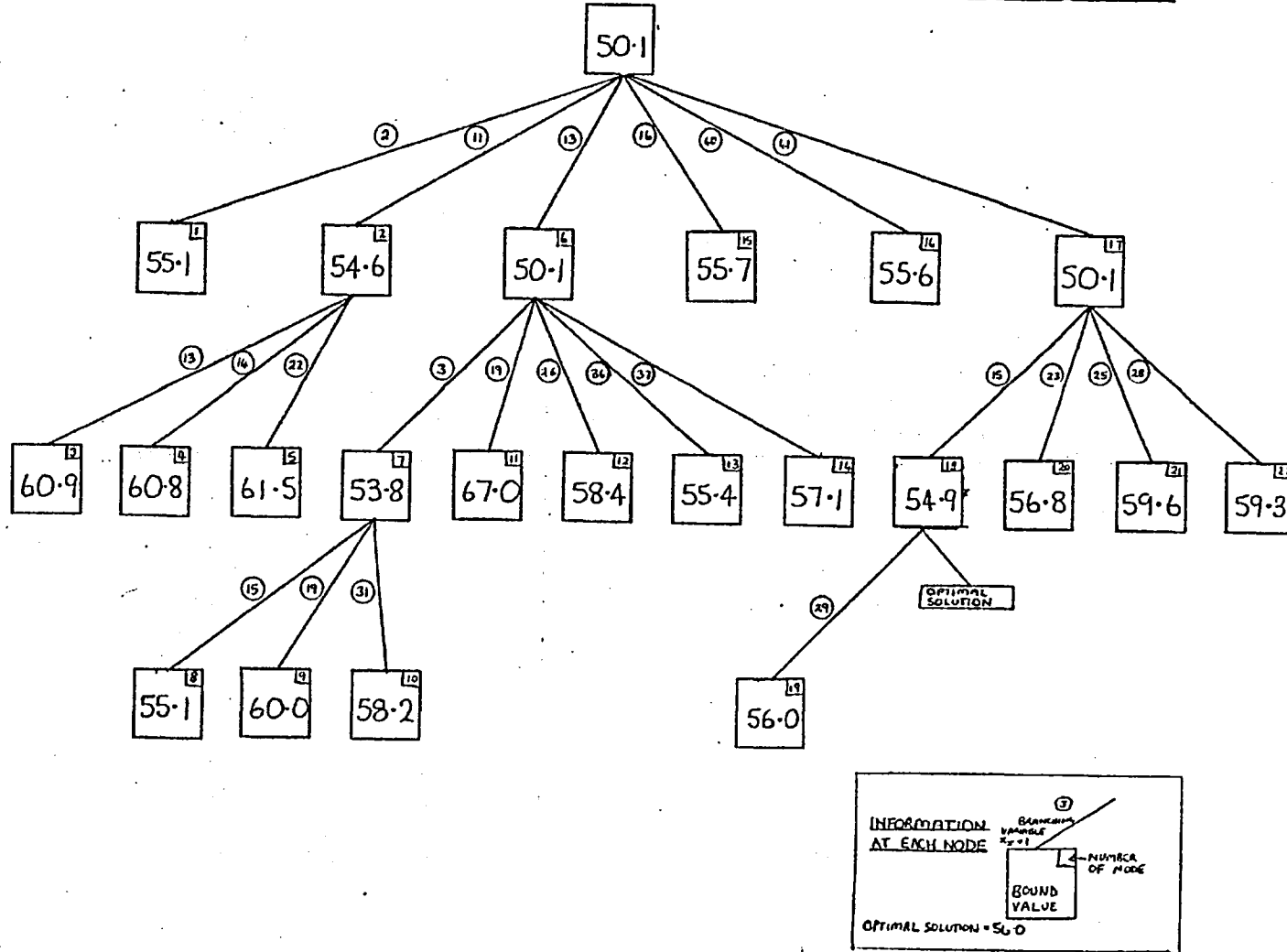


FIGURE 7.7

The Branch and Bound Tree for the Depth First Strategy for a 30 x 60 SCP of Density 0.15





is found in the depth first tree search is given by branching on  $x_{41} = 1$  at the root node and  $x_{15} = 1$  at the first level. No successor node is generated from the node at the second level because the bound estimate was greater than 55. Dominance tests were not used in the best bound strategy to reduce the number of nodes searched.

#### 7.4.2 Test Problems

Fifteen randomly generated problems were tested. The results in Table 7.8 show that the average time to perform the best bound search was 8.0 sec. whereas for the depth first search it was 14.1. The number of nodes in the tree for the best bound search was 81 which was less than that for the depth first search that had on average 164 nodes. For dense problems the differences were greater and this may have been because dominance tests were used in the best bound search.

The best bound search also tended to find the optimal solution earlier in the search than the depth first method. The average time to find the optimal solution, shown in Table 7.9, was 4.5 (CDC 6500) sec. for the best bound search and 10.3 sec. for the depth first search. The time taken to find the optimal solution as a percentage of total computing time is also shown in Table 7.9. The best bound search found the optimal solution half way through the search on average. The depth first search found 2\* in about three quarters of the total time on average. There were wide deviations between the average results and the results for individual problems however the best bound search was nearly always faster and generated fewer nodes.

The maximum number of nodes stored by the best bound search was close to the total number of nodes examined as columns (vii) and (viii) of Table 7.8 shown. It would therefore be worthwhile to use pointers to

TABLE 7.8

The Number of Nodes generated by the Depth First and Best Bound Tree Searches using Branching on Rows

PROBLEM				Optimal Solution z*	Heuristic Bound as % of z* 100 x z <sub>2</sub> / z*	Best Bound Search			Depth First Search		
No.	m	n	ρ			Max. No. of Nodes Stored	Total No. of Nodes	Total Time	Max. Depth of Search Tree	Total No. of Nodes	Total Time
(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)	(xii)
81	30	100	.12	53	85	46	61	3.9	4	109	5.2
82	40	100	.10	75	89	64	78	5.6	5	102	6.0
83	40	100	.12	68	87	135	146	9.2	6	240	10.7
70	40	150	.08	92	93	53	64	6.6	6	616	36.1
84	40	150	.08	90	97	67	74	4.6	2	22	2.5
85	40	150	.10	67	93	104	110	10.7	5	175	14.0
86	40	150	.10	69	94	72	72	4.6	5	213	19.0
71	50	100	.06	146	90	3	3	1.1	2	3	1.1
87	50	100	.08	126	88	111	175	24.8	5	186	19.8
88	50	100	.08	112	90	21	26	2.8	5	39	3.5
89	50	100	.08	111	90	66	81	8.0	5	72	7.2
90	50	100	.08	103	88	65	92	10.7	8	328	18.9
91	50	100	.10	89	91	49	69	11.3	4	65	13.5
92	50	100	.10	86	91	32	47	7.7	5	103	13.5
93	50	100	.10	91	90	116	120	11.2	8	194	18.2
Average					90.4		81	8.0		164	14.1

Table 7.9 To Show the Time Taken to find the Optimal Solution by the Depth First and Best Bound Tree Searches

PROBLEM				OPTIMAL SOLUTION	HEURISTIC LOWER BOUND (as % of optimal solution)	DEPTH FIRST		BEST BOUND	
No. (i)	m (ii)	n (iii)	$\rho$ (iv)			Time taken to find $z^*$ (vii)	Time taken to find $z^*$ as % of total search time (viii)	Time taken to find $z^*$ (ix)	Time taken to find $z^*$ as % of total search time (x)
				$z^*$ (v)	$100 \times z^e / z^k$ (vi)				
81	30	100	0.12	53	85	1.2	31	1.7	33
82	40	100	0.10	75	89	4.7	84	5.8	97
83	40	100	0.12	68	87	6.7	73	9.8	92
70	40	150	0.08	92	93	1.9	29	35.3	98
84	40	150	0.08	90	97	3.4	74	1.5	60
85	40	150	0.10	67	93	6.6	62	11.4	81
86	40	150	0.10	69	94	2.6	57	18.8	99
71	50	100	0.06	146	90	0.9	86	0.9	84
87	50	100	0.08	126	88	1.4	6	19.1	96
88	50	100	0.08	112	90	1.7	61	2.0	57
89	50	100	0.08	111	90	0.7	9	0.7	10
90	50	100	0.08	103	88	2.7	25	18.2	96
91	50	100	0.10	89	91	2.2	19	2.2	24
92	50	100	0.10	86	91	2.5	32	11.8	87
93	50	100	0.10	91	90	7.8	70	15.7	86
Average					90.4	4.5	47.8	10.3	73.3

successor nodes so that nodes are no longer stored once all their successors have been fathomed. It should also be realised that the computational work in redefining a subproblem was greater for the best bound search than the depth first. If the methods of defining the subproblem were improved the computation times would be even quicker for the best bound search. Details of how this can be done for the travelling salesman problem are given in Carpaneto and Toth [c2].

CHAPTER 8IMPLEMENTATION OF AN ALGORITHM FOR SOLVING  
THE SCP USING GRAPH COVERING RELAXATIONS8.1 Introduction

A FORTRAN program for solving the SCP using the graph covering relaxations of Chapter 5 embedded in a tree search is described in this chapter. The design of the program and the data structures used for storing the SCP are discussed in §8.2 and §8.3. The solution of the graph covering problem and reduced cost tests are described in §8.4. Parameters of the problem are discussed in §8.5.

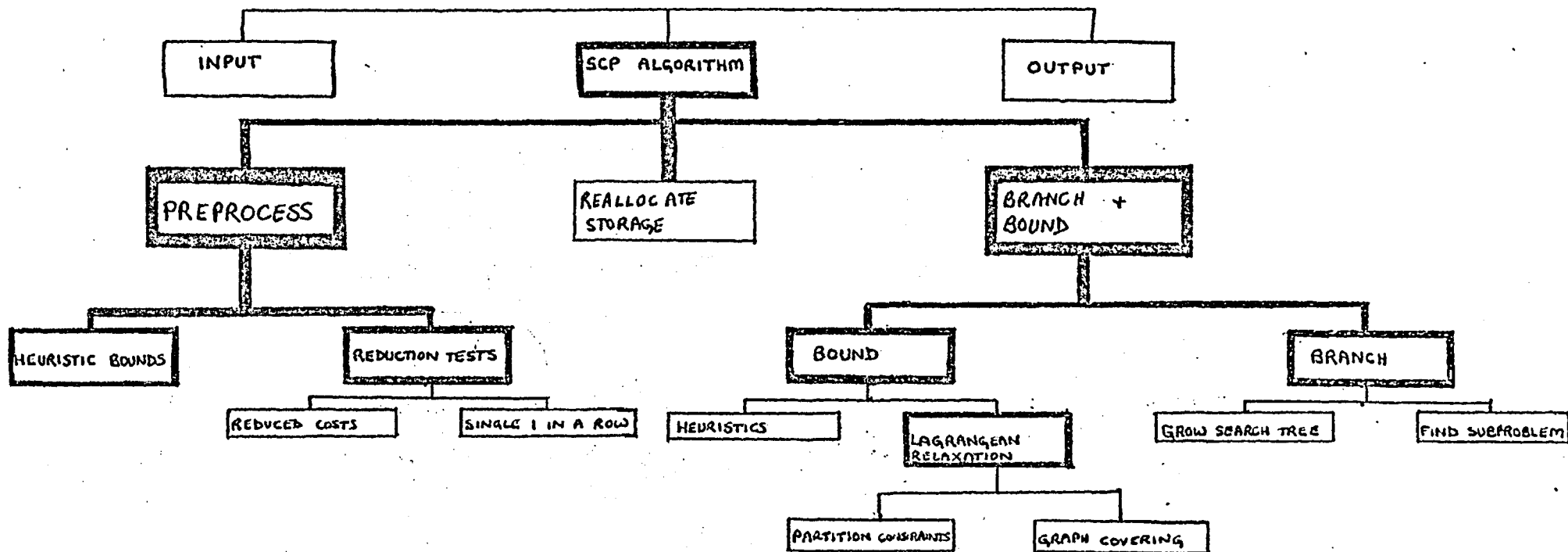
8.2 Design Of A FORTRAN Program

The aim of this research was to produce a program that could easily be modified so as to allow a wide variety of branching and bounding strategies to be tested. The program is run entirely in-core. The amount of available storage proved to be a limitation and hence the ability to pack arrays or read data from the disc would have to be added to enable large problems (of greater than 10000 non-zero entries in the constraint matrix  $A$ , say) to be solved. There are three main sections to the code as shown in Fig.8.1.

The first is the preprocessing stage in which the heuristics of Chapter 2 are used to reduce the size of the problem. Only two reduction tests were found to be particularly useful for random problems. These were the reduced cost tests and the single 1 in a row test of §1.3.3.1.

FIGURE 8.1

Design Of A FORTRAN Computer Program For The SCP



If the size of the problem decreases after the pre-processing stage then the columns that have been removed from the problem are discarded completely and the storage reallocated. This has to be done because the Lagrangean relaxation and branching strategy use much more storage than the heuristic algorithms.

Thirdly there is the branch and bound phase which is divided into three parts:

- (a) calculating a lower bound
- (b) branching forward
- (c) backtracking.

### 8.3 Data Structures For The Graph Covering Algorithms

#### 8.3.1 The Set Covering Problem

To store the costs of the set covering problem a single vector of dimension  $n$  is required. The constraint matrix is stored twice, both by row and by column to speed up the computation. The matrix is stored as a list of non-zero columns in a vector ITJ together with a pointer vector of dimension  $m$ , IP, which gives for each row  $i$  the total number of elements stored in ITJ up to the end of row  $i$ . This is illustrated in Appendix 3. Likewise a vector JTI which lists the non-zero rows is stored and JP is a vector of dimension  $n$  that points to the end of each column. Hence the total storage for the costs  $c$  and the constraint matrix  $A$  is approximately  $2mp + 2n + m$  where  $\rho$  is the density of the constraint matrix.

To mark columns that have been removed the cost  $c_j$  is set equal to

$c_j + \text{BIG}$  where BIG is a suitably large number. So that columns that have been removed can be added back to the problem at a later date a vector LREP of maximum dimension  $n$  gives the index of each column removed. This can store a negative number,  $-j$ , if  $x_j$  is set equal to 0 and  $j$ , a positive number, if  $x_j$  is set equal to 1.

To indicate which rows have been removed a vector ILK of dimension  $m$  is required. This is a linked list of rows currently in the SCP and ILK(I) is set equal to a negative number if the row has been removed. This vector allows rows that are in the SCP to be accessed quickly and also enables one to determine whether or not a row  $i$  has been removed by testing the sign of ILK(I). In addition the first row of the SCP must be stored.

The storage required to mark whether rows or columns have been removed is  $1 + m + n$ . The total storage required for an SCP in which some rows and some columns may be removed is about  $2mnp + 3n + 2m + 1$ .

### 8.3.2 Lagrangean Relaxation

#### 8.3.2.1 Graph covering relaxation GCR1, the row relaxation

Indices of rows removed in a Lagrangean relaxation of the SCP can be stored in a list LREL, of dimension at most  $m$ . They are also marked as removed in the vector ILK. The Lagrange multipliers require  $m$  words of storage and in addition it is convenient to store their sum. There is no need to store both costs of the relaxed problem and  $c_j$  as one can be calculated from the other.

In testing the feasibility of the graph covering solution for the SCP firstly a list LCD, of length at most  $m$  (in the absence of negative costs in the GCP), of indices of columns of the SCP corresponding to



arcs in the graph covering solution and the lower bound value is needed.

A list of relaxed rows for which  $u_i(a^i x - 1) \neq 0$  is stored, together with a search direction,  $v$ , and steplength,  $\sigma$ . Thus the additional storage required for this Lagrangean relaxation excluding the amount required to solve the graph covering problem is at most  $3m + 2$ .

### 8.3.2.2 Graph covering relaxation, GCR2, the column relaxation

A key to the arcs that are in the graph covering problem of length at most  $\sum_{j=1}^n \lceil \bar{h}_j / 2 \rceil$ , which is less than  $(mnp + \bar{n}/2)$ , is stored in the vector JLK.

If arc  $k$  is the first arc to be derived from column  $j$  then  $JLK(K) = J$ . Otherwise each time an arc that is not the first arc to be derived from a column  $j$  is created a counter NNOW, that starts at  $n$ , is increased by 1. Then  $JLK(K) = NNOW$ . Thus if arcs are generated from each column  $j$  in turn the list JLK is a list of integers and each time an integer less than or equal to  $n$  appears in the list the corresponding arc is derived from a new column of the SCP. Hence for an SCP with two columns  $a_1$  and  $a_2$  below:

1	1
0	1
1	0
1	0
1	0
1	1
1	0

Column  $a_1$  would give 3 arcs and column  $a_2$  would give 2 arcs hence JLK would equal (1, 3, 4, 2, 5).

As in GCR1 a list of arcs LCD, equal to (1, 2, 3, 4) say, and lower bound value is given by the graph covering algorithm. To indicate that an arc is in the graph covering solution the corresponding component of JLK is then multiplied by -1. In this case JLK would equal (-1, -3, -4, -2, 5). It is then easy to count the number of arcs derived from a particular column that are in or not in the graph covering solution. For the example arcs 1, 2 and 3 derived from column  $a_1$  are all in the graph covering solution thus  $x_1$  in the SCP can be set equal to 1. The cost vector of arcs in the GCP is of the same length as JLK. The costs are changed in the subgradient optimization phase by using a list of columns of the SCP JCH from which at least one arc in the graph covering solution has been derived. For each such column both the total number of arcs in the GCP derived from the column, NCH, and the number of arcs that are in the graph covering solution, NWON, are stored. If NWON and NCH are equal then the feasibility conditions for the SCP are satisfied. Otherwise it is easy to calculate the penalties with which to change the costs in subgradient optimization from these parameters. The maximum storage required for this relaxation is then approximately  $mnp + n$  for JLK and the costs of the relaxed problem,  $m + 1$  for the graph covering solution plus  $3m$  for the information needed to change the costs. The total storage is then  $mnp + 4m + 1$ .

### 8.3.3 Branching Strategies

#### 8.3.3.1 Depth first search on rows

The depth first search branching on rows of §7 requires firstly that variables fixed equal to 1 and 0 are marked. This can be done using the list LREP of §8.3.1. In addition LEV is a variable that gives the depth of the search tree. IBR(LEV) gives the branching row at

level LEV. The position KST(LEV) at which the branching variable is stored in the list ITJ is registered. This means that  $J = ITJ(KST(LEV))$  gives the index of the variable  $x_j$  that is fixed equal to 1 and  $ITJ(K1), \dots, ITJ(KL)$  where  $K1 = IP(IBR(LEV) - 1) + 1$  and  $KL = KST(LEV) - 1$  gives variables fixed equal to 0. (The convention  $IP(0) = 0$  is used). In addition it is necessary to record KREP(LEV) the number of variables stored in LREP at level LEV in the tree search. If LEVMAX is the maximum depth of the search tree the additional storage is thus  $3 \text{ LEVMAX}$ . Additionally the cost of the fixed variables at each level can be stored to speed up computation. Also the rows that are covered at each level can be stored in bits.

#### 8.3.3.2 Best bound search on rows

The best bound search requires more storage than the depth first method. For each node, NODE, in the search tree the position of the branching variable in the list ITJ and the branching row, KST(NODE) and IBR(NODE), are stored. In addition the estimated bound value, VLD(NODE), is stored. To record the tree structure the father node, JPR(NODE), is stored, with  $JPR(NODE) = 0$  signifying the NODE is the root node. This gives sufficient information for variables to be fixed at a node. Firstly NODE is chosen as the node for which VLD(NODE) is least. Then  $I = IBR(NODE)$ , the branching row is calculated. Variables  $x_j$  in row I are fixed equal to 0 until variable  $x_{j_0}$  for which  $J_0 = ITJ(KST(NODE))$  is reached. This is then fixed equal to 1. This method of fixing variables is then repeated for the father node of NODE until the root of the tree is reached. To facilitate the computation it is advantageous to store the cost of the fixed variables.

The speed with which the node that has the best bound can be retrieved can be greatly improved by using a linked list to link the nodes in

order of increasing estimated bound value. When a node is generated and inserted into the linked list time is taken in finding a location for it. This can be reduced by using a heap instead of a linked list to store the nodes, at the expense of additional storage. Further details are given for the travelling salesman problem by Carpaneto and Toth [C2].

The search is also quicker if the rows covered by variables fixed equal to 1 are stored in bits at each node of the search tree. In a best bound search this information can be useful for dominance tests. Nodes are discarded in a best bound search either because they are fathomed or because all their successors have been fathomed. To free the storage space left when the latter case has occurred at each node the first successor node should also be stored.

The minimum total storage for a best bound search is then  $4 \text{ MXND}$  where  $\text{MXND}$  is the maximum number of tree search nodes allowed at any one time. Additional storage of  $\text{MXND} + \text{MXND} \times \text{MWORD}$ , where  $\text{MWORD}$  is the number of words needed to store the bit pattern of  $m$  bits which indicates which rows have been removed, is required to implement the dominance tests. To store a linked list and successors nodes requires  $2 \text{ MXND}$  words. Hence the total amount of storage required is  $7\text{MXND} + \text{MXND} \times \text{MWORD}$ .

#### 8.3.4 Total Storage Required

To solve the SCP with both graph covering lower bounds in a depth first tree search requires at least  $3mp + 3n + 9m + 3 \text{ LEVMAX}$  words of storage plus the amount of storage required to solve the GCP which is at most  $1mp + 13m$  including storing the graph in adjacency lists which use  $mp + 2m$ . Hence the total storage is  $4mp + 3n + 22m + 3 \text{ LEVMAX}$ .

In addition some work arrays in which to perform calculations are required, but some of the graph covering algorithm and Lagrangean relaxation can share the same storage, so in practice this formula gives an estimate of the required amount. For a best bound implementation this increases to  $4mp + 3n + 22m + 7MXND + MXND + MWORD$  which unless using packed arrays and disc storage means that the best bound search requires too much storage to be practical.

## 8.4 Solving The Graph Covering Problem

### 8.4.1 The Graph Covering Algorithm

The algorithm used for the GCP was based on the matching algorithm of Edmonds' [E1, E2] which is described in Minieka [M4a] and Lawler [L2]. As no graph covering code was available, an algorithm of Derigs and Kazakidis [D3a] for solving the minimum perfect matching problem on a graph was modified. This is described in §8.4.2. Before the graph covering code could be used all arcs of negative cost were removed from the problem. Their values and indices were stored and then they were replaced with arcs of cost equal to 0. The graph was stored in adjacency lists as described in Aho et al [A1] and Derigs and Heske [D3]. The advantage of this representation is that it not only takes advantage of sparsity but it also allows graphs with multiple arcs to be handled. Instead of using reduced costs to reduce problem size after solving the GCP, lower bounds on the reduced costs were available and were used instead as described in §8.4.3. Sensitivity analysis of the GCP was considered but was not very useful because it was difficult to implement except in very simple cases. The graph covering algorithm starts by finding a matching in the 0-graph. Using start procedures that make the 0-graph as large as possible was found to be an effective

way of reducing computing time as described in §8.4.4.

#### 8.4.2 Converting an Algorithm for Solving the Minimum Perfect Matching Problem to a Graph Covering Algorithm

##### 8.4.2.1 Introduction

In a graph  $G(V,E)$  a perfect matching is a set of arcs for which each vertex is incident with exactly one arc in the set. A matching is a set of arcs that meets each vertex at most once. In a graph that has a cost associated with each arc and possesses a perfect matching, the problem of finding a minimum cost perfect matching is the integer program, MPMP:

$$\text{MPMP} \begin{cases} \min cx \\ Ax = 1 \\ x_j \in \{0,1\} \quad j = 1,2,\dots,n \end{cases}$$

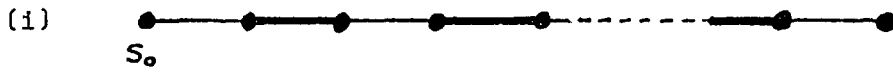
where  $A$  is the vertex-arc incidence matrix of the graph and  $c_j$  the cost of the  $j$ th arc. It is assumed  $c_j \geq 0$  for all  $j$ .

To get a graph covering program that can be used repeatedly to solve Lagrangean subproblems in a reasonable time a FORTRAN code for the MPMP was modified. The graph covering problem is MPMP with  $Ax = 1$  replaced by  $A'x \geq 1$ .  $A'$  is the vertex-arc incidence matrix of a graph  $G'$ . To modify the code note that the GCP can be reformulated as MPMP, [K4], by the addition of an extra variable for each constraint. For the  $i$ th constraint a column  $a_{j(i)}$  is added with a single 1 in row  $i$  and the cost of the corresponding column is  $c_{j(i)} = \min_{j \in N_i} c_j$ .

The resulting problem has the form of MPMP except that some columns including the extra columns that have been added may then have only one non-zero entry. Thus an extra constraint is added with 1's in columns that ensure that the constraint matrix has exactly two 1's in each column. Let the resulting constraint matrix be  $A'$ . This is the

FIGURE 8.2

An Augmenting Path



The path is changed to match  $s_0$  as shown below.

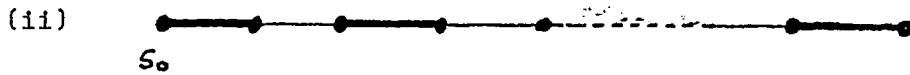
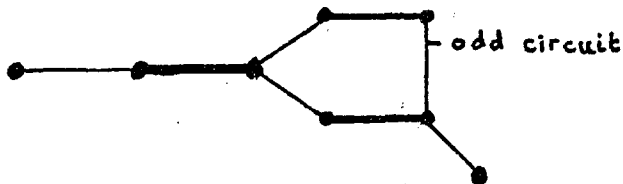
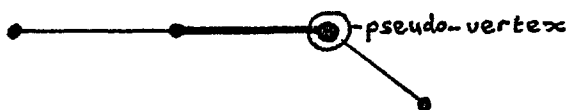


FIGURE 8.3

Formation Of A Pseudo Vertex



The graph is shrunk to give



Edges in the Matching are shown in heavy lines.

vertex-arc incidence matrix of a graph  $G$ . The MPMP code was modified so that the vertex corresponding to the extra constraint did not necessarily have to be covered at the end of the algorithm and it had a corresponding dual variable that was fixed equal to 0.

#### 8.4.2.2 Outline of the matching algorithm (Edmonds' Algorithm)

The MPMP code followed the algorithm of Edmonds for which data structures are described in detail in Lawler [L2]. The algorithm starts by first giving each vertex  $v_i$  a weight (dual variable),  $w_i$ , that satisfies  $c_j - w_i - w_k \geq 0$  where  $c_j$  is the cost of  $j$  that joins vertices  $v_i$  and  $v_k$ . This gives subgraph of  $G(V,E)$  for which the reduced costs of each arc are equal to 0, the 0-graph. Edmonds' algorithm starts with a matching of the 0-graph. This is also a matching of  $G(V,E)$ . If all the vertices are covered then the algorithm terminates with an optimal cover. Otherwise a vertex is found that is not covered,  $s_0$ , say. This vertex forms the root of a tree made up of arcs of  $G$  that are alternately not in the matching and in the matching. The tree is grown in the 0-graph from the root  $s_0$  by adding a non-matching and matching edge until one of the following three conditions occurs, illustrated in Fig. 8.2 and Fig. 8.3:

- (a) An augmenting path is detected.
- (b) A pseudo-vertex is formed.
- (c) The tree can be grown no further

Case (a) means that a path of arcs in the 0-graph has been found of odd length with non-matching edges at each end. The matching edges in this path are changed to non-matching edges and vice versa thus covering the two vertices at each end of the path. Case (b) means that an odd circuit of vertices in the 0-graph has been found. This is shrunk to form one pseudo-vertex. If case (c) arises the dual



variables must be changed. They are only changed for vertices in a tree, that is for vertices that are connected by a path made up of matching and non-matching arcs alternately until a root vertex  $s_0$  is reached.

#### 8.4.2.3 Modifying Edmonds' algorithm

The following modifications to Edmonds' algorithm must be made to give a graph covering algorithm. The extra vertex that is added to ensure that each column of the vertex-arc incidence matrix has exactly 2 1's will be labelled  $v_*$ . Firstly  $v_*$  can never be the root of a tree. Secondly  $v_*$  is never allowed to be shrunk into a pseudo-vertex. The first property is straightforward to implement and the second is handled by considering two cases when  $v_*$  is joined to a tree. These are:

##### Case 1 - Vertex $v_*$ is Matched by a Vertex $v_s$ in the Tree

This case is illustrated in Figure 8.4. There is an augmenting path from the root,  $s_0$ , of the tree containing  $v_s$  to  $v_*$ . The roles of edges in this path can be changed as in (a) so that vertex  $s_0$  is matched. Then vertex  $v_s$  is matched by an arc in the path and not by arc  $(v_s, v_*)$  which is removed from the matching. The tree with root  $s_0$  is no longer a tree in the next graph covering iteration.

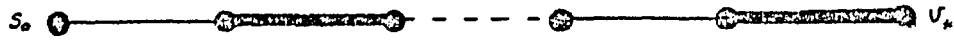
##### Case 2 - An Augmenting Path from $s_0$ to $v_*$ Has Been Found

In this case, illustrated in Fig. 8.5, matching and non-matching edges are changed as in (a). There is a vertex  $v_s$  in the path from  $s_0$  to  $v_*$  such that  $(v_s, v_*)$  becomes a matching edge. The tree rooted at  $s_0$  is destroyed as in Case 1. This means that  $v_s$  can never become

FIGURE 8.4

Case 1 Vertex  $v_*$  Is Matched In The Tree

(a) Detecting a Path  $(s_0, v_*)$



(b) Changing the Path to Match  $s_0$

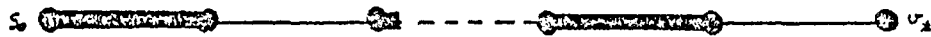
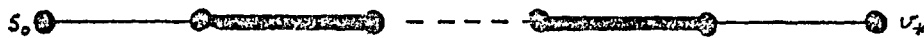


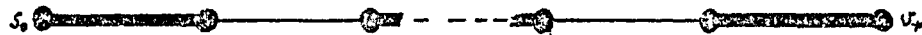
FIGURE 8.5

Case 2 Vertex  $v_*$  Is Not Matched

(a) Detecting an Augmenting Path  $(s_0, v_*)$



(b) Changing the Path to Match  $s_0$



————— Edge not in the matching  
 ————— Edge in the matching

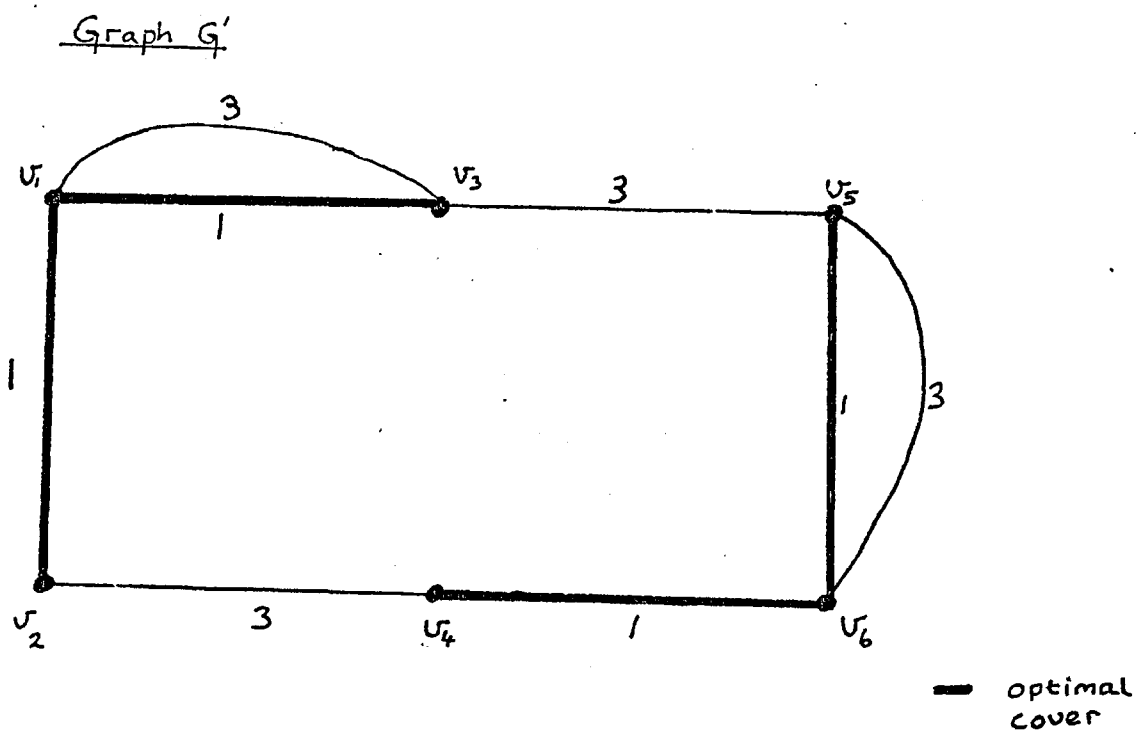
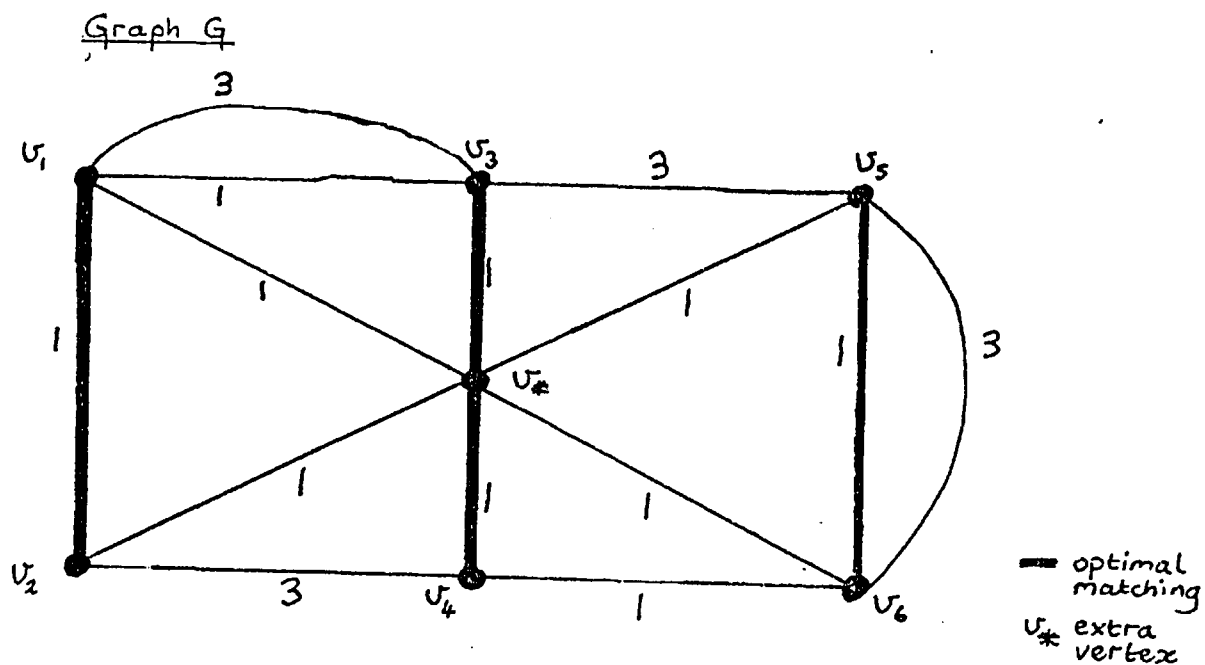
part of a tree as long as it is matched by  $v_*$ . Hence the reduced cost of arc  $(v_s, v_k)$  remains equal to 0 as long as  $(v_s, v_*)$  is a matching edge.

Just as Edmonds' algorithm terminates with a matching in a shrunken graph so the graph covering algorithm terminates with a set of arcs in a shrunken 0-graph that meets each vertex of the original graph exactly once except possibly the extra vertex  $v_*$ , which may or may not be covered any number of times. The matching then induces a covering in the original graph in exactly the same way as Edmonds' algorithm induces a matching from the matching in the shrunken graph. To transform the graph to one on the original graph the arcs that are in the cover and joined to the extra vertex must be mapped back to the arcs from which they were originally derived. For example if arc  $(s, v_*)$  is in the solution and has been added to the graph because the cost  $c_{jo} = \min_{j \in S} c_j$  (where  $S$  is the set of arcs of the original GCP which have one end equal to  $s$ ) then the original arc  $jo$  in the GCP is marked as being in the optimal cover. The optimal graph covering solution has the property that if  $c_j \geq 0$  then all arcs in the solution are incident with at least one vertex that is only covered by one arc in the solution. This corresponds to a prime cover of the SCP.

The example in Fig.8.6 shows an optimal matching in a Graph  $G'$  which corresponds to an optimal cover in a graph  $G$ . The cost of each arc is shown alongside it and  $v_*$  is the extra vertex. The arcs in the optimal matching and optimal cover are shown in heavy lines. The original GCP was the problem.

FIGURE 8.6

A Graph  $G$  in Which an Optimal Matching  
Corresponds to an Optimal Cover in  $G'$





$$c_j - u_i - u_k \quad (8.1)$$

Thus a column  $a_j$  of the SCP has a lower bound on the reduced cost of:

$$\bar{s}_j = c_j - \sum_{i=1}^m u_i a_{ij} \quad (8.2)$$

where the weights  $u_i$  correspond to vertex weights in a column splitting relaxation of the SCP. If  $z_l$  is the lower bound from GCR2 then a column can be removed if:

$$\bar{s}_j \geq z_u - z_l \quad (8.3)$$

where  $z_u$  is an upper bound to the SCP.

For the combined relaxations (where rows  $R$  are relaxed) if  $z_l$  is the lower bound and all the Lagrange multipliers satisfy:

$$\lambda_i \geq 0 \quad \text{for all } i \in R$$

$$c_j - \sum_{i \in R} \lambda_i a_{ij} \geq 0 \quad \text{for all } j \quad (8.4)$$

then if

$$c_j - \sum_{i \in R} \lambda_i a_{ij} - \sum_{i \notin R} u_i a_{ij} \geq z_u - z_l$$

the corresponding column can be removed from the SCP. To ensure the reduced cost test, (8.3), is valid, firstly the constraints (8.4) must be satisfied and secondly when the best value of the lower bound is calculated for a given node in the search tree the multipliers  $\lambda_i$  and  $u_i$  are stored in a single vector. This test is particularly effective in a branch and bound algorithm as it can substantially curtail the number of branches generated at a node.

#### 8.4.4 Start Procedures for the Graph Covering Algorithm

The graph covering problem, when used in subgradient optimization, is likely to have arcs of negative cost. Either the procedures of Chapter 5 for making the costs non-negative by adjusting the Lagrange multipliers can be invoked or they can be removed from the problem and stored in a list and added at the end of the GCP solution to the remaining problem. Also costs equal to 0 can be stored in a list. Thus a graph with positive costs remains. The initial dual variables are obtained by considering their optimal value at the previous subgradient iteration and then testing that the reduced cost,  $c_j - u_i - u_k$ , is not negative for all arcs  $e_j$  joining vertices  $v_i$  and  $v_k$ . If  $c_j - u_i - u_k < 0$  then  $u_i$  is reduced until either  $c_j - u_i - u_k \geq 0$  or  $u_i$  is equal to 0. This procedure is repeated until all arcs have non-negative reduced cost. Then the minimum cost,  $\delta$ , of arcs incident to each vertex is found. If this is not equal to 0 then the vertex weight,  $u_i$ , can be increased by  $\delta$ . This gives a 0-graph in which a matching is found from which to start the graph covering algorithm.

#### 8.5 Parameters Of The Program

Two tolerances are used in the graph covering algorithm. The first is EPS and is approximately  $10^{-3}$  and is the tolerance in which the optimal solution must lie. The second is ETA and any variable with absolute value less than ETA is regarded as being equal to 0.

Iteration counters were used to limit the number of tree search nodes, graph covering iterations, subgradient iterations and number of times the relaxation could be changed. In addition two parameters were used to limit the number of subgradient optimization iterations

allowed without an increase in bound. These had a critical effect on the execution time and it is essential that if a subgradient ascent is not producing an increase in the bound that the ascent is terminated.

The stepsize parameters and parameters  $\theta$  have been discussed in Chapter 3 and choice of  $\sigma$  was found to have a significant effect on the search.

The number of rows relaxed in GCR1 prior to splitting columns to give GCR2 did not seem to be a crucial factor. The most important factor seemed to be the actual number of graph covering problems solved. However relaxing a large number of rows initially means that small GCP's are solved and this can reduce the computation time.

The option of whether or not to perform the preliminary reduction phase was tested. It was found best, in almost all cases, to use the heuristic procedures firstly to reduce problem size and secondly to get good bounds before the graph covering relaxation was used.

It would be advantageous to control the parameters dynamically in the algorithms because, for example, the best stepsize parameters at tree search nodes near the root were not always the best parameters further down the tree. Also if there is a large gap between an estimated lower bound value and upper bound at a node in a tree search it is probably quicker to continue branching forward for one or two stages without even calculating a bound.



CHAPTER 9CONCLUSIONS9.1 Summary

The most successful algorithm for small problems uses the heuristic methods of Chapter 2 in a ~~single~~ tree search. However they are unsuitable for larger problems for two reasons. Firstly the upper bounds they give are often much higher than the optimal solution. Secondly the lower bound produced by the heuristics is bounded above by the LP bound which often does not solve the SCP.

Although the attempts in Chapter 3 to accelerate the subgradient ascent were not very successful some improvement was possible with careful choice of parameters. The slow ~~rate~~ of increase in the bound for the subgradient optimization ascent was a problem in Chapter 5 where theoretically the graph covering bound is greater than the LP bound but in practice it is difficult to get the former bound to exceed the latter.

The network flow relaxations were not particularly useful as they are superseded by E~~ch~~berry's method [E4].

In Chapter 5 the graph covering relaxations are rarely able to compete with the bounds obtained using the commercial LP code, APEX.

There are several reasons for this. The first, already mentioned, is that the search direction and steplength used in subgradient optimization were not particularly effective in increasing the lower bound. Secondly the LP code uses a basis, that is a subset of the columns for the bulk of the calculations. Thirdly the branching

strategy used by the APEX code is superior to that used in Chapter 5.

The decomposition algorithm of Chapter 6 is too costly for randomly generated problems. However it is likely to perform much better on problems for which the constraint matrix is almost in block diagonal form. This is often the case in practical vehicle scheduling and routing problems. The state space relaxation method was only briefly studied and it gave lower bounds that were too low for practical application. However combining it with the heuristics of Chapter 2 could produce good bounds.

The best bound strategy performed better than the depth first. However it had storage limitations which made it impossible to use it on large problems. Both of the branching on rows strategies used could be improved by instead of solving for the bound at nodes that are unlikely to be fathomed taking forward steps to generate new nodes instead.

## 9.2 Extensions And Ideas For Future Research

### 9.2.1 Extensions of the Graph Covering Algorithms

The blossom constraints in the graph covering relaxations are of the form:

$$\bigvee_{i \in R} \sum a_{ij} x_j \geq v(S(R))$$

where  $\bigvee$  is the logical 'or' sign

$R$  is a subset of the rows of the SCP

where  $v(S(R))$  is the optimal solution to the unicost problem  $S(R)$ :

$$\begin{aligned} \min \quad & \sum_{j=1}^n x_j \\ S(R) \quad & \sum_{j=1}^n a_{ij} x_j \geq 1 \quad \text{for all } i \in R \\ & x_j \in \{0,1\} \end{aligned}$$

It may be possible to find SCP's with special structure for which this type of constraints can be used to raise the bound. For the graph covering relaxations  $R$  is a set of row of the SCP of odd cardinality with at most 2 non-zero entries per column.

### 9.2.2 Aggregating Constraints

Although constraint aggregation may not in practice produce a bound much better than that obtainable by Lagrangean relaxation it may be computationally beneficial. Thus constraints can be multiplied by variables and added together to give an integer knapsack problem. The bound is at least as good as the LP bound and may be better.

### 9.2.3 Extensions to 0-1 Integer Programs

Several of the methods illustrated here are applicable to a more general class of integer programs. For example the heuristics could be used to get dual feasible solutions to the LP relaxation of the problem:

$$\begin{aligned} \min \quad & cx \\ & Ax \geq b \\ & x_j \in \{0,1\} \quad j = 1,2,\dots,n \end{aligned}$$

especially when the coefficients  $a_{ij}$  and right hand sides  $b_i$  take small non-negative integer values.

#### 9.2.4 Improvement of the State Space Relaxations

The state space relaxations can be improved using the heuristics to either give integer weights to each row or a dual feasible solution to the LP. These can then be used to extend the state space as mentioned in Chapter 6.

#### 9.2.5 Duality

The application of integer programming duality was only briefly discussed in Chapter 6. It is useful because it enables reduced costs to be calculated. Variables with large reduced costs are rejected. Alternatively if a subset of columns is used for the calculations variables with negative reduced cost can be added to the problem. Alternatives to LP duality should be investigated. These could use cutting planes from which integer programming dual variables can be defined.

#### 9.2.6 Methods For Improving The Code

This section deals with topics that have already been used by previous researchers. It gives ideas for improving the SCP code.

- 1) The subgradient ascent could be improved by using a linesearch to calculate  $\sigma$ , the steplength. For example a cubic linesearch could be used.
- 2) Using a subset of columns for the major calculations of the SCP algorithm would speed up the program. This is analogous to using a basis in LP. Reduced cost tests at nodes in a branch and bound tree would then enable columns with negative reduced

costs to be brought into the problem.

- 3) Storage is a major problem in solving large sparse SCP's.

The CDC computer has a large word length which means that the arrays can be packed. It would be possible to pack the arrays that store the constraint matrix to roughly 1/6th their current size. A slower way of handling larger problems would be to store the constraint matrix out of core.

- 4) Often the upper bound produced by the heuristics was not optimal and this meant that the tree searches were unnecessarily long. Thus it would be useful to calculate, initially an upper bound from an  $r$ -optimal method.

- 5) Disjunctive cuts as used in Balas and Ho can be generated from the graph covering dual variables. These would raise the bound further at little extra cost, but possibly extra storage.

REFERENCES

- A1 A.V. AHO, J.E. HOPCROFT and J.D. ULLMAN "The Design and Analysis of Computer Algorithms" Addison-Wesley, 1974.
- A2 J.P. ARABEYRE, J. FEARNLEY, F.C. STEIGER and W. TEATHER "The Airline Crew Scheduling Problem: A Survey" Trans. Sci. 3, 1969, 140-163.
- A3 J.A. ARAOZ-DURAND "Polyhedral Neopolarities" PhD Thesis, Dept. of Applied Analysis and Computer Science, University of Waterloo, Ontario, 1974.
- A4 D. AVIS "A Note on Some Computationally difficult Set covering problems" Math. Prog. 18, 1980, 138-145.
- B1 E.K. BAKER, L.D. BODIN, W.F. FINNEGAN, R.J. PONDER "Efficient Heuristic Solutions to an Airline Crew Scheduling Problem" AIIE Transactions 11, 1979, 79-85.
- B2 K.R. BAKER "Scheduling a Full Time Workforce to meet Cyclic Staffing Requirements" Man. Sci. 20, 1974.
- B3 E. BALAS "An Additive Algorithm for Solving LP with 0-1 Variables" Opns.Res. 13, 1965, 517-546.
- B4 E. BALAS "Set Covering with Cutting Planes from Conditional Bounds" MSRR 399, Carnegie-Mellon University, 1977.
- B5 E. BALAS "Cutting Planes from Conditional Bounds: A new approach to set covering" Math. Prog. Study 12, Combinatorial Optimization, 1980, 19-36.

- B6 E. BALAS and N. CHRISTOFIDES "A Restricted Lagrangean Approach to the Travelling Salesman Problem" Working Paper, Imperial College, Feb. 1979.
- B7 E. BALAS and A. HO "Set Covering Algorithms using Cutting Planes, Heuristics, and Subgradient Optimization: A Computational study" Math. Prog. Study 12, Combinatorial Optimization, 1980, 37-60.
- B8 E. BALAS and M. PADBERG "Set Partitioning: A Survey" SIAM Review 18, 1976, 710-760  
also  
"Set Partitioning: A Survey" in "Combinatorial Optimization" N. CHRISTOFIDES et. al. (Ed.), Wiley, 1979.
- B9 E. BALAS and H. SAMUELSSON "A Node covering Algorithm" NRLQ 21, 1974, 213-233.
- B10 M.L. BALINSKI "On Maximum Matching, Minimum Covering and their Connections" Proc. Int. Symp. on Math. Prog. H.W. KUHN (ed.) Princeton UP, 1970, 303-311.
- B11 M.L. BALINSKI and R.E. QUANDT "On an Integer Program for a Delivery Problem" Opns. Res. 12, 1964, 300-304.
- B12 R.S. BARR, F. GLOVER, D. KLINGMAN "A Generalised Alternating Path Algorithm for Transportation Problems" REJOR 2, 1978, 137-144.
- B13 R. BARR, F. GLOVER and D. KLINGMAN "Enhancements of Spanning Tree Labelling procedures for Network Optimization" INFOR 7, 1979, 16-34.
- B14 J.J. BARTHOLDI, J.B. ORLIN, H.D. RATLIFF "Circular 1's and Cyclic Staffing" Research Report 77-11, University of Florida, Gainesville.

- B15 J.J. BARTHOLDI and H.D. RATLIFF "A Field Guide to Identifying Matching and Network Flow Problems" Research Report, 77-12, University of Florida, Gainesville, 1977.
- B15a J.J. BARTHOLDI and H.D. RATLIFF "Unnetworks, with Applications to Idle Time Scheduling" Man. Sci. 8, 1978, 850-858.
- B15b M.S. BAZARAA, C.M. SHETTY "Nonlinear Programming, Theory and Applications" Wiley, 1979.
- B16 M. BEALE (Ed.) "Applications of Mathematical Programming Techniques" EUP, 1970.
- B17 D.E. BELL and J.F. SHAPIRO "A Convergent Duality Theory for Integer Programming" Opns. Res. 25, 1977.
- B18 M. BELLMORE, H.J. GREENBERG and J.J. JARVIS "Multi-Commodity Disconnecting Sets" Man. Sci. 16, 1970, B427-B433.
- B19 M. BELLMORE and H.D. RATLIFF "Optimal Defense of Multi-Commodity Networks" Man. Sci. 18, 1971, 174-185.
- B20 M. BELLMORE and H.D. RATLIFF "Set Covering and Involutory Bases" Man. Sci. 18, 1971, 194-206.
- B21 C. BERGE "Graphs and Hypergraphs" Dunod, Paris, 1970 (English Translation: North-Holland Amsterdam, 1973).
- B22 G.N. BERLIN and J.C. LIEBMAN "Mathematical Analysis of Emergency Ambulance Location" Socio-Economic Planning Sciences 8, 1974, 323-328.
- B23 A.T. BERZTISS "Data Structures: Theory and Practice" Academic Press, 1971.
- B24 F. BESSIERE "Sur la recherche du nombre chromatique d'un graph par un programme lineaire un nombres entiers RAIRO 9; 1965, 143-148.



- B25 O. BILDE and J. KRARUP "Plant location, Set covering and Economic lot size. An  $O(mn)$  algorithm for Structured Problems" Res. Report IMSOR, The Technical University of Denmark, 1975, and Report No. 75/6, Institute of Datalogy, University of Copenhagen, 1975  
also  
"Sharp lower bounds and efficient Algorithms for the Simple Plant Location Problem" Annals of Discrete Mathematics, 1, 1977, 79-97.
- B26 J.M. BODER "A method for Solving Crew Scheduling Problems" ORQ 12, 1975, 55-62.
- B27 L.D. BODIN and R.B. DIAL "Hierarchical Procedures for Determining Vehicle and Crew Requirements for Mass Transit Systems" Report College of Business and Management, University of Maryland, Jan 1979.
- B28 N. BONDE and J. TIND "Bounds in Set Partitioning" Research Report Institute of Operations Research, Univeristy of Aarhus, Denmark, April 1976.
- B29 V.J. BOWMAN and J. STARR "Set Covering by Ordinal Cuts I: Linear Objective Functions" MSRR, No. 321, Carnegie Mellon University, Pittsburgh, PA, June 1973.
- B30 G.H. BRADLEY, G.G. BROWN and G.W. GRAVES "Design and Implementation of large Scale Primal Transshipment Algorithms" Man. Sci. 24, 1977, 1-34.
- B31 R. BREU and C.A. BURDET "Branch and Bound Experiments in 0-1 Programming" Math. Prog. Study 2, 1974, 1-50.
- B32 M.A. BREUER "Simplification of the Covering Problem with Application to Boolean Expressions" JACM 17, 1970, 160-181.

- C1 P.M. CAMERINI, L. FRATTA and F. MAFFIOLI "On Improving Relaxation Methods by Modified Gradient Techniques" Math. Prog. Study 3, 1975, 26-34.
- C2 G. CARPANETO and P. TOTH "An efficient Algorithm for the Asymmetric Travelling Salesman Problem" Presented at ORSA/TIMS Atlanta, November 1977.
- C3 C. CHARALAMBOUS and A.R. CONN "An efficient Method to solve the minimax problem directly" SIAM JI. Num. Anal. 15, 1978, 162-187.
- C4 J.W. CHRISSIS and R.P. DAVIS "Some Computational Experience with Dynamic Set Covering Problems" Technical Report No. 7908, Dept. of Industrial Engineering and Operations Research, Virginia Polytechnic Institute, 1979.
- C5 N. CHRISTOFIDES "Graph Theory: an Algorithmic Approach" Wiley, 1975;
- C6 N. CHRISTOFIDES "A minimax facility location problem and the Cardinality constrained Set covering Problem" MSRR 375, Carriegie-Mellon University, Pittsburgh, Oct. 1975.
- C7 N. CHRISTOFIDES "The Vehicle Routing Problem" RAIRO 10, 1976, 55-70.
- C8 N. CHRISTOFIDES and S. KORMAN "A Computational Survey of Methods for the Set Covering Problem" Man. Sci. 21, 1975, 591-599.
- C9 N. CHRISTOFIDES, A. MINGOZZI AND P. TOTH "State space relaxation Methods for the Vehicle Routing Problem" Imperial College, Dept. of Management Science Report IC-OR-79-09, 1979.

- C10 N. CHRISTOFIDES and S.K. MITRA "A new algorithm for 0-1 Programming"  
Imperial College, DMSS, Jan. 1974.
- C11 V. CHVATAL "A Greedy Heuristic for the Set Covering Problem"  
Math of OR 4, 1979, 233-235.
- C12 A. COBHAM, F. FRIDSHALL, J.H. NORTH "An application of Linear  
Programming to the Minimization of Boolean functions" Proc. 2nd  
Annual Symposium on Switching and Circuit Theory and Logical Design  
AIEE, pub. 5134, 1961, 3-9 (see also IBM Res. RC-47 (1961)).
- C13 R.A. CODY and E.G. COFFMAN "Record Allocation for Minimising  
Expected Retrieval Costs on Drumlike Storage Devices" JI. ACM 23,  
1976, 103-115.
- C14 G. CORNUEJOLS, M.L. FISHER and G. NEMHAUSER "The Location of Bank  
Accounts to Optimise Float" Man. Sci 23, 1977, 789-810.
- C15 G. CORNUEJOLS, G.L. NEMHAUSER and L.A. WOLSEY "Worst-case and  
Probabilistic Analysis of Algorithms for a Location Problem" MSRR  
443, Carnegie-Mellon University, Pittsburgh, 1979
- C15a H. CROWDER "Computational Improvements of Subgradient Optimization"  
IBM Technical Report RC4907, Yorktown Heights, New York, 1974.
- C16 W.H. CUNNINGHAM and A.B. MARSH "A Primal Algorithm for Optimum  
Matching" Math. Prog. Study 8, 1978, 50-72.
- D1 R.H. DAY "On optimal extracting from a multiple file data storage  
system: an application of integer programming" Opns. Res. 13, 1965,  
482-494.
- D2 U. DERIGS "Algebraische Matching Problems" Doctoral Thesis,  
Mathematisches Institut, Universität Zu Köln, 1978.

- D3 U. DERIGS and A. HESKE "A Computational Study on some Methods for Solving the Cardinality Matching Problem" Report 79/2 Mathematisches Institut, Universität Zu Köln, January 1979.
- D3c U. DERIGS and G. KAZAKIDIS "On two Methods for Solving minimal perfect matching problems", Report, Industrieseminar, Universität Zu Köln, May 1979.
- D4 I.S. DUFF and J.K. REID "Some design features of a Sparse Matrix Code", ACM Trans. Math. Soft. 5, 1979, 18-35.
- E1 J. EDMONDS "Covering and Packing in a Family of Sets" Bulletin AMS 68, 1962, 494-499.
- E2 J. EDMONDS "Paths, Trees and Flowers", Canadian J1. of Math. 17, 1965, 449-467.
- E3 J. EDMONDS and E.L. JOHNSON "Matching, Euler Tours and the Chinese Postman" Math. Prog. 5, 1973, 88-124.
- E4 D.R. ERLKOTTER "A Dual Based Procedure for Uncapacitated Facility Location" Opns. Res., 26, 1979, 992-1009.
- E5 J. ETCHEBERRY, "The Set Covering Problem. A new implicit enumeration algorithm" Opns. Res. 25, 1977, 760-772.
- E6 S. EVEN and O. KARIV "An  $(n^{2.5})$  - algorithm for Maximum Matching in General Graphs" in "Proceedings fo the 16th Annual Symposium on Foundations of Computer Science" IEEE , New York, 1975, 100-112.
- F1 R. FLETCHER.(Ed.) "Optimization" Academic Press, 1969.
- F2 L.R. FORD and D.R. FULKERSON "Flows in Networks", Princeton, New Jersey and OUP, 1962.

- F3 D.R. FULKERSON "Blocking and Anti-blocking Pairs of Polyhedra"  
Math. Prog. 1, 1971, 168-194.
- F4 D.R. FULKERSON, G.L. NEMHAUSER, L.E. TROTTER "Two Computationally  
Difficult Set Covering Problems that arise in Computing the 1-width  
of incidence Matrices of Steiner triple systems" Math. Prog. Study  
2, 1974, 72-81.
- F5 D.R. FULKERSON and D.B. WEINBERGER "Blocking pairs of Polyhedra  
Arising from Network Flows" J1. Comb Th. (b)18,1975,265-283.
- G1 H. GABOW "An Efficient Implementation of Edmonds' Algorithm  
for Maximum Matching on Graphs" JACM 23, 1975, 221-234.
- G2 M. GAREY and D.S. JOHNSON "Computers and Intractability: A Guide  
to the theory of NP completeness" Freeman & Co., San Fransisco, 1978.
- G3 R.S. GARFINKEL and G.L. NEMHAUSER "Optimal Set Covering: A Survey"  
in "Perspectives on Optimization" A.M. GEOFFRION (Ed.) Addison-Wesley,  
1972, 164-183.
- G4 R.S. GARFINKEL and G.L. NEMHAUSER "Integer Programming" Wiley,  
1972.
- G5 B. GAVISH, P. SCHWEITZER and E. SHLIFER "Assigning buses to Schedules  
in a Metropolitan area" Comp. and Opns Res 5, pp129-138, 1978.
- G6 E. GAVRIL "Algorithms for Minimum coloring, Maximum clique,  
Minimum covering by cliques and Maximum independent set of a  
Chordal Graph" SIAM J1. of Computation 1, 1972, 180-187.
- G7 F. GAVRIL "Algorithms for a Maximum Clique and a Maximum Independent  
Set of the Circle Graph" Networks, 3, 1973, 261-273.

- G8 A.M. GEOFFRION "Lagrangean Relaxation for Integer Programming"  
Math. Prog. Study 2. (M. BALINSKI, Ed) North-Holland, Amsterdam,  
1974, 97-107.
- G9 A.M. GEOFFRION and R. McBRIDE "Lagrangean Relaxation Applied to  
Capacitated Facility Location Problems" AIIE Transactions 10, 1978,  
40-47.
- G10 A.M. GEOFFRION and R. NAUSS "Parametric and Postoptimality Analysis  
in ILP" Man. Sci 123, 1977, 453-466.
- G11 J.F. GIMPEL "A Reduction Technique for Prime Implicant Tables"  
IEEE Transactions Elec. Comp.. EC,14, 1965, S35-541.
- G12 J.F. GIMPEL "A Stochastic Approach to the Solution of Large  
Covering Problems" IEEE Switching and Automata Theory, 1967, 78-83.
- G13 F. GLOVER "Parametric Branch and Bound" OMEGA 6, 1975, 145-152
- G14 F. GLOVER, D. KARNEY, D. KLINGMAN and A. NAPIER "A Computational  
Study on Start Procedures, Basis Change Criteria and Solution  
Algorithms for Transportation Problems" Man. Sci. 20, 1974, 793-813.
- G15 F. GLOVER and J. MULVEY "Equivalence of the 0-1 Integer Programming  
problem to Discrete Generalised and Pure Networks" MSRS No. 75-19,  
Princeton University, 1977.
- G16 M. GONDRAN "Set Partitioning and Covering Algorithms Applications  
and Algorithms" Bull. Dir. Etud. et Rech 2, 1976 (French), 59-68.
- G17 F. GRANOT and P. HAMMER "On the role of Generalised Covering Problems"  
CCERO 16, 1976, 207-216 and 277-289.
- H1 J. HALPERN "The Sequential Covering Problem under Uncertainty"  
INFOR 15, 1977, 76-93.

- H2 F. HARARY "Graph Theory" Addison - We]sey, 1971.
- H3 D. HAUSMANN and B. KORTE "Adjacency on 0-1 Polyhedra" Math. Prog. Study 8, 1978, 106-127.
- H4 M. HELD and R.KARP "The Travelling Salesman Problem and Minimum Spanning Trees II" Math. Prog. 1, 1971, 6-25.
- H4a M. HELD, P. WOLFE, H. CROWDER. "Validation of Subgradient Optimization" Math. Prog. 6, 1974, 62-88.
- H5 F.S. HILUER "Efficient Heuristic Procedures for Integer Linear Programming with an interior" Opns. Res 17, 1969, 600-637  
also  
Tech. Rep. 2, Dept. of OR, Stanford University contains a FORTRAN listing of the algorithm,
- H6 F.S. HILUER and G.LIEBERMAN "Introduction to Operations Research" Holden Day, 1974.
- H7 A. HO "Worst Case Analysis of a Class of Set Covering Heuristics" "Working paper, June 1979, Carnegie-Mellon University, Pittsburgh.
- H8 E. HOROWITZ and S. SAHNI "Fundamentals of Data Structures" Computer Science Press, Maryland, 1975.
- H9 E.J. HOROWITZ and S. SAHNI "Fundamentals of Computer Algorithms" Pitmans Publishing, 1979, London (English Edition) (First Published 1978 Computer Science Press, Maryland).
- H10 R.W. HOUSE, L.D. NELSON, J.RADO "Computer Studies of a Certain Class of Linear Integer Problems" in "Recent Advances in Optimization Techniques" A. LAVI and T VOGL (Ed.) Wiley NY, 1966, 241-280.

- I1 T. IBARAKI "Theoretical Comparisons of Search Strategies in Branch and Bound Algorithms" Intl. Jl. Comp. and Inf. Sci. 5.
- I2 T. IBARAKI "Approximate Branch and Bound Algorithms" Math. of OR, 1, 1976.
- I3 O. H. IBARRA and C.E. KIM "Fast Approximate Algorithms for the Knapsack and Sum of Subsets Problem" JACM 22, 1975, 463-468.
- I4 J.P. IGNIZIO "A Heuristic Solution to Generalised Covering Problems" Ph.D Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1977.
- J1 D.S. JOHNSON "Approximation Algorithms for Combinatorial Problems" Jl. Comp. and Syst. Sci. 9, 1974, 256-278.
- J2 E.L. JOHNSON "On the Group Problem and a Subadditive Approach to Integer Programming" Annals of Discrete Mathematics 5, 1979, 97-112.
- J3 E.L. JOHNSTON "Support Functions, Blocking Pairs and Anti-blocking Pairs" Math. Prog. Study 8, 1978, 167-196.
- K1 R.M. KARP "On the Computational Complexity of Combinatorial Problems" Networks 5, 1975, 45-68.
- K2 R.M. KARP "The Probabilistic Analysis of some Combinatorial Search Algorithms" in "Algorithms and Complexity" J.F. TRAUB (Ed.), Academic Press, 1976, 1-19.
- K2a J. KING and V. NAKORNCHAI "An extension of ROC in Production Flow Analysis" Dept. of Management Science, Imperial College, Report, 1980
- K3 G.A. KOCHENBERGER, B.A. McCARL and F.P. WYMAN "A Heuristic for General Integer Programming" Decision Sciences, 5, 1974, 36-44.



- K4 S.M. KORMAN "Graph Colouring and Related Problems in OR" Ph.D Thesis  
Dept. of Man. Sci., Imperial College, London, 1975.
- K5 S.M. KORMAN "The Graph Colouring Problem" in "Combinatorial Optimization"  
N. Christofides et. al. (Ed.), Wiley, 1979.
- K6 D. KNUTH "Fundamental Algorithms" The Art of Computer Programming, 1,  
Addison-Wesley, 1968.
- L1 A. LAND and S. POWELL "Fortran Codes for Mathematical Programming"  
Wiley, 1973.
- L2 E.L. LAWLER "Combinatorial Optimization: Networks and Matroids" Holt,  
Rinehart and Winston Inc., New York, 1976.
- L3 E.L. LAWLER "Covering Problems. Duality relations and a new Method of  
Solution" SIAM JI. 14, 1966, 115-1132.
- L4 E.L. LAWLER and D.E. WOOD "Branch and Bound Methods: a survey" Opns. Res.  
14, 1966, 699-719.
- L4a C.E. LEMKE "Bi Matrix Equilibrium Points and Mathematical Programming"  
Man. Sci., 11, 1965, 681-689.
- L5 C.E. LEMKE, H.M. SALKIN and K. SPIELBERG "Set Covering by Single Branch  
Enumeration with LP Subproblems" Opns. Res. 19, 1971, 978-1022.
- L6 B. LEV and A.L. SOYSTER "Integer Programming with Bounded Variables by  
Canonical Separation" JORS 29, 1978, 477-485.
- L7 S. LIN "Computer solutions of the Travelling Salesman Problem" Bell. Syst.  
Tech. JI. 44, 1965, 2245-2269.
- M1 R.E. MARSTEN "An Algorithm for Large Set Partitioning Problems" Man.  
Sci. 20, 1974, 774-787.

- M2 G.T. MARTIN "An Accelerated Euclidean Algorithm for Integer Linear Programming" in "Recent Advances in Mathematical Programming" G. GRAVES and P. WOLFE(Ed.) McGraw-Hill, New York, 1963, 311-317.
- M3 A. MEIR and J.W. MOON "Relations between Packing and Covering Numbers of a Tree" Pacific J. of Math. 61, 1975, 225-233.
- M4 P. MEVERT and M ROHDE "Set Partitioning and Side Constraints" Working paper 24/78, Freie Universitat, Berlin, 1978.
- M4a E. MINIEKA "Optimization Algorithms for Networks and Graphs" M. Dekker, New York, 1978.
- M5 J.A. MORELAND "Scheduling Air Flight Crews" Masters Thesis, MIT, Dept. of Aeronautics and Astronautics, 1966.
- M6 J.M. MULVEY and H.P. CROWDER "Cluster-analysis: An Application of Lagrangean Relaxation" Man. Sci. 25, 1979, 329-340.
- M7 K. MURTY "On the Set Representation Problem and the Set Covering Problem" in "Symposium of the Theory of Scheduling and its Application" S.E. Elmagharaby (Ed.) Springer-Verlag,Heidleberg, 1973, 143-161.
- N1 G.L. NEMHAUSER, L.E. TROTTER and R.M. NAUSS "Set Partitioning and Chain Decomposition" Man. Sci. 20, 1974, 1413-1423.
- N2 G.L. NEMHAUSER and G.M. WEBER "Optimal Set Partitionings, Matchings and Lagrangian duality" NRLQ 26, 1979, 553-563.
- N3 R.G. NIGMATULLIN "The Fastest Descent Method for Covering Problems" in Proc. Symposium on Questions of Precision and Efficiency of Computer Algorithms, Book 5, Kiev, 1969, 116-126 (in Russian)

- N4 S.C. NTAFOU and S.L. HAKIMI "On Path Cover Problems in Digraphs and Applications to Program Testing" IEEE on Software Engineering, 1979.
- P1 M.W. PADBERG "On the Facial Structure of the Set Packing Polyhedra" Math. Prog. 5, 1973, 199-215.
- P2 M.W. PADBERG "Characterizations of Totally Unimodular, Balanced and Perfect Matrices" in "Combinatorial Programming" B. ROY (Ed.) D. Reidel Publishing Co, 1976, 275-284.
- P3 M.W. PADBERG "Covering, Packing and Knapsack Problems" Annals of Discrete Mathematics 4, 1979, North-Holland.
- P4 M.W. PADBERG and S. HONG "On the Travelling Salesman Problem: A Computational Study" T.J. Watson Research Center Report, IBM Research, 1977.
- P4a J.F. PIERCE "A Two Stage Approach to the Solution of the Vehicle Dispatching Problem" Presented at the 17th TMS Intl. Conf., LONDON, 1970.
- P5 J.F. PIERCE and J.S. LASKY "Improved Combinatorial Programming Algorithms for a class of all 0-1 IP problems" Man. Sci. 19, 1973, 528-543.
- P6 C.J. PIPER and A.A. ZOLTNER "Some easy postoptimality Analysis for 0-1 programming" Man. Sci. 22, 1976, 759-65.
- P7 W.R. PULLEYBLANK "Minimum node covers and 2-Critical Graphs" Math. Prog. 17, 1979, 91-103.
- P8 W.R. PULLEYBLANK and J. EDMONDS "Facets of 1-Matching Polyhedra" in Hypergraph Seminar, Springer Verlag lecture notes on Mathematics, 411, 1974.

- Q1 W.V. QUINE "The Problem of Simplifying Truth Functions" Am. Math. Monthly 59, 1952, 521-531.
- R1 C. REVELLE and R. SWAIN "Central Facilities Location" Geographical Analysis 2, 1970, 30-42.
- R1a J.B. ROSEN "The Gradient Projection Method" SIAM JI. 8, 1960, 181-217
- R2 R. ROTH "Computer Solutions to Minimum-Cover Problems" Opns. Res. 17, 1969, 455-465.
- R3 J. RUBIN "A Technique for the Solution of Massive Set Covering Problems with Applications to Airline Crew Scheduling" Trans. Sci. 7, 1973, 34-48.
- R4 R.A. RUTMAN "An Algorithm for Placement of Interconnected Elements based on Minimum Wire Length" Proc. of AFIPS Cong. 20, 1964, 477.
- S1 S. SAHNI and T. GONZALES "P-complete Approximation Problems" JACM, 23, 1976, 555-565.
- S2 H.M. SALKIN and R.D. KONCAL "Set Covering by an all Integer Algorithm - Computational experience" JACM 20, 1973, 189-193.
- S3 H.M. SALKIN and J. SAHA "Set Covering: Uses Algorithms, Results" Dept. of Tech. Memorandum No. 272, Case Western Reserve, University, 1972.
- S4 M.E. SALVESON "The Assembly Line Balancing Problem" Transactions ASME, 77, 1955, 939-947.
- S5 H.M. SAMUELSSON "Solving Large Set Covering Problems" Working Paper 196, SUNY at Buffalo, 1974.
- S6 S. SENJU and B. TOYODA "Heuristic Method for 0-1 Programming: An approach to LP with 0-1 variables" Man. Sci. 15, 1968, B196-B207.

- S7 J.F. SHAPIRO "A Survey of Lagrangean Techniques for Discrete Optimization" Tech. Report 133, OR Center, Massachusetts Institute of Technology, 1976.
- S8 F. SHEPARDSON and R.E. MARSTEN "A Lagrangean Relaxation Algorithm for the 2 duty Period Scheduling Problem" Technical Report 532, Management Information Systems Dept., University of Arizona, Tucson, AZ, 85721, 1978.
- S9 L. STEINBERG "The Backboard Wiring Problem: a placement algorithm" SIAM Review 3, 1961, 37.
- T1 H.A. TAHA "Integer Programming: Theory, applications and computations" Academic Press, NY, 1975.
- T2 H.M. THIRIEZ "The SCP: A Group Theoretic Approach" RAIRO 3, 1971, 83-104.
- T3 R. TIBREWALA, D. PHILIPPE and J. BROWNE "Manpower Scheduling" Man. Sci. 19, 1972, 71-75
- T4 J. TIND "Blocking and Antiblocking Sets" Math. Prog. 6, 1974, 157-166.
- T5 G.C. TITTERINGTON "An Algorithm to Select a near Minimal Spanning set of vectors" in Bulletin of the Institute of Mathematics and its Application, 13, Feb 1977.
- T6 K.R. TOPALIAN "Tree Search Methods and the Set Covering Problem" MSc Thesis, Department of Management Science, Imperial College, 1971.
- T7 C. TORREGAS and C. REVELLE "Optimal Location under time or distance Constraints" Papers of the Regional Science Association 28, 1972, 133-143.

- T8 C. TORREGAS, R. SWAIN, C. REVELLE and L. BERGMAN "The Location of Emergency Service Facilities" *Opns. Res.* 19, 1971, 1363-1373.
- T9 Y. TOYODA "A Simplified Algorithm for Obtaining Approximate Solutions to 0-1 Programming Problems" *Man. Sci.* 12, 1975, 1417-1427.
- T10 M. TRYPIA and N. CHRISTOFIDES "Sequential Method for 0-1 Programming" *SIAM JI. Appl. Math.* 31, 1976, 271-285.
- T11 N.P. TUAN "A Flexible Tree Search Method" *Opns. Res.* 11, 1977, 972-989.
- V1 B. VON HOHENBALKEN "Least Distance Networks for the Scheme of Polytopes" *Math. Prog.* 15, 1978, 1-11.
- W1 W. WALKER "Application of the Set Covering Problem to the Assignment of Ladder Trucks to Fire Houses" *Opns. Res.* 22, 1974, 275-277.
- W2 G.M. WEBER "Sensitivity Analysis of Optimal Matchings" *Tech. Report 427, School of OR and Ind. Eng., Cornell University, May 1979.*
- W3 D. WEINBERGER "Network Flows, Minimum Coverings and the 4-colour conjecture" *Opns. Res.*, 24, 1976, 272-290.
- W4 L.J. WHITE "Minimum Cover of Fixed Cardinality in Weighted Graphs" 21, *SIAM JI.* 1973, 104-113.
- W5 L.J. WHITE and M. GILLENSON "An Efficient Algorithm for Minimum k-covers in Weighted Graphs" *Math. Prog.*, 8, 1975, 20-42.
- W6 L. WOLSEY "Valid Inequalities and Superadditivity for 0-1 IP's" *Math. of OR*, 2, 1977, 66-77.

- W7 F.P. WYMAN "Binary Programming: a decision rule for selecting Optimal Versus Heuristic Techniques" *Comp. J.* 16, 1973, 135-140.
- Z1 S. ZANAKIS "Heuristic 0-1 Linear Programming, an Experimental Comparison of 3 methods" *Man. Sci* 24, 1977, 91-104.
- Z2 E. ZEMEL "Lifting the Facets of 0-1 polytopes" *Math. Prog.* 15, 1978, 268-277.
- Z3 K. ZORYCHTA "On Converting the 0-1 Linear Programming Problem to an SCP" *Bull. Acad. Polon. Sci. Ser. Sci. Math. Astron. Phys.* 25, 1977, 919-923.

JOURNAL      ABBREVIATIONS

ACM Trans. Math. Soft.	Association for Computing Machinery Transactions on Mathematical Software
AIIE Trans.	American Institute of Industrial Engineers Transactions
Am. Math. Month.	American Mathematical Monthly
Bell Syst. Tech. Jl.	Bell System Technical Journal
Bulletin AMS	Bulletin of the American Mathematical Society
Canadian Jl. of Math.	Canadian Journal of Mathematics
CCERO	Cahiers du Centre d'Etudes de Recherche Operationnelle
Comp. and Opns. Res.	Computers and Operations Research
Comp. Jl.	Computer Journal
EJOR	European Journal of Operational Research
INFOR	Canadian Journal of Operational Research
Int. Jl. Comp. and Inf. Sci	International Journal Computer and Information Science
JACM	Journal for the Association of Computing Machinery
Jl. Comb. Th.(B)	Journal of Combinatorial Theory, Series B
Jl. Comp. and Syst. Sci.	Journal of Computer and System Science
JORS	Journal of the Operational Research Society (formerly ORQ)
Man. Sci.	Management Science
Math. of OR	Mathematics of Operational Research



Math. Prog.	Mathematical Programming
NRLQ	Naval Research Logistics Quarterly
Opns. Res.	Operations Research
ORQ	Operational Research Quarterly
RAIRO	Revue d'Automatique d'Informatique et de Recherche Operationnelle - Operational Research
SIAM JI.	Journal of the Society for Industrial and Applied Mathematics (SIAM)
SIAM JI. Appl. Math.	SIAM Journal of Applied Mathematics
SIAM JI. of Computation	SIAM Journal of Computation
SIAM JI. Num. Anal.	SIAM Journal of Numerical Analysis
SIAM Review	Siam Review
Trans. Sci	Transportation Science.

APPENDIX 1

Analysis Of Preliminary Reduction Strategies

To Prove

For an  $m \times n$  SCP in which the fixed probability that  $a_{ij} = 1$  is  $p$  the probability given any two rows  $i_1$  and  $i_2$  that either row  $i_1$  dominates row  $i_2$  or vice versa is:

$$2(1 - p + p^2)^n - (1 - 2p + 2p^2)^n$$

Proof

Suppose row  $i_1$  has  $r$  non-zero entries then the probability that row  $i_2$  has  $r$  non-zero entries in the same position and hence row  $i_1$  dominates row  $i_2$  is:

$$p^r \tag{A1.1}$$

This case is shown below:

row $i_1$	1	1	1	1...1	0	0	0	...	0
row $i_2$	1	1	1	1...1	1	0	0	...	1
	$r$ non-zero entries								

row  $i_1$  dominates row  $i_2$

The second case is when row  $i_2$  dominates row  $i_1$  and the rows are not equal. Given that row  $i_1$  has  $r$  non-zero entries the probability that row  $i_2$  has 0 whenever row  $i_1$  has 0 in a column and row  $i_2$  has 0 in at least one of the  $r$  non-zero columns of  $i_1$  is:

$$(1 - p^r)(1 - p)^{n-r} \tag{A1.2}$$

The case is shown below:



APPENDIX 2INDEX OF TERMINOLOGY

This list gives the section of the thesis in which a term is described. The page number is the page on which the term is first used.

<u>TERM</u>	<u>SECTION</u>	<u>PAGE</u>
Active node	1.2	4
Affine independence	1.3.3.4	27
Applications of the SCP	1.3.1	7
Balanced matrix	1.3.4.3	29
Bank float location problem (BLP)	1.3.2.2	14
Best bound tree search	1.2	6
Branch and bound	1.2, 1.3.3.5	4
Branched node	1.2	6
Branching strategies	1.3.3.5, 7	23
Breadth-first tree search	1.2	6
Brother node	1.2	6
Chord	1.3.2.1	9
Chordal graph	1.3.2.1	9
Circle graph	1.3.2.1	9
Circuit	1.1	2
Claw free graphs	1.3.2.1	9
Complexity results	1.3.4.1	26

<u>Term</u>	<u>Section</u>	<u>Page</u>
Cover	1.2, 5.2	4
Crew scheduling	1.3.1	7
Cutting planes	1.3.3.6	25
Cyclic matrix, $A(p,q)$	1.3.2.1	12
Data structures	1.3.5	30
Decomposition	6	134
Depth-first tree search	1.2	6
Dual feasibility	2.2	35
Dual integer program	1.3.4.4	29
Dual linear program (DLP)	1.2	2
Dual set covering problem (DSCP)	1.3.4.4.	29
Dynamic programming relaxations	6	134
Dynamic set covering problem (DYSCP)	1.3.2.2	15
Facets	1.3.4.2	28
Father node	1.2	4
Fathomed	1.2	4
Feasible solution	1.2	4
Generalised Lagrangean Relaxation (GLR(F))	1.3.4.4.	29
Generalised set covering problem (GSCP)	1.3.2.2	15
Graph	1.1	1
Graph Covering Problem(GCP)	1.1, 1.3.2.1, 5.2	1
Graph Covering relaxations	5	96
Graph Covering	1.1, 5.2	1

<u>TERM</u>	<u>SECTION</u>	<u>PAGE</u>
Heuristics	2	31
Heuristics for Lower Bounds to the SCP	2.2	32
Heuristics for Upper Bounds to the SCP	2.2,2.3	32
Information Retrieval	1.3.1	7
Integer Program (IP)	1.2	2
Interval graph	1.3.2.1	8
Knapsack Problem (KP)	1.2	2
Lagrange Multiplier, $\lambda$	1.2, 3	3
Lagrangian Relaxation, LR( $\lambda$ )	1.2, 3.1	2
Line Graph	1.3.2.1	9
Linear Programming Relaxation(LP)	1.2	2
Lower Bound, $z_\epsilon$	1.2, 1.3.3.3	4
Minimal Cover	1.1	1
Near-alternative solutions	3.4.3.	63
Network Flow Relaxations	4.1	78
Network Flow Relaxation, NF1	4.2	83
Node Covering Problem (NCP)	1.3.2.1	8
Non-deterministic Algorithm	1.3.4.1	26
NP Complete	1.3.2.1	8
Partitioning the constraints	5.3.4	100
Path	1.1	1
Polyhedra	1.3.4.2	27
Preliminary reductions	1.3.3.1	18
Prime cover	1.2	4
Production Planning	1.3.1	8

<u>TERM</u>	<u>SECTION</u>	<u>PAGE</u>
Projection Method	3.2	52
Related Problems (to the SCP)	1.3.2	8
Relaxation	1.2	2
Root Node	1.2	4
Rotation of constraints	5.3.4	4
Routing Problems	1.3.1	7
Row Relaxation	3, 5.3	7
Scheduling Problems	1.3.1	7
Search Directions	3.3	55
Sensitivity Analysis	1.3.5	30
Set Covering Problem (SCP)	1.1	1
Set Partitioning Problem (SPP)	1.1	1
Shortest Path Problem	1.3.2.1	10
Sorting the Constraint Matrix	1.3.3.2	21
State Space Relaxation	6	134
Stepsize	3.4	61
Subgradient	1.2, 3	3
Subgradient Optimization	1.2, 3.2	3
Successor Node	1.2	2
Target Value	3.4.2	62
Theoretical Results	1.3.4	26
Tree	1.1	2
Tree Search	1.2	2
Uncapacitated Plant Location Problems (UPLP)	1.3.2.2	3
Unimodularity	1.3.4.3	28
Upper Bound, $z_u$	1.2, 1.3.34	4
Vertex arc incidence matrix	5.2	96
Well-solved cases of the SCP	1.3.2.1	8

APPENDIX 3AN EXAMPLEA3.1 Introduction

This appendix illustrates the algorithms developed in each chapter on the following example, SCP:

EXAMPLE A1

$$\begin{array}{l}
 \text{SCP} \left[ \begin{array}{l}
 \text{minimise } [4 \quad 5 \quad 3 \quad 2 \quad 4 \quad 3] x \\
 \text{subject to } \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \\
 x_j = 0 \text{ or } 1 \quad j = 1, 2, \dots, n
 \end{array} \right.
 \end{array}$$

The optimal solution to the SCP is given by  $x^* = (0, 1, 0, 0, 0, 1)$  and has a value  $v(\text{SCP})$ , equal to 8. The optimal solution of the corresponding LP,  $x_{LP} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, 0)$  and has value 7. The solution to DLP is  $u^* = (2, 0, 1, 1, 2, 1)$  and the bound obtained is obviously also 7.

A3.2 The Heuristic Lower Bounds, Chapter 2

The heuristic algorithms for upper and lower bounds are described.

Procedure 1 INITIAL BOUNDS starts with the column sums:

$$h = (3, 4, 3, 3, 3, 3)$$



and the dual variables are all zero giving reduced costs:

$$s = (4, 5, 3, 2, 4, 3).$$

### Iteration 1

The least cost per row satisfied is given by  $\Delta = 0.6$  from column 4 which covers rows 2, 4 and 6. Thus  $u_2 = u_4 = u_6 = 0.6$ .

Rows 2, 4 and 6 are removed from the problem and updating the reduced costs and column sums gives:

$$s = (3.3, 3.6, 2.3, 0, 3.3, 1.6)$$

$$h = (2, 2, 2, 0, 2, 1)$$

### Iteration 2

The least reduced cost per row covered is given by  $\Delta = 1.16$  from column 3. Thus  $u_3 = u_5 = 1.16$ . Rows 3 and 5 are removed and reduced costs and column sums are updated to give:

$$s = (2.16, 2.5, 0, 0, 1, 0.5)$$

$$h = (1, 1, 0, 0, 0, 0)$$

### Iteration 3

The least reduced cost per row covered is given by  $\Delta = 2.16$  from column 1. Thus  $u_1 = 2.16$ . All rows are covered and a prime cover is given by:

$$x = (1, 0, 1, 1, 0, 0) \quad \text{with cost } z_u = 9$$

and a dual feasible solution to DLP is:

$$u = (2.16, 0.6, 1.16, 0.6, 1.16, 0.6) \quad \text{with lower bound } z_l = 6.5.$$

Procedure 2 LPBOUND then finds rows for which  $u_i \{a_i^i x - 1\} \neq 0$ . In this case rows 2, 3 and 6 do not satisfy the complementary slackness conditions.

For row 2  $u_2$  is set equal to 0 and the reduced costs are then:

$$s = (0, 1.0, 0.6, 0.6, 1, 0.5).$$

Considering row 4,  $u_4$  is increased to 1.16 and the reduced costs become:

$$s = (0, 0.5, 0.6, 0.16, 0.5, 0)$$

Then  $u_5$  is set equal to 1.6. Hence the lower bound is increased to 6.83. Reduced costs are now:

$$s = (0, 0, 0.16, 0.16, 0, 0)$$

and  $u = (2.16, 0, 1.16, 1.16, 1.6, 0.6)$

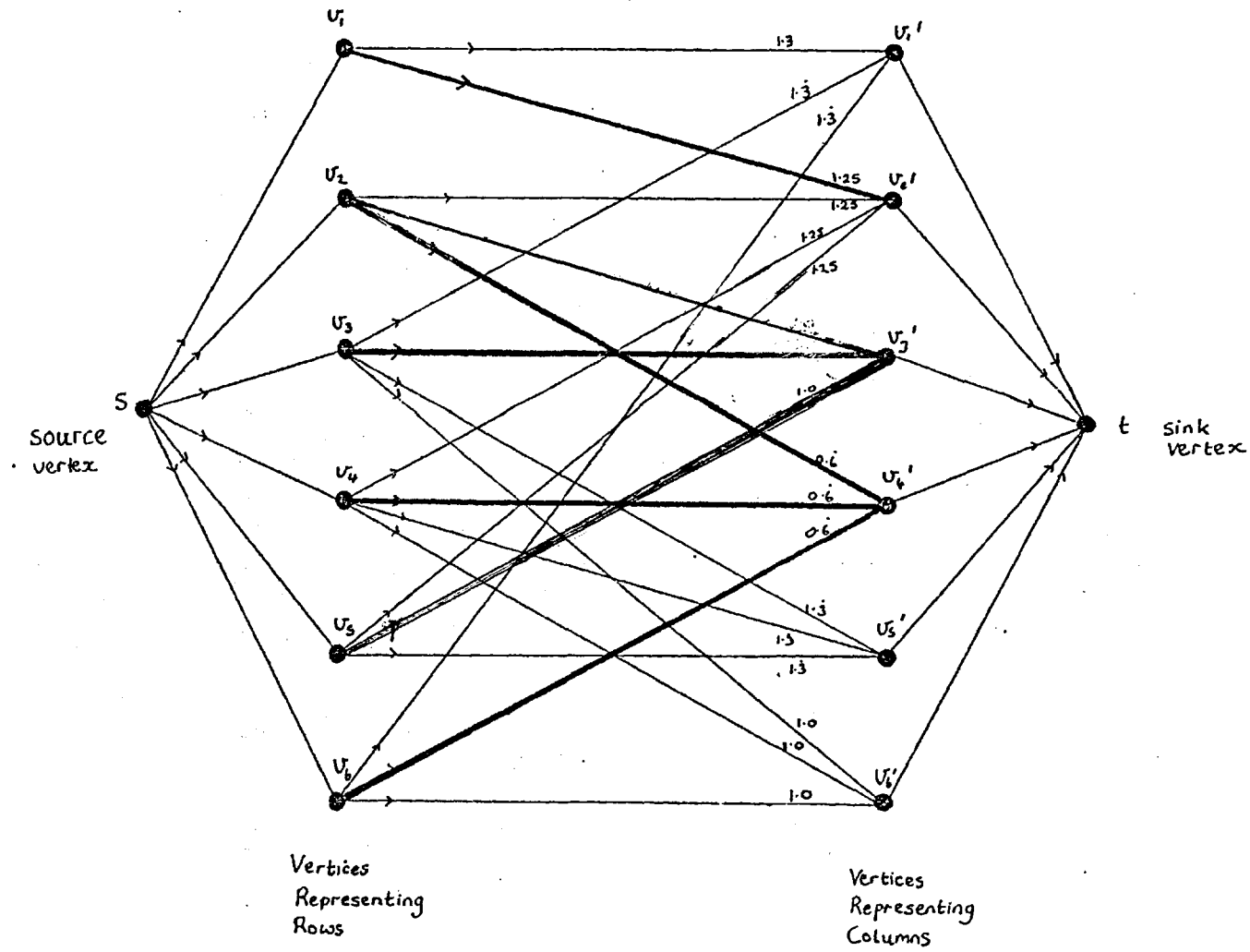
A new prime cover is found because  $s_j x_j \neq 0$  for  $j = 3, 4$ . Thus the new cover must be chosen from the columns 1, 2, 5 and 6, for example  $x_1 = x_2 = 1$  with cost 9. Subsequent iterations do not yield an improvement in the lower bound value.

### A3.3 The Network Flow Lower Bounds $v(\text{NF1})$ , Chapter 4

This section illustrates the first of two network flow relaxations, NF1(d) together with subgradient optimization.

An initial matrix of costs  $d_{ij}^0 = c_j/h_j$  for all  $j, i \in M_j$ , is given below:

Fig A3.1 Graph for Network Flow Relaxation NF1.



$i \backslash j$	1	2	3	4	5	6
1	1.33	<u>1.25</u>				
2		1.25	1.00	<u>0.66</u>		
3	1.33		<u>1.00</u>		1.33	1.00
4		1.25		<u>0.67</u>	1.33	1.00
5		1.25	<u>1.00</u>		1.34	
6	1.34			<u>0.67</u>		1.00

Zeros are omitted and  $d_{ij}^0$  is underlined whenever  $\xi_{ij} = 1$ . The value of the lower bound is 5.25. The network flow solution is shown in Fig.A3.1. The costs  $d_{ij}^0$  are changed in columns 2 and 3, for which feasibility for the SCP is not satisfied. Suppose that for the subgradient ascent  $\alpha$  is chosen so that  $\alpha(z_u^F - z_l^F)/\|w\|^2 = 0.5$  then  $p_2 = 1, h_2 = 4, p_3 = 2, h_3 = 3$  giving:

$$\pi_{12} = 0.475 \quad \text{and} \quad \pi_{22} = \pi_{42} = \pi_{52} = -0.125$$

$$\text{and} \quad \pi_{32} = -0.333 \quad \text{and} \quad \pi_{33} = \pi_{35} = 0.166$$

The costs are then changed to  $d_{ij}^0$  given below:

$i \backslash j$	1	2	3	4	5	6
1	<u>1.33</u>	1.63				
2		1.12	0.66	<u>0.66</u>		
3	1.33		1.17		1.33	<u>1.00</u>
4		1.12		<u>0.67</u>	1.33	1.00
5		<u>1.13</u>	1.17		1.34	
6	1.34			<u>0.67</u>		1.00

The bound then increases to 5.46.

At the final iteration the costs  $d_{ij}^0$  were given by:

$i \backslash j$	1	2	3	4	5	6
1	2.06	<u>2.02</u>				
2		<u>-0.02</u>	0.04	0.00		
3	<u>0.92</u>		0.96		0.97	1.00
4		0.95		<u>0.94</u>	0.94	0.95
5		2.05	<u>2.00</u>		<u>2.09</u>	
6	<u>1.02</u>			1.06		1.05

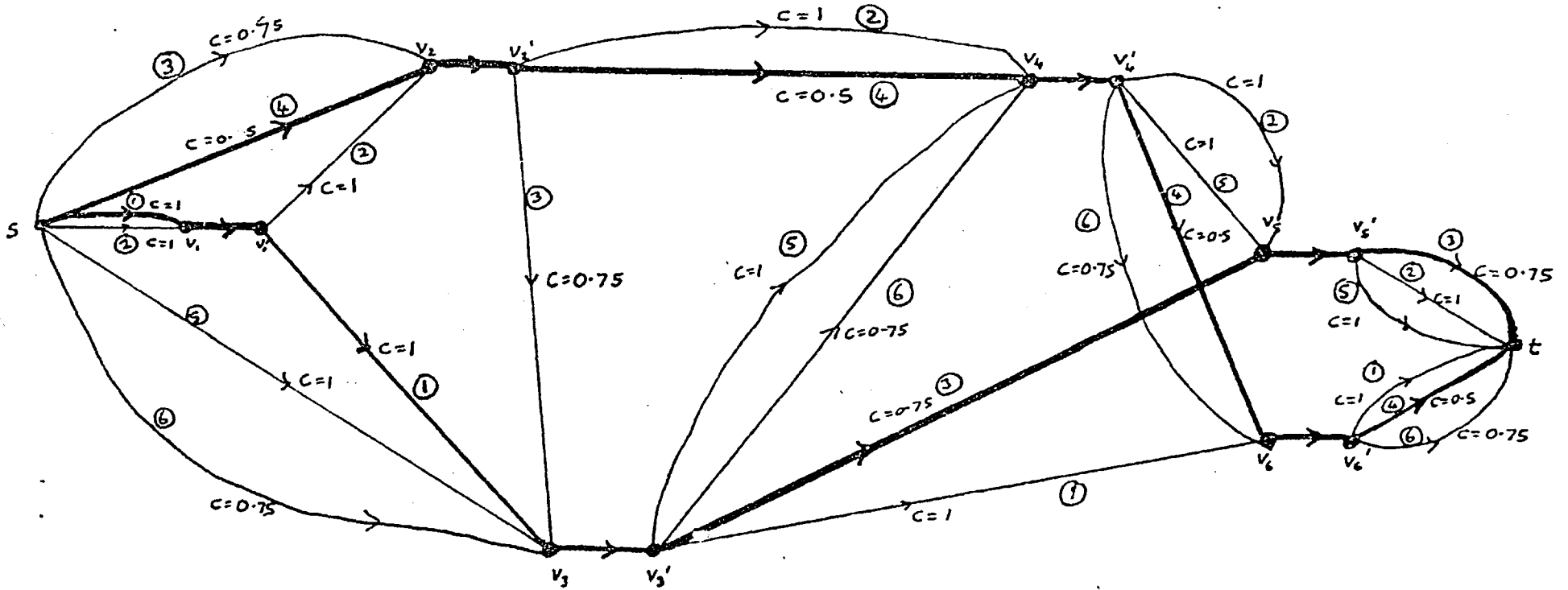
The lower bound from this solution is 6.88. The costs can be changed to non-negative values so that  $d_{52}$  becomes  $2.05 - 0.02 = 2.03$  and  $d_{22}$  becomes 0. The dual feasible solution  $u$  obtained from these costs is then  $(2.02, 0.00, 0.92, 0.94, 2.00, 1.02)$  giving a slightly improved bound of 6.90. The corresponding reduced costs  $s_j$  are given by  $s = (0.04, 0.04, 0.08, 0.04, 0.14, 0.12)$  and since  $\min_{j \in N_i} s_j > 0$  for all  $i$  it is possible to increase  $u_5$  and  $u_6$  each by 0.04, giving  $u = (2.02, 0.00, 0.92, 0.94, 2.04, 1.06)$ . The corresponding reduced costs are then  $(0.00, 0.00, 0.04, 0.00, 0.10, 0.08)$  and the lower bound is 6.98. The costs  $d_{ij}$  are now as shown below:

$i \backslash j$	1	2	3	4	5	6	$u$
1	<u>2.02</u>	2.02					2
2		<u>0.00</u>	0.02	0.00			0
3	<u>0.92</u>		0.93		0.95	0.95	1
4		<u>0.94</u>		0.94	0.97	0.97	1
5		<u>2.04</u>	2.05		2.08		2
6	<u>1.06</u>			1.06		1.08	1

It is not easy to find an LP feasible solution from this result. The feasible solution suggested for the SCP here is  $x_1 = x_2 = 1$ . This solution has a cost of 9.

FIGURE A3.2

Graph For Example, NF2



The optimal flow is shown in heavy lines and has a cost of 5.5.

① column of the SCP from which the arc is derived

### A3.4 The Second Network Flow Lower Bound, $v(\text{NF2})$

The graph for the second network flow relaxation is shown in Fig.A3.2.

Initial costs  $d_{ik}^j$  are given by  $c_j / (h_j + 1)$ . Each arc is labelled with a cost and the index,  $j$ , of the column from which it is derived is circled. The initial bound has value 5.5.

In this example set covering feasibility constraints are violated for  $j=1$  and 3. This means that costs in arcs  $(s, v_1)^1$ ,  $(v_1', v_3)^1$ ,  $(v_3', v_5)^3$ ,  $(v_5', t)^3$  are increased and costs in arcs  $(v_3, v_6)^1$ ,  $(v_6', t)^1$ ,  $(s, v_2)^3$  and  $(v_2', v_3)^3$  are decreased. The number of arcs derived from column 1 in which the flows are non-zero,  $p_1 = 2$  and for column 3,  $p_3 = 2$ .

Suppose  $\alpha$  is chosen so that  $\frac{\alpha(z_u - z_l)}{\|w\|^2} = 0.5$

$$\text{then } \pi_{s1}^1 = \pi_{13}^1 = \pi_{25}^3 = \pi_{5t}^3 = 0.25$$

$$\text{and } \pi_{36}^1 = \pi_{6t}^1 = \pi_{s2}^3 = \pi_{23}^3 = -0.25$$

The corresponding costs are changed to give values as follows:

$$d_{s1}^1 = d_{13}^1 = 1.25$$

$$d_{35}^3 = d_{5t}^3 = 1.00$$

$$d_{36}^1 = d_{6t}^1 = 0.75$$

$$d_{s2}^3 = d_{23}^3 = 0.5$$

A minimum cost flow in the resulting graph is non-zero in the same arcs as before and has cost 6.5 which is greater than the initial flow of 5.5.

FIGURE A3.3

Graph  $\bar{G}$ , The Complement Of The Row Intersection Graph

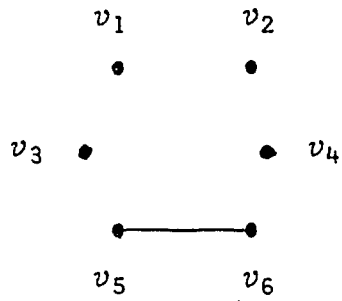
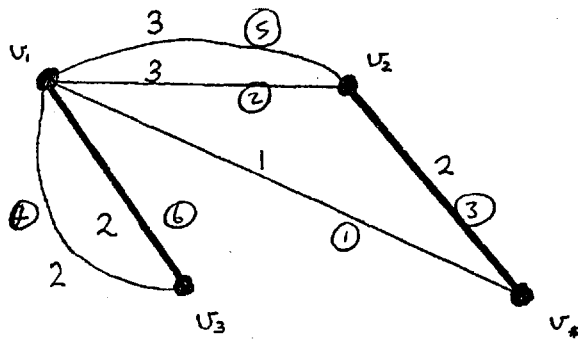


FIGURE A3.4

Graph Covering Problem For GCR1





A3.5 The First Graph Covering Relaxation, GCR1, Chapter 5

A row intersection graph for the example is the complement of the graph  $\bar{G}$  shown in Fig.A3.3. The  $i$ th row is represented by a vertex,  $v_i$ , of  $\bar{G}$  and  $(v_i, v_k)$  is an arc whenever rows  $i$  and  $k$  do not intersect. A maximal clique in  $\bar{G}$  is  $\{v_5, v_6\}$  corresponding to rows 5 and 6 which are disjoint in the SCP. Adding row 4 to rows 5 and 6 gives 3 rows which make a graph covering problem. Assume the multipliers for the relaxed constraints are equal to the LP optimal dual variables. Hence  $\lambda_1 = 2$ ,  $\lambda_2 = 0$  and  $\lambda_3 = 1$ . Then the graph covering problem GCR1 is:

$$\begin{aligned} & \min [ 1 \quad 3 \quad 2 \quad 2 \quad 3 \quad 2 ] x + 3 \\ & \text{subject to } \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} x \geq \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\ & \quad \quad \quad x_j \in \{0,1\} \quad j = 1,2,\dots,6 \end{aligned}$$

The graph covering problem is shown in Fig.A3.4 and an extra vertex  $v_*$  has been added to ensure that each column has at least two non-zero entries in it. Each arc is labelled with its cost and the column from which it is derived is encircled. The graph covering solution has cost 4, given for example by  $x_3 = x_4 = 1$ , and added to the Lagrange multipliers this gives a lower bound of 7 to the SCP.

Other solutions are  $x_1 = x_2 = 1$ ,  $x_3 = x_6 = 1$  and  $x_1 = x_5 = 1$ . The subgradients  $(1 - Ax)$  for relaxed constraints corresponding to these 4 solutions are  $(1, -1, 0)$ ,  $(-1, 0, 0)$ ,  $(1, 0, -1)$  and  $(0, 1, -1)$ . The optimal value of the Lagrangean function is 7.

### A3.6 The Second Graph Covering Relaxation, GCR2, Chapter 5

The initial costs are derived from the optimal dual variables to the LP relaxation to give the following graph covering problem:

Column No.    1    2    3    4    5    6    7    8    9    10    11    12

$$\min d \cdot y = [3 \quad 1 \quad 2 \quad 3 \quad 1 \quad 2 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 1] y$$

$$\text{subject to } \begin{bmatrix} 1 & & 1 & & & & & & & & & & \\ & & 1 & & 1 & & 1 & & & & & & \\ & & & & & & & & & & & & \\ 1 & & & & 1 & & & & & 1 & & 1 & \\ & & & 1 & & & 1 & & & & & 1 & \\ & & & & 1 & & 1 & & & & 1 & & \\ & & & & & & & & & & & & \\ & 1 & & & & & & & & & & & 1 \end{bmatrix} y \geq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$y_t \in \{0,1\} \quad t = 1,2,\dots,12$$

(0's have been omitted from the constraint matrix)

An optimal solution is given by  $y_3 = y_6 = y_{11} = y_{12} = 1$ , giving a lower bound equal to 7. The solution is not feasible for the SCP because  $y_4 = 0$  and  $y_5 = 0$  therefore the costs  $d_4$  and  $d_5$  decreased to give costs:

$$d = [3 \quad 1 \quad 2.5 \quad 2.5 \quad 1.5 \quad 1.5 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 1]$$

The lower bound given by the graph covering relaxation is then 7.

### A3.7 A Second Example To Illustrate A Combination Of The Two Graph Covering Relaxations, Chapter 5

To illustrate the use of both graph covering relaxations together a second example will be used.

Example A2

Consider the SCP with costs  $c$ , and constraint matrix  $A$  given below.

Column No.	1	2	3	4	5	6	7	8	9	10	11	12
Cost:	1	1	1	2	2	2	6	3	6	6	3	6
Row No.	1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	1	1	0	0	0	0	0	0	0	0
2	1	1	0	0	1	0	0	0	0	0	0	0
3	0	1	1	0	0	1	0	0	0	0	0	0
4	0	0	0	1	0	0	1	1	1	0	0	0
5	0	0	0	0	1	0	0	1	1	1	0	0
6	0	0	0	0	0	1	1	1	0	0	1	0
7	0	0	0	0	0	0	1	0	1	1	0	1
8	0	0	0	0	0	0	1	0	1	0	1	1
9	0	0	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	1	1	0	1	1

A heuristic solution to the dual of the LP relaxation of SCP is given by:  $u = (1, 0, 0, 1, 1, 1, 1, 1, 0, 0)$ , giving a lower bound of 6.

The procedure for partitioning  $A$  gives a row intersection graph  $G_0$  from which a maximal independent set  $I^0 = \{1, 5\}$  can be removed. In the resulting graph  $I^1 = \{2, 4\}$  is a second maximal independent set. Thus rows 1, 2, 4 and 5, corresponding to  $I^0 \cup I^1$ , from  $A_2$ . To these rows can be added row 3. Then  $M_1$ , the set of relaxed constraints, is  $\{6, 7, 8, 9, 10\}$  and  $M$ , the set of graph covering constraints is  $\{1, 2, 3, 4, 5\}$ .

Using the values of the given vector  $u$  to get initial multipliers  $\lambda$ , the resulting problem GCR1 is:

Column No.	1	2	3	4	5	6	7	8	9	10	11	12	
$\min$	[ 1	, 1	, 1	, 2	, 2	, 1	, 3	, 2	, 4	, 5	, 1	, 4 ]	$x + 3$

Subject to

$$\begin{array}{r}
 \text{Row No. } 1 \\
 \phantom{\text{Row No. }} 2 \\
 \phantom{\text{Row No. }} 3 \\
 \phantom{\text{Row No. }} 4 \\
 \phantom{\text{Row No. }} 5
 \end{array}
 \begin{bmatrix}
 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0
 \end{bmatrix}
 x \geq
 \begin{bmatrix}
 1 \\
 1 \\
 1 \\
 1 \\
 1
 \end{bmatrix}$$

A solution is  $x_1 = x_6 = x_8 = 1$  which gives a bound of 7 to the SCP.

Constraints 7 and 8 are not satisfied and constraint 6 is oversatisfied since  $x_6 + x_8 > 1$ . This means that  $\lambda_6$  must be decreased and  $\lambda_7$  and  $\lambda_8$  increased. Suppose the values of  $\lambda$  are changed to  $\lambda_6 = 0, \lambda_7 = 2, \lambda_8 = 2$ . The objective function in GC1 then becomes:

$$(1, 1, 1, 2, 2, 2, 2, 3, 2, 4, 1, 2) x + 4$$

which has a solution  $x_1 = x_2 = x_9 = 1$  and the lower bound is 8.

Subgradient optimization is continued until no further increase in the bound is possible. The relaxation can now be rotated. Since  $x_1 = x_2 = x_9 = 1$  constraints 6 and 9 are not satisfied. In this example, the constraints that are not satisfied have equal multipliers,  $\lambda_6 = \lambda_9 = 0$ ; therefore suppose constraint 6 is chosen arbitrarily to be added to the graph covering constraints. Column 8 then has 3 1's in it so the program is no longer a GCP. Either constraint 4 or constraint 5 can be removed from the GCP, suppose constraint 4 is relaxed. Then constraint 7 can also be added to the graph covering constraints. Therefore  $M_1 = \{4, 8, 9, 10\}$  and  $M_2 = \{1, 2, 3, 5, 6, 7\}$ . We compute initial values of  $\lambda$ :  $\lambda_4 = 1, \lambda_8 = 2, \lambda_9 = \lambda_{10} = 0$ .

Then the GCP after the rotation is:

$$\begin{array}{r}
 \text{Column No. } 1 \\
 \phantom{\text{Column No. }} 2 \\
 \phantom{\text{Column No. }} 3 \\
 \phantom{\text{Column No. }} 4 \\
 \phantom{\text{Column No. }} 5 \\
 \phantom{\text{Column No. }} 6 \\
 \phantom{\text{Column No. }} 7 \\
 \phantom{\text{Column No. }} 8 \\
 \phantom{\text{Column No. }} 9 \\
 \phantom{\text{Column No. }} 10 \\
 \phantom{\text{Column No. }} 11 \\
 \phantom{\text{Column No. }} 12
 \end{array}
 \min [ 1, 1, 1, 1, 2, 2, 3, 2, 3, 6, 1, 4 ] x + 3$$



Col.No. of SCP  $j$  1 2 3 4 5 6 7 8 8 9 9 10 10 11 12 12

Col.No. of GCP  $t$  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

min	Cost	1	1	1	1	2	2	3	2	0	3	0	5.7	0.3	1	3.7	0.3
	Row No																
	1	1		1	1												
	2	1	1			1											
	3		1	1			1										
	5					1			1		1		1				
	6						1	1	1						1		
	7							1			1		1				1
	9									1				1			1
	10									1		1			1		1

The solution  $y_3 = y_5 = y_7 = y_9 = 1$  gives a lower bound of 9 and  $y_8 \neq y_9$ ; therefore the set covering problem is not solved and  $d_8$  must be decreased and  $d_9$  increased. Taking  $\pi_8 = -0.5$  and  $\pi_9 = 0.5$  the costs are changed to  $d_8 = 1.5$  and  $d_9 = 0.5$ . The optimal solution is  $y_1 = y_6 = y_{10} = y_{11} = y_{13} = 1$ . The bound is 9.3. Since  $y_{12} = 0$  and  $y_{13} = 1$ ,  $d_{12}$  can be decreased to 5.2 and  $d_{13}$  increased to 0.8. The costs are then:

Col.No. of GCP, $t$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	+ 3
$d_t$	1	1	1	1	2	2	3	1.5	0.5	3	0	5.2	0.8	1	3.7	0.3	

The solution is  $y_3 = y_5 = y_7 = y_9 = 1$  giving a bound of 9.5. Again the set covering feasibility are not satisfied for  $j = 8$ . Thus, increasing  $d_9$  to 1.0 and reducing  $d_8$  to 1.0 gives:

Col.No. of GCP, $t$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	+ 3
$d_t$	1	1	1	1	2	2	3	1	1	3	0	5.2	0.8	1	3.7	0.3	

A solution is  $y_1 = y_2 = y_8 = y_{11} = y_{15} = 1$ . The cost of this solution is 9.7; etc.

It is worthwhile to note here that the value of the LP solution to the example used in this section is 9.5.

### A3.8 Lower Bounds From Decomposition, Chapter 6

Using example A1 again and splitting the constraints as in the first graph covering relaxation GCR1 one obtains the 2 SCP's, S1 from rows 1 to 3 of the SCP and S2 from the remaining rows:

$$S1 \left[ \begin{array}{l} \min_{y'} c'y' = [3 \quad 2 \quad 1 \quad 0 \quad 1 \quad 1] y' \\ \text{subject to} \end{array} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} y' \geq \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right.$$

$$y'_j \in \{0,1\} \quad j = 1,2,\dots,6$$

and

$$S2 \left[ \begin{array}{l} \min_{y''} c''y'' = [1 \quad 3 \quad 2 \quad 2 \quad 3 \quad 2] \\ \text{Subject to} \end{array} \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} y'' \geq \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right.$$

$$y''_j \in \{0,1\} \quad j = 1,2,\dots,6$$

An optimal solution to S1 with value 3 is  $y' = (0, 1, 0, 0, 0, 1)$  and to S2 with value 4 is  $y'' = (0, 0, 1, 0, 0, 1)$ . The lower bound to the SCP is then 7.

Decreasing the costs  $c''_2$  and  $c'_3$  and increasing costs  $c'_2$  and  $c''_3$  because  $y''_2 = y'_3 = 0$  and  $y'_2 = y''_3 = 1$  gives costs:

$$c' = [ 3 \quad 2.5 \quad 0.5 \quad 0 \quad 1 \quad 1 ]$$

$$c'' = [ 1 \quad 2.5 \quad 2.5 \quad 2 \quad 3 \quad 2 ]$$

then  $v(SC1) = v(SC2) = 3.5$  giving again a lower bound equal to 7.

A3.9 Lower Bounds From State Space Relaxation, SSR1

A state  $S$  is given a mapping  $g(S) = (|S|, i)$  where  $i$  is the last non-zero index of  $S$ .  $F_k(g(S))$  is the value of state  $S$  at iteration  $k$  of the state space relaxation.  $F_0(g(S)) = 0$  for all  $S$  and

$$F_k(g(S)) = \min_{j \in N_k} [F_{k-1}(g(S-a_j)) + c_j].$$

The SCP is:

$$\min [ 4 \quad 5 \quad 3 \quad 2 \quad 4 \quad 3 ] x$$

subject to

$$\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & 1 & & \\ & 1 & & 1 & & 1 & 1 \\ & & 1 & & 1 & 1 & \\ & & & 1 & 1 & & \\ 1 & & & & & & 1 \end{bmatrix} x \geq 1$$

$$F_0(0,0) = 0$$

$$x_j \in \{0,1\} \quad j = 1,2,\dots,n$$

Iteration 1

$$F_1(1,1) = \min [0+4, 0+5] = 4 \quad \text{from row 1}$$

Iteration 2

$$F_2(1,2) = \min [F_0(0,0) + 2, F_0(0,0) + 3] = 2 \quad \text{from row 2}$$

$$F_2(2,2) = \min [F_0(0,0) + 5, F_1(1,1) + 3, F_1(1,1) + 2] = 5 \quad \text{from row 2}$$



Iteration 3

$$F_3(1,3) = \min [F_0(0,0) + 4, F_0(0,0) + 3] = 3$$

$$F_3(2,3) = \min [F_0(0,0) + 4, F_0(0,0) + 3, F_1(1,1) + 4, F_1(1,1) + 3, \\ F_2(1,2) + 4, F_2(1,2) + 3] = 3$$

$$F_3(3,3) = \min [F_2(1,2) + 4, F_1(1,1) + 3, F_2(2,2) + 4, F_2(2,2) + 3] = 6$$

The remaining states have values:

$$F_4(1,4) = 2, F_4(2,4) = 2, F_4(3,4) = 5, F_4(4,4) = 5$$

$$F_5(1,5) = 3, F_5(2,5) = 3, F_5(3,5) = 3, F_5(4,5) = 5, F_5(5,5) = 5$$

$$F_6(1,6) = 2, F_6(2,6) = 2, F_6(3,6) = 2, F_6(4,6) = 5, F_6(5,6) = 6, F_6(6,6) = 7$$

Again the lower bound equals the LP bound having a value of 7.

A3.10 Branching Strategies For The SCP, Chapter 7

A depth first tree search strategy branching on rows is shown in Fig.

A3.5. The first branching row is row 1 and  $x_1$  is fixed equal to 1.

The next branching row is row 2 and  $x_2$  is fixed equal to 1. This

covers all rows hence in the absence of lower bounds. The tree search

backtracks  $x_2$  is set equal to 0 and  $x_3$  is set equal to 1. Row 4 is

the next branching row and  $x_4$  is set equal to 1. The search backtracks

and  $x_4$  is set equal to 0 and  $x_5 = 1$ . Then  $x_5$  is set equal to 0 and

$x_6 = 1$ . Thus, row 4 has been completely considered and backtracking to

row 2 takes place where  $x_3$  is set equal to 0 and  $x_4$  becomes 1. The

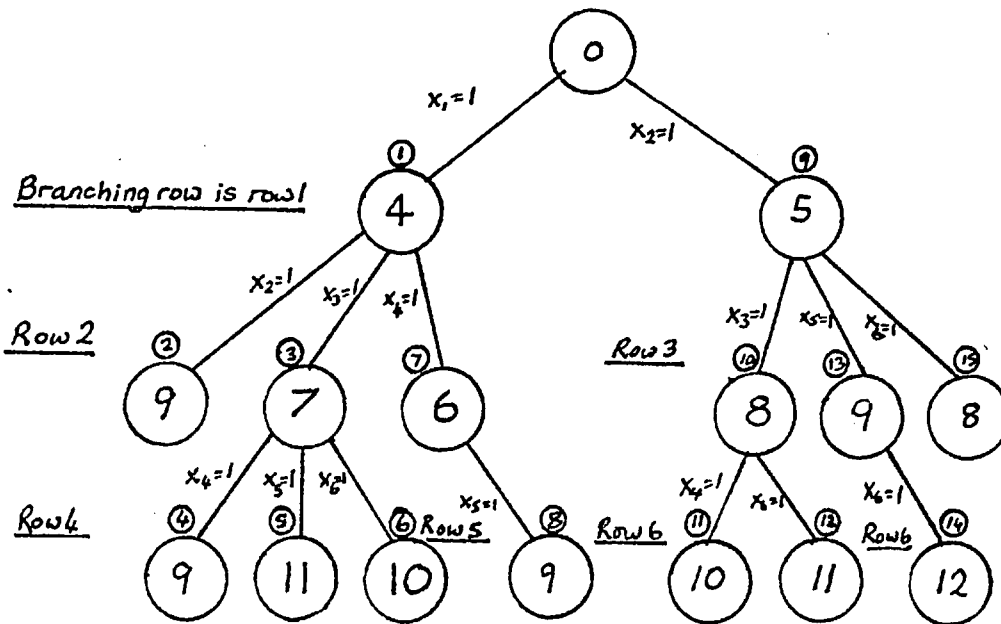
rest of the search is shown in Fig.A3.5.

A breadth first tree search would trace the same nodes in the order:

1, 9, 2, 3, 7, 10, 13, 15, 4, 5, 6, 8, 11, 12, 14.

FIGURE A3.5

A Depth First Tree Search Branching On Rows For The SCP



$z_F$  gives the total cost of the variables fixed equal to 1.

Optimal solution  $x_2 = x_6 = 1$  cost = 8

Ⓜ node number  $n$

A3.11 Implementation, Chapter 8

Storage of the constraint matrix in the list of non-zero rows JTI with pointer JP to the end of each column gives for example A1:

$$\begin{aligned} \text{JTI} &= (1, 3, 6, 1, 2, 4, 5, 2, 3, 5, 2, 4, 6, 3, 4, 5, 3, 4, 6) \\ \text{JP} &= ( \quad 3, \quad \quad 7, \quad \quad 10, \quad \quad 13, \quad \quad 16, \quad \quad 19) \end{aligned}$$

and the list of non-zero columns ITJ with pointer IP to the end of each row is:

$$\begin{aligned} \text{ITJ} &= (1, 2, 2, 3, 4, 1, 3, 5, 6, 2, 4, 5, 6, 2, 3, 5, 1, 4, 6) \\ \text{IP} &= ( \quad 2, \quad \quad 5, \quad \quad 9, \quad \quad 13, \quad \quad 16, \quad \quad 19) \end{aligned}$$

In the first graph covering relaxation GCR1 the first three constraints are relaxed giving MREL, the number of relaxed constraints equal to 3 and the list of relaxed constraints equals:

$$\text{LREL} = (1, 2, 3)$$

The linked list ILK that links rows not in the problem starts at row 4 and equals:

$$\text{ILK} = (-1, -1, -1, 5, 6, 0)$$

For the second graph covering relaxation, GCR2, the linked list that defines how the columns are split, JLK, is given as:

$$\text{JLK} = (1, 7, 2, 8, 3, 9, 4, 10, 5, 11, 6, 12, 0)$$

In a branching strategy consider node 5 for the depth first tree search of Fig.A3.6. This is at depth 3 in the tree, hence LEV, the depth of the search tree is equal to 3.

The list of branching rows, IBR, is equal to (1, 2, 4). For each row KST is a vector that gives the position in ITJ of the branching

variable. Therefore:

$$KST = (1, 4, 12)$$

being the positions where the branching variables that are fixed equal to 1,  $x_1$ ,  $x_3$  and  $x_5$  are stored in  $ITJ$ . The list of variables fixed is:

$$LREP = (1, -2, 3, -4, 5)$$

indicating that  $x_2 = x_4 = 0$  and  $x_1 = x_3 = x_5 = 1$ .

The number of variables fixed at each level of the tree search is:

$$KREP = (1, 3, 5)$$

The cost of the fixed variables is 11 and this is stored.

For a best bound tree search for each node the position of the father node is stored in a list. Here node 5 has father node 3 which has father node 0 which is the root. The branching row is stored for each node, as is the position of the branching variable in the list  $ITJ$ . In addition the bound value must be stored at each node.

## The Test Problems Used

### 4.1 Randomly Generated Problems

The problems used in Chapters 2, 3, 4, 6 and 7 were all randomly generated. The constraint matrix was of two types, type A and type B. The first type had a fixed probability,  $\rho$ , that  $a_{ij}$  was equal to 0 or 1. The second type had a constraint matrix that increased in density as the index of the constraint increased. This density ranged from  $0.5\rho$  in the first constraint to  $1.5\rho$  in the last constraint. The costs were of three types. The first was U in which all the costs equalled 1. The second was H in which  $c_j$  was equal to  $\sum_{i=1}^m a_{ij}$ . The third type of cost was obtained by setting  $c_j$  to  $2 \sum_{i=1}^m a_{ij} + 5$  and if this value was greater than 15 it was reduced by 10. This was denoted by x in the tables.

### A4.2 Korman's Test Problems; Table 5.5

These problems were generated from random graphs. The columns of the SCP represented cliques in a graph. All these problems had costs equal to 1.

### A4.3 Problems AHSC14 - AHSC17; Table 5.6

These problems were used by Salkin and Koncal (S2) and are numbered 3.5 to 3.8 in Balas and Ho (B7). They had coefficient matrices of 2% density. In addition every column had at least one, and every row at least two, non-zero entries. The costs ranged between 1 and 100.

#### A4.4 Problems LSSC1, LSSC9; Table 5.8

These two 200x2000 problems had density of 2% and are problems 5.1 and 5.9 of Balas and Ho (B7). They were randomly generated.

#### A4.5 Problems LSSC16, LSSC20, LSSC21, LSSC22; Table 5.8

These problems were also randomly generated and had 2% density. They correspond to problems 4.1, 4.5, 4.6 and 4.7 of (B7)

#### A4.6 Problems SALK12, SALK13; Table 5.8

Problem SALK12 is attributed to A.M. Geoffrion. It was randomly generated with a coefficient matrix density of 7%. However the reduction tests gave a problem with a density of 4%. It is problem 1.12 of (B7). Problem SALK13 is described by Salkin and Koncal (S2) as coming from American Airlines.

## APPENDIX 5

### The Language used in the Procedures

The description of Pidgin ALGOL given below explains the language used in the Procedures.

Pidgin ALGOL is unlike any conventional programming language in that it allows the use of any type of mathematical statement as long as its meaning is clear and the translation into RAM or RASP code is evident. Similarly, the language does not have a fixed set of data types. Variables can represent integers, strings, and arrays. Additional data types such as sets, graphs, lists, and queues can be introduced as needed. Formal declarations of data types are avoided as much as possible. The data type of a variable and its scope† should be evident either from its name or from its context.

Pidgin ALGOL uses traditional mathematical and programming language constructs such as expressions, conditions, statements, and procedures. Informal descriptions of some of these constructs are given below.

A Pidgin ALGOL *program* is a statement of one of the following types.

1. variable ← expression
2. if condition then statement else statement‡
- 3a. while condition do statement
- b. repeat statement until condition
4. for variable ← initial-value step step-size§ until final-value do statement
5. label: statement
6. goto label
7. begin
  - . statement;
  - statement;
  - .
  - .
  - .
  - statement;
  - statement
  - end
- 8a. procedure name (list of parameters): statement
  - b. return expression
  - c. procedure-name (arguments)
- 9a. read variable
  - b. write expression
10. comment comment
11. any other miscellaneous statement

† The *scope* of a variable is the environment in which it has a meaning. For example, the scope of an index of a summation is defined only within the summation and has no meaning outside the summation.

‡ "else statement" is optional. This option leads to the usual "dangling else" ambiguity. We take the traditional way out and assume else to be matched with the closest unmatched then.

§ "step step-size" is optional if step-size is 1.

We shall give a brief synopsis of each of these statement types.

1. The assignment statement

variable  $\leftarrow$  expression

causes the expression to the right of  $\leftarrow$  to be evaluated and the resulting value to be assigned to the variable on the left. The time complexity of the assignment statement is the time taken to evaluate the expression and to assign the value to the variable. If the value of the expression is not a basic data type, such as an integer, one may in some cases reduce the cost by means of pointers. For example, the assignment  $A \leftarrow B$  where  $A$  and  $B$  are  $n \times n$  matrices would normally require  $O(n^2)$  time. However, if  $B$  is no longer used, then the time can be made finite and independent of  $n$  by simply renaming the array.

2. In the if statement

if condition then statement else statement

the condition following the if can be any expression that has a value true or false. If the condition has the value true, the statement following then is to be executed. Otherwise, the statement following else (if present) is to be executed. The cost of the if statement is the sum of the costs required to evaluate and test the expression plus the cost of the statement following then or the cost of the statement following else, whichever is actually executed.

3. The purpose of the while statement

while condition do statement

and the repeat statement

repeat statement until condition

is to create a loop. In the while statement the condition following while is evaluated. If the condition is true, the statement after the do is executed. This process is repeated until the condition becomes false. If the condition is originally true, then eventually an execution of the statement must cause the condition to become false if the execution of the while statement is to terminate. The cost of the while statement is the sum of the costs of evaluating the condition as many times as it is evaluated plus the sum of the costs of executing the statement as many times as it is executed.

The repeat statement is similar except that the statement following repeat is executed before the condition is evaluated.

4. In the for statement

for variable  $\leftarrow$  initial-value step step-size until final-value do statement

initial-value, step-size, and final-value are all expressions. In the case where step-size is positive the variable (called the *index*) is set equal to the value of the initial-value expression. If this value exceeds the final-value, then execution terminates. Otherwise the statement following do is executed, the value of the variable is incremented by step-size and compared with the final-value. The process is repeated until the value of the variable exceeds the final-value. The case where the step-size is negative is similar, but termination occurs when the value of the variable is less than the final-value. The cost of the for statement should be obvious in light of the preceding analysis of the while statement.



The above description completely ignores such details as when the expressions for initial-value, step-size, and final-value are evaluated. It is possible that the execution of the statement following `do` modifies the value of the expression `step-size`, in which case evaluating the expression for `step-size` every time the variable is incremented has an effect different from evaluating `step-size` once and for all. Similarly, evaluating `step-size` can affect the value of `final-value`, and a change in sign of `step-size` changes the test for termination. We resolve these problems by not writing programs where such phenomena would make the meaning unclear.

5. Any statement can be made into a *labeled statement* by prefixing it with a label followed by a colon. The primary purpose of the label is to establish a target for a `goto` statement. There is no cost associated with the label.

6. The `goto` statement

`goto label`

causes the statement with the given label to be executed next. The statement so labeled is not allowed to be inside a block-statement (7) unless the `goto` statement is inside the same block-statement. The cost of the `goto` statement is one. `goto` statements should be used sparingly, since they generally make programs difficult to understand. The primary use of `goto` statements is to break out of `while` statements.

7. A sequence of statements separated by semicolons and nested between the keywords `begin` and `end` is a statement which is called a *block*. Since a block is a statement, it can be used wherever a statement can be used. Normally, a program will be a block. The cost of a block is the sum of the costs of the statements appearing within the block.

8. *Procedures*. In Pidgin ALGOL procedures can be defined and subsequently invoked. Procedures are defined by the *procedure-definition statement* which is of the form:

procedure name (list of parameters): statement

The list of parameters is a sequence of dummy variables called *formal parameters*. For example, the following statement defines a function *procedure*

procedure MIN(x, y):  
if x > y then return y else return x

The arguments  $x$  and  $y$  are formal parameters.

Procedures are used in one of two ways. One way is as a *function*. After a function procedure has been defined, it can be invoked in an expression by using its name with the desired arguments. In this case the last statement executed in the procedure must be a `return` statement 8(b). The `return` statement causes the expression following the keyword `return` to be evaluated and execution of the procedure to terminate. The value of the function is the value of this expression. For example,

$A \leftarrow \text{MIN}(2 + 3, 7)$

causes  $A$  to receive the value 5. The expressions  $2 + 3$  and  $7$  are called the *actual parameters* of this procedure invocation.

The second method of using a procedure is to call it by means of the procedure-calling statement 8(c). This statement is merely the name of the procedure followed by a list of actual parameters. The procedure-calling statement can (and usually does) modify the data of the calling program. A procedure called this way does not need a `return` statement in its definition. Completion of execution of the last statement in the procedure completes the execution of the procedure-calling statement. For example, the following statement defines a procedure named INTERCHANGE.

```

procedure INTERCHANGE(x, y):
begin
    t ← x;
    x ← y;
    y ← t
end

```

To invoke this procedure we could write a procedure-calling statement such as

```
INTERCHANGE(A[i], A[j])
```

There are two methods by which a procedure can communicate with other procedures. One way is by global variables. We assume that global variables are implicitly declared in some universal environment. Within this environment is a subenvironment in which procedures are defined.

The other method of communicating with procedures is by means of the parameters. ALGOL 60 uses call-by-value and call-by-name. In *call-by-value* the formal parameters of a procedure are treated as local variables which are initialized to the values of the actual parameters. In *call-by-name* formal parameters serve as place holders in the program, actual parameters being substituted for every occurrence of the corresponding formal parameters. For simplicity we depart from ALGOL 60 and use call-by-reference. In *call-by-reference* parameters are passed by means of pointers to the actual parameters. If an actual parameter is an expression (possibly a constant), then the corresponding formal parameter is treated as a local variable initialized to the value of the expression.

9. The read statement and write statement have the obvious meaning. The read statement has a cost of one. The write statement has a cost of one plus the cost of evaluating the expression following the keyword *write*.

10. The comment statement allows insertion of remarks to aid in the understanding of the program and has zero cost.

11. In addition to the conventional programming language statements we include under "miscellaneous" any statement which makes an algorithm more understandable than an equivalent sequence of programming language statements. Such statements are used when the details of implementation are either irrelevant or obvious, or when a higher level of description is desirable. Some examples of commonly used miscellaneous statements are:

- a) let  $a$  be the smallest element of set  $S$
- b) mark element  $a$  as being "old"†
- c) without loss of generality (wlg) assume that . . . otherwise . . . in statement  
For example,

wlg assume  $a \leq b$  otherwise interchange  $c$  and  $d$  in statement

means that if  $a \leq b$  the following statement is to be executed as written. If  $a > b$ , a duplicate of the statement with the roles of  $c$  and  $d$  interchanged is to be executed.

Copyright © 1974 by Bell Telephone Laboratories, Incorporated, J. F. Hopcroft, and J. D. Ullman. Philippines copyright 1974 by Bell Telephone Laboratories, Incorporated, J. E. Hopcroft, and J. D. Ullman.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada. Library of Congress Catalog Card No. 74-3995.