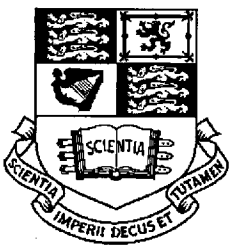
Design of Reliable Networks with Flows

by

Efstratios Rappos

A thesis submitted for the degree of
Doctor of Philosophy of the University of London
and the
Diploma of Imperial College

September 2003



The Management (Business) School
Imperial College
University of London

Abstract

This thesis deals with optimisation algorithms for the Reliable Network Design Problem (RNDP) with flows which is defined as follows. Given a set of vertices (one single source and many demand vertices) and a set of potential edges, the objective is to design a distribution network at a minimum cost, subject to demand requirements and reliability constraints. The cost function consists of a fixed term and a variable part proportional to the flow through each edge. Reliability is implemented through the requirement that the network will be able to supply the required demand if any single edge is removed from the layout. The RNDP with flows is NP-hard in the general case. Both the uncapacitated and capacitated (where flows are bounded by edge capacities) versions of the problem are examined in the thesis.

A new Mixed Integer Programming (MIP) formulation of the uncapacitated RNDP with flows is presented and a new Branch and Cut algorithm is developed to optimally solve the problem. For this case the reliability constraint is equivalent to the network having the two-edge connectivity property, that is, the requirement that at least two edge-disjoint paths from the source to every consumer must exist. Several classes of valid inequalities are studied, and their effect on the objective function value is investigated. The algorithmic implementation of the proposed algorithm is described and computational experience is reported.

A new heuristic algorithm based on ant colony optimisation is also developed for the uncapacitated RNDP with flows. This approach uses the idea of an ant group to represent a solution and can assist the solution of larger networks that cannot normally be solved to optimality. The performance of the algorithm is evaluated through computational experimentation.

The capacitated RNDP with flows is also considered in the thesis. A new Branch and Cut algorithm is developed to find the optimal solution for this problem based on a new MIP formulation. New valid inequalities are derived and implemented.

Numerous computational results on randomly generated networks of different sizes and characteristics are reported to evaluate the computational performance of the proposed algorithms.

Acknowledgements

First of all I would like to thank my supervisor, Dr Eleni Hadjiconstantinou for her help, support and guidance throughout this research. I would also like to thank the Engineering and Physical Science Research Council (EPSRC) and my industrial sponsors WRc plc who funded this project under the Postgraduate Training Partnership scheme.

On a more personal level, special thanks should go first of all to my parents Efthimios and Georgia and my sister Helen for their patience, optimism and financial support throughout these years.

I would also like to thank all the "operational research" people at Imperial and beyond, for all the fruitful academic discussions we had together, their comments, ideas and help they provided me with during my stay in London, so thank you Dalila Fontes, Roberto Baldacci, Marco Boschetti, Fernando Alonso Blázquez and Hélène Pinol.

Finally I would like to thank my colleagues at the Management School research office: Marwan Fayad, Eugenia Gargana, Marco and Ariadna Barassi, Ali Al-Eyd, Parichart Thienchai, Karen Handley, Mounir Naffi, Gustavo Rinaldi, Kiran Aggarwal, Reda Lebcir, Mariam Harfush Pardo, Nicos Nicolaou, Keong Choong and Jair Muñoz Bugarin. I cannot also forget to thank the following people as well: Christina, Manolis and George.

Last but not least, I would like to thank everyone that I didn't have time to spend with in the last years, for their patience and understanding.

Contents

Abstract 2

Acknowledgements 3

List of Models 10

List of Figures 11

List of Tables 14

Summary of Notation 17

1 Introduction 18

1.1 Network design problems 18

1.1.1 Basic definitions and terminology 19

1.1.2 Problem definition 20

1.2 Network reliability 22

1.3 Research goals and motives 23

1.4 Methodological preliminaries 24

1.4.1 Overview of a mixed integer program (MIP) 24

1.4.2 Polyhedral structure and cutting planes 25

1.4.3 Cutting plane algorithms 26

1.4.4 Solving MIP using branch and cut 29

1.4.5 The Gomory-Chvátal cuts 30

1.5 Thesis overview 33

2 Reliable network design problems **34**

2.1 Introduction 34

2.2 Classification of network design problems 34

2.3 The single commodity uncapacitated network design problem with fixed charge cost 36

2.3.1 The fixed charge problem (FCP) 38

2.3.2 Solution methods for the FCP 39

2.4 Topological approaches to network reliability 42

2.4.1 Definitions of network connectivity 42

2.4.2 Mathematical properties 44

2.4.3 Connectivity in directed graphs 47

2.4.4 A general model with connectivity constraints 49

2.4.5 Exact solution methods 51

2.4.6 Heuristic solution methods 53

2.5 Alternative approaches to network reliability 54

2.5.1 Network augmentation 54

2.5.2 Probabilistic methods 55

2.5.3 Application-based approaches 57

2.6 The reliable network design problem considered in this thesis 60

2.7 Test problems 60

2.7.1 Random graphs 62

2.7.2 Probability of obtaining an infeasible instance 63

2.7.3 Choice of demands and costs 66

2.7.4 Generation of uncapacitated test problems 67

2.7.5 Generation of capacitated test problems 68

2.7.6 Computational implementation details 68

2.8 Summary 69

3 A mixed integer programming approach for the uncapacitated RNDP with flows **70**

3.1 Introduction 70

3.2 A mixed integer programming formulation 71
3.3 An example optimal solution 73
3.4 The initial LP relaxation 76
3.4.1 An exact separation algorithm for the reliability constraints 78
3.4.2 Minimum cuts and the Gomory-Hu tree 79
3.4.3 Computational results 80
3.5 Partition inequalities 80
3.5.1 Separation heuristic 82
3.5.2 A numerical example 83
3.5.3 Generation of partition inequalities 84
3.6 The maximum flow inequalities 84
3.6.1 Separation heuristic 86
3.6.2 Facet defining inequalities 88
3.6.3 A numerical example 96
3.6.4 Computational results 98
3.7 The r-cover inequalities 99
3.7.1 Derivation of the r-cover inequalities 100
3.7.1.1 The b-matching problem 100
3.7.1.2 The capacitated case 101
3.7.1.3 The odd cut problem 102
3.7.1.4 Separating matching inequalities and the Padberg-Rao algorithm 102
3.7.1.5 Reliability and r-covers 103
3.7.2 A numerical example 104
3.7.3 Computational results 104
3.8 The Gomory cuts 107
3.8.1 Computational results 107
3.9 Lift and project cuts 108
3.9.1 Numerical example 112
3.9.2 Computational results 114
3.10 Description of a lower bound ascent algorithm 115

3.10.1 Extensive computational results for the URNDP with flows . . . 116
3.10.2 Discussion . . . 125
3.11 Summary . . . 126

4 A branch and cut algorithm for the uncapacitated RNDP with flows 127
4.1 Introduction . . . 127
4.2 A general overview of the branch and cut procedure . . . 127
4.3 Branching strategies . . . 129
4.3.1 Variable selection . . . 129
4.3.2 Node selection . . . 131
4.4 Cut generation . . . 132
4.4.1 Cut validity . . . 132
4.4.2 Cut lifting . . . 133
4.5 Computational implementation of the branch and cut algorithm . . . 134
4.5.1 Outline of the B&C algorithm . . . 134
4.5.2 The cut generation procedure . . . 134
4.5.3 Evaluation of variable selection strategies . . . 136
4.5.4 Evaluation of node selection strategies . . . 139
4.5.5 Evaluation of branch direction strategies . . . 139
4.5.6 Alternative branching strategy . . . 140
4.6 Generating a feasible solution to the URNDP with flows . . . 141
4.6.1 An illustrative example . . . 142
4.6.2 Computational performance of the heuristic . . . 142
4.7 A solved example instance . . . 144
4.8 Computational results . . . 148
4.8.1 Discussion . . . 153
4.9 Summary . . . 155

5 An ant colony heuristic for the URNDP with flows 156
5.1 Introduction . . . 156
5.2 An overview of ant colony optimisation . . . 156
5.3 Ant colony optimisation algorithms . . . 158

5.3.1 An ACO algorithm for the TSP 159
5.3.2 ACO algorithms for other combinatorial optimisation problems . 161
5.4 An outline of an ant colony heuristic for the URNDP with flows 161
5.4.1 The ant colony setup 162
5.4.2 Trail deposition 164
5.4.3 Trail updating 165
5.4.4 Path selection—flow ants 166
5.4.5 Path selection—reliability ants 167
5.4.6 Obtaining a URNDP solution 168
5.5 Computational implementation of the ant colony algorithm 168
5.5.1 Algorithm initialisation and termination 169
5.5.2 Ant group generation 169
5.5.3 Global trail updating 170
5.5.4 The situation where an agent cannot leave a vertex 170
5.6 Computational results 172
5.7 Summary 177

6 A branch and cut algorithm for the capacitated RNDP with flows 180
6.1 Introduction 180
6.2 Capacitated network design problems 180
6.2.1 Relation to uncapacitated problem 181
6.2.2 Overview of capacitated network design problems 182
6.3 A mixed integer programming formulation 183
6.3.1 Layout of the solution 187
6.4 The initial LP relaxation 188
6.4.1 An exact separation algorithm for the capacitated reliability constraints 189
6.5 Valid inequalities for the CRNDP 191
6.5.1 Valid inequalities derived from the URNDP 191
6.5.2 The capacitated degree inequalities with equal capacities 191
6.5.3 The capacitated reliability inequalities 192

6.5.4 The capacitated degree inequalities with different capacities . . . 194
6.5.5 Strengthening the capacitated degree inequalities using the b-matching problem . . . 195
6.6 Description of a branch and cut algorithm . . . 196
6.6.1 A lower bound ascent procedure . . . 196
6.6.2 Cut generation during branch and cut . . . 197
6.7 A solved example instance . . . 197
6.8 Computational results . . . 198
6.8.1 Discussion . . . 204
6.9 Summary . . . 206

7 Conclusion **207**
7.1 Contributions . . . 207
7.2 Concluding remarks . . . 208
7.3 Future work . . . 209

A Sample test instances **213**

B A description of the URNDP polytope for K_4 **216**

References **227**

Index **245**

List of Models

M1 The general Mixed Integer Program 25

M2 The single commodity uncapacitated network design problem 37

M3 The fixed charge problem (FCP) 39

M4 The multicommodity extended formulation for the FCP 41

M5 A MIP introducing topological reliability 50

M6 The edge-connectivity network design problem 51

M7 The uncapacitated reliable network design problem with flows 71

M8 The initial relaxation of the URNDP 77

M9 The LP relaxation of the URNDP with the maximum flow inequalities ($|S| = 1$) 86

M10 The generation of lift and project cuts (A) 111

M11 The generation of lift and project cuts (B) 111

M12 The capacitated reliable network design problem with flows 184

M13 The capacitated reliable network design problem with flows - equivalent formulation 186

M14 The initial relaxation of the CRNDP 189

M15 The reliable network design problem with lower capacities 211

List of Figures

1.1 The convex hull of feasible points of an integer program 26
1.2 Faces and facets of a convex polytope 27
1.3 Adding a cut improves the upper bound of a maximisation problem . . . 28
1.4 An overview of the Branch and Cut algorithm 31

2.1 An example of survivability considerations 59
2.2 In unlabelled graphs the vertices are not distinguishable 63

3.1 In a two-edge connected network every cut must contain at least two edges 72
3.2 An example URNDP with flows: input network 73
3.3 An example URNDP with flows: optimum network 74
3.4 An example URNDP with flows: all possible reliable configurations . . . 75
3.5 A 2-edge connected graph with no tour 76
3.6 A *non* 2-edge connected graph with all vertices of degree 2 or more . . . 77
3.7 An example of the Gomory-Hu procedure 80
3.8 Partition inequalities 81
3.9 Partition inequalities versus reliability constraints 84
3.10 The maximum flow inequalities 85
3.11 An example of the separation heuristic for the maximum flow inequalities 87
3.12 Solution FS1 satisfying with equality the maximum flow inequalities . . . 94
3.13 Solution FS2 satisfying with equality the maximum flow inequalities . . . 94
3.14 Solution FS3 satisfying with equality the maximum flow inequalities . . . 94
3.15 Solution FS4 satisfying with equality the maximum flow inequalities . . . 94
3.16 Solution FS5 satisfying with equality the maximum flow inequalities . . . 94

3.17 The r-cover inequalities 99
3.18 An example of violated r-cover inequalities 104
3.19 An outline of the lift and project technique 109
3.20 Example of the lift and project technique (A) 112
3.21 Example of the lift and project technique (B) 113

4.1 Outline of the Branch and Cut algorithm for the URNDP with flows . . 135
4.2 Heuristic step 1: The minimum cost tree solution 142
4.3 Heuristic step 2: The minimum cost tree on the leaves 142
4.4 Heuristic step 3: Merging the two trees 143
4.5 Heuristic step 4: The final layout of the solution 143
4.6 The layout of the input graph 145
4.7 The solution layout associated with the best lower bound obtained at the root node 146
4.8 The layout of the optimal solution 147
4.9 The layout of the optimal solution (expanded) 147
4.10 Graph of computational time versus network size for different values of γ. 154
4.11 Graph of computational time versus γ for different network sizes. 154

5.1 How real ants find the shortest way round an obstacle 158
5.2 A tree with N vertices is equivalent to $N-1$ paths leaving the source . . 162
5.3 Flow ants and reliability ants 164
5.4 Outline of the algorithm used by an ant agent 172
5.5 Outline of the algorithm used by an ant group 173
5.6 Outline of the overall ACO algorithm 174

6.1 Example of an edge failure in the CRNDP with flows 185
6.2 The number of saturated edges incident to a vertex 188
6.3 The separation of the capacitated reliability constraints 191
6.4 The layout of the optimal solution of the capacitated RNDP test instance 198
6.5 Graph of time versus network size for different values of γ. 205
6.6 Graph of time versus γ for different network sizes. 205

B.1 The variables used in the description of the facets of the URNDP convex hull for K_4 216

List of Tables

2.1 Classification of network design problems 35

2.2 The variable names used in the MIP formulation of the fixed charge problem 40

2.3 Equivalent definitions for edge and node connectivity 45

2.4 The proportion of feasible problems among randomly generated instances 65

2.5 The proportion of feasible problems among randomly generated instances with a fixed number of edges 65

2.6 The naming convention used for the test instances 67

3.1 Notation used in model **M7** of the URNDP with flows 71

3.2 Root node lower bounds LB_0 using the initial LP relaxation (**M8**) 81

3.3 The constraints defining the URNDP with flows for K_4 96

3.4 Analysis of the maximum flow inequalities and the convex hull for K_4 . . 97

3.5 Root node lower bounds LB_1 with the maximum flow inequalities added 98

3.6 Root node lower bounds with the partition and r-cover inequalities added 106

3.7 Root node lower bounds LB_4 with the Gomory inequalities added . . . 108

3.8 Root node lower bounds LB_5 with the lift and project inequalities added 115

3.9 Computational results for the URNDP: 12–25 vertices (averages over five instances) 119

3.10 Computational results for the URNDP: 30–50 vertices (averages over five instances) 120

3.11 Computational results for the URNDP: Test instances with 12–25 vertices (lower bounds) 121

3.12 Computational results for the URNDP: Test instances with 30–50 vertices (lower bounds) 122
3.13 Computational results for the URNDP: Test instances with 12–25 vertices (number of cuts) 123
3.14 Computational results for the URNDP: Test instances with 30–50 vertices (number of cuts) 124

4.1 Performance of the algorithm for different variable selection rules 138
4.2 Implementation of two alternative node selection strategies 139
4.3 Implementation of two alternative node selection strategies 140
4.4 Quality of the heuristic solution at the beginning of the B&C algorithm . 143
4.5 Execution of the B&C algorithm for a test instance 145
4.6 Computational results for the URNDP: 12–25 vertices (averages over five instances) 149
4.7 Computational results for the URNDP: 30–50 vertices (averages over five instances) 150
4.8 Computational results for the URNDP: 12–25 vertices 151
4.9 Computational results for the URNDP: 30–50 vertices 152
4.10 Investigation on the contribution of the r-cover inequalities to the B&C algorithm 155

5.1 The parameters used in the ACO algorithm for the TSP 160
5.2 The notation used for the ACO algorithm for the URNDP with flows . 163
5.3 The parameters in the ACO algorithm for the URNDP with flows . . . 174
5.4 Computational results for different values of β 175
5.5 Computational results for different values of q_0 176
5.6 Comparison of the different strategies used when an ant cannot leave a vertex 176
5.7 Parameters used in the ACO algorithm for the various groups of test problems 176
5.8 Computational results: ACO algorithm (averages) 178
5.9 Computational results: ACO algorithm (test instances) 179

6.1 The notation used in the MIP formulation of the CRNDP with flows . . 184

6.2 Computational results for the CRNDP: Root node (averages over three instances) 200

6.3 Computational results for the CRNDP: Root node 201

6.4 Computational results for the CRNDP: Branch and Cut (averages over three instances) 202

6.5 Computational results for the CRNDP: Branch and Cut 203

A.1 Characteristics of the test instance 214

A.2 Edge capacities for the CRNDP test instance 215

B.1 The constraints of the URNDP with flows formulation for K_4 217

Summary of Notation

A list of the most commonly used symbols, acronyms and their interpretation is presented below.

$\mathbb{N}$	$=$	$\{0, 1, 2, \ldots\}$
$\mathbb{Z}$	$=$	$\{0, \pm 1, \pm 2, \ldots\}$
$\mathbb{R}_+$	$=$	$\{x : x \geq 0\}$
$:=$	:	definition
$\leftarrow$	:	assignment (in a computer pseudocode)
V, N	:	V is the set of vertices (or nodes) of a graph and $N = \|V\|$
V'	:	the set of customer vertices (or nodes) of a graph $V' = V \backslash \{0\}$
K_n, $K_{m,n}$	:	the complete and complete bipartite graph, respectively
e_j	:	the j^{th} unit vector, (u_i) where $u_j = 1$, and $u_i = 0$ for $i \neq j$.
$\delta(S)$	:	the cut containing the subset S of graph vertices in one shore
$\mathcal{P}$	:	the class of problems solvable in polynomial time
$\mathcal{NP}$	:	the class of Non-deterministic Polynomial problems (problems for which a 'yes' answer to their decision problem can be verified in polynomial time)
B&C	:	Branch and Cut
LP	:	Linear Program or Linear Programming
MIP	:	Mixed Integer Program
RNDP	:	Reliable Network Design Problem
URNDP	:	Uncapacitated RNDP
CRNDP	:	Capacitated RNDP
$\mathcal{B}$, $\mathcal{NB}$	:	the set of Basic and non-Basic variables of a LP
CH	:	(For a MIP) the convex hull of the incidence vectors of the solutions to the MIP
$\mathbb{P}(\ldots)$	:	probability
G-H tree	:	Gomory-Hu tree
ACO	:	Ant Colony Optimisation

"But where shall I begin?" she asks.
"Start at the beginning and continue on until
you reach the end. Then stop."

Alice in Wonderland (1865)
Charles Dodgson (Lewis Carroll)

Chapter 1

Introduction

1.1 Network design problems

Network design constitutes a central problem in combinatorial optimisation. Network design problems play an important role in modern optimisation theory due to the great number of situations that can be modelled by a network: transport networks (road, rail, traffic flow), utility networks (power, water, waste, gas), distribution networks (flight/vehicle scheduling and routing, supply chains) just to name a few. For this reason, network problems are of great theoretical and practical interest to areas such as computer science (computer networks, VLSI design), transportation services (routing and the travelling salesman problem), production planning (location of facilities), scheduling (vehicle fleet and air traffic scheduling) and the telecommunication and utility sector (design of power grids and water distribution networks, telephone networks and the internet). The rapid expansion of technology has stipulated the need to solve new and more difficult network design problems. This is because the size of the networks involved has increased and, moreover, the development of new materials for the network components and new construction techniques imposes different design requirements than before. Furthermore, the advance of computer technology has enabled scientists to be able to cope with these added requirements and currently a significant amount of research is devoted to the area of network design, both in an academic environment and in the private sector.

Although a basic knowledge of graph theoretical principles is assumed for the interested reader, there exists some discrepancy in the way some of these terms are used in the literature. The purpose of the next section is to clarify a few of the mathematical terms that will be used in this thesis (some dating back to the days of Euler, Kirkman, Kirchoff—early eighteenth century—when graph theory was born).

1.1.1 Basic definitions and terminology

A *graph* is defined as an abstract entity consisting of a set of points called the *nodes* or *vertices*, $V = \{0, 1, \ldots, N-1\}$, and a set of *arcs*, each connecting a pair of vertices, i.e., $A = \{(i,j), i,j \in V\}$. We always assume that a graph has at least two vertices. Graphs can be *directed* (often shortened to *digraphs*) or *undirected* according to whether the direction of the arcs is of importance or not; in the case of undirected graphs the arcs are traditionally referred to as *edges*. It is generally simpler to consider undirected graphs, however, there are situations where the direction of the arc is important either in order to model the problem or to obtain meaningful results; in these cases directed graphs must be employed. Note that, for directed graphs the word 'edge' will be used if we wish to imply that the distinction between the two possible arcs is of no relevance, that is, using the term 'edge (i,j)' implies 'arc (i,j) or arc (j,i).' A *path* is a sequence of vertices $v_1, \ldots, v_k$ such that consecutive vertices $v_i v_{i+1}$ define arcs. A *simple path* is a path that does not revisit the same vertex twice. A *cycle* is a simple path where the start and end vertices are the same.

An undirected graph is called *connected* if, for any decomposition of its vertex set V into two disjoint sets V_1 and V_2 such that $V = V_1 \cup V_2$, there exists an edge (i,j) with $i \in V_1$ and $j \in V_2$. An undirected graph is a *forest* if it has no cycles, if it is also connected then it is called a *tree* (hence a tree is minimally connected).

The *degree* of a vertex i of a graph, $\deg(i)$, is the number of edges incident to that vertex. Two graphs are *homeotopic* to each other if one can be obtained from the other when its edges are subdivided into paths by inserting new vertices of degree 2, i.e., if one graph can be transformed to the other by adding or removing degree 2 vertices. A *series-parallel* graph is one that does not contain a subgraph that is homeotopic to K_4, the complete graph with 4 vertices. A *branching* is a forest where every vertex is the head of at most one arc, and an *arborescence* is a branching that is a tree. In a spanning arborescence every vertex except one (the *root* node) is the head of one arc. The *components* of a graph are the maximal connected subgraphs of the graph. An edge e is called a *bridge* (or *isthmus*

or *cut edge*) if $G\backslash e$ has more components than G.

A *network* is a generalised graph in the sense that it is assumed that there is some commodity flowing through the edges. This can be visualised for example, by considering a water distribution network; the underlying graph is the structure of pipes and the commodity is the water flowing through the pipes. The amount of the commodity that passes through an arc (per unit of time) is commonly referred to as the *flow* of the arc. In that setup, one or more vertices are the *source* vertices, i.e., the suppliers of the flow material, some other are *sink* vertices where flow is absorbed and the remaining are *transportation* vertices where the flow is conserved (at each time instant the amount of flow entering the vertex is equal to the amount of flow leaving it). The sink vertices often represent customer locations that consume the commodity and for this reason they are often referred to as *demand* or *customer* vertices.

Naturally, arcs can only flow outwards from a source, but may flow both into or out of a demand vertex. In general however, this is not a necessary requirement to impose since cost minimisation will automatically impose this.[1]

1.1.2 Problem definition

A *network design problem* can be defined as follows (Dell'Amico, Maffioli & Martello (1997)):

> "Given a set of available (perhaps capacitated) edges, the problem seeks a least cost network, composed of both variable flow costs and edge fixed costs, that meets prescribed flow requirements."

In combinatorial optimisation, undirected graphs have a (real, usually positive) number associated with each edge which we shall call the *weight* of the edge. We refrain from naming it 'cost' (this is reserved for networks) for two reasons: firstly, 'cost' naturally implies a minimisation problem but the graph problems that will be discussed can be both minimisation and maximisation problems; and secondly the weights of a particular graph problem discussed may be related to the costs of the underlying network, so using the same word to refer to both of them could cause confusion.

[1] We can always do better (costwise) than a network that shifts water (say) in a loop returning to the source. For, if there is flow going into a source vertex this means that this flow originated from some (possibly different) source vertex or vertices; hence decreasing the amount of flow accordingly we can remove completely that flow without increasing the total cost (we assume that there is a positive relationship between the amount of flow passing through a pipe and the associated cost).

Associated with each demand vertex is a number, known as the *demand* of the vertex which represents the amount of the commodity required at this vertex per unit of time. Throughout this work we shall assume that the demands are integer. This is not a serious restriction since a problem with rational demands can be easily converted to an integer-demand one by multiplying the demands with the least common multiple of all the denominators present and, after solving the problem, adjust the flows accordingly to revert to the original scale. The set of real (irrational) numbers is not a convenient set to work with for many reasons: firstly, all algorithm implementations are programmed in a computer where all data is represented by means of finite arithmetic and therefore irrational arithmetic is not possible, and secondly, real-life problem data that is fed into algorithms is never irrational. Moreover, many algorithms in combinatorial optimisation are not designed for and cannot handle irrational data very well.[2]

Associated with each arc is a *cost function*, that is, an integer-valued (for the reasons mentioned in the previous paragraph) function that represents the cost incurred per unit time when the medium flows through the arc. For example, this function can be a function of the physical properties of the pipe (length, diameter) which are known a priori as well as the amount of flow that is to stream through that pipe which is not known beforehand, as it is part of the problem unknowns. We also assume that there is no loss of material in the network, that is, the supply of flow at the sources equals exactly the sum of the demands at the demand vertices.

For most practical reasons, we can assume that there is only one source vertex; for, if otherwise we can join all the sources to an artificial 'supersource', that provides all the flow, with arcs of zero cost. The source nodes now become customers of zero demand.

In addition, network edges may have an integer capacity, which represents the maximum amount of flow that can pass through the given edge. A transformation to convert a problem with edge capacities to an uncapacitated one exists, and details of this procedure and other techniques relevant to capacitated network design problems will be discussed later in the thesis.

To illustrate the above definition of a network design problem in a more practical context, let us consider the case of a water distribution network: the edges of the graph represent potential locations of the pipes and the vertices represent (known) locations of customers (or groups of customers). The demands are the (known) amounts of water

[2]The elementary Ford-Fulkerson algorithm in its standard flow-increasing version, for example, can be made to run into an infinite loop without reaching an optimum for a 6-vertex network and a particular choice of irrational (related to the golden section constant ϕ) capacities (Weber (1998)).

that each customer requires (per unit of time) and the edge costs represent the cost of installing and maintaining the pipe for a given period of time.

The essence of a network design problem is to determine which subset of the available edges should be used in the design of the network (among all possible choices) and the amount of the commodity that should flow through each of these arcs per unit time such that each demand vertex receives exactly the amount of commodity that it needs (i.e., the difference between the total flow that enters into the node minus the total flow that leaves the vertex equals the demand of that vertex), at a minimum cost.

1.2 Network reliability

In any distribution network *reliability* (also referred to as *survivability*) is an important design requirement as a network designed without reliability considerations will have a larger running cost resulting from the consequences that a breakdown or failure will incur. Moreover, in a modern society certain network operators (e.g., utility providers such as water, gas, electricity and telecommunication companies) are required to provide an acceptable level of service that entails certain reliability considerations. Indeed, Goulter (1995) in the framework of water network design mentions that any design of such a network should be based on the following two primary objectives: (i) minimisation of the cost, and (ii) maximisation of reliability, thus establishing that for some networks implementing reliability in a network is as important as minimising the cost.

Although reliability itself is an important benefit if it is taken into consideration in the design of a network, the way it is implemented varies significantly among different types of network and among researchers, due to the vast number of ways in which a network can be made more reliable. In general terms, network reliability can be defined as (Cullinane, Lansey & Mays (1992)):

> "The ability of the system to provide service with an *acceptable level* of interruption in spite of abnormal conditions."

The exact details of the definition are open to interpretation: for a water distribution network alone, Wagner, Shamir & Marks (1988*b*) mention no less than twenty different measures of reliability, ranging from water volume deficit to inter-failure times and repair durations. If one adds the research done in implementing hydraulic reliability (e.g., Ormsbee & Kessler (1990)), and other probabilistic definitions of reliability (Wagner,

Shamir & Marks (1988*a*)) we end up with an even larger number (and this by restricting the focus to water networks only).

It becomes clear that a exhaustive analysis of reliability in the context of network design is neither possible nor desirable, because for each approach to reliability completely different solution techniques are generally employed. It is important to note, however, that many of the suggested definitions of reliability only work with certain simulation or scenario-based approaches, and therefore have limited scope for improvement or extension. They are (or were) nevertheless extremely useful to engineers for predicting the behaviour of a network and estimating probabilities of failure (and expected repair budgets) but of little theoretical interest because new such models appear all the time as the quality of materials improves and engineering practices change with the technological progress.

In recent years there has been an increase of graph-theory related techniques. The main reasons for this is that they provide a more consistent measure of reliability which can be solved analytically rather than using simulation methods. Goulter (1995) observes this trend and the popularity of graph connectivity methods in assessing reliability in networks, but reports that such techniques are often slow (most graph connectivity network design problems are $\mathcal{NP}$-hard) and sometimes cannot solve the problem sizes that are considered 'typical' for some applications. With the recent progress in computer technology and combinatorial optimisation over the last decade, graph theoretical approaches to reliability are nowadays the most commonly used. The expansion of the internet is also partly to blame, as the new, world-wide communication types of network that it introduced are often modelled with a graph theoretical approach to reliability (connectivity). On the same lines, fiber-optic communication networks are routinely designed using edge or node connectivity considerations to incorporate reliability into the network layout.

1.3 Research goals and motives

The core problem studied in this thesis is a network design problem with flows (i.e., a single commodity problem) where we seek the layout of a network that will satisfy, at a minimum cost, some reliability requirements.

While a significant amount of research has been done in both fields of reliability and flow/commodity networks, there exists a gap in the literature in research where the two are combined. One reason could be that a network design problem containing just one

of these characteristics is already very complex, hence studying a combination of the two would be extremely difficult.

However, given the technological evolution in computers many problems that, until recently, have been considered intractable can now be solved quickly. Moreover, the existence of powerful computers has spawned new scientific research into the algorithmic techniques that can be used to solve them (faster linear programming and branch and bound software), so a much larger number of combinatorial optimisation problems can be solved nowadays than even before.

We develop an *exact* algorithm for solving the problem of designing a *reliable* network with the presence of *flows*. The existence of an exact algorithm is important for two reasons: first of all, it is always more beneficial to know the *best* (optimal) solution to a problem than an approximate one, as in a real-life problem a small deviation from the optimum may correspond to a huge increase in cost. Secondly, the knowledge of the optimal solution provides a means of comparing different heuristics for solving the problem, by measuring how close their solution is to the best one.

A model of a network problem on the design, at a minimum cost, of a network capable of supplying all its demand vertices while possessing some level of reliability could be of great interest in the areas of utility networks. The network owners could use this model to plan network expansions, to compare the layout of the existing network to the optimal one, and to direct maintenance works to the most vulnerable areas.

1.4 Methodological preliminaries

It is beneficial, at this point, to present an overview of the techniques that are commonly used in combinatorial optimisation to solve network design problems to optimality. These techniques will be referred to frequently in the overview of the literature and in later chapters of the thesis.

1.4.1 Overview of a mixed integer program (MIP)

A *Mixed Integer Program (MIP)* is a linear program (LP) with some variables restricted to take integer values. The formulation of a general MIP is shown in **M1** below.

MODEL M1

$$\max \quad \mathbf{c}^T\mathbf{x} + \mathbf{b}^T\boldsymbol{\xi} \tag{M1.1}$$

subject to

$$\mathbf{A}\mathbf{x} + \mathbf{B}\boldsymbol{\xi} \leq \mathbf{d} \tag{M1.2}$$

$$\mathbf{x} \geq 0 \tag{M1.3}$$

$$\boldsymbol{\xi} \in \mathbb{N}^k \tag{M1.4}$$

THE GENERAL MIXED INTEGER PROGRAM

where $\mathbf{c}, \mathbf{x}$ are m-vectors, $\mathbf{b}, \boldsymbol{\xi}$ are k-vectors, $\mathbf{d}$ is an n-vector and the matrices $\mathbf{A}, \mathbf{B}$ have dimensions $n \times m$ and $n \times k$ respectively. The *objective function* **M1**.1 sense can be maximisation or minimisation, and the solution of the MIP is a set of values for the variables $\mathbf{x}$ and $\boldsymbol{\xi}$ that satisfy all the problem *constraints* **M1**.2–**M1**.4.

If all the variables of the problem are required to be integer then it is often called an *Integer* or *Pure Integer Program*. Linear and Mixed Integer Programs have been extensively studied due to the large number of problems that can be formulated as such, among different fields of science (e.g., see Tripathy (1984), da Silva (2001), Powell (1999), Evans & Fairbairn (1989), Richards, Schouwenaars, How & Feron (2002)).

For MIPs, various techniques for their solution have been developed and are well documented, for instance see Darby-Dowman & Wilson (2002), Nemhauser & Wolsey (1988), Beasley (1996), Eiselt & Sandblom (2000) and Wolsey (1998) for an overview of MIP or Schrijver (1986) for a more theoretical analysis. It is known however that the general MIP, in contrast to linear programs,[3] is $\mathcal{NP}$-hard and as such, an efficient polynomial algorithm is unlikely to exist.

1.4.2 Polyhedral structure and cutting planes

Consider the general Mixed Integer Program **M1**. Its LP relaxation is obtained by relaxing the integrality constraints **M1**.4 to their continuous counterparts:

$$\max \left\{ \mathbf{c}^T\mathbf{x} + \mathbf{b}^T\boldsymbol{\xi} \;:\; \mathbf{A}\mathbf{x} + \mathbf{B}\boldsymbol{\xi} \leq \mathbf{d},\ \mathbf{x} \in \mathbb{R}^m_+,\ \boldsymbol{\xi} \in \mathbb{R}^k_+ \right\} \tag{1.1}$$

[3]The LP decision problem is in $\mathcal{P}$ as it can be solved in polynomial time e.g., using the ellipsoid algorithm by Khachiîân (1979).

The set of constraints define a polytope[4] in $(p+q)$ space:

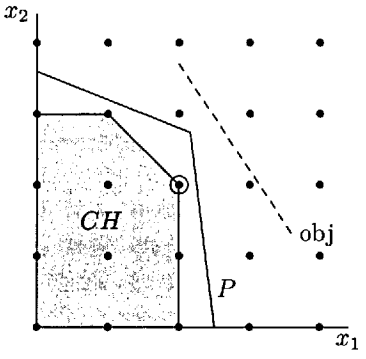
$$P = P(\mathbf{A}, \mathbf{B}, \mathbf{d}) := \left\{(\mathbf{x}, \boldsymbol{\xi}) \in \mathbb{R}_+^{p+q} : \mathbf{A}\mathbf{x} + \mathbf{B}\boldsymbol{\xi} \leq \mathbf{b}\right\} \tag{1.2}$$

The *convex hull* (CH) of a set $X \subseteq \mathbb{R}^n$ is defined as the polytope

$$CH(X) = \left\{x \in \mathbb{R}^n : x = \sum_{i=1}^{t} \lambda_i X_i \text{ with } \lambda_1 + \cdots + \lambda_t = 1,\ \lambda_i \geq 0 \text{ for } i = 1, \ldots, t, \text{ over all finite subsets } \{X_1, \ldots, X_t\} \text{ of } X\right\} \tag{1.3}$$

(Wolsey (1998)). The convex hull of all the feasible solutions (CH) of model **M1**, as a polytope, can be described using a finite set of linear inequalities. If this description is known, then optimising the LP constrained by these inequalities with respect to the original objective function will yield the optimal solution to the MIP. Finding however the complete description of this convex hull is difficult, and amounts to a task no easier than solving the original MIP.

As an example, figure 1.1 presents the polyhedra P and CH for a simple two dimensional integer program. The optimal solution of the MIP is circled.



$$\max\{3x_1 + 2x_2\} \tag{1.4}$$

$$8x_1 + x_2 \leq 20 \tag{1.5}$$

$$2x_1 + 5x_2 \leq 18 \tag{1.6}$$

$$x_1, x_2 \geq 0 \tag{1.7}$$

$$x_1, x_2 \text{ integer.} \tag{1.8}$$

Figure 1.1: The convex hull of feasible points of an integer program

1.4.3 Cutting plane algorithms

Given that no efficient, direct algorithm for solving MIPs is known, solution techniques focus on the relationship of the two polytopes P and CH described in the previous section. The reason is that if the description of the polytope CH is known (i.e., knowing all

[4]Both a *polyhedron* and a *polytope* are defined as the intersection of a finite number of hyperplanes and/or closed halfspaces. Although the notation varies, most authors require polyhedra to be bounded; polytopes on the other hand may be unbounded.

the inequalities that define its bounding faces) then the MIP reduces to a linear program that can be solved easily. (The linear program will have the same objective function and be constrained by the inequalities describing CH.) However, the only polytope description that is readily available is that of P, obtained by relaxing the integrality constraints of the MIP. Algorithms that try to obtain CH, starting from P, were among the first ones to be developed in order to solve MIPs. Such algorithms are known as *cutting plane algorithms* because they successively add 'cuts' (inequalities) to P to approximate CH.

For example, consider the general MIP **M1**. Any linear inequality $\boldsymbol{\alpha}^T\mathbf{x} + \boldsymbol{\beta}^T\boldsymbol{\xi} \leq \lambda$ which is satisfied by all points in $CH(X)$ is called a *valid inequality* for the mixed integer program. We say that a valid inequality defines a *face* of CH if the intersection of the hyperplane bounding a valid inequality with CH is not the empty set. If, moreover, this intersection is of maximal dimension (i.e., has a dimension of one less the dimension of CH) then the valid inequality is said to define a *facet* of CH, as shown in figure 1.2.

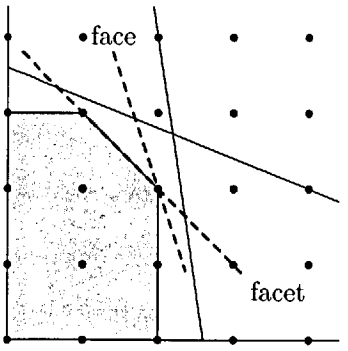


Figure 1.2: Faces and facets of a convex polytope

In the context of cutting plane algorithms, a *cut* (or a *cutting plane*) is a valid inequality that 'cuts off' a part of P, i.e.,

$$P \supset P \cap \{(\mathbf{x}, \boldsymbol{\xi}) \in \mathbb{R}_+^{m+k} : \boldsymbol{\alpha}^T\mathbf{x} + \boldsymbol{\beta}^T\boldsymbol{\xi} \leq \lambda\} \tag{1.9}$$

If the solution of the linear program

$$\max\{\mathbf{c}^T\mathbf{x} + \mathbf{b}^T\boldsymbol{\xi} : (\mathbf{x}, \boldsymbol{\xi}) \in P\} \tag{1.10}$$

is feasible for the MIP then it is the required optimal solution. If, on the other hand, it is not feasible then the current solution, then cuts can be generated and added to the LP until the optimal solution is found. An example of the addition of such a cut is shown in figure 1.3. The sequence of the LP values provides a sequence of decreasing upper

bounds on the optimal value of the MIP.[5], which will eventually converge to the required optimal MIP solution.

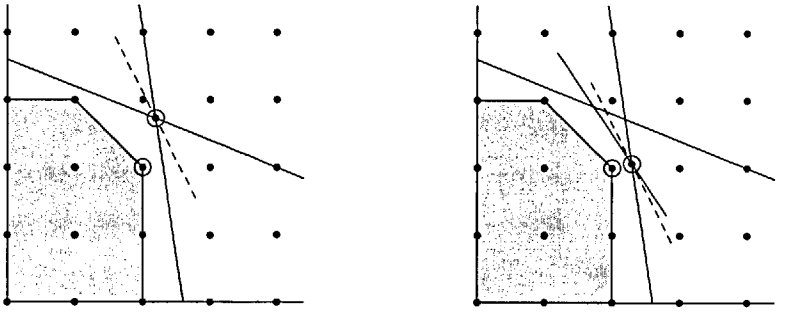


Figure 1.3: Adding a cut improves the upper bound of a maximisation problem

Theoretically such cuts can always be found provided that the current LP solution is not feasible for the MIP. One of the earliest types of cut (and cutting plane algorithm) is the Gomory cut, described in section 1.4.5. Several categories of cuts are known, some of which can be obtained and are valid for general MIPs, and others that are specific for particular problems. An overview of the 'classical' cuts used in mixed integer programming can be found in Padberg (2001), Wolsey (1998), Ceria, Cordier, Marchand & Wolsey (1998) and Marchand, Martin, Weismantel & Wolsey (1999).

However, cutting plane algorithms are not efficient. The generation of cutting planes is often computationally expensive and the convergence to the optimal solution may be slow, although the algorithms themselves may be guaranteed to terminate in a finite number of steps.

To speed up cutting plane algorithms, it is common to add specialised cuts specific to the problem being solved (if such cuts are known). They are usually more effective than generic cuts that can be generated for any MIP, especially if they can be designed to define facets of CH. Even so, it is very unlikely that an efficient algorithm for obtaining the complete description of the polytope CH for a general MIP will be found since the general problem is $\mathcal{NP}$-hard. Some exceptions to this rule do exist but only for a few special categories of mixed integer programs (e.g., the b-matching problem).

For these reasons other techniques for solving MIPs have been developed, the most important of which is the Branch and Cut system which is outlined in the next section.

[5] or a sequence of increasing lower bounds, in the case of a minimisation problem.

1.4.4 Solving MIP using branch and cut

Divide and conquer approaches for solving MIPs are very popular. They consist of methods dividing the problem into a sequence of sub-problems and continuously optimising these sub-problems until a solution to the original MIP is found. Various such enumeration techniques are possible although some are problem dependent. The most commonly used in practice are LP based approaches, such as Branch and Bound,[6] Branch and Cut and Branch and Price (a recent overview of linear programming-based algorithms for the solution of MIPs can be found in Johnson, Nemhauser & Savelsbergh (2000)). The problem is divided in a number of mutually disjoint subproblems by relaxing the integrality constraints thus obtaining a sequence of LPs that can be quickly solved. The sequence of LPs is often represented by a binary tree called the *binary enumeration tree*, and the algorithm tracks the MIP-feasible solutions (those that satisfy the integrality constraints) as they are found and stops when the best feasible solution found can no longer be improved and is therefore the required optimal solution to the problem. Other forms of relaxations exist for non-linear objective functions, such as Lagrangian relaxation or problem-specific combinatorial relaxations where one problem is relaxed to an easier combinatorial problem. In complexity theory terms these divide and conquer processes are no simpler than completely enumerating all the possible solutions and picking the best one, but the advantage of the above techniques lies in the fact that we can *ignore* some parts of the solution space that are *known* not to contain the optimal solution thus significantly speeding up the process and increasing the size of the problems that can be solved in a reasonable amount of time.

Usually the Branch and Bound procedure is implemented as follows: the LP relaxation of an MIP is solved, and if an integer variable ξ, say, has a fractional value $\bar{\xi}$ in the solution, then the problem is divided into two sub-problems by adding one of the constraints

$$\xi \leq \lfloor \bar{\xi} \rfloor \qquad \text{and} \qquad \xi \geq \lceil \bar{\xi} \rceil,$$

and the process is then iterated. By keeping track of the feasible MIP solutions encountered, we can ignore a subproblem whose LP value is worse than a known feasible solution (as this subproblem cannot contain the optimum solution), thus speeding up the solution process.

Branch and Cut (B&C) combines cutting plane algorithms with Branch and Bound. The algorithm is essentially an enumeration process, similar to Branch and Bound, but

[6]The Branch and Bound technique dates back to the early sixties, Land & Doig (1960) and Little, Murty, Sweeney & Karel (1963).

where the bounds obtained from LP relaxations are strengthened using cuts. The overall advantage is that the Branch and Cut algorithm is faster than branch and bound and does not suffer from the 'slow convergence' feature of cutting plane algorithms because if at any point the added cutting planes do not contribute significantly to the bound, then we can stop the cut generation and proceed with branching. An outline of the steps of a Branch and Cut algorithm for a minimisation problem (Wolsey (1998)) is presented in figure 1.4.

1.4.5 The Gomory-Chvátal cuts

Probably the earliest type of cut for integer programs is the Gomory cut (Gomory (1960)) that can be used to solve pure or mixed integer linear programs. These cuts can always be generated during B&C until the MIP is solved to optimality. The original use of Gomory cuts was in a cutting plane algorithm where cuts are generated and added to the LP relaxation continuously until an integer feasible solution is obtained.

The cutting plane algorithm thus obtained has been notoriously slow to converge, as usually a huge number of cuts is needed to be added to obtain an MIP feasible solution. In a Branch and Cut framework however, there is no need to generate all possible Gomory cuts but instead only a few may be generated in order to improve the LP relaxation bound.

The Gomory integer cut is described below. Consider a linear program with m constraints that includes the slack variables:

$$\max \{\mathbf{c}^T\mathbf{x} : \mathbf{Ax} = \mathbf{b}, \quad \mathbf{x} \in \mathbb{N}^n\} \tag{1.11}$$

Solving this LP (e.g., see Dantzig & Thapa (1997)) we obtain an optimal solution $\bar{x}$ and the LP tableau can be written as

$$\max \left\{\bar{a}_{00} + \sum_{j \in \mathcal{NB}} \bar{a}_{0j} x_j\right\} \tag{1.12}$$

$$x_{B_u} + \sum_{j \in \mathcal{NB}} \bar{a}_{uj} x_j = \bar{a}_{u0}, \qquad \text{for } u = 1, \ldots, m, \tag{1.13}$$

where $\bar{a}_{uj}$, B_u are the entries and the basic variable for the u-th row of the tableau and $\mathcal{B}$, $\mathcal{NB}$ are the sets of basic and non-basic variables in the LP solution.

If the solution $\bar{x}$ is not integer there exists some row i for which $\bar{a}_{i0} \notin \mathbb{N}$.[7] For that

[7]Note that for a row i where a variable x_k is basic the coefficient $\bar{a}_{i0}$ equals the current value of x_k, namely $\bar{x}_k$.

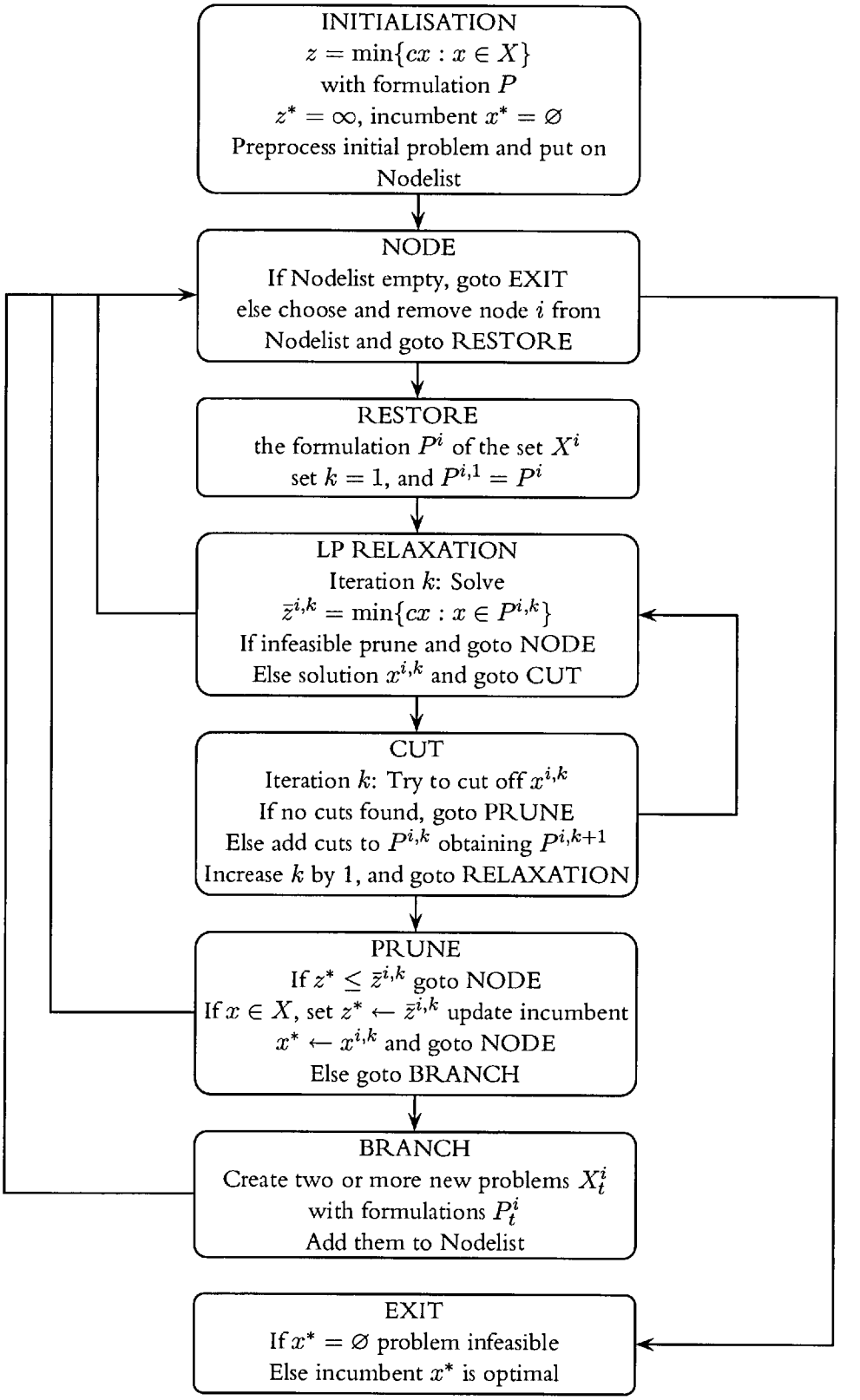


Figure 1.4: An overview of the Branch and Cut algorithm

row, the Gomory cut is the following fractional cut

$$x_{B_i} + \sum_{j \in \mathcal{NB}} \lfloor \bar{a}_{ij} \rfloor x_i \leq \lfloor \bar{a}_{i0} \rfloor \tag{1.14}$$

which can be written as

$$\sum_{j \in \mathcal{NB}} f_{ij} x_i \geq f_{i0} \tag{1.15}$$

where $f_{ij} = \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$ and $f_{i0} = \bar{a}_{i0} - \lfloor \bar{a}_{i0} \rfloor$ are the fractional parts of $\bar{a}_{ij}$ and $\bar{a}_{u0}$ respectively. This inequality is valid for (1.11) and cuts off the current LP solution. Such a cut can be generated from any row of the tableau corresponding to a basic variable with a fractional value.

Note that in order to add the Gomory cut in the LP formulation it is necessary to eliminate the terms from (1.15) that correspond to the slack variables. This can be done using the definition of the slacks and *ad hoc* substituting into (1.15); and this means that the Gomory cut will, in general, not relate the variables of the original problem that are non-basic only but some of the basic ones too.

A similar inequality is the Gomory Mixed Integer Cut, which is valid for MIPs (e.g., Wolsey (1998)). For a general mixed integer program

$$\max \{\mathbf{c}^T \boldsymbol{\xi} + \mathbf{d}^T \mathbf{x} : \mathbf{A}\mathbf{x} + \mathbf{B}\boldsymbol{\xi} = \mathbf{b}, \quad \mathbf{x} \geq 0, \quad \boldsymbol{\xi} \in \mathbb{N}^n\} \tag{1.16}$$

the following inequality is valid and cuts off the current LP point:

$$\sum_{f_j \leq f_0} f_j \xi_j + \sum_{f_j > f_0} \frac{f_0(1-f_j)}{1-f_0} \xi_j + \sum_{\bar{a}_{uj} > 0} \bar{a}_{uj} x_j + \sum_{\bar{a}_{uj} < 0} \frac{f_0}{1-f_0} \bar{a}_{uj} x_j \geq f_0 \tag{1.17}$$

where the row u of the tableau corresponds to a basic integer variable which is fractional in the LP solution.

These cuts belong to a general category of cuts for integer programs, called *Gomory-Chvátal cuts*, of the form

$$\lfloor \boldsymbol{\lambda}^T \mathbf{A} \rfloor \mathbf{x} \geq \lfloor \boldsymbol{\lambda}^T \mathbf{b} \rfloor \tag{1.18}$$

which are valid for an integer program

$$\min \{\mathbf{c}^T \mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \quad \mathbf{x} \in \mathbb{N}^n\} \tag{1.19}$$

for any vector $\boldsymbol{\lambda} \in \mathbb{R}^m_+$ (where m is the number of constraints). Indeed it is known that any cut of the form (1.14)—including therefore Gomory cuts—can be expressed as a Gomory-Chvátal cut (see Nemhauser & Wolsey (1988)); and, moreover, the converse is

true, subject to some conditions (see Cornuéjols & Li (2001), Letchford & Lodi (2002)). The cuts (1.18) were derived in Chvátal (1973) as an observation that the slack variables were not required in order to obtain Gomory-type cuts.[8] Obtaining vectors λ (the *Gomory-Chvátal multipliers*) that produce good quality cuts is not easy however; for some cases multipliers that yield facets for some problems are known (e.g., Capara, Fischetti & Letchford (2000)) but the general behaviour of Gomory-Chvátal cuts is anything but well understood.

1.5 Thesis overview

This thesis is divided in four main parts. Chapter 2 presents an overview of the literature in the area of reliable network design, concentrating on problems with fixed and variable costs, and the various ways that reliability can be implemented in a network design problem. The techniques that can be used to solve these problems are also outlined. The chapter concludes with the definition of the network design problem that will be studied in this thesis.

In chapters 3 and 4, the uncapacitated version of the reliable network design problem examined in this thesis is discussed. In chapter 3, a new mixed integer programming formulation is presented for this problem and an exact solution method is proposed based on Branch and Cut. Several known classes of valid inequalities are described, implemented and the computational experience obtained is reported. Other cuts, specific to the research problem are also presented in this chapter. In chapter 4, a Branch and Cut algorithm is described for the solution of the research problem to optimality.

Chapter 5 presents a novel heuristic approach for the solution of this problem using techniques from the area of ant colony metaheuristics. We develop a new solution framework, suited to the network design problem examined in the thesis, and computational results obtained with the algorithm are reported.

In chapter 6 the capacitated version of the reliable network design problem is studied, and a new MIP formulation (to account for the increased reliability requirements that the added edge capacities impose) is presented. This formulation is solved to optimality using a Branch and Cut algorithm.

The thesis concludes with chapter 7 where the contributions of this research are outlined together with suggestions for future extensions.

[8] It is also shown that it is enough to consider vectors λ with at most $\min\{\#\text{vars}, \#\text{constraints}\}$ non-zero elements in order to generate any inequality of the form (1.18).

Network: "Anything reticulated or decussated at equal distances, with interstices between the intersections."

Dictionary of the English Language (1755)

Chapter 2

Reliable network design problems

2.1 Introduction

In this chapter the fields of network design and network reliability will be reviewed. The chapter begins with an overview of the different types of network design problems with emphasis to problems containing flows and both fixed and variable costs. The different classifications of network reliability found in the literature will be presented next, followed by a definition of the research problem studied in this thesis.

2.2 Classification of network design problems

This section is concerned with general network design problems and their various characteristics (objective functions, topological constraints, flow properties). In the vast majority of this work we shall only discuss linear problems. The section will focus on uncapacitated problems; capacitated network design will be studied in more detail in chapter 6, as it deserves special mention due to its different characteristics and complexity.

Table 2.1 shows a possible classification of network design problems by Balakrishnan, Magnanti & Mirchandani (1997) and represents in general terms the different active areas of research. Here UND stands for 'uncapacitated network design', ST for 'Steiner tree problem' MLND is 'multi-level network design', CND means 'capacitated network

design', NL is the 'network loading problem', NS is 'network survivability' and NR is the 'network restoration problem'.

	Uncapacitated			*Capacitated*			
Characteristic	*UND*	*ST*	*MLND*	*CND*	*NL*	*NS*	*NR*
Cost (Fixed, Variable)	F,V	F	F,V	F,V	F,V	F	F
Capacity (Bundle, Commodity Based)	–	–	–	B	B	CB	CB
Topology (Tree, Connectivity, Multi-level, Duplication)	–	T	M	–	D	C	–
Flow (Simultaneous, Non-simultaneous)	S	S	S	S	S	N	N

Table 2.1: Classification of network design problems

The *Steiner Tree Problem* is described as follows: given a graph $G(V, E)$, edge weights and a set of vertices $T \subseteq V$, the goal is to find the minimum weight tree that spans all the vertices in T, allowing the use of other (Steiner) vertices, not in T, if necessary. The Steiner tree problem is $\mathcal{NP}$-hard, so a polynomial time algorithm is unlikely to be found.[9] (See Magnanti & Wolsey (1995), Hwang & Richards (1992) and Maculan (1987) for an overview of the Steiner tree problem.)

An extension to the Steiner Tree Problem is the *Multi-Level Network Design Problem* (sometimes called *Multi-Weighted Steiner Problem*). In this setup, each vertex has a level l_i representing a certain type of facility. The objective is to connect all the vertices at each level using (at least) facilities of grade (quality) l, the common grade level of the vertices. Variants include the *Hierarchical Network Design Problem* where there are only two levels and there are only two vertices that require a grade 2 connection (Current, ReVelle & Cohon (1986), Pirkul, Current & Nagarajan (1991)). In their paper, Duin & Volgenant (1989) present a transformation of this problem to the Steiner tree problem on a graph with twice the number of vertices and edges.

The *Network Loading Problem* seeks a solution of minimum cost that allows point-to-point commodity demand to flow simultaneously on capacitated facilities that we install on the edges, where we are allowed to install an integer number of facilities per edge. Often the capacities are also modular, i.e., if they are sorted in ascending order $c_1 < c_2 < \ldots < c_k$ then c_{i+1} is a multiple of c_i. This problem is $\mathcal{NP}$-hard even in the case with one commodity and two facility types (Magnanti & Mirchandani (1993)). This type of

[9]We adopt the widely accepted opinion that $\mathcal{P} \neq \mathcal{NP}$, at least for a philosophical reason: for if $\mathcal{P} = \mathcal{NP}$ then this, among other things, would imply that the constructive effort of creating mathematical proofs (including Ph.D thesis) is no more complex a task than the mechanical verification effort taken by a computer to verify the correctness of a statement; such an assumption renders mathematical proofs to a mundane computational task (Hemaspaandra & Ogihara (2002)).

problem is common in telecommunications (for instance, multiple transmission facilities with different capacities on the same network) and also in the modelling of congestion, although these problems are in general non-linear. This problem has been extensively studied and the most important results and surveys in the area can be found in Barahona (1996), Bienstock & Günlük (1996) and Magnanti, Mirchandani & Vachani (1995).

The *Network Restoration Problem* deals with the installation of sufficient spare capacity on a network (given a network flow configuration) so that the flow can be re-routed should any edge fail. Many alternatives can be found (e.g., depending on whether the flow that must be re-routed can be split or not etc.) and many articles restricting the problem on particular network types, alternative types of network failures, performance assessments and the study of impact and severity of failures (e.g., see issue 12 (1) of the *IEEE Journal on Selected Areas in Communications* (1994) for an overview) of such variants.

In the next part of this chapter we are going to concentrate on the uncapacitated network design problem with both fixed and variable costs, which forms the basis of the research problem.

2.3 The single commodity uncapacitated network design problem with fixed charge cost

The general single commodity uncapacitated network design problem can be defined as follows: Consider a directed network $G(V, E)$ (with no loops or parallel edges) and let $\mathbf{d} \in \mathbb{R}^{|V|}$ be the vector of the vertex net demands of the commodity that is to be routed on the network. The (unknown) edge flows are represented with the vector $\mathbf{x} \in \mathbb{R}^{|E|}$ and the edge costs per unit of flow are $\mathbf{c} \in \mathbb{R}^{|E|}$. The (unknown) vector $\boldsymbol{\xi} \in \Xi \subseteq \{0,1\}^{|E|}$ represents whether a particular edge is present (1) or not (0), and the vector $\mathbf{b} \in \mathbb{R}^{|E|}$ represents the edge installation cost for each edge. The mathematical formulation of the above can be given by means of the following linear model:

MODEL M2

$$\textbf{min} \quad \mathbf{c}^T\mathbf{x} + \mathbf{b}^T\boldsymbol{\xi} \tag{M2.1}$$

subject to

$$\mathbf{N}\mathbf{x} = \mathbf{d} \tag{M2.2}$$

$$0 \leq x_i \leq Q_i\xi_i \tag{M2.3}$$

$$\boldsymbol{\xi} \in \Xi \tag{M2.4}$$

THE SINGLE COMMODITY UNCAPACITATED NETWORK DESIGN PROBLEM

The constant vector $\mathbf{Q}$ represents upper limits on the values of the flows. In this uncapacitated version of the problem, these numbers are large enough so as not to restrict the flow, in other words, Q_i is at least as large as the largest possible flow in the network. The matrix $\mathbf{N}$ is the vertex-edge incidence matrix of G. Constraint **M2**.2 specifies that there is no loss of flow in the network and constraint **M2**.3 states that no flow can exist if an edge is not present. The purpose of the constraint **M2**.4 is to include topological restrictions on the layout. As a minimum, it is usually required that the network should be connected, although in more complex cases this may be implied indirectly from other topological constraints.[10] If no further restrictions are required then $\Xi = \{0,1\}^{|E|}$. Moreover, if we only require the network to be connected, then we obtain a network design problem known as the *Fixed Charge Problem*, which shall be studied in detail in the next section.

Furthermore, if all the demands d_i are zero then this problem reduces to the simple *Minimum Spanning Tree Problem*, where we seek a connected subset of the edges that contain all the vertices. The minimum spanning tree problem is a particularly easy problem which can be solved in a straightforward manner in polynomial time using greedy-type algorithms, e.g., the algorithms described by Prim (1957) and those for some special cases derived earlier by Kruskal (1956) and Borůvka (1926). The reason for this behaviour is that spanning trees are bases of a matroid and, as such, can be solved to optimality with a greedy-type algorithm (Edmonds (1971)). The quick procedure for generating spanning trees can yield fast heuristics for the reliable network design problem as we shall see later.

The number of distinct spanning trees of a graph, however, increases exponentially with the vertex size. For any graph this number equals the determinant of the matrix BB^T, where B is the vertex-arc incidence matrix of (any orientation of) the graph

[10] If connectivity is not desired, i.e., it is possible to use only a subset of the available vertices then this yields the Steiner problem versions of the problem, c.f., the Steiner tree problem presented in section 2.2.

with one of its rows deleted (Kirchhoff (1847)). This gives n^{n-2} spanning subtrees for the complete graph K_n and $m^{n-1}n^{m-1}$ for the complete bipartite graph $K_{m,n}$ (e.g., see Jungnickel (1999)), although they can be enumerated quite fast with a small extra overhead.[11]

Other special cases that can be solved in polynomial time include the case where the variable costs c_i are zero. In that case the flow variables do not affect the overall cost of the solution and can therefore be ignored, reducing the problem to a minimum spanning tree. The values of the flow variables could be found later and added without violating feasibility (since there are no edge capacities) and without affecting the optimality of the solution. The same is also true if the fixed costs b_i are zero (e.g., see Hochbaum & Segev (1989)).

2.3.1 The fixed charge problem (FCP)

The (uncapacitated) fixed charge network design problem (FCP) constitutes the simplest problem with a cost function consisting of a constant cost per edge used in the design of the network and a variable cost typically related to the flow through the edge.

The fixed costs of the problem represent the costs incurred with the installation of a given edge, whereas the variable costs are associated with the flow passing through i.e., the running costs of the network. It is reasonable to assume that the latter will be different according to the direction of the flow, in other words, $c_{ij} \neq c_{ji}$. This is the case, for example, for water networks where the cost will be different if the water is flowing upstream or downstream. We assume, however, that the fixed installation costs are associated with each *edge* and are the same irrespectively of the direction the flow is moving.

The fixed charge problem is $\mathcal{NP}$-hard even when there is a single source and the fixed costs are a constant multiple of the variable costs, since the Steiner Tree Problem can be reduced to a fixed charge problem of that form (see Hochbaum & Segev (1989)).

The solution of the uncapacitated FCP with linear costs is always a tree. This is true in the more general case of cost functions that are concave with respect to the flows (Zangwill (1968)). The intuitive reason behind this property is that, for an uncapacitated network, given a solution that is not a tree, we can do better (costwise) by removing a cycle and increasing the other flows accordingly. From the concavity property, the newly

[11] Kapoor & Ramesh (1995) present an algorithm with a running time of $O(N + V + E)$ for undirected graphs and $O(NV + V^3)$ for directed graphs (Kapoor & Ramesh (2000)), where N is the number of spanning trees, and V, E are the number of vertices and edges respectively.

adjusted combined flow will be cheaper than the sum of the original flows that generated it.

We can formulate the FCP as follows, using one design variable ξ per edge and one flow variable x per arc, assuming that there are no arcs going into the source vertex.

MODEL M3

$$\textbf{min} \quad \sum_{\substack{(i,j)\in E \\ i<j}} \left(b_{ij}\xi_{ij} + c_{ij}x_{ij} + c_{ji}x_{ji}\right) \tag{M3.1}$$

subject to

$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = d_i \qquad \text{for all } i \in V' \tag{M3.2}$$

$$\sum_{j:(0,j)\in E} x_{0j} = \sum_{i\in V'} d_i \tag{M3.3}$$

$$x_{ij} \leq Q\xi_{ij} \qquad \text{for all } (i,j) \in E,\ i<j \tag{M3.4}$$

$$x_{ji} \leq Q\xi_{ij} \qquad \text{for all } (i,j) \in E,\ i<j \tag{M3.5}$$

$$x_{ij} \geq 0 \qquad \text{for all } (i,j) \in E \tag{M3.6}$$

$$0 \leq \xi_{ij} \leq 1 \qquad \text{for all } (i,j) \in E,\ i<j \tag{M3.7}$$

$$\xi_{ij} \in \mathbb{Z} \qquad \text{for all } (i,j) \in E,\ i<j \tag{M3.8}$$

THE FIXED CHARGE PROBLEM (FCP)

An interpretation of the notation used is shown in table 2.2; the parameter Q can be assumed to be $\sum_{i\in V'} d_i$, since no flow can be larger than the total demand of all the customers.

Given that the FCP is $\mathcal{NP}$-hard, no efficient algorithm for its exact solution is likely to be found. Techniques for its exact solution will inevitably include some form of LP-based approach, for example a cutting plane algorithm or enumeration algorithms such as branch and bound or branch and cut. These approaches are briefly described in the next section.

2.3.2 Solution methods for the FCP

Exact solution methods for solving the FCP concentrate mainly in cutting plane algorithms, 'Branch and Bound' and 'Branch and Cut'. These techniques will require

Symbol	*Interpretation*
V':	$V\backslash\{0\}$, the set of vertices excluding the source
d_i:	demand of vertex i, where $i \in V'$
b_{ij}:	fixed cost, $(i,j) \in E$, $i < j$
c_{ij}:	linear cost per unit flow, $(i,j) \in E$
Q:	a large number representing the maximum flow through any arc
ξ_{ij}	are (unknown) binary variables, defined for $(i,j) \in E$, $i < j$, representing whether a given edge is present or not
x_{ij}	are the (unknown) flow variables, defined for $(i,j) \in E$.

Table 2.2: The variable names used in the MIP formulation of the fixed charge problem

the solution of the relaxation of the formulation **M3** obtained by dropping the integrality constraints **M3**.8. The initial LP relaxation can usually be solved quickly, but it can be further simplified, as suggested by Nemhauser & Wolsey (1988) and Stallaert (2000), by noticing that it is equivalent to the LP obtained by adding the equalities

$$\xi_{ij} = \begin{cases} 1 & \text{if } b_{ij} \leq 0 \\ (x_{ij} + x_{ji})/b_{ij} & \text{if } b_{ij} = 0 \end{cases} \tag{2.1}$$

and thus eliminating the variables ξ_{ij}.

An alternative standard approach to the FCP, proposed by Rardin & Choe (1979), is to re-formulate the FCP as a multicommodity network problem represented by the formulation **M4**. This is done by introducing new variables w_{ij}^t which represent the fraction of flow that is destined for customer t passing through arc (i,j). This technique is also used in Ng & Rardin (1997), and polyhedral results for the multicommodity formulation of the FCP are obtained in Hellstrand, Larson & Migdalas (1992) and Rardin & Wolsey (1993).

MODEL M4

$$\textbf{min} \quad \sum_{\substack{(i,j)\in E \\ i<j}} \left(b_{ij}\xi_{ij} + c_{ij}x_{ij} + c_{ji}x_{ji}\right) \tag{M4.1}$$

subject to

$$\sum_{j:(j,i)\in E} w_{ji}^t - \sum_{j:(i,j)\in E} w_{ij}^t = 1 \qquad \text{for all } i, t \in V' \tag{M4.2}$$

$$\sum_{j:(0,j)\in E} w_{0j}^t = 1 \qquad t \in V' \tag{M4.3}$$

$$\sum_{t\in V'} d_t w_{ij}^t = x_{ij} \qquad \text{for all } (i,j) \in E \tag{M4.4}$$

$$0 \le w_{ij}^t \le \xi_{ij} \qquad \text{for all } (i,j) \in E,\ i<j \tag{M4.5}$$

$$0 \le w_{ji}^t \le \xi_{ij} \qquad \text{for all } (i,j) \in E,\ i<j \tag{M4.6}$$

$$0 \le \xi_{ij} \le 1 \qquad \text{for all } (i,j) \in E,\ i<j \tag{M4.7}$$

$$\xi_{ij} \in \mathbb{Z} \qquad \text{for all } (i,j) \in E,\ i<j \tag{M4.8}$$

THE MULTICOMMODITY EXTENDED FORMULATION FOR THE FCP

Recently, Ortega & Wolsey (2003) developed a Branch and Cut algorithm for the uncapacitated FCP. For this algorithm, the dicut valid inequalities are introduced and a heuristic for their separation is presented. The Branch and Cut algorithm was used to solve randomly generated test problems based on networks on grids, series-parallel graphs and planar graphs.

Other exact approaches for solving the FCP include dynamic programming (see Fontes (2000)), where an algorithm for solving the FCP with general concave cost functions is presented.

Heuristic algorithms for the solution of the FCP have also been developed, for instance, Kim & Pardalos (1999) present a dynamic slope scaling procedure for the solution of the problem, derived as a special case of the minimum concave-cost network flow problem. In a more recent approach, Kim & Hooker (2002) propose a hybrid optimisation method combining constraint programming with LP relaxation procedures, and report a 50% improvement in performance compared to commercial MIP solvers.

Finally it should be noted that, apart from network design problem examined in this section, other types of problems are often modelled using both fixed and variable costs. The most important category is the *fixed charge transportation problem*, where the total

cost of transporting a product consists of a fixed part and a variable part dependent on the amount of goods being transported. This problem is well-studied and an overview can be found in Herer, Rosenblatt & Hefter (1996), Adlakha & Kowalski (1999), Sun, Aronson, McKeown & Drinka (1998) and Adlakha & Kowalski (2003).

2.4 Topological approaches to network reliability

In this section the different ways to implement reliability in a network design problem will be presented. As already mentioned in chapter 1, there is an extremely wide range of possible definitions of reliability (survivability) found in the literature. They can be divided into the following categories:

- topological approaches to reliability
- probabilistic approaches to reliability
- empirical approaches (based on customer satisfaction etc.)
- other specialised approaches (entropy reliability measures, game theory based etc.)

For the purposes of this research, we provide a detailed analysis of the *topological* approaches to reliability. Some discussion on the remaining ones will be presented in section 2.5 for reasons of completeness. The last two categories, however, have restricted applicability in an operational research network design framework.

Topological reliability requirements relate to the solution layout of a network design problem. These requirements are usually defined via notions of network connectivity defined in a graph theoretical way. Connectivity measures are extremely popular for two main reasons: Firstly, a variety of models for network design use simple linear or concave cost functions and in either case the optimal solution is a tree. Given that a tree is minimally connected, if any edge or vertex breaks down then the network will become disconnected. To prevent against such a situation a connectivity requirement is added.

2.4.1 Definitions of network connectivity

There are many survey articles on the connectivity properties of a network and related graph theoretical results. Perhaps the most complete among the sources that contain theoretical results are by Tutte (1966), Tutte (1984), Bollobás (1978), Mader (1979) and

Frank (1995). Other general review articles about network connectivity and their applications can be found in Raghavan & Magnanti (1997), Gavish (1992), Gavish & Neuman (1992), and Cardwell, Monma & Wu (1989).

Connectivity is the most obvious and most commonly used measure of topological reliability in many types of networks. Consider an undirected network with an underlying graph $G(V, E)$ containing no parallel edges or loops. The following notions of topological connectivity can be defined:

- *Node connectivity*
- *Edge connectivity*
- *Tree connectivity*

A set of paths connecting two vertices i and j is called *node disjoint* (respectively *edge disjoint*) if no vertex[12] (respectively edge) appears in more than one path.[13] Two vertices are k-node (edge) connected if k node (edge) disjoint paths joining them exist.

A graph is called *k-node-connected* (respectively *k-edge-connected*) if every pair of vertices is k-node (k-edge) connected. The *node connectivity*, $\kappa(G)$ (respectively, edge connectivity, $\lambda(G)$) of a graph is defined as the largest k such that the graph G is k-node (respectively k-edge) connected. i.e.,

$$\kappa(G) = \max\{k \mid \text{for all pairs of vertices there are } k \text{ node-disjoint paths joining them}\} \quad (2.2)$$

$$\lambda(G) = \max\{k \mid \text{for all pairs of vertices there are } k \text{ edge-disjoint paths joining them}\} \quad (2.3)$$

Equivalently, a graph is called k-node/edge connected if its node/edge connectivity is at least k.

Whitney (1932*a*) showed that the node connectivity is a stronger requirement than edge connectivity, in the sense that

$$\kappa(G) \leq \lambda(G) \leq \deg(G) \quad (2.4)$$

where $\deg(G) = \min\{\deg(i),\ i \in V\}$ is the minimum degree of all vertices of G. Later Chartrand & Harary (1968) proved the following 'converse' statement: for any three given integers such that

$$0 < k \leq l \leq d, \quad (2.5)$$

[12] except of course the endpoints i, j.

[13] Note also that no vertex can appear in the same path twice, since paths cannot 'loop', i.e., revisit a given vertex again.

it is possible to find an undirected graph such that $\kappa(G) = k$, $\lambda(G) = l$ and $\deg(G) = d$.

It is also worth mentioning that the node connectivity $\kappa(G)$ can be calculated with complexity $O(|V|^{1/2}|E|^2)$, proven in Even & Tarjan (1975), and the edge connectivity $\lambda(G)$ of a (directed) graph can be calculated with complexity $O(|V||E|)$, a result due to Matula (1987) and Mansour & Schieber (1989). To determine, however, whether a given graph is k-node connected or not for a given k, can be done with complexity of $O(|V||E|)$ (see Even (1977), Galil (1980)), although for the cases $k = 1, 2$ or 3 the results by Tarjan (1972) and Hopcroft & Tarjan (1973) prove that it is possible to obtain a complexity of only $O(|E|)$.

A graph is called *k-tree-connected* if it contains k edge-disjoint spanning trees (N.B., not necessarily rooted at the same vertex). Edge connectivity is related to tree-connectivity via theorem 11 of section 2.4.3.

2.4.2 Mathematical properties

This section describes some mathematical properties of connected graphs which will be useful in understanding the structure of the solution and the description of the algorithms presented in the thesis.

The most important theorem on graph connectivity is Menger's theorem (Menger (1927)), which can be summarised by

> "Either there are k openly disjoint paths between two specified vertices or there is a set of less than k elements separating these two vertices."

Theorem 1 (Menger 1927) *The following statements are true:*

- *Let G be a graph (directed or undirected), and let s and t be two vertices. Then there are k edge-disjoint s-t paths if and only if after deleting any $k-1$ edges t is still reachable from s.*
- *Let G be a graph (directed or undirected), and let s and t be two non-adjacent vertices. Then there are k node-disjoint s-t paths if and only if after deleting any $k-1$ vertices (distinct from s and t) t is still reachable from s.*

This theorem enables us to define edge and node connectivity via the number of elements that we need to remove in order to separate two vertices, indeed, Whitney (1932*b*)

presents the following alternative definitions of connectivity that follow from Menger's theorem:[14]

Theorem 2 (Whitney 1932) *(Alternative definitions of edge and node connectivity)*

- *An undirected graph is k-edge-connected if and only if it remains connected after $k-1$ edges are deleted.*
- *An undirected graph is k-node-connected if and only if it remains connected after $k-1$ vertices are deleted.*

These observations are summed up in table 2.3, which contains equivalent definitions of connectivity for undirected graphs.

	Definition A	*Definition B*	*Definition C*
k-edge-connected:	$\forall\ s,t$, there exist k edge disjoint paths joining s, t.	the graph remains connected if $k-1$ edges are removed.	$\lambda(G) \geq k$
k-node-connected:	$\forall\ s,t$, there exist k node disjoint paths joining s, t.	the graph remains connected if $k-1$ vertices are removed.	$\kappa(G) \geq k$

Table 2.3: Equivalent definitions for edge and node connectivity

A procedure to construct any k-edge connected graphs is presented below. The following operations are defined:

- **Operation Q_k:** Pick k arbitrary edges, subdivide them with vertices $z_1, \ldots, z_k$ and identify $z_1, \ldots, z_k$ into a new vertex z.
- **Operation Q_k^+:** Proceed as in Q_k, then choose a vertex x and add a new edge joining x and z.
- **Operation Q_k^2:** Proceed as in Q_k, then choose k distinct edges of this new graph, not all incident to z, subdivide each of these edges by a vertex $z'_1, \ldots, z'_k$, identify these vertices into a new vertex z', and join z and z' with a new edge.

The following two theorems are true by Lovász (1979) and Mader (1978) respectively.

Theorem 3 *An undirected graph is $2k$-edge-connected if and only if it can be obtained from a vertex by adding edges and applying Operation Q_k.*

[14]Note that it is necessary to overcome the requirement of Menger's theorem that the two vertices in the node-disjoint case are not adjacent.

Theorem 4 *An undirected graph is $(2k+1)$-edge-connected if and only if it can be obtained from a vertex by successive addition of edges and repeated applications of Operations Q_k^+ and Q_K^2.*

For 2-edge connected graphs in particular, the following implications are true (e.g., see Frank (1995)):

Theorem 5 *For a connected graph $G = (V, E)$ the following are equivalent:*

(i) G is 2-edge-connected (defined as a graph without cut edges[15]*).*

(ii) For any pair of vertices there are two edge-disjoint paths connecting them.

(iii) Any edge is contained in a circuit.

(iv) G can be built up from a vertex by sequentially adjoining edges (loops allowed) and subdividing edges.

A similar result holds for node-connectivity. Property (iv) is more commonly known as "G has an *ear decomposition.*" An ear decomposition of G is defined as a sequence $G_0, G_1, \ldots, G_k$ of subgraphs of G where G_0 consists of one vertex and no edge, and G_i arises from G_{i-1} by adding one path for which the two endpoints belong to G_{i-1} while the interior vertices do not.[16]

Finally some other interesting consequences are stated below (Jungnickel (1999)).

Theorem 6 *A k-node-connected graph on n vertices contains at least $\lceil \frac{kn}{2} \rceil$ edges.*

This bound was proven by Harary (1962) to be tight.

Theorem 7 *A planar graph can be at most 5-node connected.*

This is derived from the fact that every planar graph contains a vertex of degree at most 5, which is obtained from Euler's formula relating the number of vertices, edges and faces of a planar graph.

Theorem 8

- *Let G be a graph with n vertices. If* $\deg(G) \geq n/2$ *then* $\lambda(G) = \deg(G)$. *(Chartrand (1966))*
- *For $\{0,2\}$-regular graphs,*[17] $\kappa(G) = \deg(G)$. *(Brouwer & Mudler (1997))*

[15] an edge e of a graph G is a *cut edge* if $G \backslash e$ has more components than G, see section 1.1.1.

[16] c.f.: a graph is 2-node-connected if and only if it has an *open* ear-decomposition (Korte & Vyger (2000)).

[17] A $\{0,2\}$-regular graph is a connected graph such that any pair of vertices has either 0 or 2 common neighbours (e.g., K_4 or the icosahedron).

2.4.3 Connectivity in directed graphs

Many of the results previously presented can be extended for the case of digraphs (e.g., Jungnickel (1999)). The notion of *strong connectivity* in a directed graph is defined as the property that every vertex is accessible from any other. Clearly, this is true if and only if every vertex is a root, and hence a tree is never strongly connected. The definition of edge connectivity is often replaced by *arc* connectivity to distinguish the two cases; a digraph is k-arc connected if the deletion of less than k arcs leaves a strongly connected digraph. Similar theorems to theorems 3, 4, and 5, can be derived for digraphs (Frank (1995)).

Theorem 9 (Edmonds, 1973) *Given a digraph $D = (V, A)$ with a specified vertex s, there are k pairwise arc-disjoint arborescences rooted at s if and only if $\delta^-(X) \geq k$ for all $X \subseteq V \backslash s$, where $\delta^-(X)$ is the set of arcs with the end vertex in X and the start vertex not in X.*

A common question relating edge-disjoint paths with edge-disjoint trees is answered by the following conjecture and theorem (Frank (1995)). They are also related to the previous theorem.

Conjecture 10 *The following statements have not been fully studied.*

(i) "Suppose that, given a digraph $D = (V, A)$ and a specified vertex $s \in V$, there are k openly disjoint paths from s to any other vertex of D. Then there are k arc-disjoint s-arborescences of D such that for any vertex $v \in V \backslash s$ the k paths from s to v uniquely determined by the arborescences are openly disjoint."

(ii) "Suppose that, given a graph $G = (V, E)$ and a specified vertex $s \in V$, there are k openly disjoint paths from s to any other vertex of G. Then there are k spanning trees of G such that for any vertex $v \in V \backslash s$ the k paths from s to v uniquely determined by the k trees are openly disjoint."

(iii) The same as (ii) but replace "openly disjoint" with "edge-disjoint."

The last part of this conjecture (edge-disjoint paths) is of major interest. Firstly it is easy to see that (i) implies (ii). It is known by Whitty (1987) that (i) holds for $k = 2$ and that (ii) holds for $k = 3$ (Zehavi & Itai (1989)). Huck (1995) proved that (i) holds for acyclic digraphs but disproved it for general digraphs when $k \geq 3$. Conjecture (iii) is proved for $k = 2$, using ear-decompositions for two-edge-connected graphs. This means that, given

a graph $G = (V, E)$ and a vertex $s \in V$ (the source) such that there are two edge disjoint paths from s to any other vertex, then there are two spanning trees such that, for any vertex (other than the source), the two paths from the source to the vertex determined by the two trees are edge-disjoint.

We finally briefly present some known results regarding the properties of tree-connected digraphs.

Theorem 11 *A graph $G = (V, E)$ is k-tree-connected if and only if G can be obtained from a vertex by sequentially adding edges (connecting old vertices) and applying the following operation: choose j ($0 \leq j \leq k$) distinct edges and $k - j$ (not necessarily distinct) vertices, subdivide the j edges by j vertices, identify these j vertices to a new vertex z and connect z and the $k - j$ old vertices by $k - j$ edges.*

Theorem 12 *A digraph is called k-arc-connected (or strongly k-arc-connected) if deleting any subset of arcs of less than k elements leaves a strongly connected digraph. This is equivalent to having $\delta^+(X) \geq k$ for all $\emptyset \neq X \subset V$. A digraph is* k-arc-connected from s *if $\delta^-(X) \geq k$ for any subset $X \subseteq V - s$. This is equivalent to (Menger's theorem) the property that there are k arc-disjoint paths from s to any other vertex, which, by the previous theorem is equivalent to that there are k spanning arborescences rooted at s.*

Theorem 13 (Farber, Richter & Shank (1985)) *Any pair of edge-disjoint spanning trees can be obtained from any other by a sequence of single edge exchanges in a way that preserves, at each step, the property of being edge-disjoint spanning trees.*

Theorem 14 (Li (1998)) *Any pair of edge-disjoint minimum-weight connected spanning k-edge subgraphs can be obtained from any other by a sequence of single-edge exchanges in a way that preserves, at each step, the property of being edge-disjoint minimum-weight connected spanning k-edge subgraphs.*

Finally, a key property of the (minimum cost) solution to the k-edge or k-node connected network design problem is the following:

Theorem 15 *If the costs satisfy the triangle inequality, there exists an optimal k-edge connected spanning network satisfying the following properties:*

(i) all the vertices are of degree k or degree $k + 1$, and

(ii) removing any set of at most k edges does not leave any of the resulting connected components k-edge connected.

The above was proven by Monma, Munson & Pulleyblank (1990) for the complete graph K_n, for the case $k = 2$, shortly followed by the proof by Bienstock, Brickell & Monma (1990) that the theorem is true for arbitrary k, and moreover, that "k-edge" can be replaced throughout with "k-node" and the theorem will still be true, subject to the technical restriction that $|V| \geq 2k$ for (i) to hold.

2.4.4 A general model with connectivity constraints

Perhaps the most extensive survey on topological reliability implementation for network design problems without flows is by Grötschel, Monma & Stoer (1995*a*). The different approaches described there are all variations of the general MIP formulation **M5** presented below. Using their notation, the input data to the model is an undirected graph $G(V, E)$ and three non-negative integers r_{ij}, k_{ij} and d_{ij} per edge. The reliability requirements are that for any pair of vertices s and t, the following must hold:

- The network must have at least r_{st} edge-disjoint paths from vertex s to t.
- Removing at most k_{st} vertices (different from s, t) must leave d_{st} edge-disjoint paths joining s and t.

We define the following functions, for all $Z, W \subseteq V$:

$$r(W) := \max\{r_{st} \mid s \in W,\ t \in V \backslash W\} \tag{2.6}$$

$$d(Z, W) := \max\{d_{st} \mid s \in W \backslash Z,\ t \in V \backslash (Z \cup W)\} \tag{2.7}$$

A general model for reliable network design based on edge and node connectivity is given by the following formulation:

MODEL M5

$$\textbf{min} \quad \sum_{(i,j)\in E} c_{ij}\xi_{ij} \qquad \textbf{(M5.1)}$$

subject to

$$\sum_{i\in W}\sum_{j\in V\setminus W} \xi_{ij} \geq r(W) \qquad \text{for all } W \subseteq V, \emptyset \neq W \neq V \qquad \textbf{(M5.2)}$$

$$\sum_{i\in W}\sum_{j\in V\setminus(Z\cup W)} \xi_{ij} \geq d(Z,W) \qquad \text{for all } \textit{eligible}^{18}\ (Z,W) \text{ of subsets of } V \qquad \textbf{(M5.3)}$$

$$0 \leq \xi_{ij} \leq 1 \qquad \text{for all } (i,j) \in E \qquad \textbf{(M5.4)}$$

$$\xi_{ij} \in \mathbb{Z} \qquad \text{for all } (i,j) \in E \qquad \textbf{(M5.5)}$$

A MIP INTRODUCING TOPOLOGICAL RELIABILITY

In the formulation **M5**, c_{ij} denotes the cost of each edge (the model does not consider flows, so only fixed costs are present) and ξ_{ij} are the decision variables representing whether an edge is included in the layout or not. The objective is to design a network, at a minimum cost, such as the edge and node connectivity requirements (represented by constraints **M5**.2 and **M5**.3 respectively) are satisfied.

This general formulation contains many problems as its subcases. For example, the minimum spanning tree problem arises when constraints **M5**.3 are removed and $r_{st} = 1$ for all s, t. The minimum cost k-edge connected network design problem is obtained by dropping constraints **M5**.3 and $r_{st} = k$ for all s, t. The Steiner Tree Problem can be obtained by setting $r_{st} = 1$ for all pairs of vertices s, t that are in a given set $S \subseteq V$, $r_{st} = 0$ otherwise, and dropping the node connectivity requirements **M5**.3. Since the Steiner problem belongs to the class of $\mathcal{NP}$-hard problems (see e.g., Garey & Johnson (1979)) the network design model **M5** is also $\mathcal{NP}$-hard.

Another variation of the above model is when one is interested in the edge-connectivity requirements r_{st} only. We can define a *cut*, $\delta(W)$, where $W \subseteq V$ as the set of edges with one endpoint in W and the other endpoint in $V\setminus W$. We also define $\text{con}(W) = \sum_{st\in\delta(W)} r_{st}$, and $x(F) = \sum_{ij\in F} x_{ij}$. This version of the problem appears below.

[18] In this setup, a pair (Z,W) is *eligible* when $Z \cap W = \emptyset$ and $|Z| = k_{st}$ for at least one pair of vertices s,t with $s \in W$ and $t \in V\setminus(Z\cup W)$.

MODEL M6

$$\textbf{min} \quad \sum_{(i,j)\in E} c_{ij}\xi_{ij} \tag{M6.1}$$

subject to

$$\xi(\delta(W)) \geq \text{con}(W), \qquad \text{for all } \emptyset \neq W \subset V \tag{M6.2}$$

$$0 \leq \xi_{ij} \leq 1 \qquad \text{for all } (i,j) \in E \tag{M6.3}$$

$$\xi_{ij} \quad \text{integral} \qquad \text{for all } (i,j) \in E \tag{M6.4}$$

THE EDGE-CONNECTIVITY NETWORK DESIGN PROBLEM

For many practical applications low connectivity requirements are needed. Restricting attention to 2-edge or 2-node connectivity is quite common in the literature since simultaneous failures of more than one component are unlikely in the normal operation of many networks (mainly due to the fact that breakages are often repaired fairly quickly).

The edge-connectivity network design problem **M6** has been studied extensively in the context of fiber optic telecommunication networks where a two-edge connected design ($\text{con}(W) = 2$) is commonly used.

2.4.5 Exact solution methods

Exact solution techniques for the integer programs **M5** and **M6** focus on attempts to derive valid inequalities that can be used in the development of cutting plane or Branch and Cut algorithms.

Grötschel & Monma (1990) derived some polyhedral results for the general formulation of **M5** and showed that the problem of deciding whether the convex hull of the incidence vectors of the solutions to **M5** (CH) is nonempty or not is an $\mathcal{NP}$-hard problem. The same is true for the problem of finding the degree of CH.

Grötschel, Monma & Stoer (1995*a*) and Grötschel, Monma & Stoer (1995*b*) present a cutting plane algorithm for the design of 2-edge connected and 2-node connected networks. They derive valid inequalities for these problems, namely the partition (or multicut) inequalities, vertex partition inequalities, and the r-cover inequalities. Special cases of r-cover inequalities for 2-edge reliability (which they call odd-wheel inequalities) have are also been studied by Mahjoub (1994). Other valid inequalities for the 2-edge connectivity problem have been derived from transformations of the comb inequalities for the Travelling Salesman Problem (TSP) (Grötschel, Monma & Stoer (1992*a*) and Boyd

& Hao (1993)). A detailed description of these classes of valid inequalities, which include the partition and the r-cover inequalities, together with their separation procedures will be presented in chapter 3.

Some polyhedral results regarding the 2-edge and the 2-node connected network design problems are also known for special types of the input graphs. Using the fact that the Steiner tree problem can be solved polynomially for series-parallel and Halin[19] graphs, it is possible to obtain polynomial descriptions for the 2-edge and 2-node connectivity convex hulls (Winter (1985) and Winter (1986)). A similar approach can be found in Fonlupt & Mahjoub (1999) where the authors define a class of graphs, called *perfectly two-edge connected*, to consist of those graphs whose convex hull of incidence vectors of the 2-edge connectivity problem (CH) is equal to the polytope obtained from the constraints of **M6** by dropping the integrality constraints. Mahjoub (1994) proves that series-parallel graphs are perfectly 2-edge connected, and Barahona & Mahjoub (1995) extend the above by providing a similar description of the convex hull for Halin graphs. For these graphs a complete description of the convex hull is known.

More recently Mahjoub (1997) extended the class of graphs which are perfectly 2-edge connected as follows: let Γ be the class of graphs $G = (V, E)$ for which there is a subset of the vertices $S \subseteq V$ such that:

(i) $|S| = 3$ and S covers all the edges of G,

(ii) $|V \backslash S| \geq 3$ and if $|V \backslash S| = 3$ then $G = K_{3,3}$,

(iii) $\delta(S)$ does not contain multiple edges.

Then the graphs in Γ are perfectly 2-edge connected.

A similar problem is studied in Chopra (1994) where the convex hull of k-edge connected spanning subgraphs is considered. The author considers the case of outerplanar graphs (planar graphs that can be embedded on the plane such that all the vertices lie on the outermost face) and a complete description of the convex hull is obtained for the cases where k is odd.

[19] A Halin graph $G = (V, T \cup C)$ is a graph consisting of a tree T that has no vertex of degree 2 together with a simple cycle C whose vertices are the pendant vertices of T. These graphs can be embedded in the plane, with C as an exterior face, are minimally 3-connected and were first described in Halin (1971).

2.4.6 Heuristic solution methods

Many simple heuristic techniques for two-edge or k-edge reliable network design problems (without flows) can be found in the literature.

Frederickson & Ja'ja' (1982) used a modification of the TSP heuristic by Christofides (1976) and proved that, for the two-edge connected network design problem and provided that the edge costs satisfy the triangle inequality, the solution value SOL will have the same worst-case bound as Christofides algorithm has for the TSP; hence

$$\text{SOL} \leq \frac{3}{2}\text{OPT} \tag{2.8}$$

Other heuristic methods for the 2-edge connected problem are based on tree heuristics, using the property that the solution network can be decomposed into two arc-disjoint spanning trees. Such tree heuristics were derived by Goemans & Bertsimas (1993) and Agrawal, Klein & Ravi (1995) for the k-edge connected problem, although the use of parallel edges is allowed in the solution.

A number of both construction and improvement heuristics for two-connected survivability constraints is presented in Monma & Shallcross (1989). The construction heuristics are based on the ear-decomposition properties of vertex and edge connected networks and the improvement heuristics on 2OPT, 3OPT, and some special transformations (pretzel and quetzel).

Another simple heuristic approximation algorithm is the following by Vazirani (2003): Choose a vertex $r \in V$ of a graph G. We then bidirect each edge of G with two arcs, i.e., we replace each edge (i, j) with the two arcs (i, j) and (j, i), each having the same cost as the original edge. Let this modified graph be H. Edmonds showed that every two-edge connected graph can be divided in two spanning trees rooted at r, and the minimum weight subgraph which has two edge-disjoint arborescences can be solved in polynomial time. It is shown that if this algorithm is applied to graph H then the solution will satisfy

$$OPT(G) \leq OPT(H) \leq 2 \cdot OPT(G). \tag{2.9}$$

Williamson, Goemans, Mihail & Vazirani (1995) and Gabow, Goemans & Willamson (1998) present polynomial running time approximation algorithms for the problem of finding a minimum cost spanning subgraph such that the edge connectivity between any pair of vertices is no less than a specified amount.

Loganathan, Sherali & Shah (1990) describe a two phase heuristic that uses linear programming to modify a spanning tree network in such a way to reduce the cost without

taking into consideration the reliability constraint. In the second stage the algorithm systematically adds arcs to the tree to guarantee 2-edge-connectivity.

Many metaheuristic methods can also be found in the literature but they usually consider specific, application-based problems. For instance, Deeter & Smith (1998) present a genetic algorithm to find the most economic solution to a network design problem where edges fail with known probability. The overall reliability of the network is calculated using Monte-Carlo simulation. The fitness function used depends on the cost of the design and the reliability, allowing infeasible solutions (solutions that have a reliability below a given threshold) but penalising them accordingly.

2.5 Alternative approaches to network reliability

2.5.1 Network augmentation

Network augmentation refers to the category of problems that seek to add edges to an existing network in order to increase the level of reliability.

The first important article on network augmentation problems to achieve certain connectivity requirements was by Eswaran & Tarjan (1976), who introduced the problem of augmenting a graph to make it 2-edge connected, 2-node connected, or to make a digraph strongly connected. They also showed that for general edge costs these problems are $\mathcal{NP}$-hard.

The general k-edge-connectivity augmentation problem is presented in the review article by Frank (1992) where the problem of achieving a given level of edge or arc connectivity is discussed for both directed and undirected graphs, including polynomially solvable cases; this is followed by a later survey article (Frank (1994)) where the node-connectivity problem is also discussed.

A more specific version of network augmentation that allows parallel edges is the following: Given a undirected and unweighted graph G with edge connectivity λ, they solve the problem of finding the minimum number of edges to be added to the graph such that the augmented graph is $(\lambda+\delta)$-edge connected where $\delta \geq 1$ is a given integer. This problem was proven to be polynomially solvable by Watanabe & Nakamura (1987) and the fastest algorithm to date by Naor, Gusfield & Martel (1997) to solve it has a running time of $O(\delta^2 nm + \delta^3 n^2) + nF(G)$, where n, m and $F(G)$ is the number of vertices, edges and the time needed to perform one maximum flow in G. This can be simplified to $O(mn)$ when $\delta = 1$. For a general graph and if edge replication is not

allowed, the vertex and edge augmentation problems are $\mathcal{NP}$-hard.

The edge-biconnectivity augmentation problem is defined as the problem of finding the cheapest possible set of edges to add to a given network to make the network 2-edge connected. This problem is $\mathcal{NP}$-hard even if the edge costs are chosen from the set $\{1, 2\}$ only, as proven by Eswaran & Tarjan (1976), although they proposed a polynomial algorithm for the special case of a complete graph with equal edge costs. Frederickson & Ja'ja' (1981) showed that this augmentation problem is equivalent to augmenting a tree, obtained by shrinking the already 2-edge connected components. A heuristic procedure based on this reduction was presented, with a cost no more than twice the optimal cost.

Raidl & Ljubić (2002) describe a heuristic method based on evolutionary local search using specifically designed mutation and recombination operators to improve performance. Finally some other heuristic approximation algorithms have been described by Khuller & Vishkin (1994) and Garg, Santosh & Singla (1993).

2.5.2 Probabilistic methods

In this section a short overview of probabilistic definitions of reliability in the context of network design will be presented. There are approaches that use some type of probabilistic definition to measure reliability, which is usually related to an input of individual probabilities of failure for each element (vertex or edge) of the network. The network design problem is formulated as the problem of constructing a network at minimum cost such that some constraints are satisfied and the reliability is greater than a given value. These approaches play an important role in the design of certain types of networks, e.g., telecommunication.

An overview of probabilistic measures can be found in Colbourn (1987) or Ball, Colbourn & Scott Provan (1995) and a summary of those most commonly used is given below:

(i) *k-terminal* reliability,

(ii) *two-terminal* reliability (the same as (i) with $k = 2$),

(iii) *all-terminal* reliability (the same as (i) with $k = |V|$, the number of vertices),

(iv) *reachability* (the probability that every vertex is reachable from a given vertex),

(v) some directed variant of the above,

(vi) (s, K)-*terminal* reliability,

(vii) probability that the network has a topological reliability property, e.g., is two-edge connected or two-node connected.

(viii) probability that the network can meet a specified level of flow at each demand point.

In the probabilistic setup, a stochastic network model is assumed where each edge can be 'operating' or 'failed' and that a path is 'operating' if every edge on the path is. The k-terminal reliability for undirected networks is defined by:

$$r := \mathbb{P}\{\text{there exist operating paths between any pair of vertices in } K\} \tag{2.10}$$

where K is a set of 'target' vertices (with $k = |K|$) and s is the source vertex.

For the case of directed networks, k-terminal reliability is defined quite differently, namely:

$$r := \mathbb{P}\{\text{there exist operating paths from } s \text{ to each vertex in } K\} \tag{2.11}$$

where s is a given 'source' vertex. Some authors however (e.g., Ball (1986)) use the latter definition for undirected networks, but to prevent ambiguity and following the notation used by Colbourn (1987) definition (2.11) will be referred to as (s, K)-terminal reliability instead, where s is the source vertex and K the set of target vertices. The 2-terminal reliability for directed networks is often called s, t connectedness.

A similar analysis is performed by Ball (1986), who presents an overview of some of these measures including the computational complexity of the corresponding network design problems. The exact calculation of the above measures of reliability is $\mathcal{NP}$-hard for general networks.

When all edges have the same probability p of being operative the reliability can be expressed as a polynomial in p. The probability that a given subgraph F of the edges ($F \subseteq E$) is operative is $p^{|F|}(1-p)^{|E-F|}$, and therefore depends only on the number of edges it contains. If S_i is the number of operational subgraphs with i edges, then the probability of network operation is defined as the *reliability polynomial* and equals

$$\text{Rel}(p) = \sum_{i=0}^{N} S_i p^i (1-p)^{N-i}, \tag{2.12}$$

where N is the number of vertices of the network.

The reliability polynomial was introduced by Moore & Shannon (1956) in the context of electrical relays and has been used extensively in assessing network reliability in electrical and computer networks. The disadvantage of the reliability polynomial is that it can only compare the reliability of networks at a given value of p. Indeed, it is possible for the polynomials of two networks to 'cross', i.e., for one to be more reliable for one value of p, whereas the other may be more reliable for a different value (Kel'mans (1979)). Moreover as Myrvold, Cheung, Page & Perry (1991) proved, for any given number of vertices and edges, in certain cases there is no graph that is most reliable for all values of p.

The analysis is performed on networks where the failure probabilities of each arc are known. The solution methodologies concentrate mainly on an algorithm to determine bounds on the quantity r of expression (2.11), although some comparison to graph theoretical models is made. In a similar approach, Lucet, Manouvrier & Carlier (2000) present an algorithm for the problem of evaluating the same probability r in the case where all-terminal reliability is replaced by 2-edge reliability, that is, calculating the probability that the network is two-edge connected.

Probabilistic techniques have been widely used in engineering, e.g., power, telephone, computer and telecommunication networks, but the input data for the network components in these areas do not vary with the flow that passes through them, as it is the case in water networks. Some useful insight into reliability in other areas of engineering can be found in Altinkemer & Yu (1992), Gross & Saccoman (1998), Sherali, Ozbay & Subramanian (1998), Xue (1997) and Zemel (1982).

2.5.3 Application-based approaches

In this section we present some algorithms for reliable network design derived for particular types of networks. These are mainly heuristic procedures since application-based problems are often very complex to be solved exactly. We present an overview of the application-based approaches for the design of water distribution networks, an area where reliability is often included in the design requirements.

Water networks present the difficulty that, for a satisfactory solution, water must be provided at a certain pressure, hence connectivity to a source is not enough. The pressure loss equations are represented by the Hazen-Williams equations which are non-linear and are traditionally solved using nonlinear gradient methods (e.g., see Alperovits & Shamir (1977)). Goulter & Morgan (1985) were one of the first authors to consider reliability as one of the design requirements, describing the "design of looped water distribution

networks", where the requirement of a 'looped' network incorporates a certain level of reliability in the network. Their procedure consists of two algorithms: one to find a 'branched' network (i.e., a tree) and the other to extend it to a 'looped' network (i.e., a more reliable layout).

Kessler & Shamir (1991) present an algorithm for obtaining a local optimal solution, again taking into consideration the hydraulics, mentioning that their model constitutes an $\mathcal{NP}$-complete problem, and as such it is unlikely that an algorithm with a computational time bounded by a polynomial will be found. Ostfeld & Shamir (1996) use a different technique and try to find two spanning trees in the configuration whose distance is maximal. The distance between two trees is defined as the number of arcs that are contained in one tree but not in the other.

Ormsbee & Kessler (1990) present an implementation of a 2-node-connected network for the design or water networks. Since other hydraulic and component failure considerations are also taken into account, no algorithm for explicitly making a network 2-node-connected is presented (some pipes are arbitrarily added) and then an algorithm by Itai & Rodeh (1984) is used to find two node-disjoint spanning trees that are needed to ensure hydraulic redundancy. A very similar two-tree approach for the cases if 2-node and 2-edge connectivity is presented in Kessler, Ormsbee & Shamir (1990).

Fujiwara & Khang (1990) present a two-phase decomposition method for the design of water networks, where the first stage is to obtain a locally optimal solution using a nonlinear gradient programming algorithm and the second stage is to improve the solution by solving a concave program with linear constraints.

On the same lines, Loganathan, Greene & Ahn (1995) present a two phase heuristic suggesting a multistart local search technique (where the feasible region is saturated with randomly generated starting points, and then a local minimisation algorithm is used to identify the local optima) or a simulated annealing method to solve the concave program.

It is also wise to mention the importance of *simulation* methods as opposed to the *analytic* ones mentioned earlier. For example, consider the case of water distribution network design: in many approaches, a case-by-case study is performed with different sets of demands and network layouts to determine the network flows and the pressures. The performance of the network can then be determined via the ability to meet the demands over the complete set of cases. Alternatively, one can simulate using the underlying probability distributions for the failure of each component and repair times over a set of different scenarios. Wagner, Shamir & Marks (1988*b*) and also Goulter (1995) list about twenty different measures of reliability that can be used in simulation analysis in network

design problems.

Finally, Ostfeld & Shamir (1993) provide a good overview of the work done on the area of simulation methods for the design of reliable water distribution networks (until 1993) and some more recent developments are summarised in Khomsi (1994) and Goulter (1995).

Novel approaches to the subject also exist, for instance, a recent paper by Deeter & Smith (1998) uses a genetic algorithm to identify the topology of the arcs in the network. They assume that arcs fail with equal probability, possibly more than one at a time and arc repair is not considered. Many of these ideas are also present in the earlier work by Walters & Smith (1995) who concentrate on finding the optimal tree layout using genetic algorithms. Awumah, Goulter & Bhatt (1991) made an attempt to use an entropy approach defining the entropy of each vertex as the ability of the vertex to receive water when a link incident on it fails, and then trying to maximise the total entropy of the system.

An alternative approach is considered by Myung, Kim & Tcha (1999), who take into account both edge-connectivity constraints and some *link-survivability* constraints. Link survivability represents the proportion of loss incurred when an edge fails. For example, in the networks presented in figure 2.1, if the dashed edge breaks down (this is the worst case scenario in both networks), there is a 60% loss of flow in the first network but only a 40% loss in the second.[20] They define the *k-link-survivable* network as the minimum cost network design problem that satisfies the edge-connectivity constraints but also its survivability does not fall below a given percentage. They also describe a polynomial time to calculate the k-terminal-survivability and an MIP formulation for an exact approach, followed by a simple constructive heuristic. This problem is $\mathcal{NP}$-hard as it contains the minimum cost, two-edge connected network problem as a special case.

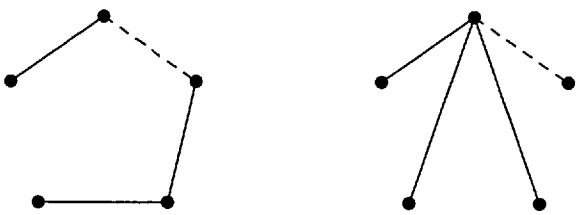


Figure 2.1: An example of survivability considerations

[20]The authors assume that there is a unit level of traffic between every pair of vertices (the article deals with the k-terminal problem in the context of communication networks).

2.6 The reliable network design problem considered in this thesis

In this section the main research problem, the *Reliable Network Design Problem (RNDP)* with flows, is described. We distinguish two versions of the problem: the *Uncapacitated* RNDP (URNDP) which is studied in chapters 3, 4 and 5, and the *Capacitated* RNDP (CRNDP) in which capacities restricting the flow through an edge are imposed. The CRNDP is discussed in chapter 6.

We consider a network design problem, where the underlying graph contains no parallel edges or loops. We consider this graph to be undirected, in the sense that if the an arc (i, j) is present, then so is the opposite arc (j, i).

We assign one vertex of the graph to be the source vertex, vertex 0, and the remaining are demand vertices. Associated with each demand vertex is an integer number d_i (also called the 'demand' of that vertex) representing the amount of flow that this vertex requires. Attached to each edge are two numbers, representing the fixed and variable costs associated to the use of that edge. The cost of using an edge is the sum of the fixed cost plus the variable cost multiplied with the flow that passes through the edge. The total cost of a network is the sum of the costs of all the edges it contains.

The flows that pass through the edges must be such that each customer receives exactly the amount of flow that it requires provided that the flow is conserved (no flow is lost in the process). Given that all the flow is supplied by the source vertex, we can consider this vertex as a vertex of negative demand $-\sum_{i \neq 0} d_i$.

We further impose reliability design requirements. The adopted approach to network reliability is the following: *We seek a network that can satisfy the flow constraints if any single edge is removed from the network.*

The optimisation objective is to minimise the cost of the solution network among all possibilities that satisfy the design requirements.

2.7 Test problems

An in depth study of any combinatorial optimisation problem requires the availability of a large number of test problems to which the mathematical techniques will be applied.

Unfortunately, most of the literature that does not use real data focuses more on the algorithmic developments (e.g., the best design of the procedure or the different

parameter values) and less on the method employed to generate the problems that were used to verify and compare the various algorithms. Even in the cases where test problem generation is discussed, the weight is, in most cases, on the way that the values of various parameters (costs, demands, etc) are generated and the structure of the underlying graph is barely mentioned. There is no doubt that these parameters are of great importance but the significance of the layout of the input graph should not be underestimated. For instance, many researchers use the euclidian distance as the cost of an edge where the vertices are randomly distributed on a square, others use costs that are uniformly distributed on an interval of the real line. Little mention is given on the criteria that determine how the edges of the network were selected and often the complete graph is used.

For the RNDP studied in this thesis, there are the following parameters to be decided:

- decide the layout of the underlying graph
- decide the values of the demands for each vertex
- decide the values of the linear and fixed costs
- decide the values of the capacities (for the capacitated problem)

The way that the test problems are generated is extremely important. An algorithm may appear to perform better or worse than another but this may just be because the particular instances that were used to compare them were too easy, too difficult, or just too 'strange'.

Some insight on the relative complexity of a test problem for the reliable network design problem can be obtained intuitively. For example, one should expect that if the costs corresponding to the flow are much larger that the fixed costs, then the solution will look very similar to a network flow problem with a few added edges added to ensure reliability. This will be a very easy problem to solve. On the other hand if the variable costs are very small, the solution will be similar to that obtained by solving the two-edge connected network design problem without flows and adding the cheapest flows to this solution. Although the latter problem is more difficult than the former, neither of them reflects the real complexity of the problem which is indeed greater for cases in between the above two extremes.

The layout of the input graph is also very important. If the graph is very sparse, then there are few possibilities for feasible solutions and the problem is solved more quickly. If the network is too big then an algorithm searching the solution space may move and

find very quickly the region that contains a probable solution, and this may also be an easy problem. With a network density in between the problem may be harder because the solution can be hidden 'deeper' and the algorithm in question may not be efficient in detecting it. Furthermore, for the RNDP it is necessary that the layout of the initial problem is two-edge connected otherwise no feasible solution will exist. An interesting question is therefore whether it is possible to know a priori the probability of a randomly generated test instance to be infeasible as a function of the density or the number of edges present.

No suitable research was found regarding these issues in the literature of exact solution methods where computational results are studied. In many articles which describe the generation of the test problems, no mention about how infeasible instances were handled or the number of such instances is presented, so it is likely to assume that the researchers just ignored those instances.

Given that many algorithms are tested using *randomly* generated test problems it is wise to see from a graph theoretical perspective, how random graphs are produced and their properties. The next section presents a short overview of the most important results in the area and, in particular, those that are related with the reliable network design problem.

2.7.1 Random graphs

This section is concerned with the problem of the random generation of the layout of the test instances.

Let $\mathcal{G}^N$ is the set of all possible graphs with N vertices. There are, in general, two ways to create a random graph (Bollobás (2001)):

(i) Each edge of the complete graph can be included in the graph with probability p.

(ii) Fix the number of edges E. Select one such graph randomly (so that every possible graph with E edges has equal probability of appearing).

Method (i) will generate graphs with roughly the same density, whereas method (ii) will generate graphs with the same number of edges. In the first case, setting $p = 1/2$ corresponds to the case where every pair of graphs of $\mathcal{G}^N$ is equiprobable.

A theorem by Pittel (1982) states that the two above methods are practically the same, provided that E is close to $pN(N-1)/2$ (subject to a few technical requirements). For

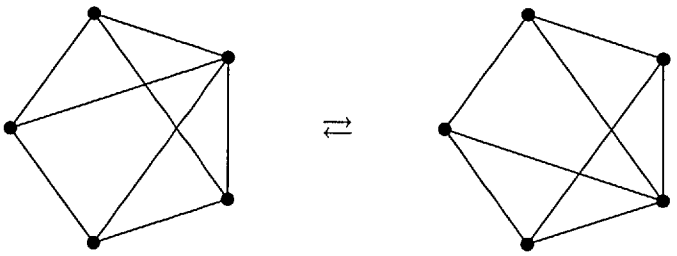


Figure 2.2: In unlabelled graphs the vertices are not distinguishable

the purposes of this thesis, we chose to generate the test instances with method (i), so that the networks produced will have roughly the same density.

Finally a note on random *unlabelled graphs*. These are graphs whose vertices are not considered to be distinguishable (figure 2.2). In this research the underlying graph of the network is always assumed to be labelled. This is because the networks used are also equipped with demands and costs which are generated independently of the layout and in a 'labelled' (vertex specific) fashion; thus the fact that two networks may have isomorphic underlying graphs has little effect on the properties of the two instances. Moreover, unlabelled graphs are known to possess some seemingly very strange properties. For example the probability that a random unlabelled graph with N vertices is connected is not a monotonic function of the number of edges E, in other words adding edges may actually *reduce* the probability that the random graph will be connected (Wright (1972)). Another interesting result is that if one considers all unlabelled graphs with N vertices, almost all of them are k-node connected. On the other hand, among all unlabelled graphs with E edges (without an isolated vertex) almost all are connected but almost none of them is two-connected. (An overview of the properties of unlabelled graphs can be found in Bollobás (2001).)

2.7.2 Probability of obtaining an infeasible instance

For the RNDP, the density of the network affects the complexity of the problem. If a network is very sparse, then only few feasible solutions exist and an exact algorithm to solve the RNDP is likely to terminate very quickly, and the opposite is true when the density increases. However, if the density of the input network is very small then no feasible solution may exist.

In this section we present the results of an experimentation by generating random graphs with different densities and investigating whether an instance is feasible or not. For simple connectivity (as opposed to two-edge connectivity) several results are known

to this effect. This was performed to find the regions of 'feasibility' in order not to generate infeasible or very sparse test problems.

Tables 2.4 and 2.5 show the results of an experiment where random graph layouts were generated. These instances were studied for feasibility in the context of the RNDP. The instances were generated using both the probability p of an edge being present and with a fixed number of edges E, for different number of vertices. Fifty instances were generated for each case. The shaded area in each entry represents the percentage of the instances where a feasible solution of the RNDP exists.

As this table suggests, more than 9000 instances were tested but this procedure was very fast because each problem was not required to be solved to optimality, so once a feasible solution was identified then that problem could be classified as feasible.[21] The values of the demands and costs were irrelevant for this experimentation and they were all set to 1.

[21]Moreover, CPLEX was used for this experimentation (see also section 2.7.6 for more details) which identified quickly infeasible solutions, most of the times by detecting conflicting constraints during preprocessing, without the need to solve even a single LP.

	Network density (p)														
Vertices	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75
10	○	○	○	◔	◔	◔	◑	◑	◕	◕	◕	●	●	●	●
15	○	○	○	◔	◔	◑	◕	◕	●	●	●	●	●	●	●
20	○	○	◔	◔	◑	◕	◕	●	●	●	●	●	●	●	●
25	○	○	◔	◑	◕	◕	◕	●	●	●	●	●	●	●	●
30	○	○	◔	◕	●	●	●	●	●	●	●	●	●	●	●
35	○	◔	◔	◕	●	●	●	●	●	●	●	●	●	●	●
40	○	◔	◑	◕	●	●	●	●	●	●	●	●	●	●	●

Table 2.4: The shaded area represents the proportion of feasible problems among randomly generated instances

	Number of edges (E)																
Vertices	10	20	40	60	80	100	125	150	175	200	250	300	350	400	500	600	700
10	○	◕	●	–	–	–	–	–	–	–	–	–	–	–	–	–	–
15	○	○	◕	●	●	●	–	–	–	–	–	–	–	–	–	–	–
20	○	○	◔	◕	●	●	●	●	●	–	–	–	–	–	–	–	–
25	○	○	◔	◑	◕	◕	●	●	●	●	●	●	–	–	–	–	–
30	○	○	○	◔	◑	◕	●	●	●	●	●	●	●	●	–	–	–
35	○	○	○	○	◔	◑	◕	●	●	●	●	●	●	●	●	–	–
40	○	○	○	○	◔	◔	◕	◕	●	●	●	●	●	●	●	●	●

Table 2.5: The proportion of feasible problems among randomly generated instances with a fixed number of edges

2.7.3 Choice of demands and costs

Test problems or methods for generating fixed and variable costs for test instances for the URNDP and their relative complexity were not found in the literature. However, given the similarities of the problem with the fixed charge problem when the reliability requirements are dropped, test instances from the fixed charge problem can be used. The most complete numerical experimentation for the FCP can be found in Hochbaum & Segev (1989). The authors consider the fixed charge problem and develop techniques for generating upper and lower bounds on the optimal solution value. The methods they use for generated the test instances are useful for the URNDP but the techniques they use for the bounds cannot be applied because some of the inequalities employed (e.g., cycle prevention) are not valid. The test problems had 25 or 50 vertices, and an important parameter was the ratio of the fixed cost over the variable (linear) cost, $\beta = b_{ij}/c_{ij}$ using the URNDP notation. The networks have one source vertex and the demands of the other vertices are chosen uniformly from the set $\{1, 2, \ldots, M\}$. The dependence of the complexity of the instances is investigated as a function of these two parameters. The costs were generated as follows: the linear costs c_{ij} were either selected from a uniform distribution on $[0, 1]$ or they were taken to be the distances of N points uniformly distributed on the unit square. Then for each edge the value of the fixed cost is chosen so that the ratio b_{ij}/c_{ij} is equal to the given value of β.

The authors findings verify that the ratio

$$\gamma = \frac{M}{2\beta} \tag{2.13}$$

is important as far as complexity is concerned. The reason behind this is that if either cost type (fixed or linear) is much smaller than the other then the instance will be easy because for the two extreme cases the fixed charge problem reduces to polynomially solvable cases (shortest path or optimal branching). The value of γ represents roughly the ratio of the variable over fixed cost costs. (this can be seen from the definition of β and from the fact that by construction the average demand is $M/2$.) For the fixed charge problem the region of γ where the most difficult instances were encountered was the interval $\gamma \in [0.05, 0.1]$, having tested values in the range $[0.0013, 5]$.

It is reasonable to assume that such dependence on the two cost types will apply to the URNDP as well. The design requirements are stronger because reliability constraints are imposed but the structure of the cost function remains the same. An important difference is, however, that the URNDP does not become polynomial when either cost type dominates the other; in particular, if the fixed costs are ignored the problem is

Problem	*Vertices*	γ	*Instance (1 to 5)*
g1n12g001i1	12	0.01	1
g1n12g001i2	12	0.01	2
⋮			

Table 2.6: The naming convention used for the test instances

polynomially solvable, whereas if the linear costs disappear the problem is $\mathcal{NP}$-hard (see section 3.2 on page 73) although the latter is significantly easier than the full URNDP. Computational experimentation with the URNDP verifies this observation.

2.7.4 Generation of uncapacitated test problems

In this section the way that the test instances for the uncapacitated RNDP were generated is described. First the number of vertices N is specified. Networks with 12, 15, 20, 25, 30, 40 and 50 vertices were generated. The layout of the network is determined using a fixed probability for each edge being present in the network. The value of $p = 0.75$ was chosen as it is large enough to be away from the 'infeasible' region (table 2.4) which could have made the problem appear easier due to the small number of feasible configurations that exist for sparse networks. This gives an average density (the ratio of edges present in the graph versus the maximum number of edges possible) of 75%.

For the generation of the demands and costs a similar method to Hochbaum & Segev (1989) was used. The demands were uniformly chosen from the interval $\{1, 2, \ldots, 10\}$. The value of γ was taken to be 0.01,0.005, 0.1, 1 or 2. The linear costs are chosen uniformly in the region $\{1, 2, \ldots, 10\}$ and the fixed costs were chosen to make the value of γ the given one. A total of five instances were generated for each combination of vertex size and γ.

The test problems were named according to the number of vertices, the value of γ and the instance number (1–5), using the notation

g1n<vertices>g<gamma>i<instance>,

therefore, for example, the test instance 'g1n12g005i3' refers to the third instance with 12 vertices and $\gamma = 0.05$ (table 2.6).

2.7.5 Generation of capacitated test problems

Choosing capacities is a very subtle business. Imposing weak capacities means that the capacity restrictions are never attained and the problem is essentially (in a complexity sense) equivalent to the uncapacitated version of the problem. On the other hand, if too strict capacities are imposed, this will make the problem infeasible or too easy since very few feasible possibilities exist.

The fixed and variable costs and the demands were generated as in the uncapacitated case. The procedure used to generate the capacities is the following: we note that, on average, there are $0.75 \times (N-1)$ arcs leaving the source vertex, where N is the number of vertices. The total demand that they must supply is, on average, $5.5 \times (N-1)$, since the average demand of every vertex equals 5.5. We set the average capacity to be the ratio of the total average demand over *half* the (average) number of edges leaving the source. This imposes reasonable capacity restrictions on both the edges leaving the source and the remaining edges of the network. So the average capacity, $\bar{U}$, equals

$$\bar{U} = \frac{5.5 \times 2 \times (N-1)}{0.75 \times (N-1)} = 14.67. \tag{2.14}$$

We generate the values of the capacities uniformly on the interval $[\bar{U}/2, 2\bar{U}]$, rounded to the nearest integer, i.e, $[7, 29]$. This method has the advantage that it ensures uniformity among the different instances with a different number vertices, making the comparison of the computational results across different network sizes easier. The naming convention used for the capacitated instances is the same as the one used for the uncapacitated ones, but the name is prefixed by 'g2' (group 2) instead.

Appendix A contains the layouts, demands, costs and capacities for two sample test instances on 12 vertices that are used as examples in the following chapters.

2.7.6 Computational implementation details

The mathematical formulations presented in this thesis were implemented in a computer program using C++ and all the results were obtained on a *Pentium III* PC with a clock speed of 600MHz.

The LP solver used was CPLEX 7.5 (see ILOG™ S.A. (2001) in the references) and all reported times are CPU times in seconds unless otherwise mentioned.

2.8 Summary

In this chapter an overview of the literature regarding network design problems in combinatorial optimisation was presented. We focused on problems containing flows and both fixed and variable costs, followed by an overview of the implementation of reliability into the design of a network and the possible mathematical formulations of such problems. Problem applications and solution techniques were discussed. The chapter concluded with a description of the *Reliable Network Design Problem with flows*, which is the base research problem studied in the thesis. A discussion on the generation of test instances suitable for this problem was also presented. In the next chapter a *mathematical formulation* for the uncapacitated Reliable Network Design Problem with flows will be presented, which will be solved *to optimality* using a Branch and Cut algorithm.

Sometimes, formulating a problem is more difficult than solving it.

Albert Einstein (1879–1955)

Chapter 3

A mixed integer programming approach for the uncapacitated RNDP with flows

3.1 Introduction

This chapter deals with the uncapacitated reliable network design problem (URNDP) with flows that was described in the previous chapter. The chapter begins with a Mixed Integer Programming (MIP) formulation which is used to model this problem, and the formulation is solved to optimality using a Branch and Cut procedure. For this purpose it is necessary to generate classes of valid inequalities that are used to improve the lower bounds obtained through various LP relaxations of the original formulation.

This chapter describes several categories of valid inequalities that can be derived from the MIP formulation producing the best lower bound on the optimal solution of the problem obtained at the root node of a Branch and Cut algorithm. The complete Branch and Cut algorithm will be described in chapter 4.

3.2 A mixed integer programming formulation

The *Uncapacitated Reliable Network Design Problem with flows* is defined by the MIP **M7**. The interpretation of the variables used in this formulation is shown in table 3.1.

Symbol	*Interpretation*
V':	$V\backslash\{0\}$, the set of vertices excluding the source
d_i:	(integer) demand of vertex i, where $i \in V'$
b_{ij}:	fixed cost, $(i,j) \in E$, $i < j$
c_{ij}:	linear cost per unit flow, $(i,j) \in E$
Q:	a large integer representing the maximum flow through any arc
ξ_{ij}	are (unknown) binary variables, defined for $(i,j) \in E$, $i < j$, representing whether a given edge is present in the layout or not
x_{ij}	are the (unknown) flow variables, defined for $(i,j) \in E$.

Table 3.1: Notation used in model **M7** of the URNDP with flows

MODEL M7

$$\textbf{min} \quad \sum_{\substack{(i,j)\in E \\ i<j}} (b_{ij}\xi_{ij} + c_{ij}x_{ij} + c_{ji}x_{ji}) \tag{M7.1}$$

subject to

$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = d_i \qquad \text{for all } i \in V' \tag{M7.2}$$

$$\sum_{j:(0,j)\in E} x_{0j} = \sum_{i\in V'} d_i \tag{M7.3}$$

$$\sum_{\substack{i\in S, j\in S' \\ (i,j)\in E \\ i<j}} \xi_{ij} + \sum_{\substack{i\in S, j\in S' \\ (i,j)\in E \\ i>j}} \xi_{ji} \geq 2 \qquad \text{for all } 0 \in S \subseteq V,\ S' = V\backslash S \tag{M7.4}$$

$$x_{ij} \leq Q\xi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M7.5}$$

$$x_{ji} \leq Q\xi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M7.6}$$

$$x_{ij} \geq 0 \qquad \text{for all } (i,j) \in E \tag{M7.7}$$

$$\xi_{ij} \in \{0,1\} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M7.8}$$

THE UNCAPACITATED RELIABLE NETWORK DESIGN PROBLEM WITH FLOWS

The *flow balance* constraints (**M7**.2) guarantee that there is no loss of flow in the network, i.e., the total amount of flow that enters a customer vertex is equal to the demand of that vertex plus the amount of flow that leaves the vertex. The same applies to the flows that leave the source vertex (constraint (**M7**.3)).

The *reliability* constraints (**M7**.4) guarantee that the network will have the required reliability, that is, that the network should be able to sustain a single edge failure. This requirement, for the uncapacitated case, translates to the layout of the solution being two-edge connected (theorem 2) which means that every cut in the network will contain at least two edges. (See figure 3.1).

The *trivial* constraints (**M7**.5) and (**M7**.6) simply state that there can be no flow through an edge if that edge does not exist. These constraints must be included in the formulation, otherwise infeasible solutions may be produced. The associated value of Q can be any large integer, large enough so as not to restrict the flow through any arc. The value of Q was chosen to be the sum of all the customer demands, i.e.,

$$Q = \sum_{i \in V'} d_i \tag{3.1}$$

since in any optimal solution no flow can be greater than the total demand.

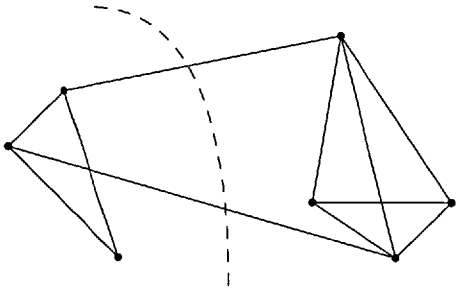


Figure 3.1: In a two-edge connected network every cut must contain at least two edges

In the above formulation, binary variables ξ are defined one for each edge, since an edge can either be included in the design or not, whereas the flow variables x are defined one for each arc (i.e., two per edge) because if an edge exists then flow can travel in either direction. In an optimal solution however, it is not possible to have both $x_{ij} > 0$ and $x_{ji} > 0$ since, in such case, replacing the two flows with the net flow

$$x_{ij} = \max\{x_{ij} - x_{ji},\ 0\} \tag{3.2}$$

$$x_{ji} = \max\{x_{ji} - x_{ij},\ 0\} \tag{3.3}$$

will result in another feasible solution whose total cost will be reduced by

$$|x_{ij} - x_{ji}| \times c_{ij}.$$

There are two obvious observations regarding this formulation. First of all, the above problem is $\mathcal{NP}$-hard, since it contains both the 2-edge connected network design problem (when $d_i = 0$ for all i) and the fixed charge problem (no reliability) as special cases, both of which are $\mathcal{NP}$-hard as discussed in chapter 2. On the other hand, if the fixed costs are all set to zero then the URNDP does become polynomially solvable, but this situation is hardly of any interest as it is no longer a network design problem but just a network flow calculation problem (one can always choose to include all the available edges in the layout as this does not affect the cost).

The second observation is the exponential number of reliability constraints **M7**.4 and the resulting increased complexity when trying to solve large problems. The number of reliability constraints is $2^{N-1} - 1$, where $N = |V|$ is the number of vertices in the network. With such a formulation, even attempting to solve the LP relaxation of **M7** is hard for large networks. To overcome this computational complexity, a relaxation procedure was devised where most of the reliability constraints are removed from the formulation and then added as necessary. This procedure will be described in section 3.4.

3.3 An example optimal solution

As an example, consider the input network with 6 vertices and 10 edges of figure 3.2. The edges of the graph represent the locations of potential links and the pair (c, b) on

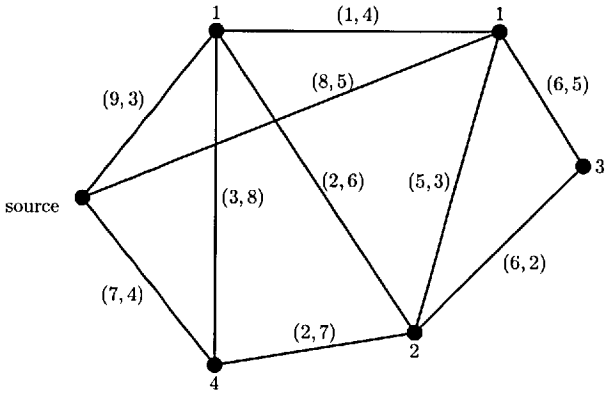


Figure 3.2: An example URNDP with flows: input network

every edge represents the variable and fixed costs of that edge, respectively. The demand

of each vertex, except from the source vertex, is shown by the number adjacent to each vertex. The optimal solution network of this example is shown in figure 3.3, with an optimal solution value of 135.

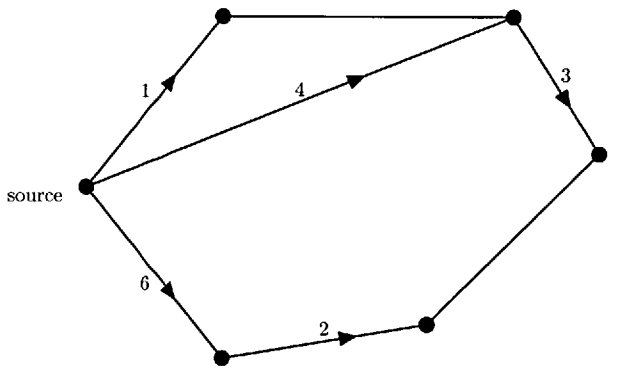


Figure 3.3: An example URNDP with flows: optimum network

As a measure of complexity, we present all the possible solution layouts that correspond to the input graph. Since the input graph has ten edges, there are $2^{10} - 1 = 1023$ possible sub-graphs (excluding the empty graph with no edges) but actually only forty of them are two-edge connected. The complete set of all forty two-edge connected graphs for the example problem is shown in figure 3.4. Underneath each layout, the total cost is displayed; this cost is calculated by adding the fixed cost component of all the edges included in the network design and the cost associated with the minimum cost flow problem on this layout. We can see that for this example there are actually *two* configurations that attain the minimum cost of 135 (multiple optima).

The following observations can be made regarding the structure of the optimum network layout:

(i) The optimal network will consist of some edges that carry a positive amount of flow (the *flow edges*) which supply the customer vertices. Given the linear cost structure of the objective function, these edges will form a tree structure just as in the case of the fixed charge problem.

(ii) The remaining edges in the layout (the *reliability edges*) have zero flow and are needed to ensure that the reliability design requirements are satisfied, since a tree layout is never two-edge connected.

The reliability constraints of the URNDP are similar to the subtour elimination constraints of the Travelling Salesman Problem (TSP), the difference being that in the TSP

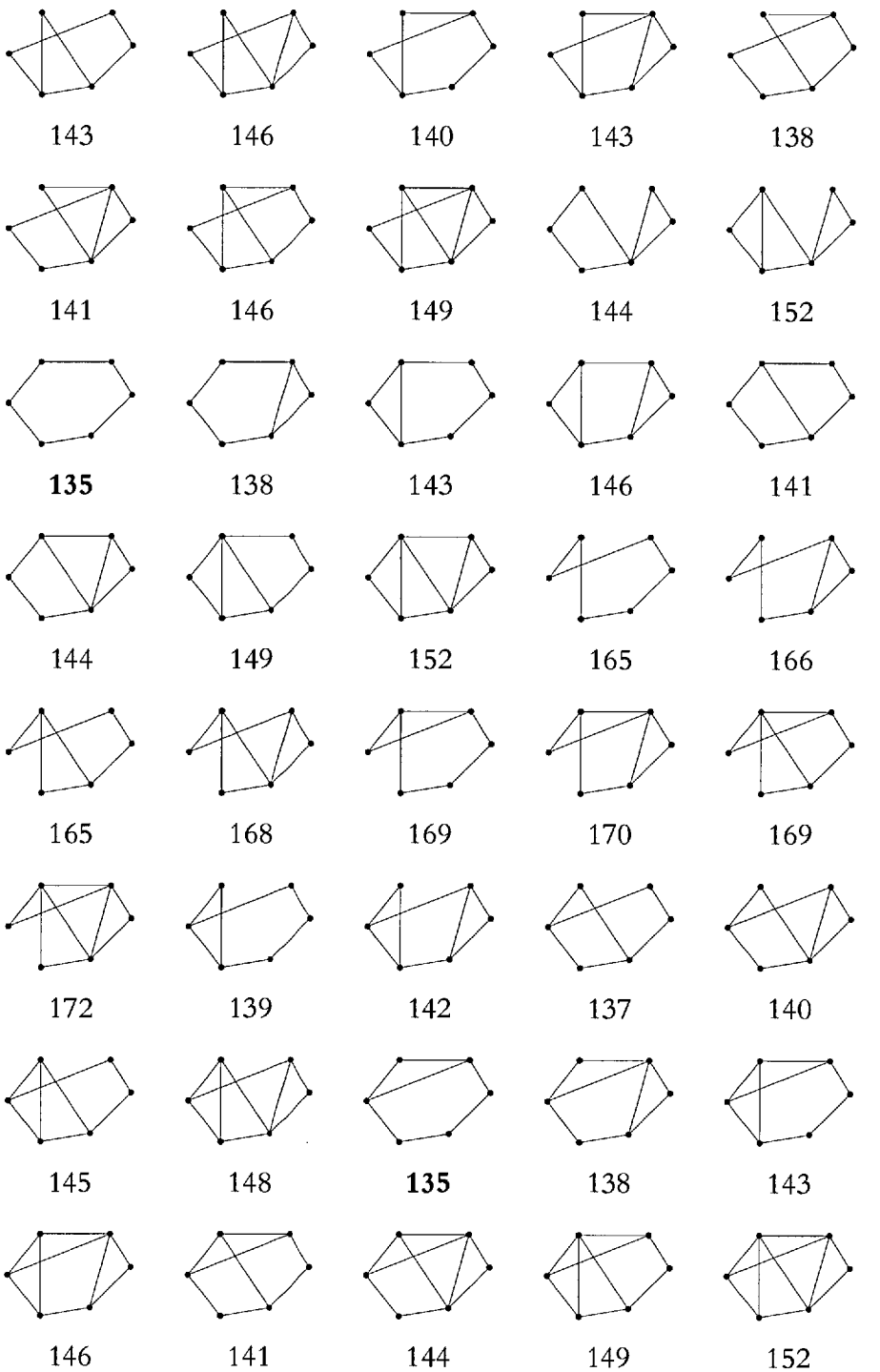


Figure 3.4: An example URNDP with flows: all possible reliable configurations

the "≥ 2" constraints are replaced by "$= 2$". This means that the set of feasible layouts for the URNDP is larger than that of the TSP; in particular, there exist layouts that have the two-edge connectivity property but are not feasible for the TSP. One such layout is the Petersen graph shown in figure 3.5 which is two-edge connected but contains no tour (and hence no feasible TSP solution).

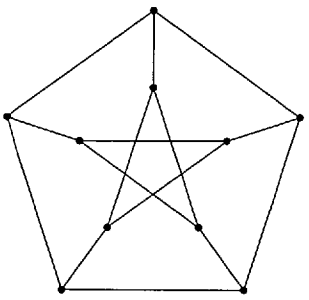


Figure 3.5: A 2-edge connected graph with no tour

3.4 The initial LP relaxation

This section presents the initial linear programming (LP) relaxation of model **M7** that serves as the starting point towards obtaining a lower bound to the URNDP with flows. We begin by relaxing the integrality constraints **M7**.8 to their continuous equivalent. Moreover, the reliability constraints **M7**.4 are replaced with a subset of them consisting of the *degree constraints*, which are those corresponding to $|S| = 1$. The degree constraints require that every vertex should have a degree of at least two.

The complete LP relaxation is presented in model **M8** below.

MODEL M8

$$\textbf{min} \quad \sum_{\substack{(i,j)\in E \\ i<j}} (b_{ij}\xi_{ij} + c_{ij}x_{ij} + c_{ji}x_{ji}) \tag{M8.1}$$

subject to

$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = d_i \qquad \text{for all } i \in V' \tag{M8.2}$$

$$\sum_{j:(0,j)\in E} x_{0j} = \sum_{i\in V'} d_i \tag{M8.3}$$

$$\sum_{j:(j,i)\in E} \xi_{ji} \geq 2 \qquad \text{for all } i \in V \tag{M8.4}$$

$$x_{ij} \leq Q\xi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M8.5}$$

$$x_{ji} \leq Q\xi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M8.6}$$

$$x_{ij} \geq 0 \qquad \text{for all } (i,j) \in E \tag{M8.7}$$

$$0 \leq \xi_{ij} \leq 1 \qquad \text{for all } (i,j) \in E,\ i < j \tag{M8.8}$$

THE INITIAL RELAXATION OF THE URNDP

We note that replacing the reliability constraints **M7**.4 in model **M7** with the degree constraints **M8**.4 does not guarantee two-edge connectivity; for instance, the graph shown in figure 3.6 has all vertices of degree at least two but, nonetheless, it is not two-edge connected.

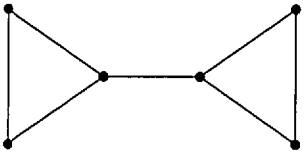


Figure 3.6: A *non* 2-edge connected graph with all vertices of degree 2 or more

Having removed some of the constraints from the formulation, a procedure is necessary to identify whether a given solution violates any of the constraints that have been removed. The procedure presented in the next section will be used to identify violated reliability constraints.

3.4.1 An exact separation algorithm for the reliability constraints

We consider the problem of deciding whether a given solution $\bar{\xi}$, obtained by considering a subset of the reliability constraints, satisfies all the reliability constraints **M7**.4 or not. If not, then any violated constraints must be identified because the solution $\bar{\xi}$ cannot be feasible and therefore these constraints have to be added to the formulation. The procedure of identifying violated constraints is referred to as the *separation problem of the reliability constraints.*

We present an algorithm, with a polynomial running time that can solve the separation problem. The idea is to construct a graph consisting of the same vertex and edge set as the original network, where every edge (i, j) is given a weight equal to $\bar{\xi}_{ij} = 1$. We then find the *global minimum cut* of this graph, i.e., a set $T \subset V$ such as the sum of the weights of all edges with one end-vertex in T and the other in $V \backslash T$ has a minimum value among all possible choices of T. If this value is less that 2, this means that there exist a violated reliability constraint **M7**.4 where the set S is $S = T$ (if $0 \in T$) or $S = V \backslash T$ (if $0 \notin T$). If, on the other hand, the minimum cut has a value of 2 or more, then this means that all cuts have a value greater than or equal to 2 and hence the solution is two-edge connected. In this case the current solution vector $\bar{\xi}$ satisfies all the reliability constraints **M7**.4.

The separation algorithm for the reliability constraints is summarised below:

1. Consider the graph with edge weights given by the values of the binary variables, $\bar{\xi}_{ij}$.
2. Find the global minimum cut (GMC) of this graph.
3. If the value of the GMC is *two or more* stop. No violated reliability constraints **M7**.4 exist.
4. Otherwise, a violated reliability constraint is obtained, by setting the sets S and S' of **M7**.4 to be the two shores separated by the GMC.

We note that there is no need for the binary variables $\bar{\xi}$ to have integer values for the separation algorithm to be used. The separation procedure can identify such violated reliability constraints even when the values of $\bar{\xi}$ are fractional.

3.4.2 Minimum cuts and the Gomory-Hu tree

There are many algorithms to find global minimum cuts. The oldest one is due to Gomory and Hu (Gomory & Hu (1961)) and has a polynomial running time and calculates the *all-pairs minimum cut*, i.e., the minimum cut separating *any* pair of vertices. The global minimum cut can be easily obtained from the minimum value among all the vertex pairs. Faster algorithms for the global minimum cut exist, for instance Nagamochi & Ibaraki (1992), Nagamochi, Ono & Ibaraki (1994), Karger & Stein (1993) or the improvements to the Gomory-Hu algorithm by Padberg & Rinaldi (1990). (See also Chekuri, Goldberg, Karger, Levine & Stein (1996) for a numerical comparison of the different GMC algorithms.) For the implementation used in this thesis, the Gomory-Hu algorithm is used (and in particular the faster implementation by Gusfield (1990)). This is because *(i)* the other algorithms are only slightly faster and much more complex to implement and *(ii)* they only produce the minimum global cut. On the other hand, the Gomory-Hu procedure, by calculating the all-pairs minimum cuts, provides us with information that allows us to generate the *partition inequalities* at the same time. Partition inequalities are a class of valid inequalities for the URNDP with flows and are generalisations of the reliability constraints and can also be obtained using the Gomory-Hu algorithm. Since they are often stronger than the reliability constraints, we choose to add partition cuts, whenever possible, instead of reliability constraints. This issue and the partition inequalities are discussed in section 3.5.

As mentioned earlier, the Gomory-Hu algorithm is a polynomial algorithm to obtain the minimum cut separating any pair of vertices of a graph. The input to the algorithm is an undirected graph with positive edge weights, and the output is a tree, the *Gomory-Hu tree* (G-H tree) having the same number of vertices as the original graph. An example is shown in figure 3.7. The numbers on the input graph are the edge weights, and the output shows the G-H tree with the associated edge weights. These weights on the output tree have the following property: The minimum cut separating two vertices i, j of the original network is equal to the minimum of the weights of the G-H tree along the (unique) path joining i and j in the G-H tree.

Gomory and Hu also proved that every network has a G-H tree. This also implies that in every network with N vertices there are at most $N - 1$ different possible values for any minimum cut separating two vertices.

It should be also mentioned that the Gomory-Hu tree is not necessarily a sub-graph

of the input graph.[22]

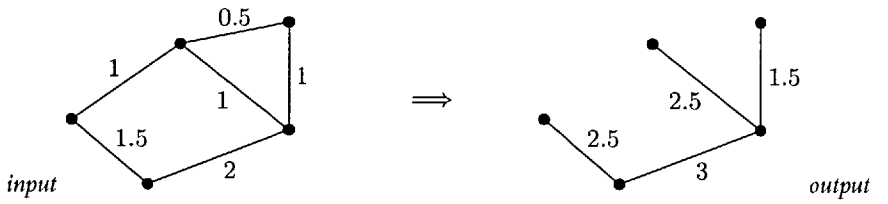


Figure 3.7: An example of the Gomory-Hu procedure

It is interesting to note that in contrast to the minimum cut problem described above that can be solved in polynomial time, the problem of finding the *maximum cut* of a graph is proven to be $\mathcal{NP}$-hard, even in the case of positive weights only (see Korte & Vyger (2000)), although some polynomial solvable special cases do exist, e.g., for planar graphs (Hadlock (1975)). This is the reason why the Gomory-Hu and the other polynomial minimum cut algorithms can *only* work with the edge weights being positive.

3.4.3 Computational results

Table 3.2 presents the lower bounds obtained for the initial LP relaxation **M8** using the test instances described in section 2.7. The notation used in the table is the following:

- LB_0 Lower bound obtained after adding the flow balance, trivial and degree inequalities (**M8**).
- z_{opt} The cost of the optimal solution to the URNDP or the cost of the best feasible solution found at the end of the Branch and Cut algorithm.
- $\%LB_0$ Percentage ratio LB_0/z_{opt}.

As expected, the initial lower bounds are of poor quality and further improvements are required to reduce the values of the percentage errors. This can be achieved by adding cuts to the LP relaxation, and classes of cuts that are suitable for this problem are presented in the remaining sections of this chapter.

3.5 Partition inequalities

Partition inequalities are a generalisation of the reliability constraints **M7**.4. The partition inequalities are derived from the fact that in a feasible solution, every cut must contain at

[22]The standard example in the literature is $K_{3,3}$, the complete bipartite graph with 3 vertices in each shore. If all edges have the same weight w, the G-H tree is a star-like tree consisting of one vertex joined to the remaining five with a weight of $3w$.

least two edges. These inequalities have been studied in Grötschel & Monma (1990) and later in Grötschel, Monma & Stoer (1995*b*) and Grötschel, Monma & Stoer (1995*a*) in the context of two-edge connected networks without flows.

We consider a network with a vertex set V. If V is divided (partitioned) into p subsets $W_1, W_2, \ldots, W_p$, then the following inequality, called the *partition inequality*, is valid:

$$\xi(\delta(W_1, W_2, \ldots, W_p)) := \frac{1}{2} \sum_{i=1}^{p} \xi(\delta(W_i)) \geq p. \tag{3.4}$$

An example is shown in figure 3.8, where the vertex set is divided into three partitions, and for that case the partition inequalities translate to the sum of the cut values defined by the partitions to be greater than 6.

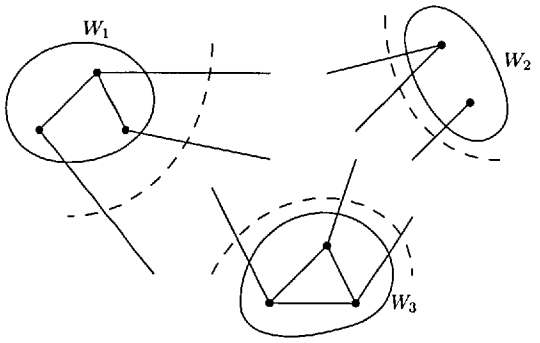


Figure 3.8: Partition inequalities

The reliability constraints **M7**.4 are a special case of the partition inequalities, corres-

Problem	*Vertices*	*Edges*	γ	z_{opt}	LB_0	% LB_0
g1n12g001i5	12	47	0.01	17434	17282.0	99.12
g1n12g01i2	12	55	0.1	2177	2084.0	95.73
g1n12g1i3	12	49	1	370	346.9	93.77
g1n15g01i4	15	85	0.1	1475	1365.0	92.55
g1n20g01i3	20	143	0.1	2582	2394.9	92.75
g1n25g01i4	25	230	0.1	2586	2370.4	91.66
g1n30g2i2	30	336	2	668	635.2	95.09
g1n40g005i5	40	575	0.05	6193	5572.1	89.97
g1n50g1i2	50	955	1	1179	1067.2	90.52
g1n50g01i1	50	907	0.1	3834	3260.0	85.03
Average						*92.62*

Table 3.2: Root node lower bounds LB_0 using the initial LP relaxation (**M8**)

ponding to the case where V is divided into two partitions ($p = 2$). Partition inequalities however, are generally stronger than reliability constraints. (See example of section 3.5.2.) For this reason, whenever a violated reliability constraint is found, we search for a stronger partition inequality. If a violated partition inequality (with $p \geq 3$) *is* found then the latter inequality is added, otherwise the reliability constraint is used instead.

The separation problem for partition inequalities is proven to be $\mathcal{NP}$-hard (e.g., see Grötschel, Monma & Stoer (1992*b*)). However, an easy heuristic procedure to test for violated partition inequalities in polynomial time can be derived using a simple modification of the Gomory-Hu algorithm. This procedure is described in the following section.

3.5.1 Separation heuristic

The method used to generate violated partition inequalities is similar to the one used by Grötschel, Monma & Stoer (1992*a*) and Grötschel, Monma & Stoer (1995*b*). We use the values of the binary variables $\bar{\xi}$ as weights for the Gomory-Hu algorithm, just as in the separation algorithm for reliability constraints (section 3.4.1). After that, we apply the Gomory-Hu algorithm to calculate the G-H tree and the associated edge values. If the minimum edge value of the tree is ≥ 2 then no violated partition inequalities are generated and the procedure stops. (A value of ≥ 2 also shows that all reliability constraints **M7**.4 are satisfied.) If, on the other hand, there are $p - 1$ edges with values < 2 (where $p \geq 2$) then we remove these edges from the G-H tree, which is then divided into p disjoint components. These components define the vertices of the p subsets of the partition inequalities (3.4).

The separation heuristic for the partition inequalities is summarised below:

1. Consider the graph with edge weights given by the values of the binary variables, $\bar{\xi}_{ij}$.
2. Find the Gomory-Hu tree of this graph.
3. If all the tree edge values are *two or more* (≥ 2) stop. No violated partition inequalities (3.4) can be found.
4. Otherwise, remove the tree edges with values *less than two* (< 2) to obtain the required partition of the vertex set.

The above process is a heuristic separation procedure because it is possible for violated partition inequalities of the form (3.4) to exist, but they may not be detected with this procedure. The procedure however, as described in section 3.4.1 *always* detects violated reliability constraints (which are a subset of the partition inequalities) if they exist. This is sufficient to guarantee that the final optimal solution will always be feasible.

In summary, the possible outcomes using the Gomory-Hu approach to detect violated reliability constraints and partition inequalities given a (possibly fractional) solution $\bar{\xi}$ are:

- The algorithm detects a violated reliability constraint. In this case:
  - ⋆ *either:* A violated partition inequality ($p \geq 3$) is also found (so we add this to the LP instead)
  - ⋆ *or:* No violated partition inequality ($p \geq 3$) is found (and hence we add the reliability constraint to the LP)
- The algorithm reports that no violated reliability constraints exist. This does not exclude the possibility that violated partition inequalities may actually exist.

3.5.2 A numerical example

Consider the example shown in figure 3.9. In part (i) of this figure a LP solution is shown where the edges drawn have values $\bar{\xi} = 1$. The global minimum cut for this graph has a value of 1, and therefore a violated reliability constraint exists (shown with a dashed line).

If this constraint is added the new resulting solution will look like the one shown in part (ii) of the figure, where the dashed edges correspond to edges with value $\bar{\xi} = 0.5$. This solution satisfies the newly added reliability constraint and also has the same cost as the first solution (assuming zero demands and that all edges have the same fixed cost component).

If instead of the reliability constraint, the partition inequality defined by the three sets (separated by dotted lines in part (i)) is added, then the induced solution, shown in part (iii) of the figure, is actually a feasible solution to the problem. Note also that the degree constraints are always satisfied by all solutions shown in this figure.

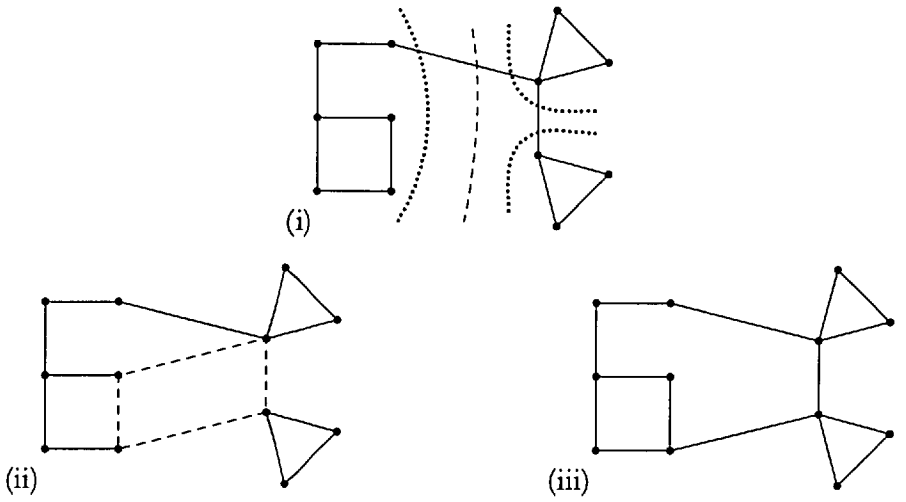


Figure 3.9: Partition inequalities versus reliability constraints

3.5.3 Generation of partition inequalities

Computational experimentation showed that few partition inequalities (and violated reliability constraints) are detected and they usually do not produce a significant improvement to the lower bound, at least compared to other classes of valid inequalities.

They are, nonetheless, an essential part in the solution of the URNDP with flows, as they prevent infeasible solutions (solutions that do not satisfy all the reliability constraints **M7**.4) from being produced. Indeed, most of the partition inequalities are generated not in the root node stage but during the branch and cut algorithm when an integer LP feasible solution is found and the separation heuristic is called upon to decide whether the solution is actually feasible with respect to the reliability constraints or not. These issues are discussed in the branch and cut implementation of chapter 4. At the root node, computational experimentation showed that it is best to generate partition inequalities after the maximum flow inequalities are identified since, in this case, a greater number of violated inequalities is often found.

3.6 The maximum flow inequalities

In this section a class of valid inequalities for the URNDP with flows is developed, which relates both sets of flow and design variables. These inequalities will be referred to as the *maximum flow inequalities*, and they are described below.

Let $V' = V \backslash \{0\}$ be the set of customer vertices and let $\emptyset \neq S \subseteq V'$. We consider any two vertices $i \notin S$ and $j \in S$ (see figure 3.10). From the flow balance equations

(**M7**.2) we must have

$$x_{ij} \leq \sum_{k \in S} d_k + \sum_{m \in S, n \notin S} x_{mn}. \tag{3.5}$$

These inequalities are weaker than the flow equations as such, but if we combine them with the constraints (**M7**.5), (**M7**.6) of model **M7**

$$x_{ij} \leq Q\xi_{ij} \tag{3.6}$$

we obtain

$$x_{ij} \leq \left[\sum_{k \in S} d_k + \sum_{m \in S, n \notin S} x_{mn} \right] \xi_{ij} \tag{3.7}$$

and, linearising, we end up with

$$x_{ij} \leq \left(\sum_{k \in S} d_k \right) \xi_{ij} + \sum_{m \in S, n \notin S} x_{mn}. \tag{3.8}$$

The above inequalities (the maximum flow inequalities) must hold for any feasible solution to the problem. An example of these inequalities is shown in figure 3.10, and they state that the flow through an arc (i, j) supplying a set of vertices S is at most equal to the total demand of S plus the sum of the flows of all arcs that leave the set S.

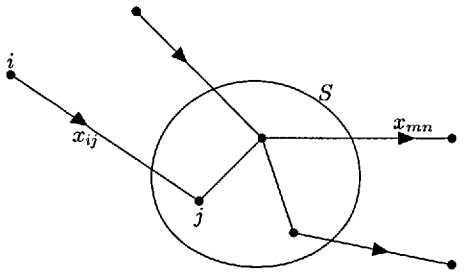


Figure 3.10: The maximum flow inequalities

The maximum flow inequalities are satisfied with equality only if this arc is the only arc supplying the set of vertices S.

From the form of the above inequalities it is clear that these inequalities are most effective if $|S|$ is small. During computational experimentation it was observed that in the case $|S| = 1$ these inequalities contribute significantly to the improvement of the lower bound at a small cost (at most $\leq 2|E|$ extra inequalities in the formulation, where $|E|$ is the number of edges of the network). For this reason, it was decided to include all maximum flow inequalities that correspond to $|S| = 1$ in the LP relaxations, regardless of

whether they are currently violated or not. For the cases $|S| \geq 2$, a separation heuristic procedure is used to generate violated maximum flow inequalities, which is presented in the next section.

The complete LP relaxation including the maximum flow inequalities with $|S| = 1$ is shown in model **M9**.

MODEL M9

$$\textbf{min} \quad \sum_{\substack{(i,j)\in E \\ i<j}} (b_{ij}\xi_{ij} + c_{ij}x_{ij} + c_{ji}x_{ji}) \tag{M9.1}$$

subject to

$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = d_i \qquad \text{for all } i \in V' \tag{M9.2}$$

$$\sum_{j:(0,j)\in E} x_{0j} = \sum_{i\in V'} d_i \tag{M9.3}$$

$$\sum_{j:(j,i)\in E} \xi_{ji} \geq 2 \qquad \text{for all } i \in V \tag{M9.4}$$

$$x_{ij} - d_j\xi_{ij} \leq \sum_{k} x_{jk} \qquad \text{for all } i \in V,\ j \in V' \tag{M9.5}$$

$$x_{ij} \leq Q\xi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M9.6}$$

$$x_{ji} \leq Q\xi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M9.7}$$

$$x_{ij} \geq 0 \qquad \text{for all } (i,j) \in E \tag{M9.8}$$

$$0 \leq \xi_{ij} \leq 1 \qquad \text{for all } (i,j) \in E,\ i < j \tag{M9.9}$$

THE LP RELAXATION OF THE URNDP WITH THE MAXIMUM FLOW INEQUALITIES ($|S| = 1$)

3.6.1 Separation heuristic

The following new heuristic procedure is used to find a set $S \subseteq V'$ and an edge (i,j) with $i \notin S$, $j \in S$ given an input set of flow and binary values $(\bar{\mathbf{x}}, \bar{\boldsymbol{\xi}})$, such that the following maximum flow inequality is violated:

$$\left(\sum_{k\in S} d_k\right) \xi_{ij} + \sum_{m\in S, n\notin S} x_{mn} - x_{ij} \geq 0. \tag{3.9}$$

We observe that a small value for $\sum_{m\in S,n\notin S} x_{mn}$ is important for an effective cut, so we use the Hao-Orlin algorithm (Hao & Orlin (1994)) to find the global minimum cut in the *directed* graph with the flow values $\bar{x}$ as weights. This procedure produces two sets S and S' such that the sum of the arc weights (i.e., the flows) going from S to S' is minimum. These sets will be the sets S and $V\backslash S$ of the generated cuts (3.9).

Note that in the implementation of the Hao-Orlin algorithm it is necessary to enforce $0 \in S'$ as required by the maximum flow inequalities, as well as $|S'| \geq 2$; this is because we assume that there are no arcs going into the source vertex and for this reason the value of the directed cut from V' to $\{0\}$ will always be zero (and hence this would always be the minimum directed cut produced by the Hao-Orlin algorithm). This can be easily achieved by assigning large weights to arcs going into the source vertex.

Having obtained the two sets S and S', we then minimise the quantity

$$\min \quad \left\{ \left(\sum_{k\in S} d_k \right) \bar{\xi}_{ij} + \sum_{m\in S,n\notin S} \bar{x}_{mn} - \bar{x}_{ij} \right\} \tag{3.10}$$

over all possible arcs $i \in S'$, $j \in S$. If this minimum value is negative we then add the corresponding violated maximum flow cut in the LP, if it is not, no violated maximum flow inequalities can be generated using the heuristic. An example of the separation procedure is shown in figure 3.11. In this figure, the edge numbers represent the flow values and the values next to the vertices are the demands. The solid edges represent edges with $\bar{\xi}_{ij} = 1$ whereas the dashed edge represents an edge with $\bar{\xi}_{ij} = \lambda < 1$. (Actually $\lambda > 3/23$ is required from the trivial constraints of the formulation.) The minimum directed cut is shown by the dashed line, with a value of 3, and the minimum value of expression (3.10) is $3\lambda - 3$. Since this value is negative, a violated maximum flow inequality is produced.

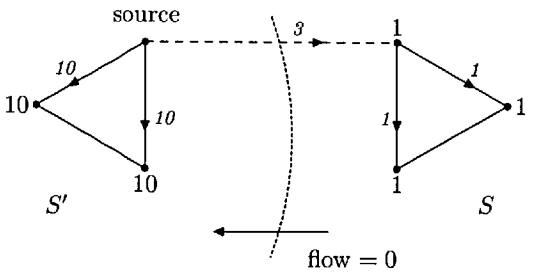


Figure 3.11: An example of the separation heuristic for the maximum flow inequalities

The separation heuristic for the maximum flow inequalities is summarised below:

1. Consider a directed graph with arc weights given by the values of the flows, $\bar{x}_{ij}$.
2. Use the Hao-Orlin algorithm to obtain the minimum directed cut on this graph, and let S, S' be the two shores obtained, with the minimum cut going from S' to S.
3. Mimimise the quantity (3.10) over all possible arcs of the cut, and let (i', j') be the arc that achieves this minimum.
4. If the minimum value is negative, the maximum flow inequality defined by the set S and the arc (i', j') is violated by the current solution.

We observe that the procedure described above is a heuristic procedure, since it is possible that violated maximum flow inequalities exist but they are not generated by the above algorithm.

3.6.2 Facet defining inequalities

In this section a proof is given that for certain types of input graphs the maximum flow inequalities (3.8), obtained for the case $|S| = 1$ are facet defining. These graphs are the complete graph K_N on N vertices (with $N \geq 4$) and, more generally, any three-edge connected graphs.

Theorem 16 *The maximum flow inequalities (3.8) corresponding to $|S| = 1$ are facet defining for the complete graph.*

Proof

Let $G(V, E)$ be the complete graph and label the vertices of the graph $0, 1, 2, \ldots, N-1$ where vertex 0 is the source vertex. The maximum flow inequalities corresponding to $|S| = 1$ are

$$x_{ij} - d_j \xi_{ij} - \sum_k x_{jk} \leq 0 \quad \text{for all } i,\ j\ (j \neq 0). \tag{3.11}$$

We shall prove that these inequalities are facet defining for the 'two-edge connected with flows' polytope, P, at each vertex $j \in V'$ and all incoming arcs $(i, j) \in E$. Without loss of generality we consider the case at vertex 1 and the incoming arc $(0, 1)$. The same technique of proof applies to any other arc. The inequality for this case is:

$$x_{01} - d_1 \xi_{01} - \sum_k x_{1k} \leq 0. \tag{3.12}$$

The polytope P is described in terms of $3|E|$ variables (where $|E|$ is the number of edges), $|E|$ of them corresponding to the binary variables ξ_{ij}, $|E|$ of them to the flow variables x_{ij} and $|E|$ relate to the 'inverse' flow variables x_{ji} (where $i < j$). Note that the polytope is not fully dimensional because its description does not contain only inequalities but also equalities, (the flow balance constraints). We also assume that $d_1 > 0$ as if $d_1 = 0$ then the maximum flow inequalities are implied by the flow balance constraints. This means that, if all the flow that enters vertex 1 is coming through vertex 0 then these inequalities are always 'facet defining' (since in that case the entire polytope P is a 'facet' because these inequalities will always be satisfied at equality due to the flow balance constraints). If, on the other hand, there are many arcs supplying vertex 1 then the maximum flow inequalities are redundant and never 'tight', hence they never define facets (not even faces) of the polytope.

Inequality (3.12) defines a hyperplane in $3|E|$ dimensions (when '$\leq$' is replaced with '=') which we shall denote by Π:

$$\Pi : x_{01} - d_1\xi_{01} - \sum_k x_{1k} = 0, \tag{3.13}$$

or in general matrix notation

$$\Pi : (\pi_{ij}, \pi_{ji}, \rho_{ij}) \cdot (x_{ij}, x_{ji}, \xi_{ij})^T = \pi_0. \tag{3.14}$$

The method that will be used to show that (3.12) is facet defining is the following: We shall consider a generic hyperplane in $3|E|$ dimensions

$$\sum_{i<j} \mu_{ij}x_{ij} + \sum_{i<j} \mu_{ji}x_{ji} + \sum_{i<j} \nu_{ij}\xi_{ij} = \mu_0 \tag{3.15}$$

and select enough points $\mathbf{x}_w \in P$ (i.e., feasible URNDP solutions, which will be referred to as *FS*) that satisfy (3.12) *with equalities* (i.e., points that lie on the hyperplane Π). Using the points $\mathbf{x}_w$ we solve the linear equation system

$$(\mu_{ij}, \mu_{ji}, \nu_{ij}) \cdot \mathbf{x}_w^T = \mu_0 \quad \text{for all } w, \tag{3.16}$$

for the unknowns μ, ν, μ_0. To prove that Π is facet defining, it is enough to prove that the *only* solution to the above is

$$(\pi_{ij}, \pi_{ji}, \rho_{ij}, \pi_0) = \lambda(\mu_{ij}, \mu_{ji}, \nu_{ij}, \mu_0), \tag{3.17}$$

where λ is a scalar with $\lambda \neq 0$.

Consider, therefore, the feasible solution to the URNDP with flows (FS1) obtained as follows:

$$\xi_{ij} = 1,\ x_{0j} = d_j, \quad \text{and } x_{ij} = 0 \quad \text{for any other arc.} \tag{3.18}$$

This solution is shown in figure 3.12. This solution lies on the hyperplane Π as it satisfies the maximum flow inequalities with equality. For this solution equation (3.15) gives

$$\sum_k \mu_{0k} d_k + T = \mu_0 \tag{3.19}$$

where

$$T = \sum_{i<j} \nu_{ij} \tag{3.20}$$

Now we can modify this solution by adding the arc (i, j) carrying a positive flow α and compensating by adding the opposite arc (j, i) carrying the same amount of flow. This new feasible solution will also lie on Π provided that neither i nor j is the vertex 1. This gives

$$\sum_k \mu_{0k} d_k + \alpha(\mu_{ij} + \mu_{ji}) + T = \mu_0. \tag{3.21}$$

Subtracting equalities (3.19) and (3.21) we obtain

$$\mu_{ij} + \mu_{ji} = 0. \tag{3.22}$$

For the edges that do contain vertex 1, we can only apply the above for the edge $(0, 1)$ as for this case only the maximum flow inequalities are satisfied as equalities. So we have proven so far that

$$\mu_{ij} + \mu_{ji} = 0 \quad \text{for } i \neq 1,\ j \neq 1; \qquad \mu_{01} + \mu_{10} = 0. \tag{3.23}$$

Now consider an edge (i, j) with $i \neq 1$, $j \neq 1$. We construct the following two solutions (FS2 and FS3) to the URNDP with flows that consist of all edges ($\xi_{ij} = 1$) and the flow runs along the paths

$$0 \to 1 \to 2 \to \cdots \to i-1 \to i+1 \to \cdots \to j-1 \to j+1 \to \cdots \to N-1 \to i \to j$$

and

$$0 \to 1 \to 2 \to \cdots \to i-1 \to i+1 \to \cdots \to j-1 \to j+1 \to \cdots \to N-1 \to j \to i$$

as shown in figures 3.13 and 3.14 respectively. Both these solutions lie on Π as there is only one arc supplying vertex 1. Using the solution values that correspond to these two

layouts, equation (3.15) gives

$$(d_1 + \cdots + d_{N-1})\mu_{01} + (d_2 + \cdots + d_{N-1})\mu_{12} + \cdots + (d_{N-1} + d_i + d_j)\mu_{N-2,N-1} + (d_i + d_j)\mu_{N-1,i} + d_j\mu_{ij} + T = \mu_0 \quad (3.24)$$

and

$$(d_1 + \cdots + d_{N-1})\mu_{01} + (d_2 + \cdots + d_{N-1})\mu_{12} + \cdots + (d_{N-1} + d_i + d_j)\mu_{N-2,N-1} + (d_i + d_j)\mu_{N-1,j} + d_i\mu_{ji} + T = \mu_0 \quad (3.25)$$

respectively. Subtracting equations (3.24) and (3.25) and using (3.23) we get

$$(d_i + d_j)(\mu_{N-1,i} - \mu_{N-1,j}) + (d_i + d_j)\mu_{ij} = 0 \quad (3.26)$$

i.e.,

$$\mu_{N-1,i} - \mu_{N-1,j} + \mu_{ij} = 0. \quad (3.27)$$

Since there is no distinction between the vertices $N-1$, i and[23] j, we expect that

$$\mu_{N-1,i} = \mu_{N-1,j}. \quad (3.28)$$

(It is possible to derive (3.28) by writing down equations of the type (3.27) for all possible combinations of vertices and then solving the linear system, but by noting the symmetry makes this step simpler.) Hence we have that

$$\mu_{ij} = 0 \quad \text{for all } i \neq 1,\ j \neq 1. \quad (3.29)$$

For the solution (3.18) we observe that we can increase the flow along the cycle $0 \rightarrow 1 \rightarrow k \rightarrow 0$ by an arbitrary amount α and the solution will still be feasible and satisfy the maximum flow inequalities with equality. This means that

$$\mu_{01} + \mu_{1k} + \mu_{k0} = 0, \quad (3.30)$$

and therefore using (3.29)

$$\mu_{1k} = -\mu_{01}. \quad (3.31)$$

Coefficients ν_{ij}

We now focus on the coefficients ν_{ij} corresponding to the decision variables ξ_{ij} (i.e., the edges being present in the solution). We consider the feasible solution described in

[23] the only edges that are 'special' (distinguishable) in this setup are 0 and 1, which define the incoming arc of the maximum flow inequality that is being examined.

(3.18) we modify this solution by removing any single edge of the graph (i.e., setting $\xi_{ij} = 0$). This will yield another feasible solution for the URNDP that lies on the hyperplane Π provided that the edge (i, j) is not carrying any flow. The reason why this new solution will still lie on Π is simple: the change does not include a variable appearing in the description (3.13) of Π. The reason why the new solution will be feasible is more subtle, but we note that the underlying graph is complete which means that every cut of the graph has $N - 1$ edges; this means that by removing any edge will leave a network where every cut will contain at least 2 edges, hence it will always be two-edge connected. The equations obtained by these two solutions will only differ in the term "$+\nu_{ij}$" (which will disappear from the left-hand side of the second equality). Subtracting the two implies that ν_{ij} must be equal to zero.

The only coefficients ν_{ij} that cannot be eliminated with the above reasoning are the ones that carry flow in the solution (3.18), in other words edges $(0, k)$. By considering a different solution, for example the solution FS4 (figure 3.15) where the flow runs along the vertices

$$0 \rightarrow 1 \rightarrow 2 \cdots \rightarrow N - 1$$

and using the same method (removing one edge) we can obtain the same result for these coefficients too. The only exception is edge $(0, 1)$ which cannot be removed from the solution since if it is removed then the solution will not satisfy the maximum flow inequalities with equality.

Therefore, it has been shown that:

$$\begin{aligned}
&\mu_{ij} = 0, && \text{for all } i \neq 1,\ j \neq 1 \\
&\mu_{01} + \mu_{10} = 0 && \\
&\mu_{1k} = -\mu_{01}, && \text{for all } k \neq 0 \\
&\nu_{ij} = 0, && \text{for all } (i, j) \neq (0, 1)
\end{aligned}$$

Coefficients μ_{k1}

We finally consider the coefficients μ_{k1}. These coefficients will only appear when ξ_{k1} is non-zero. If a solution is, at the same time, required to lie on Π this implies that $\xi_{01} = 0$. In this case, furthermore, there can be no flow *leaving* vertex 1 because in that case too, the maximum flow inequalities will be strict. This means that: edge $(0, 1)$ is not present, vertex 1 receives exactly the amount of flow d_1 (through a vertex other than 0) and there is no flow leaving vertex 1. For any such solution—for example the solution

FS5 (figure 3.16) where the flow runs through

$$0 \to N-1 \to N-2 \to \cdots \to 2 \to 1,$$

—we obtain the following relationship

$$d_1\mu_{k1} + \sum_{\text{edges present}} \nu_{ij} = \mu_0 \tag{3.32}$$

where k is the vertex supplying the vertex 1. Hence,

$$\mu_{k1} = \frac{\mu_0}{d_1} \tag{3.33}$$

since the only non-zero μ's are the μ_{j1} and the only non-zero ν is ν_{01}; the latter is not present in the equation as edge $(0,1)$ is not in the solution.

Using the coefficients obtained so far, equation (3.19) yields

$$\nu_{01} = \mu_0 - \mu_{01}d_1 \tag{3.34}$$

In summary we have calculated that

$$\begin{aligned}
&\mu_{01} = \kappa\,, \quad \text{(say)}\\
&\mu_{10} = -\kappa\\
&\mu_{k1} = \mu_0/d_1\\
&\mu_{1k} = -\kappa\\
&\nu_{01} = \mu_0 - \kappa d_1\\
&\text{All other } \mu,\ \nu \text{ are zero.}
\end{aligned}$$

The equation of the hyperplane (3.15) becomes

$$\kappa x_{01} + \sum_{k\neq 0} \frac{\mu_0}{d_1} x_{k1} - \sum_{k} \kappa x_{1k} + (\mu_0 - \kappa d_1)\xi_{01} = \mu_0 \tag{3.35}$$

The equation implies that the solutions lie on a family of hyperplanes, for different values of μ_0, κ.[24] However, the polytope P is not fully dimensional since some of the constraints are equalities (the flow balance equations). For this reason all the hyperplanes described by (3.35) actually collapse into one hyperplane as shown below. Equation (3.35)

[24] If this were the case then the original maximum flow inequality would define a *face* of the polytope and not a *facet*.

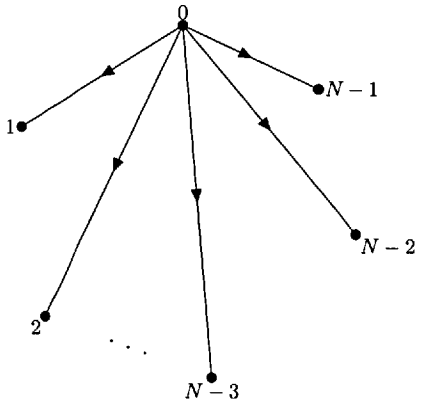


Figure 3.12: Solution FS1 satisfying with equality the maximum flow inequalities

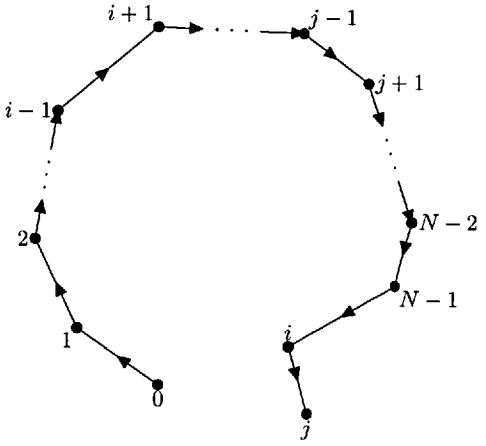


Figure 3.13: Solution FS2 satisfying with equality the maximum flow inequalities

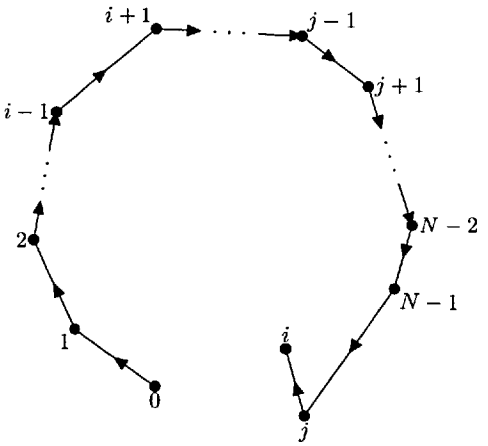


Figure 3.14: Solution FS3 satisfying with equality the maximum flow inequalities

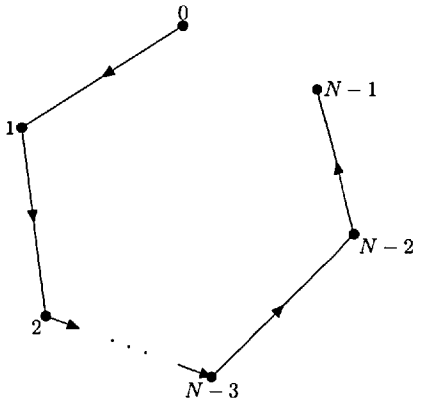


Figure 3.15: Solution FS4 satisfying with equality the maximum flow inequalities

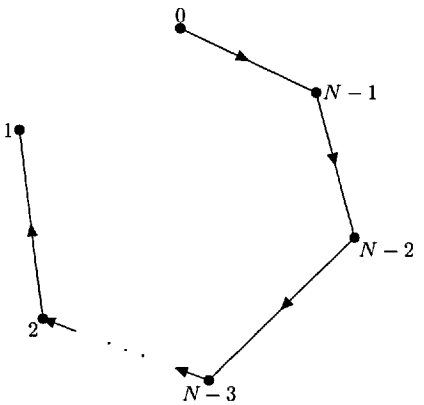


Figure 3.16: Solution FS5 satisfying with equality the maximum flow inequalities

can be re-arranged as

$$\left(\kappa - \frac{\mu_0}{d_1}\right) x_{01} - \left(\kappa - \frac{\mu_0}{d_1}\right) \sum_k x_{1k} + (\mu_0 - \kappa d_1)\xi_{01} = \\ = \frac{\mu_0}{d_1}\left(d_1 + \sum_k x_{1k} - \sum_{k \neq 0} x_{k1} - x_{01}\right) \quad (3.36)$$

The right-hand side of this equation will always be zero because of the flow balance equations at vertex 1, which any point of P must satisfy. This means that the hyperplane (3.35) is equivalent to

$$\left(\kappa - \frac{\mu_0}{d_1}\right)\left(x_{01} - \sum_k x_{1k} - d_1\xi_{01}\right) = 0 \quad (3.37)$$

and hence we have managed to reduce the generic hyperplane (3.15) to a scalar multiple of the hyperplane Π.

Therefore the maximum flow inequality (3.12) is facet defining. Since the edge $(0, 1)$ was arbitrary selected, it can be replaced with any other edge which shows that the maximum flow inequalities (3.11) are facet defining. □

Corollary 17 *The maximum flow inequalities are facet defining for any three-edge connected network.*

Proof

The proof of Theorem 16 can be applied to this case as well, with the following differences in the layouts of the feasible solutions used: FS1 can be replaced with any solution where the flows form a tree, and all the edges of the graph being present. FS2 and FS3 can similarly be easily replaced with any solution where the vertices j (for FS2) or i (for FS1) are leaves in the flow tree and all the flow supplying the network passes through edge $(0, 1)$. Solution FS5 (as specified in the proof) can be any solution where edge $(0, 1)$ is absent and vertex 1 receives exactly the amount of flow d_1; clearly such a solution exists for a three-edge connected network too.

The only difficulty is the step where two feasible solutions were used to derive that $\nu_{ij} = 0$ except for ν_{01} (page 92). These two solutions should have their flow trees disjoint, except for edge $(0, 1)$ which must be present in both trees for the process to work. In the previous proof the pair of solutions FS1 and FS4 had this property. To obtain such solutions for this case, we observe that if a network is three-edge connected then it is also two-edge connected and therefore there exist two edge disjoint spanning trees (see

Conjecture 10, part (iii), on page 47). These trees can provide the two disjoint solutions that we require. Moreover, we require that both solutions contain the edge $(0, 1)$ but this can be achieved by first shrinking the graph by merging the vertices 0 and 1, then applying the previous technique and finally adding the edge $(0, 1)$ to both the disjoint trees obtained. This can be done because the shrunk graph will be two-edge connected as well.

The three-edge connectivity property is required when, after the two solutions described above are obtained, we need the graph to remain two-edge connected when one edge is removed. This is true, by definition, for any three-edge connected graph. □

3.6.3 A numerical example

We present a brief example of calculating the dimensions of the two-edge connected polytope and the dimensions of the maximum flow inequalities generated. The input graph is the complete graph with four vertices (K_4), with all the demands set to 1. The calculations were performed using PORTA (see *PORTA 1.4.0α* (2002)), which calculates convex hulls and analyses polyhedra numerically. There are 6 edges in K_4 and therefore 18 variables (six for the binary variables ξ_{ij} and twelve for the flow variables x_{ij}). The polytope is described by the constraints shown in table 3.3, consisting of *four* flow balance equations, *seven* reliability constraints and *twelve* trivial constraints.

Flow balance constraints		*Reliability constraints*
$x_{01} + x_{02} + x_{03} - x_{10} - x_{20} - x_{30} = 3$		$\xi_{01} + \xi_{02} + \xi_{03} \geq 2$
$x_{01} + x_{21} + x_{31} - x_{10} - x_{12} - x_{13} = 1$		$\xi_{01} + \xi_{12} + \xi_{13} \geq 2$
$x_{02} + x_{12} + x_{32} - x_{20} - x_{21} - x_{23} = 1$		$\xi_{02} + \xi_{12} + \xi_{23} \geq 2$
$x_{03} + x_{13} + x_{23} - x_{30} - x_{31} - x_{32} = 1$		$\xi_{03} + \xi_{13} + \xi_{23} \geq 2$
Trivial constraints		$\xi_{02} + \xi_{03} + \xi_{12} + \xi_{13} \geq 2$
$x_{01} - 3\xi_{01} \leq 0$	$x_{10} - 3\xi_{01} \leq 0$	$\xi_{01} + \xi_{03} + \xi_{12} + \xi_{23} \geq 2$
$x_{02} - 3\xi_{02} \leq 0$	$x_{20} - 3\xi_{02} \leq 0$	$\xi_{01} + \xi_{02} + \xi_{13} + \xi_{23} \geq 2$
$x_{03} - 3\xi_{03} \leq 0$	$x_{30} - 3\xi_{03} \leq 0$	*Bounds*
$x_{12} - 3\xi_{12} \leq 0$	$x_{21} - 3\xi_{12} \leq 0$	$0 \leq \xi_{ij} \leq 1$
$x_{13} - 3\xi_{13} \leq 0$	$x_{31} - 3\xi_{13} \leq 0$	$0 \leq x_{ij} \leq 3$
$x_{23} - 3\xi_{23} \leq 0$	$x_{32} - 3\xi_{23} \leq 0$	ξ_{ij}, x_{ij} integer

Table 3.3: The constraints defining the URNDP with flows for K_4

Note that since only integer polyhedra can be analysed numerically, we have set the flow variables x_{ij} to be integer. This is not a limitation given that, with integer demands, any solution with non-integer flow values will necessarily include flow that cycles; this

solution can be converted to one with integer x_{ij} by adjusting the flows (removing cycles) and, as the numerical experimentation showed, the maximum flow inequalities are facet defining for this 'integer-flow' version of the convex hull too.

convex hull of integer solutions	
number of variables	18
dimension of convex hull	15
number of integer (feasible) points	52044

maximum flow inequality	*dimension*
$x_{13} - \xi_{13} - x_{30} - x_{31} - x_{32} \leq 0$	14
$x_{23} - \xi_{23} - x_{30} - x_{31} - x_{32} \leq 0$	14

maximum flow inequality	*dimension*
$x_{01} - \xi_{01} - x_{10} - x_{12} - x_{13} \leq 0$	14
$x_{21} - \xi_{12} - x_{10} - x_{12} - x_{13} \leq 0$	14
$x_{31} - \xi_{13} - x_{10} - x_{12} - x_{13} \leq 0$	14
$x_{02} - \xi_{02} - x_{20} - x_{21} - x_{23} \leq 0$	14
$x_{12} - \xi_{12} - x_{20} - x_{21} - x_{23} \leq 0$	14
$x_{32} - \xi_{23} - x_{20} - x_{21} - x_{23} \leq 0$	14
$x_{03} - \xi_{03} - x_{30} - x_{31} - x_{32} \leq 0$	14

Table 3.4: Analysis of the maximum flow inequalities and the convex hull for K_4

The results of the analysis are presented in table 3.4. The LP relaxation of the formulation defines a polyhedron of dimension 15 (18 variables minus 3 linearly independent constraints[25]) that contains 52044 feasible points. These points define a convex hull which has also dimension 15. The maximum flow inequalities are valid for all the feasible points, and define a hyperplane of dimension 14 when they intersect the convex hull. This shows that they are facets because their dimension is one less than the dimension of the convex hull.

Note that we could have chosen to exclude the flow variables that return to the source (x_{10}, x_{20} and x_{30}) since they are always zero in the optimal solution. In that case there are 15 variables only, the convex hull is of dimension 12 and contains 2929 integer points. The maximum flow inequalities in that case define hyperplanes of dimension 11. For this case, it is also possible to calculate *all* the 747 facets of the convex hull from the set of integer points, something that is practically impossible for the previous convex hull given the much larger number of integer points it contains. The complete results of the analysis of this polytope are shown in appendix B, which could be useful when searching for other valid inequalities for the URNDP with flows.

The above analysis took about 15 minutes (on a 600MHz Pentium III PC) to be performed. However, such enumeration techniques cannot be used to analyse polytopes corresponding to larger graphs as the computational time increases dramatically. For instance, it would take 32 years or so on an identical computer to perform a similar analysis for K_5, mainly because of the exponentially larger number of feasible points

[25]This is because the first flow balance constraint can be obtained by summing the remaining three, therefore only three linearly independent equalities exist.

contained in the polyhedron defined by the constraints.

3.6.4 Computational results

The computational results presented in this section are obtained by adding to the LP relaxation (model **M9**), which includes maximum flow inequalities corresponding to the case $|S| = 1$, any maximum flow inequalities with $|S| \geq 2$ that have been generated using the heuristic of section 3.6.1.

The notation used in table 3.5 is presented below:

- LB_0 Lower bound obtained after adding the flow balance, trivial and degree inequalities (**M8**).
- LB_1 Lower bound obtained after adding the flow balance, trivial, degree and maximum flow inequalities (3.8).
- z_{opt} The cost of the optimal solution to the URNDP or the cost of the best feasible solution found at the end of the Branch and Cut algorithm.
- $\%LB_0$ Percentage ratio LB_0/z_{opt}.
- $\%LB_1$ Percentage ratio LB_1/z_{opt}.
- *# cuts* The total number of maximum flow cuts generated.
- t_{mf} The time (in seconds) taken for the generation of the maximum flow cuts.

Problem	*Vertices*	*Edges*	z_{opt}	LB_0	LB_1	$\%\, LB_0$	$\%\, LB_1$	*# cuts*	t_{mf}
g1n12g001i5	12	47	17434	17282.0	17300.5	99.13	99.23	88	0.01
g1n12g01i2	12	55	2177	2084.0	2124.6	95.73	97.59	102	0.02
g1n12g1i3	12	49	370	346.9	359.6	93.77	97.18	91	0.01
g1n12g2i1	12	49	381	372.0	381.0	97.65	100	89	0.01
g1n15g01i4	15	85	1475	1365.0	1387.1	92.55	94.04	163	0.04
g1n20g01i3	20	143	2582	2394.9	2444.3	92.75	94.67	273	0.07
g1n25g01i4	25	220	2586	2370.4	2419.2	91.66	93.55	441	0.12
g1n30g2i2	30	337	668	635.2	661.0	95.09	98.94	653	0.25
g1n40g005i5	40	575	6193	5572.1	5660.9	89.97	91.41	1122	0.75
g1n50g1i2	50	955	1179	1067.2	1134.5	90.52	96.23	1876	2.31
g1n50g01i1	50	907	3834	3260.0	3380.8	85.03	88.18	1777	2.18
Average						*93.89*	*96.28*	*489.8*	*0.36*

Table 3.5: Root node lower bounds LB_1 with the maximum flow inequalities (**M9**) added

We note a significant improvement to the lower bounds when the maximum flow inequalities are added, compared with the initial LP relaxation LB_0, and that the total time required for the generation of the inequalities is very small. Extensive computational results for all groups of test problems are presented in section 3.10.1 at the end of the chapter.

3.7 The r-cover inequalities

Another set of inequalities that is valid for the URNDP with flows are the *r-cover inequalities*. They appeared in the article by Grötschel, Monma & Stoer (1992*a*) in the context of the design of two-edge connected networks without flows, and they relate only to the decision variables ξ. They are derived from the requirement that every vertex must have at least r edges incident to it, which in the case of the URNDP with flows is true with $r = 2$.

The r-cover inequalities are:

$$\sum_{i,j \in H} \xi_{ij} + \sum_{(i,j) \in \delta(H), (i,j) \notin T} \xi_{ij} \geq |H| - \left\lfloor \frac{|T|}{2} \right\rfloor \tag{3.38}$$

for all $H \subseteq V$ and all $T \subseteq \delta(H)$, where V is the set of vertices and $\delta(H)$ is the cut with H as a shore, i.e., all the edges with one end in H and the other end outside H. These inequalities are depicted in figure 3.17.

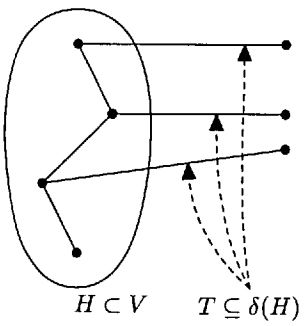


Figure 3.17: The r-cover inequalities

The set H is usually referred to as the *handle* and the edges T as the *teeth*. Note that an odd number of teeth is required for the inequalities to have a contribution, because if T is even then these inequalities are implied by the degree constraints. To see this, we can add the degree constraints corresponding to the vertices in H to obtain

$$2 \sum_{i,j \in H} \xi_{ij} + \sum_{(i,j) \in T} \xi_{ij} + \sum_{(i,j) \notin T} \xi_{ij} \geq 2|H|, \tag{3.39}$$

i.e.,

$$2\sum_{i,j\in H}\xi_{ij} + 2\sum_{(i,j)\notin T}\xi_{ij} \geq 2|H| - \left\lceil \sum_{(i,j)\in T}\xi_{ij} - \sum_{(i,j)\notin T}\xi_{ij} \right\rceil. \tag{3.40}$$

The r-cover inequality, for $|T|$ even, is the same as (3.40) with a right-hand side of $2|H| - |T|$. To show that these inequalities are implied from (3.40) it is enough to prove that

$$2|H| - \left\lceil \sum_{(i,j)\in T}\xi_{ij} - \sum_{(i,j)\notin T}\xi_{ij} \right\rceil \geq 2|H| - |T|. \tag{3.41}$$

This is equivalent to

$$\sum_{(i,j)\notin T}\xi_{ij} \geq \sum_{(i,j)\in T}\xi_{ij} - |T| \tag{3.42}$$

which is always true, since the right-hand side is zero or negative.

3.7.1 Derivation of the r-cover inequalities

This section describes the separation algorithm for the r-cover inequalities. The separation problem is related to the odd cut problem that can be solved using the Padberg-Rao algorithm (Padberg & Rao (1982)). The procedure is described, albeit very briefly, in Grötschel, Monma & Stoer (1992*a*), and Grötschel, Monma & Stoer (1992*b*). The full derivation is presented, including the relationship to the b-matching problem, as a similar technique will be used to derive new inequalities for the capacitated RNDP in chapter 6.

3.7.1.1 The b-matching problem

The b-matching problem can be described as follows:

> "Given a network $G(V, E)$ with non-negative edge weights (costs) q_{ij}, and integers b_v associated with each vertex $v \in V$, find the minimum (maximum) cost sub-network such that at least (at most) b_v edges meet at each vertex v."

We consider the maximisation version of the problem; this can easily be formulated as an IP as follows:

$$\max \quad \sum_{i,j} q_{ij}\xi_{ij} \tag{3.43}$$

subject to:

$$\sum_{j} \xi_{ij} \leq b_i, \qquad \text{for all } i \in V \tag{3.44}$$

$$\xi_{ij} \in \{0, 1, 2, \ldots\} \tag{3.45}$$

In the above formulation every edge may be selected more than once and ξ_{ij} denotes the number of times it is chosen.

Solving the LP relaxation of this problem yields fractional solutions. Edmonds & Johnson (1970) showed that if a certain class of inequalities is added (the *matching or blossom inequalities*) then we obtain an exact description of the convex hull spanned by the incidence vectors of the problem, i.e., the resulting LP has an integer solution which is therefore the required optimum. The matching inequalities are the following:

$$\sum_{i,j \in W} \xi_{ij} \leq \frac{1}{2}\left(\sum_{i \in W} b_i - 1\right), \qquad \text{for all } W \subseteq V \tag{3.46}$$

Note that the b-matching problem can also be solved in polynomial time using an algorithm also devised by Edmonds.

3.7.1.2 The capacitated case

If we allow at most d_e copies of each edge e to be present we obtain the capacitated version of the problem. We consider the case $d_e = 1$, i.e., no copies of the same edge are allowed. In this case the constraint (3.45) becomes

$$\xi_{ij} \in \{0, 1\} \tag{3.47}$$

Edmonds generalised the matching inequalities for the capacitated version. For the case of the 1-capacitated b-matching problem the inequalities required to fully describe the convex hull are:

$$\sum_{i,j \in W} \xi_{ij} + \sum_{(i,j) \in T} \xi_{ij} \leq \frac{1}{2}\left(\sum_{i \in W} b_i + |T| - 1\right) \tag{3.48}$$

for all $W \subseteq V$ and $T \subseteq \delta(W)$ that satisfy

$$\sum_{i \in W} b_i + |T| = \text{ odd.} \tag{3.49}$$

3.7.1.3 The odd cut problem

Padberg & Rao (1982) related the b-matching problem to the *odd cut problem* which is presented below:

> "Given a network $G(V, E)$ in which every vertex can be either *odd* or *even*, with an even number of odd vertices, and given a set of edge weights, find a cut of minimum weight such that each shore of the cut has an odd number of odd vertices."

The odd cut problem can be solved in polynomial time with an algorithm devised by Padberg & Rao (1982) using a modification of the Gomory-Hu algorithm, and this can be used as a tool to separate the capacitated b-matching inequalities (3.48), as presented in the next section.

3.7.1.4 Separating matching inequalities and the Padberg-Rao algorithm

The separation of the matching inequalities using the odd cut problem is described in Padberg & Rao (1982) and it is summarised below. We only describe the version relating to the 1-capacitated case.

We label every vertex as odd if b_v is odd and even otherwise. We assign the odd cut weights w_{ij} to be the values of edge variables $\bar{\xi}_{ij}$ of the b-matching problem, and modify the network as follows: we scan all edges in any order. For each edge (i, j) scanned, we insert a new vertex, I_i say which has an odd label. We join this vertex with i and edge weight $w'_{iI_i} = 1 - w_{ij}$ and also with vertex j and edge weight $w'_{I_ij} = w_{ij}$. We then reverse the label of vertex i; the label of vertex j stays the same. Finally we delete the edge (i, j). It can be shown that regardless of the order we perform the edge scanning the result will not change. The new network will also have an even number of odd vertices provided that the original network does.

We then find the *minimum* odd cut of this expanded network, $G'(V \cup V', E')$. Let this cut be $\delta(U)$. If the value of this cut is less than 1, we can find a violated matching inequality. We construct the sets W and T by defining:

$$W = U \cap V, \text{ and} \tag{3.50}$$

$$T = \left\{(i, j) \in E \mid \exists e \in \delta(U) \text{ such that } e = (I_i, j) \text{ for some } j \in V,\ w'_{I_ij} = 1 - w_{ij}\right\}. \tag{3.51}$$

This separation is done in polynomial time.

3.7.1.5 Reliability and r-covers

We now present the relationship of the b-matching problem to the design of two-edge (or more generally k-edge) connected networks. In such a network, every vertex v must have degree *at least* two, which translates to having *at least* two edges incident to it, i.e., *at most* $|\delta(v)| - 2$ edges incident to v may be *absent* from the solution. We can convert this problem to a 1-capacitated b-matching problem by setting the b-values of the vertices to be $b_v = |\delta(v)|-2$ (where $|\delta(v)|$ is the degree of vertex v), and we set the b-matching edge variables to $\xi'_{ij} = 1 - \xi_{ij}$. We show that for these variables the b-matching inequalities (3.48) produce the r-cover inequalities

$$\sum_{i,j\in H} \xi_{ij} + \sum_{(i,j)\notin T} \xi_{ij} \geq |H| - \left\lfloor \frac{|T|}{2} \right\rfloor \tag{3.52}$$

for all $H \subseteq V$ and all $T \subseteq \delta(H)$.

Using the substitutions $b_v = |\delta(v)| - 2$ and $\xi'_{ij} = 1 - \xi_{ij}$ we obtain

$$\sum_{v\in H} b_v = \sum_{v\in H}(|\delta(v)| - 2) = 2|E(H)| + |\delta(H)| - 2|H|, \tag{3.53}$$

$$\xi'(E(H)) = \sum_{e\in E(H)} \xi'_e = |E(H)| - \xi(E(H)) \tag{3.54}$$

and

$$\xi'(T') = \sum_{e\in T'} \xi'_e = |T'| - \xi(T'), \tag{3.55}$$

with $H \subseteq V$, $T' \subseteq \delta(H)$ and $E(H) := \{(i,j) \in E : i \in H, j \in H\}$. The b-matching inequality

$$\sum_{i,j\in W} \xi'_{ij} + \sum_{(i,j)\in T'} \xi'_{ij} \leq \frac{1}{2}\left(\sum_{i\in W} b_i + |T'| - 1\right) \tag{3.56}$$

becomes

$$|E(H)| - \xi(E(H)) + |T'| - \xi(T') \leq |E(H)| - |H| + \frac{|\delta(H)| + |T'| - 1}{2}, \tag{3.57}$$

i.e.,

$$\xi(E(H)) + \xi(T') - |H| + \frac{|\delta(H)| - |T'| - 1}{2} \geq 0. \tag{3.58}$$

If we set $T = \delta(H)\backslash T'$ and so $|T'| = |\delta(H)| - |T|$ we end up with

$$\xi(E(H)) + \xi(\delta(H)\backslash T) - |H| + \frac{|T| - 1}{2} \geq 0 \tag{3.59}$$

i.e.,

$$\sum_{i,j\in H} \xi_{ij} + \sum_{(i,j)\notin T} \xi_{ij} \geq |H| - \frac{|T|-1}{2}. \tag{3.60}$$

Finally, it is easy to see that the condition

$$\sum_{v\in E} b_v + |T'| = \text{odd} \tag{3.61}$$

becomes

$$2|E(H)| + |\delta(H)| - 2|H| + |\delta(H)| - |T| = \text{odd} \tag{3.62}$$

that is,

$$|T| = \text{odd}. \tag{3.63}$$

Equation (3.60) with $|T|$ = odd is identical to the r-cover inequality (3.52). □

3.7.2 A numerical example

Figure 3.18 shows an example of the implementation of the r-cover inequalities. The first diagram represents the values $\bar{\xi}$ of the decision variables in the current LP relaxation, and the second shows the sets H and T (the handle and teeth). We can observe that the indicated sets produce a violated r-cover inequality, because the left hand side of (3.38) evaluates to 1.5 whereas the right hand side equals 2.

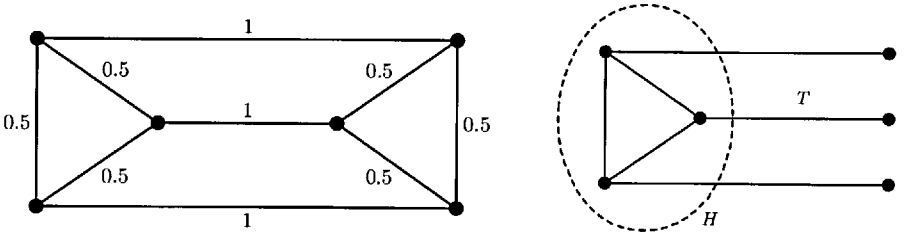


Figure 3.18: An example of violated r-cover inequalities

Note also that in the first graph of figure 3.18 all the reliability constraints are satisfied, showing that the r-cover inequalities cannot be implied by the reliability constraints alone.

3.7.3 Computational results

This section presents the effect of the r-cover inequalities when they are added to the LP relaxation (model **M9**). In the lower bound ascent method, the r-cover inequalities are

added after the maximum flow inequalities and the partition inequalities. The improvement on the objective value is shown in table 3.6, for selected instances. The notation used in this table is the following:

LB_1	Lower bound obtained after adding the flow balance, trivial, degree and maximum flow inequalities (3.8).
LB_2	Lower bound obtained after adding the flow balance, trivial, degree, maximum flow and partition inequalities (3.4).
LB_3	Lower bound obtained after adding the flow balance, trivial, degree, maximum flow, partition and r-cover inequalities (3.38).
z_{opt}	The cost of the optimal solution to the URNDP or the cost of the best feasible solution found at the end of the Branch and Cut algorithm.
$\%LB_1$	Percentage ratio LB_1/z_{opt}.
$\%LB_2$	Percentage ratio LB_2/z_{opt}.
$\%LB_3$	Percentage ratio LB_3/z_{opt}.
# cuts	The number of partition or r-cover cuts generated.
t_{part}	The time (in seconds) taken for the generation of the partition cuts.
t_{rc}	The time (in seconds) taken for the generation of the r-cover cuts.

				max flow		*partition*				*r-cover*			
Problem	*Vertices*	*Edges*	z_{opt}	LB_1	$\%\ LB_1$	LB_2	$\%\ LB_2$	*# part. cuts*	t_{part}	LB_3	$\%\ LB_3$	*# r-cov. cuts*	t_{rc}
g1n12g001i5	12	47	17434	17300.5	99.23	17300.8	99.24	20	0.06	17314.9	99.32	1	0.03
g1n12g01i2	12	55	2177	2124.6	97.59	2124.6	97.59	0	0.01	2129.9	97.84	1	0.02
g1n12g1i3	12	49	370	359.6	97.18	359.6	97.18	0	0.01	359.6	97.18	0	0.02
g1n15g01i4	15	85	1475	1387.1	94.04	1388.4	94.13	1	0.02	1388.6	94.14	1	0.05
g1n20g01i3	20	143	2582	2444.3	94.67	2445.5	94.71	2	0.06	2445.5	94.71	1	0.08
g1n25g01i4	25	220	2586	2419.2	93.55	2419.2	93.55	0	0.07	2419.4	93.56	1	0.22
g1n30g2i2	30	337	668	661.0	98.94	662.3	99.15	6	0.25	662.3	99.15	0	0.31
g1n40g005i5	40	575	6193	5660.9	91.41	5660.9	91.41	0	0.25	5672.5	91.60	1	1.00
g1n50g1i2	50	955	1179	1134.5	96.23	1134.6	96.24	4	1.06	1134.6	96.24	0	2.74
g1n50g01i1	50	907	3834	3380.8	88.18	3386.3	88.32	4	1.44	3387.7	88.36	4	16.65
Average					*95.10*		*95.15*	*3.7*	*0.32*		*95.21*	*1.0*	*2.11*

Table 3.6: Root node lower bounds with the partition (3.4) and the r-cover inequalities (3.38) added

3.8 The Gomory cuts

The last two categories of valid inequalities that were implemented for the URNDP with flows are classical, problem independent cuts. These cuts are not specific to the URNDP but can be generated for general MIPs.

A Gomory fractional cut generator was implemented for the URNDP with flows. Using the information that all the variables (including the flows) are integer in the solution, we add cuts (1.15) generated from rows of the tableau where the corresponding basic variable has a non-integer value.

The generated cuts were of poor quality. The lower bound increased slowly (albeit steadily) when these cuts were added to the formulation.

3.8.1 Computational results

The implementation used was the following. When the LP relaxation was optimised and the tableau was updated, we generated cuts from the row of the tableau with the maximum infeasibility, i.e., we chose the row corresponding to a basic variable with a fractional value closest to 0.5 among all candidate variables. Then the LP was solved again and the process was repeated, up to a maximum of 19 times. The following observations can be made:

- There was no significant difference when the cuts were obtained from rows where the basic variable is a binary variable or a flow variable.
- The cuts generated were very complex, in particular:
  - ⋆ the coefficients of the inequalities were fractional and sometimes quite small, and
  - ⋆ the cuts included many of the variables of the formulation.

Computational results obtained with the implementation of Gomory cuts are presented in table 3.7 below and in section 3.10.1. The Gomory cuts were added to the LP relaxation after the partition, maximum flow and r-cover inequalities were introduced, although it makes little difference when they are generated because their presence does not seem to affect the generation of the other types of cuts.

The addition of too many Gomory cuts is therefore not recommended, as they make the LP very complex and each LP subproblem takes significantly longer to solve, thus

slowing down the whole algorithm. This proved to be a big price to pay, especially given the small contribution that these cuts make to the lower bound.

Table 3.7 presents the lower bounds obtained when Gomory cuts were added to the formulation. The interpretation of the notation is presented below.

- LB_3 Lower bound obtained after adding the flow balance, trivial, degree, maximum flow, partition and r-cover inequalities.
- LB_4 Lower bound obtained after adding the flow balance, trivial, degree, maximum flow, partition, r-cover and Gomory inequalities.
- z_{opt} The cost of the optimal solution to the URNDP or the cost of the best feasible solution found at the end of the Branch and Cut algorithm.
- $\%LB_3$ Percentage ratio LB_3/z_{opt}.
- $\%LB_4$ Percentage ratio LB_4/z_{opt}.
- *# cuts* The number of Gomory cuts generated.
- t_{gom} The time (in seconds) taken for the generation of the Gomory cuts.

Problem	*Vertices*	*Edges*	z_{opt}	LB_3	$\%\,LB_3$	LB_4	$\%\,LB_4$	*# cuts*	t_{gom}
g1n12g001i5	12	47	17434	17314.9	99.32	17315.6	99.32	19	0.38
g1n12g01i2	12	55	2177	2129.9	97.84	2130.8	97.88	19	0.48
g1n12g1i3	12	49	370	359.6	97.18	360.3	97.37	19	0.30
g1n15g01i4	15	85	1475	1388.6	94.14	1389.6	94.21	19	0.63
g1n20g01i3	20	143	2582	2445.5	94.71	2445.7	94.72	19	1.47
g1n25g01i4	25	220	2586	2419.4	93.56	2419.6	93.56	19	2.63
g1n30g2i2	30	337	668	662.3	99.15	662.6	99.19	19	3.61
g1n40g005i5	40	575	6193	5672.5	91.60	5673.1	91.60	19	9.72
g1n50g1i2	50	955	1179	1134.6	96.24	1134.6	96.24	19	24.13
g1n50g01i1	50	907	3834	3387.7	96.24	3387.7	96.24	19	27.12
Average					*95.99*		*96.00*	*19.0*	*7.05*

Table 3.7: Root node lower bounds LB_4 with the Gomory inequalities added

3.9 Lift and project cuts

Another category of cuts that can be generated for any MIP are the *lift and project cuts*, originating from the area of disjunctive programming. They were introduced for a cutting plane algorithm for mixed 0-1 programs in Balas, Ceria & Cornuéjols (1993), and have

also been used in a branch and cut setup (Balas, Ceria & Cornuéjols (1996)).[26] The principle of the technique is shown in figure 3.19 and consists of 'lifting' the problem to a higher dimensional space (by introducing new variables), finding some inequality that is valid for the higher dimensional problem and then 'projecting' these inequalities back to the original space to obtain a new valid inequality.

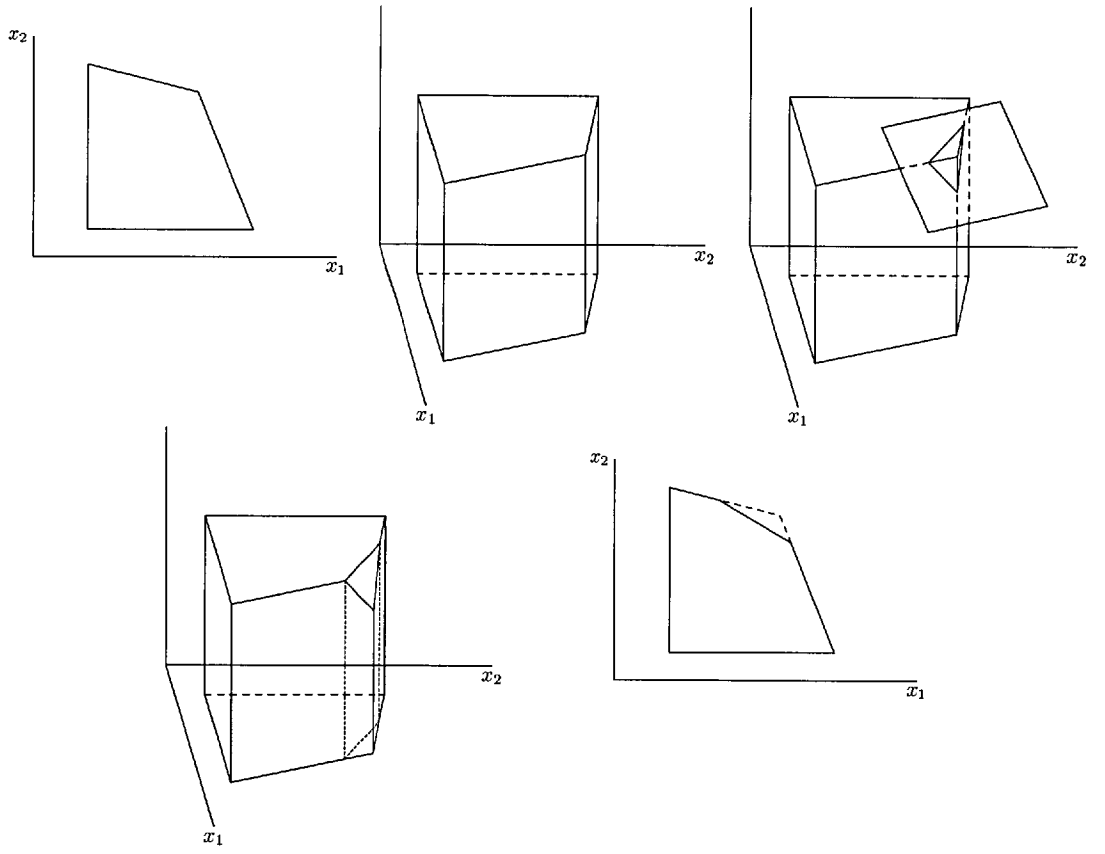


Figure 3.19: An outline of the lift and project technique

The introduction of the new variables (the 'lifting'), for 0-1 MIPs is achieved as follows.

The lift and project procedure is summarised below:

1. Choose a binary variable ξ_j which has a fractional value.
2. Multiply all constraints with ξ_j and add to the formulation.
3. Multiply all constraints with $(1 - \xi_j)$ and add to the formulation.

[26]The lift and project procedure is now being described more and more in standard MIP texts, e.g., Johnson, Nemhauser & Savelsbergh (2000).

4. Replace $(\xi_j)^2$ with ξ_j, since ξ_j can only be 0 or 1.

5. Replace $\xi_k\xi_j$ with a new variable ψ_k say, for all $k \neq j$, where $\psi_k \in \{0,1\}$.

6. If the MIP also has continuous variables, x_k say, replace $\xi_j x_k$, with a new variable $y_k \geq 0$.

7. Add the following inequalities relating ξ, ψ, x and y:

   (a) $y_k \leq x_k$

   (b) $\psi_k \leq \xi_k$

   (c) $\psi_k \leq \xi_j$

   (d) $\psi_k \geq \xi_k + \xi_j - 1$

   for all k.

Solving the LP relaxation obtained by adding the inequalities above will always cut off the current optimal LP point (provided that the variable ξ_j had a fractional value). Performing the above procedure, however, is impractical for every LP produced. A procedure is necessary that enables us to dispose of the added variables and express the cuts implied by the lift and project process in terms of the original variables only.

This procedure (the projection part of the lift and project procedure) requires the elimination of the added variables using linear combinations of the constraints and is possibly the most difficult part. Let $\mathbf{x}$ be the vector containing all the variables of the MIP (binary and continuous). In Balas, Ceria & Cornuéjols (1993) and Balas, Ceria & Cornuéjols (1996), it is shown that obtaining a lift and project valid inequality $\boldsymbol{\alpha}^T\mathbf{x} \geq \beta$ (in the original space) amounts to solving a (quite large) linear program. Assume that the original problem is $\mathbf{A}^T\mathbf{x} \geq \mathbf{b}$, where the matrix $\mathbf{A}$ also contains the upper bounds of the variables $\mathbf{x}$, and let $\bar{\mathbf{x}}$ be the current LP solution with the binary variable x_j having a factional value. Then by optimising the LP shown in **M10** with respect to the variables $(\boldsymbol{\alpha}, \beta)$ will produce the required violated inequality.

MODEL M10

$$\max \quad \beta - \boldsymbol{\alpha}^T \bar{\mathbf{x}} \tag{M10.1}$$

subject to

$$\boldsymbol{\alpha} - \mathbf{A}^T\mathbf{u} - u_0 e_j \geq 0 \tag{M10.2}$$

$$\boldsymbol{\alpha} - \mathbf{A}^T\mathbf{v} - v_0 e_j \geq 0 \tag{M10.3}$$

$$\mathbf{b}^T\mathbf{u} = \beta \tag{M10.4}$$

$$\mathbf{b}^T\mathbf{v} + v_0 = \beta \tag{M10.5}$$

$$(\boldsymbol{\alpha}, \beta) \in S \tag{M10.6}$$

$$\mathbf{u},\ \mathbf{v} \geq 0 \tag{M10.7}$$

$$u_0,\ v_0 \in \mathbb{R} \tag{M10.8}$$

THE GENERATION OF LIFT AND PROJECT CUTS (A)

The objective function seeks to find the lift and project cut that maximises the distance of the current point $\bar{\mathbf{x}}$ from the cut to be generated, in this way the 'deepest' cut will be produced. The vector e_j is the j-th unit vector and the set S corresponds to a normalisation set defining the 'allowed' values for $\boldsymbol{\alpha}$, β. Many choices are possible, but the simplest rule is to require β to be 1 or -1. The advantage of this choice is that the LP **M10** can be simplified by eliminating the α's from the formulation, the disadvantage is that the LP obtained may not have an optimal solution for some types of MIPs (Balas, Ceria & Cornuéjols (1993)).

MODEL M11

$$\min \quad \mathbf{u}^T\mathbf{A}\bar{\mathbf{x}} + u_0\bar{x}_j \tag{M11.1}$$

subject to

$$\mathbf{A}^T\mathbf{u} - \mathbf{A}^T\mathbf{v} + (u_0 - v_0)e_j \geq 0 \tag{M11.2}$$

$$\mathbf{b}^T\mathbf{u} = \beta \tag{M11.3}$$

$$\mathbf{b}^T\mathbf{v} + v_0 = \beta \tag{M11.4}$$

$$\mathbf{u},\ \mathbf{v} \geq 0 \tag{M11.5}$$

$$u_0,\ v_0 \in \mathbb{R} \tag{M11.6}$$

THE GENERATION OF LIFT AND PROJECT CUTS (B)

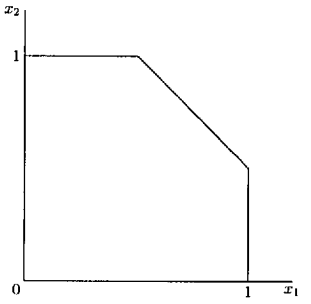
The reduced LP for the case where β is 1 or -1 is presented in **M11**. Using this formulation, the value of $\boldsymbol{\alpha}$ can be obtained from the values of $\mathbf{u}$ and u_0 using (**M10**.2) and (**M10**.3) via $\alpha_k = \max\{\alpha_k^1, \alpha_k^2\}$ where

$$\boldsymbol{\alpha}^1 = \mathbf{A}^T\mathbf{u} + u_0 e_j, \qquad \boldsymbol{\alpha}^2 = \mathbf{A}^T\mathbf{v} - v_0 e_j. \tag{3.64}$$

Recently there have been further refinements to the lift and project procedure for the generation of cuts. Balas (1997) presents a modification to the technique which, at each step, does not generate only the 'deepest' lift and project cut, but instead a class of cuts. This procedure will possibly be beneficial for large MIPs, where a maximum number of valid inequalities is needed and the solution of several, large LPs is computationally costly.

3.9.1 Numerical example

A numerical demonstration of the lift and project cuts for the URNDP is almost impossible due to the large number of constraints and variables present in the formulation. We present, instead, a shorter, fully solved example to demonstrate the generation of the lift and project cuts as well as their graphical interpretation. Consider the integer program with constraints as shown in figure 3.20, and assume that the current solution point



$$2x_1 + 2x_2 \leq 3$$
$$0 \leq x_1,\ x_2 \leq 1$$
$$x_1,\ x_2 \text{ binary.}$$

Figure 3.20: Example of the lift and project technique (A)

lies at the corner $\bar{\mathbf{x}} = (1/2, 1)$. We perform the lifting on the fractional variable x_1 by adding the lifted constraints (applying steps 1–7 of the lifting algorithm of the previous section) and introducing the variable $y = x_1x_2$. The new program and the polyhedron they define are shown in figure 3.21.

From the figure, projecting the polyhedron back to (x_1, x_2) space will induce the lift and project cut

$$x_1 + 2x_2 \leq 2. \tag{3.65}$$

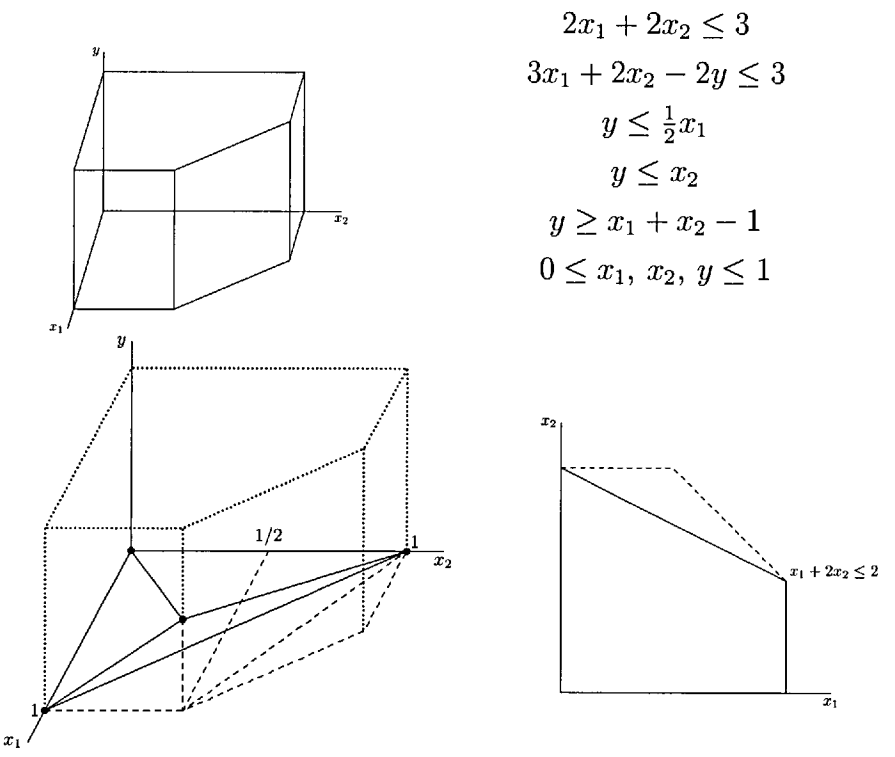


Figure 3.21: Example of the lift and project technique (B)

To see how to obtain this projection using the LP method, we need to calculate the LP constraint matrix $\mathbf{A}$ that includes the upper bounds. For this example, we have $\mathbf{A} = \begin{pmatrix} -1 & 0 & -2 \\ 0 & -1 & -2 \end{pmatrix}^T$, $\mathbf{b} = (-1\ -1\ -3)^T$, $\bar{\mathbf{x}} = \begin{pmatrix} 1/2 \\ 1 \end{pmatrix}$ and we set $\beta = -1$.

The linear program **M11** for this example is the following:

$$\min \quad -\frac{1}{2}u_1 - u_2 - 3u_3 + \frac{1}{2}u_0,$$

subject to:

$$\begin{aligned} -u_1 - 2u_3 + v_1 + 2v_3 + u_0 - v_0 &\geq 0 \\ -u_2 - 2u_3 + v_2 + 2v_3 &\geq 0 \\ -u_1 - u_2 - 3u_3 &= -1 \\ -v_1 - v_2 - 3v_3 + v_0 &= -1 \\ \mathbf{u},\ \mathbf{v} \geq 0; \qquad u_0,\ v_0 &\in \mathbb{R} \end{aligned}$$

The optimal solution to the above LP has $u_2 = 1$, $v_3 = 1/2$, $u_0 = -1/2$, $v_0 = 1/2$ and all other variables zero (and an objective function value of $= -5/4$). This gives a value

for $\boldsymbol{\alpha}$ of $\begin{pmatrix} -1/2 \\ -1 \end{pmatrix}$, and therefore the cut obtained ($\boldsymbol{\alpha}^T\mathbf{x} \geq \beta$) is

$$-\frac{1}{2}x_1 - x_2 \geq -1, \tag{3.66}$$

i.e., the same as the one obtained graphically in figure 3.21.

3.9.2 Computational results

We implement the lift and project procedure to generate cuts at the root node. The binary variable on which the technique is applied is chosen to be the one with a value nearest to 0.5 among all binary variables that have fractional values in the LP relaxation, and an upper limit of 10 cuts was imposed.

The above implementation procedure for the generation of lift and project inequalities for the URNDP with flows was applied to a number of test problem instances so as to improve the LP-based lower bound. Sample computational results are presented in table 3.8. The interpretation of the symbols used is the following:

- LB_4 Lower bound obtained after adding the flow balance, trivial, degree, maximum flow, partition, r-cover and Gomory inequalities.
- LB_5 Lower bound obtained after adding the flow balance, trivial, degree, maximum flow, partition, r-cover, Gomory and lift-and-project inequalities.
- z_{opt} The cost of the optimal solution to the URNDP or the cost of the best feasible solution found at the end of the Branch and Cut algorithm.
- $\%LB_4$ Percentage ratio LB_4/z_{opt}.
- $\%LB_5$ Percentage ratio LB_5/z_{opt}.
- *# cuts* The number of lift and project cuts generated.
- $t_{\text{lift\&pro}}$ The time (in seconds) taken for the generation of the lift and project cuts.

From the results it is obvious that the lift and project inequalities have a small effect on the improvement of the lower bound for the URNDP with flows. Although their contribution could be improved by generating more inequalities, this procedure is not efficient because of the relatively long time that is required for their generation. For the largest problems (40 and 50 vertices) the time taken for their generation is considerably larger and the improvement on the lower bound is relatively small; for this reason it was decided not to generate lift and project inequalities for the test problems with 40 vertices or more.

Problem	*Vertices*	*Edges*	z_{opt}	LB_4	$\%\, LB_4$	LB_5	$\%\, LB_5$	*# cuts*	$t_{\text{lift\&pro}}$
g1n12g001i5	12	47	17434	17315.6	99.32	17315.6	99.32	10	1.39
g1n12g01i2	12	55	2177	2130.8	97.88	2130.9	97.88	10	1.41
g1n12g1i3	12	49	370	360.3	97.37	360.3	97.37	4	0.36
g1n15g00503	15	72	3432	3390.70	98.80	3391.93	98.83	10	2.01
g1n15g01i4	15	85	1475	1389.6	94.21	1390.5	94.27	10	2.51
g1n20g01i3	20	143	2582	2445.7	94.72	2445.7	94.72	10	9.00
g1n25g001i4	25	235	17713	17166.71	96.92	17166.82	96.92	10	36.60
g1n25g01i4	25	220	2586	2419.6	93.56	2419.6	93.56	10	29.18
g1n30g2i2	30	337	668	662.6	99.19	662.6	99.19	10	46.88
g1n30g01i2	30	329	2910	2708.9	93.09	2709.0	93.09	10	65.64
Average					*96.51*		*96.52*	*9.4*	*19.50*

Table 3.8: Root node lower bounds LB_5 with the lift and project inequalities added

3.10 Description of a lower bound ascent algorithm

This section outlines the overall algorithm developed to obtain the best lower bound for the URNDP with flows, computed at the root node of a Branch and Cut method used to solve the problem to optimality.

The lower bound ascent algorithm

1. Solve the LP relaxation **M8**.

2. Add the maximum flow inequalities that correspond to $|S| = 1$ (model **M9**) and any other violated maximum flow inequalities with $|S| \geq 2$ that can be found by the separation algorithm (section 3.6.1). Every time a violated maximum flow inequality is found it is added to the formulation, the LP is re-optimised, and the process is repeated. We stop the generation of maximum flow inequalities when no more violated inequalities can be found, or when the upper limit on the number of violated inequalities (set to $|E|$, the number of edges) is reached.

3. Add violated partition inequalities (section 3.5.1). At each step a violated partition inequality is added, the LP is re-optimised and the process is iterated. We stop when no more inequalities can be detected.

4. Add violated r-cover inequalities (3.7). One inequality is added at a time, then the LP is re-optimised, and the process is repeated. We stop when no more inequalities

can be found by the separation algorithm.

5. Add violated Gomory inequalities, as described in section 3.8, and re-optimise the LP.

6. Add violated lift and project inequalities, as described in section 3.9.2, and re-optimise the LP. This constitutes the final bound of the lower bound ascent algorithm. (This step is not performed for the larger instances with 40 or 50 vertices.)

3.10.1 Extensive computational results for the URNDP with flows

This section contains computational results obtained at the root node of the Branch and Cut algorithm of the URNDP with flows. The test problems used were randomly generated as described in section 2.7, and consist of networks with 12 to 50 vertices with different ratios between the fixed and variable costs represented by the parameter γ. For each combination of network size and cost ratio five instances were generated. The results were obtained on a 600MHz *Pentium III* PC, using C++ and the CPLEX 7.5 (ILOG™ S.A. (2001)) LP solver.

The notation used in the tables is the following:

LB_0 Lower bound obtained at the root node after adding the flow balance, trivial and degree inequalities.

LB_1 Lower bound obtained at the root node after adding the flow balance, trivial, degree and maximum flow inequalities.

LB_2 Lower bound obtained at the root node after adding the flow balance, trivial, degree, maximum flow and partition inequalities.

LB_3 Lower bound obtained at the root node after adding the flow balance, trivial, degree, maximum flow, partition and r-cover inequalities.

LB_4 Lower bound obtained at the root node after adding the flow balance, trivial, degree, maximum flow, partition, r-cover and Gomory inequalities.

LB_5 Lower bound obtained at the root node after adding the flow balance, trivial, degree, maximum flow, partition, r-cover, Gomory and lift-and-project inequalities.

z_{opt} The cost of the optimal solution to the URNDP or the cost of the best feasible solution found at the end of the Branch and Cut algorithm.

$\%LB_0$ Percentage ratio LB_0/z_{opt}.

$\%LB_1$ Percentage ratio LB_1/z_{opt}.

$\%LB_2$ Percentage ratio LB_2/z_{opt}.

$\%LB_3$ Percentage ratio LB_3/z_{opt}.

$\%LB_4$ Percentage ratio LB_4/z_{opt}.

$\%LB_5$ Percentage ratio LB_5/z_{opt}.

t_{mf} The computational time (in seconds) required for the generation of maximum flow inequalities.

t_{part} The computational time (in seconds) required for the generation of partition inequalities.

t_{rc} The computational time (in seconds) required for the generation of r-cover inequalities.

t_{gom} The computational time (in seconds) required for the generation of Gomory inequalities.

$t_{\text{lift\&pro}}$ The computational time (in seconds) required for the generation of lift and project flow inequalities.

t_{root} The total computational time (in seconds) at the root node.

Tables 3.9 and 3.10 show, for each category of valid inequality, the percentage errors, the number of cuts generated and the time taken for their generation. Each row cor-

responds to a different network size/cost ratio combination, and the results represent the *average* values among the five test instances generated for each case. Table 3.9 shows the results for the test networks with 12 to 25 vertices, whereas table 3.10 shows the results for the larger networks (30 to 50 vertices).

Tables 3.11 and 3.12 present the bounds obtained during the lower bound ascent algorithm and the optimal solutions (where known[27]) for selected instances. For the same instances, tables 3.13 and 3.14 contain the number of cuts and the separation time for each category of valid inequality generated.

[27]A time limit of 10 hours was imposed on the Branch and Cut algorithm. If the algorithm did not terminate within this time, then z_{opt} represents the best feasible solution found by the algorithm.

			Lower bound ascent						*Cut details at root node*										
			initial	*max fl.*	*partit.*	*r-cov.*	*Gom.*	*l.& pro.*	*max flow*		*partition*		*r-cover*		*Gomory*		*lift and project*		
Vertices	*Edges*	γ	$\%LB_0$	$\%LB_1$	$\%LB_2$	$\%LB_3$	$\%LB_4$	$\%LB_5$	# *cuts*	t_{mf}	# *cuts*	t_{part}	# *cuts*	t_{rc}	# *cuts*	t_{gom}	# *cuts*	$t_{\text{lift\&pro}}$	t_{root}
12	47.0	0.01	99.78	99.83	99.84	99.85	99.86	99.86	86.8	0.01	4.2	0.02	0.2	0.02	7.6	0.17	4.0	0.44	0.8
12	51.4	0.05	97.26	98.06	98.19	98.21	98.27	98.27	94.8	0.01	0.2	0.01	0.2	0.02	19.0	0.38	8.8	1.04	1.6
12	49.6	0.1	94.33	96.28	97.15	97.24	97.28	97.42	91.6	0.02	1.4	0.02	1.0	0.02	19.0	0.44	10.0	1.21	1.9
12	49.8	1	94.48	98.89	98.89	98.89	99.04	99.06	92.0	0.01	0.0	0.01	0.0	0.02	11.8	0.20	4.2	0.34	0.7
12	47.6	2	96.57	99.69	99.78	99.78	99.96	99.97	87.8	0.01	0.4	0.01	0.0	0.02	7.4	0.09	0.8	0.06	0.3
Average			*96.48*	*98.55*	*98.77*	*98.79*	*98.82*	*98.92*	*90.6*	*0.01*	*1.2*	*0.01*	*0.3*	*0.02*	*13.0*	*0.26*	*5.6*	*0.62*	*1.1*
15	78.0	0.01	96.88	97.04	99.35	99.35	99.36	99.36	146.0	0.03	1.8	0.04	0.2	0.03	18.8	0.66	8.0	1.91	2.9
15	76.6	0.05	95.58	96.50	97.35	97.38	97.40	97.41	145.4	0.03	5.4	0.04	0.4	0.03	19.0	0.57	10.0	2.08	3.0
15	80.4	0.1	93.46	95.24	95.47	95.51	95.57	95.79	152.4	0.03	1.2	0.03	0.4	0.04	19.0	0.63	9.4	2.30	3.3
15	80.2	1	93.33	98.29	98.38	98.38	98.52	98.64	149.6	0.02	4.0	0.03	0.0	0.04	18.6	0.54	6.8	1.46	2.3
15	78.2	2	95.26	99.48	99.48	99.48	99.68	99.68	145.4	0.02	0.0	0.01	0.0	0.03	11.0	0.28	4.8	0.66	1.2
Average			*94.90*	*97.31*	*98.00*	*98.02*	*98.11*	*98.18*	*147.8*	*0.03*	*2.5*	*0.03*	*0.2*	*0.03*	*17.3*	*0.54*	*7.8*	*1.68*	*2.5*
20	140.0	0.01	98.15	98.38	98.73	98.97	98.97	98.97	266.2	0.06	2.8	0.08	1.0	0.11	19.0	1.38	10.0	7.16	9.4
20	143.4	0.05	95.60	96.43	96.71	96.73	96.74	96.76	274.2	0.08	2.0	0.07	0.4	0.10	19.0	1.48	10.0	7.95	10.4
20	145.0	0.1	93.59	95.82	95.97	96.11	96.13	96.15	276.4	0.07	8.6	0.11	1.0	0.12	19.0	1.26	10.0	8.41	10.5
20	145.4	1	93.26	98.36	98.38	98.38	98.49	98.61	276.8	0.05	4.4	0.06	0.0	0.08	19.0	0.92	7.8	3.66	5.2
20	140.6	2	95.12	98.73	98.77	98.77	98.97	98.97	268.6	0.06	0.4	0.04	0.0	0.07	18.6	0.98	10.0	4.79	6.3
Average			*95.15*	*97.54*	*97.71*	*97.79*	*97.86*	*97.89*	*272.4*	*0.06*	*3.6*	*0.07*	*0.5*	*0.10*	*18.9*	*1.20*	*9.6*	*6.93*	*8.4*
25	231.6	0.01	97.72	97.97	98.12	98.12	98.12	98.12	447.4	0.17	1.8	0.15	0.8	0.24	19.0	3.35	10.0	23.28	29.1
25	220.4	0.05	94.73	95.72	95.79	95.93	95.93	95.93	425.4	0.14	3.2	0.13	1.2	0.26	19.0	2.70	10.0	22.80	27.9
25	227.6	0.1	91.48	93.73	94.11	94.11	94.12	94.23	437.8	0.13	4.6	0.15	0.4	0.25	19.0	2.53	10.0	23.74	28.0
25	226.6	1	94.45	98.70	98.76	98.76	98.82	98.83	436.4	0.12	0.2	0.06	0.2	0.15	18.8	1.71	9.2	11.97	15.1
25	224.4	2	94.75	99.10	99.12	99.14	99.18	99.20	432.8	0.11	0.4	0.06	0.2	0.20	18.6	1.68	9.4	13.12	16.5
Average			*94.62*	*97.04*	*97.18*	*97.21*	*97.23*	*97.26*	*436.0*	*0.13*	*2.0*	*0.11*	*0.6*	*0.22*	*18.9*	*2.39*	*9.7*	*18.98*	*23.3*

Table 3.9: Computational results for the URNDP: 12–25 vertices (averages over five instances)

(continued.)

			Lower bound ascent						*Cut details at root node*									
			initial	*max fl.*	*partit.*	*r-cov.*	*Gom.*	*l.& pro.*	*max flow*		*partition*		*r-cover*		*Gomory*		*lift and project*		
Vertices	*Edges*	γ	$\%LB_0$	$\%LB_1$	$\%LB_2$	$\%LB_3$	$\%LB_4$	$\%LB_5$	*# cuts*	t_{mf}	*# cuts*	t_{part}	*# cuts*	t_{rc}	*# cuts*	t_{gom}	*# cuts*	$t_{\text{lift\&pro}}$	t_{root}
30	323.2	0.01	96.50	96.78	97.28	97.49	97.49	97.49	627.2	0.39	17.8	0.67	1.0	0.62	19.0	6.13	10.0	59.52	67.9
30	326.0	0.05	94.43	95.50	95.86	96.13	96.16	96.16	632.8	0.37	8.8	0.35	1.0	0.62	19.0	4.80	10.0	55.17	61.8
30	330.2	0.1	89.95	91.98	92.66	92.70	92.70	92.70	641.2	0.34	9.2	0.41	2.2	0.76	19.0	5.32	10.0	68.35	75.7
30	329.0	1	91.83	97.46	97.55	97.55	97.58	97.58	637.0	0.28	4.6	0.22	0.0	0.37	19.0	3.40	10.0	39.52	44.1
30	330.8	2	94.92	98.99	99.03	99.03	99.09	99.09	641.4	0.25	1.4	0.10	0.0	0.36	19.0	2.99	10.0	40.01	46.1
Average			*93.53*	*96.14*	*96.48*	*96.58*	*96.60*	*96.60*	*635.9*	*0.33*	*8.4*	*0.35*	*0.8*	*0.55*	*19.0*	*4.53*	*10.0*	*52.51*	*59.12*
40	581.2	0.01	95.66	95.96	95.97	95.97	95.97	—	1133.2	1.11	13.4	1.32	1.2	2.02	19.0	16.11	—	—	21.7
40	586.4	0.05	90.95	92.09	92.10	92.25	92.25	—	1143.8	0.90	2.0	0.51	0.4	1.15	19.0	11.74	—	—	19.2
40	579.4	0.1	88.99	91.28	91.36	91.41	91.41	—	1132.0	0.91	10.0	0.85	0.4	1.72	19.0	11.30	—	—	15.4
40	583.6	1	92.01	97.41	97.47	97.47	97.49	—	1139.2	0.80	4.6	0.45	0.0	1.00	19.0	7.39	—	—	10.2
40	577.6	2	94.24	98.62	98.64	98.64	98.66	—	1128.6	0.76	0.4	0.25	0.0	0.96	19.0	7.52	—	—	9.9
Average			*92.37*	*95.07*	*95.11*	*95.15*	*95.16*	—	*1135.4*	*0.90*	*6.1*	*0.68*	*0.4*	*1.37*	*19.0*	*10.81*	—	—	*15.3*
50	918.6	0.01	94.58	94.81	94.91	94.91	94.91	—	1803.6	2.83	3.2	1.79	1.2	4.88	19.0	37.73	—	—	50.0
50	917.8	0.05	88.53	89.78	89.80	89.86	89.86	—	1799.0	2.58	6.4	1.89	1.2	4.48	19.0	34.98	—	—	45.9
50	914.2	0.1	86.10	88.26	88.31	88.32	88.32	—	1791.6	2.53	6.2	1.77	1.8	7.77	19.0	32.37	—	—	46.3
50	929.8	1	91.32	97.09	97.09	97.09	97.09	—	1825.2	2.26	1.4	0.71	0.0	2.59	19.0	22.12	—	—	38.4
50	916.0	2	94.72	98.88	98.88	98.88	98.89	—	1798.8	2.06	0.2	0.50	0.0	2.42	19.0	16.02	—	—	38.1
Average			*91.05*	*93.76*	*93.80*	*93.81*	*93.81*	—	*1803.6*	*2.45*	*3.5*	*1.33*	*0.8*	*4.43*	*19.0*	*28.64*	—	—	*43.7*

Table 3.10: Computational results for the URNDP: 30–50 vertices (averages over five instances)

					Lower bound values											
					initial		*max flow*		*partition*		*r-cover*		*Gomory*		*lift & project*	
Problem	*Vertices*	*Edges*	γ	z_{opt}	z_{LB_0}	$\%LB_0$	z_{LB_1}	$\%LB_1$	z_{LB_2}	$\%LB_2$	z_{LB_3}	$\%LB_3$	z_{LB_4}	$\%LB_4$	z_{LB_5}	$\%LB_5$
g1n12g001i5	12	47	0.01	17434	17282.0	99.13	17300.5	99.23	17300.8	99.24	17314.9	99.32	17315.6	99.32	17315.6	99.32
g1n12g005i3	12	50	0.05	3372	3251.6	96.43	3256.1	96.56	3278.3	97.22	3278.3	97.22	3279.3	97.25	3279.3	97.25
g1n12g01i2	12	55	0.1	2177	2084.0	95.73	2124.6	97.59	2124.6	97.59	2129.9	97.84	2130.8	97.88	2131.0	97.89
g1n12g1i3	12	49	1	370	346.9	93.77	359.6	97.18	359.6	97.18	359.6	97.18	360.3	97.37	360.3	97.37
g1n12g2i1	12	49	2	381	372.0	97.65	381.0	100.00	381.0	100.00	381.0	100.00	381.0	100.00	381.0	100.00
g1n15g001i4	15	79	0.01	17136	16973.9	99.05	17002.4	99.22	17002.4	99.22	17002.4	99.22	17004.8	99.23	17004.9	99.24
g1n15g005i3	15	72	0.05	3432	3343.9	97.43	3385.5	98.65	3385.5	98.65	3390.5	98.79	3390.7	98.80	3391.9	98.83
g1n15g01i4	15	85	0.1	1475	1365.0	92.55	1387.1	94.04	1388.4	94.13	1388.6	94.14	1389.6	94.21	1390.5	94.27
g1n15g1i3	15	78	1	344	323.6	94.07	338.1	98.27	338.1	98.27	338.1	98.27	339.4	98.66	339.4	98.66
g1n15g2i4	15	78	2	320	299.2	93.49	317.0	99.08	317.0	99.08	317.0	99.08	318.4	99.49	318.4	99.49
g1n20g001i2	20	145	0.01	18008	17819.0	98.95	17860.2	99.18	17860.3	99.18	17860.3	99.18	17860.6	99.18	17860.6	99.18
g1n20g005i3	20	142	0.05	4426	4162.4	94.04	4196.6	94.82	4214.0	95.21	4218.1	95.30	4218.4	95.31	4223.6	95.43
g1n20g01i3	20	143	0.1	2582	2394.9	92.75	2444.3	94.67	2445.5	94.71	2445.5	94.71	2445.7	94.72	2445.7	94.72
g1n20g1i1	20	144	1	655	608.8	92.95	640.4	97.76	640.7	97.81	640.7	97.81	640.8	97.83	644.1	98.34
g1n20g2i4	20	149	2	580	554.3	95.56	572.2	98.65	572.2	98.65	572.2	98.65	572.3	98.67	572.3	98.67
g1n25g001i5	25	246	0.01	19121	18954.0	99.13	18976.5	99.24	18976.5	99.24	18976.5	99.24	18976.6	99.24	18976.6	99.24
g1n25g005i4	25	219	0.05	5356	4985.9	93.09	5032.8	93.97	5041.1	94.12	5041.5	94.13	5042.0	94.14	5042.0	94.14
g1n25g01i4	25	230	0.1	2586	2370.4	91.66	2419.2	93.55	2419.2	93.55	2419.4	93.56	2419.6	93.56	2419.6	93.56
g1n25g1i3	25	224	1	654	615.8	94.15	637.9	97.54	637.9	97.54	637.9	97.54	638.0	97.55	638.0	97.55
g1n25g2i1	25	232	2	468	437.9	93.57	461.5	98.62	461.6	98.62	461.6	98.62	461.6	98.63	461.8	98.68

Table 3.11: Computational results for the URNDP: Test instances with 12–25 vertices (lower bounds)

(ctd.)

					Lower bound values											
					initial		*max flow*		*partition*		*r-cover*		*Gomory*		*lift & project*	
Problem	*Vertices*	*Edges*	γ	z_{opt}	z_{LB_0}	$\%LB_0$	z_{LB_1}	$\%LB_1$	z_{LB_2}	$\%LB_2$	z_{LB_3}	$\%LB_3$	z_{LB_4}	$\%LB_4$	z_{LB_5}	$\%LB_5$
g1n30g001i2	30	328	0.01	22440	21768.0	97.01	21808.2	97.18	21808.2	97.18	21808.3	97.18	21808.3	97.18	21808.4	97.19
g1n30g005i4	30	321	0.05	5675	5358.3	94.42	5406.3	95.26	5406.3	95.26	5433.9	95.75	5433.9	95.75	5433.9	95.75
g1n30g01i2	30	329	0.1	2910	2649.6	91.05	2707.1	93.03	2708.7	93.08	2708.8	93.08	2708.9	93.09	2709.0	93.09
g1n30g1i2	30	314	1	850	773.5	91.00	836.0	98.35	836.0	98.35	836.0	98.35	836.7	98.43	836.7	98.43
g1n30g2i2	30	337	2	668	635.2	95.09	661.0	98.94	662.3	99.15	662.3	99.15	662.6	99.19	662.6	99.19
g1n40g001i3	40	590	0.01	23473†	22516.0	95.92	22574.7	96.17	22577.8	96.19	22577.8	96.19	22577.8	96.19	—	—
g1n40g005i5	40	575	0.05	6193†	5572.1	89.97	5660.9	91.41	5660.9	91.41	5672.5	91.60	5673.1	91.60	—	—
g1n40g01i2	40	606	0.1	3262†	2888.5	88.55	2951.2	90.47	2951.2	90.47	2951.2	90.47	2951.3	90.47	—	—
g1n40g1i4	40	579	1	956	893.8	93.50	930.5	97.33	932.5	97.55	932.5	97.55	932.8	97.57	—	—
g1n40g2i3	40	572	2	842	800.3	95.05	829.5	98.51	829.5	98.51	829.5	98.51	829.5	98.51	—	—
g1n50g001i1	50	915	0.01	29667†	28168.0	94.95	28206.4	95.08	28207.7	95.08	28207.9	95.08	28207.9	95.08	—	—
g1n50g005i5	50	917	0.05	6851†	6065.2	88.53	6170.5	90.07	6172.3	90.09	6172.3	90.09	6172.3	90.09	—	—
g1n50g01i1	50	907	0.1	3834†	3260.0	85.03	3380.8	88.18	3386.3	88.32	3387.7	88.36	3387.7	88.36	—	—
g1n50g1i2	50	955	1	1179	1067.2	90.52	1134.5	96.23	1134.6	96.24	1134.6	96.24	1134.6	96.24	—	—
g1n50g2i1	50	903	2	881	834.5	94.72	870.4	98.80	870.6	98.82	870.6	98.82	870.6	98.82	—	—

† this symbol denotes the best solution value found by the algorithm within the time limit.

Table 3.12: Computational results for the URNDP: Test instances with 30–50 vertices (lower bounds)

	Cut details at root node										
	max flow		*partition*		*r-cover*		*Gomory*		*lift & project*		
Problem	*# cuts*	t_{mf}	*# cuts*	t_{part}	*# cuts*	t_{rc}	*# cuts*	t_{gom}	*# cuts*	$t_{lift\&pro}$	t_{root}
g1n12g001i5	88	0.01	20	0.06	1	0.03	19	0.38	10	1.39	2.02
g1n12g005i3	91	0.01	1	0.01	0	0.02	19	0.39	10	1.19	1.76
g1n12g01i2	102	0.02	0	0.01	1	0.02	19	0.48	10	1.41	2.14
g1n12g1i3	91	0.01	0	0.01	0	0.02	19	0.30	4	0.36	0.83
g1n12g2i1	89	0.01	0	0.00	0	0.02	0	0.03	0	0.01	0.14
g1n15g001i4	146	0.02	0	0.02	0	0.03	18	0.62	8	1.61	2.51
g1n15g005i3	140	0.04	0	0.02	1	0.03	19	0.57	10	1.87	2.77
g1n15g01i4	163	0.04	1	0.02	1	0.05	19	0.63	10	2.51	3.49
g1n15g1i3	145	0.02	0	0.02	0	0.03	19	0.43	10	1.51	2.16
g1n15g2i4	146	0.02	0	0.02	0	0.02	18	0.41	10	1.40	2.03
g1n20g001i2	273	0.06	1	0.05	0	0.09	19	1.69	10	7.48	9.80
g1n20g005i3	274	0.08	5	0.10	1	0.09	19	1.73	10	10.11	12.71
g1n20g01i3	273	0.07	2	0.06	1	0.08	19	1.47	10	9.00	11.26
g1n20g1i1	272	0.05	1	0.04	0	0.07	19	0.90	10	4.40	5.78
g1n20g2i4	286	0.07	0	0.04	0	0.08	19	1.20	10	5.90	7.61
g1n25g001i5	474	0.17	0	0.10	0	0.23	19	3.72	10	25.25	30.35
g1n25g005i4	424	0.12	7	0.19	2	0.26	19	3.16	10	18.53	23.24
g1n25g01i4	441	0.12	0	0.07	1	0.22	19	2.68	10	29.18	33.05
g1n25g1i3	431	0.14	0	0.06	0	0.15	19	2.00	10	15.94	18.95
g1n25g2i1	448	0.11	1	0.07	0	0.17	19	1.56	10	12.13	14.62

Table 3.13: Computational results for the URNDP: Test instances with 12–25 vertices (number of cuts)

(ctd.)

	Cut details at root node										
	max flow		*partition*		*r-cover*		*Gomory*		*lift & project*		
Problem	*# cuts*	t_{mf}	*# cuts*	t_{part}	*# cuts*	t_{rc}	*# cuts*	t_{gom}	*# cuts*	$t_{\text{lift\&pro}}$	t_{root}
g1n30g001i2	637	0.31	20	0.69	2	0.69	19	6.36	10	63.21	73.50
g1n30g005i4	626	0.43	20	0.62	2	1.18	19	4.24	10	57.73	65.77
g1n30g01i2	639	0.44	1	0.19	1	0.69	19	5.05	10	65.64	73.69
g1n30g1i2	603	0.25	0	0.12	0	0.32	19	2.68	10	24.31	28.61
g1n30g2i2	653	0.25	6	0.25	0	0.31	19	3.61	10	46.88	52.37
g1n40g001i3	1152	1.06	2	0.69	2	1.93	19	15.02	—	—	22.82
g1n40g005i5	1122	0.75	0	0.25	1	1.00	19	9.72	—	—	14.28
g1n40g01i2	1186	1.18	0	0.43	0	1.12	19	11.72	—	—	17.83
g1n40g1i4	1135	0.81	2	0.38	0	1.18	19	7.54	—	—	12.28
g1n40g2i3	1117	0.75	0	0.19	0	0.93	19	8.98	—	—	12.90
g1n50g001i1	1799	2.74	4	1.50	2	5.43	19	34.91	—	—	54.30
g1n50g005i5	1793	2.24	3	1.18	2	7.04	19	31.23	—	—	49.50
g1n50g01i1	1777	2.18	4	1.44	4	16.65	19	27.12	—	—	53.86
g1n50g1i2	1876	2.31	4	1.06	0	2.74	19	24.13	—	—	34.97
g1n50g2i1	1774	1.99	1	0.56	0	2.24	19	18.33	—	—	27.49

Table 3.14: Computational results for the URNDP: Test instances with 30–50 vertices (number of cuts)

3.10.2 Discussion

From the computational results of the previous section the following observations can be made. First of all, the generic, problem independent cuts (Gomory and lift-and-project) were the least effective for this problem, producing a small improvement to the lower bound. Among the remaining cuts, the maximum flow cuts produced the biggest relative improvement. The remaining cuts (the partition and the r-cover) were quite useful for some cases but no cuts could generated for some instances. It was observed however, that a larger number of the r-cover cuts were generated during the Branch and Cut algorithm, and this justifies the reason for their generation (see section 4.8.1). For the partition inequalities, only relatively few cuts are generated overall, but they are necessary to ensure feasibility of the solution and therefore they must be included in the algorithm.

The order in which the various cuts were generated is justified as follows: the maximum flow cuts, being the ones with the largest contribution and taking a small amount of time to be obtained were chosen to be generated first. On the other hand, the Gomory and lift and project cuts were left for the end of the lower bound ascent algorithm. This is because their relative contribution to the lower bound was not significantly affected if they were generated before of after another group of cuts, and, thus, it is better to postpone their generation in order to avoid having a complex LP relaxation that will slow down the generation of the other types of valid inequalities. For the remaining two categories (partition and r-cover) it was observed that in some cases a larger number of cuts was obtained (yielding a sightly better final lower bound) if they were added in that order.

We further observe that some of the 'easiest' instances can be solved completely at the root node without a need for branching. For instance 'g1n12g2i1' for example, the solution of the LP relaxation obtained after the maximum flows are added is feasible for the MIP and is therefore the optimal solution.

With respect to the final lower bounds, we can observe that the problems with $\gamma = 0.1$ appear to be the most complex instances (with the largest final % errors) followed by the case $\gamma = 0.05$ and $\gamma = 1$. A similar behaviour is observed for the fixed charge problem (Hochbaum & Segev (1989)).

3.11 Summary

In this chapter a new mixed integer programming formulation for the Uncapacitated Reliable Network Design Problem with flows was developed, followed by a lower bound ascent algorithm for obtaining a lower bound on the objective function value at the root node of the Branch and Cut algorithm. For this purpose, as series of valid inequalities is applied; these inequalities are either general inequalities applicable to all MIPs or specific to the URNDP with flows. The separation algorithms used to detect violated valid inequalities during the execution of the lower bound ascent algorithm were also discussed and examples were presented. Theoretical results regarding the conditions when some of the inequalities are facet defining with respect to the formulation of the URNDP with flows were also derived. The chapter concluded with computational results obtained for several randomly generated test problems. The lower bound ascent procedure constitutes the root node of the Branch and Cut algorithm to solve the URNDP with flows, which is described in the following chapter.

The trees that are slow to grow
bear the best fruit.

Jean-Baptiste Poquelin (Molière)
(1622–1673)

Chapter 4

A branch and cut algorithm for the uncapacitated RNDP with flows

4.1 Introduction

In the last chapter a lower bound ascent procedure at the rood node of a Branch and Cut (B&C) algorithm for the uncapacitated RNDP with flows was described. In this chapter a Branch and Cut algorithm is developed in order to obtain the optimal solution of the problem and its main characteristics are described, namely, branching, variable selection, cut generation and management, and termination conditions. Computational results are also presented at the end of the chapter.

4.2 A general overview of the branch and cut procedure

The general principles of the Branch and Cut procedure for solving integer and mixed integer programs to optimality were presented in section 1.4.4 of the methodological preliminaries.

In summary, Branch and Cut consists of dividing the solution space of a problem into many branches (usually two) obtained by adding a different set of inequalities to the

LP formulation aiming to cut off the current fractional solution without excluding the optimal solution. This is usually done by selecting a variable that violates the integrality constraints, ξ_j say, and creating two subproblems by adding one of the constraints

$$\xi_j \leq \lfloor \bar{\xi}_j \rfloor \qquad \text{and} \qquad \xi_j \geq \lceil \bar{\xi}_j \rceil \tag{4.1}$$

to each subproblem, where $\bar{\xi}_j$ is the current fractional value. In this way the original problem (the root node of the B&C tree) is branched into two nodes, each node containing one of the subproblems generated. If we solve both subproblems to optimality (subject to the integrality constraints) and we pick the best of the two resulting solutions, this will be the required optimal solution to the original problem since no feasible solutions are excluded by (4.1). This process can be repeated, by solving the LP at one of the newly created nodes and branching again on another fractional variable. During this process the best feasible solution found so far (the *incumbent* solution) is stored and it is compared to the bound obtained by solving the LP at one branch: if the LP value is larger (for a minimisation problem) than the value of the incumbent then this branch does not contain a 'better' solution than what we already know and this node of the B&C tree can be *pruned* (or *fathomed*) i.e., eliminated from consideration. This process continues until all the nodes of the B&C tree are pruned, and the incumbent at this stage is the required optimal solution.

While solving a LP relaxation at a B&C node, cuts are generated and added to the formulation. The purpose of these cuts is to improve the bound obtained at the particular node and thus pruning that particular branch more quickly. Generating cuts, however, may be time consuming and since the system can work without the addition of any cuts (as in a branch and bound setup), there ought to exist a balance between cut generation and branching. In other words, as more cuts are added to the LP relaxation the number of B&C nodes needed tends to decrease (and thus reducing the total LP solving time), but at the same time some computational time is required for their generation so a compromise between the two is necessary to obtain the best performance of the B&C algorithm.

In the next sections we present a literature overview of the different aspects that affect the performance of the Branch and Cut algorithm. These strategies will be evaluated numerically for the URNDP with flows in section 4.5.

4.3 Branching strategies

Branching strategies refer to the rules for selecting the variable which will be used to define the two new subproblems. This variable should be one that does not have an integer value in the solution of the current LP relaxation.

For the URNDP we have two choices for the branching variable. We can either choose a binary variable ξ whose value is not 0 or 1 or alternatively we choose a flow variable x with a fractional value. This is because, although the formulation does not require so, the flow variables in the optimal solution will be integer due to the integer data of the problem. Nevertheless, the integrality of the flow variables is automatically guaranteed by the optimisation process, and there is no obvious need to try to impose this in the Branch and Cut. The binary variables on the other hand are in general less likely to take integer values, as the optimisation process will tend to assign them the smallest value as long as the trivial constraints (and the reliability constraints) allow. This is indeed where the difficulty of the URNDP with flows lies: how to force the binary variables to take the value of one if the slightest positive amount of flow passes through the corresponding edge.

For the above reasons, branching is only performed on the binary variables ξ, with two branches created for every node, one containing the constraint $\xi = 0$ and the other the constraint $\xi = 1$.

4.3.1 Variable selection

Variable selection refers to the rules of deciding which binary variable to choose when branching among all the fractional variables present in the LP solution.

There are several rules in the literature on how to select the particular binary variable in order to create the subproblems. The goal is to make the B&C algorithm terminate as fast as possible by branching on variables that are more likely to lead to good bounds, infeasible instances or feasible solutions. Some of the most commonly employed rules are presented below (Jünger, Reinelt & Thienel (1995)):

(i) **[maximum infeasibility]** Select the variable with a value closest to 0.5.

(ii) **[maximum cost]** Select the fractional variable with a the largest objective function coefficient (the largest 'cost').

(iii) A combination of (i) and (ii) above (Padberg & Rinaldi (1991)): Select a variable close to 0.5 with a big objective function coefficient in the following way. Find L and H with

$$L = \min\{0.5, \max\{\bar{\xi}_j \mid \bar{\xi}_j \leq 0.5\}\} \tag{4.2}$$

$$H = \max\{0.5, \min\{\bar{\xi}_j \mid \bar{\xi}_j \geq 0.5\}\}. \tag{4.3}$$

Define

$$C = \{j \in E \mid 0.75L \leq \bar{\xi}_j \leq H + 0.25(1 - H)\} \tag{4.4}$$

as the set of variables that are 'close' to 0.5. Among these variables, select the one with the largest objective function coefficient.

(iv) **[minimum infeasibility]** Select the variable with a value closest to 0 or 1.

(v) Select the variable that is closest to 1, i.e., choose ξ_j such that

$$\bar{\xi}_j = \max_k\{\bar{\xi}_k \mid \bar{\xi}_k \leq 0.999\}. \tag{4.5}$$

(vi) Select the variable with the largest objective function among all fractional variables that have a value 1 in the currently best known feasible solution. If no such variables exist, apply strategy (ii).

(vii) **[strong branching]** Use a LP approach: Let $\mathbf{Ax} \leq \mathbf{b}$ be all the constraints of the last solved LP. For each potential variable x_i solve the following two optimisation problems:

$$v_0^i = \max\{\mathbf{c}^T\mathbf{x}, \text{ such that } \mathbf{Ax} \leq \mathbf{b}, \ x_i = 0\} \tag{4.6}$$

$$v_1^i = \max\{\mathbf{c}^T\mathbf{x}, \text{ such that } \mathbf{Ax} \leq \mathbf{b}, \ x_i = 1\} \tag{4.7}$$

and choose the branching variable x_k with

$$\max\{v_0^k, v_1^k\} = \min_i \max\{v_0^i, v_1^i\}. \tag{4.8}$$

This technique is known as *strong branching*. In practice, however, the above optimisation problems are not solved fully; instead the dual algorithm is used for a small number of iterations and the bound obtained is used as an estimate for v_0 and v_1.

The best rule to use varies according to the combinatorial optimisation problem, but the following observations can be made: The minimum infeasibility rules (iv) and (v) will generate feasible solutions quickly since they modify variables that are *almost* integer but there is no reason why they will lead to small overall computational times. The maximum

infeasibility rule makes 'radical' changes in the two branches early in the tree and this is more likely to lead to faster results. The more complex rules, such as strong branching (vii) require a long time to decide, but they tend to produce very good choices that yield to smaller B&C trees and therefore they are probably suited for large or difficult problems, where a significant decrease in the number of LPs solved justifies the extra time taken to perform the branching.

4.3.2 Node selection

Node selection refers to the task of selecting the node to move to among the two newly created branches, and also the backtracking strategy once a node has been pruned.

For the backtracking procedure, the following two strategies are usually employed. Firstly, in the *depth-first-search (DFS)*, the algorithm chooses to move to a child of the previous node and, when it is pruned, the other child is selected. When both the children are pruned, the algorithm selects the most recently created node (which is the other child of the parent of the current node). Alternatively, a *best-bound* procedure can be followed. We choose the node with the best LP value among all nodes that have not been explored ('best' means the smallest value for a minimisation problem). In general it is acknowledged that the depth first usually performs worse than best-bound for many problems because it tends to spend a lot of time in branches for which there is possibly little improvement to the lower or upper bounds. In 'best-bound' this is not so common, as by choosing the best node, the lower bound will always be improved upon. It is possible however to have a combination of the two (based on Padberg & Rinaldi (1991)) where a *target* is specified (with a value between the current lower and upper bound) and we follow a DFS strategy until the LP value at the branch reaches the target value, then we 'backtrack' and choose the best-bound node.[28] This strategy is called *depth-first-search with backtracking (DFS-b)*. In the numerical experimentation, we set the target to be L units above the current lower bound, where L is 0.1 times the difference between the upper and the lower bounds. (i.e., the target is set to the 10% point between the lower bound and the upper bound.)

[28] If the target is set too close to the upper bound (i.e., the incumbent) then this search will be equivalent to DFS, if it is set close to the lower bound then it will resemble more best-bound.

4.4 Cut generation

This section deals with the process of generating cuts during the B&C algorithm. Recall that several cuts were generated at the root node of the B&C which was presented in chapter 3. The same separation procedures can be used to generate cuts at the various nodes of the B&C tree, using the LP relaxations obtained at a given B&C node. There are however some important complications with this procedure that are addressed in the next section.

4.4.1 Cut validity

There is a significant difference between generating cuts in a cutting plane algorithm and in a B&C setup. In B&C, the constraints added to generate the subproblems during branching (i.e., $\xi_j = 0$ or $\xi_j = 1$ for a binary MIP) are not necessarily valid for the original problem. (For instance, one of the branches may lead to an infeasible LP.) There is therefore always a risk that a cut obtained with one of these constraints added (e.g., $\xi_j = 0$, say) may not be valid when it is replaced with the other constraint ($\xi_j = 1$). In other words, cuts generated at one branch of the tree may not be valid for another branch.

This situation usually occurs if the generation of the cut relies on the value that the branching variable is set to. Among the cuts described in chapter 3, the *partition*, the *maximum flow* and the *r-cover* do not use the actual values that the binary variables are set to and, therefore, are globally valid no matter where in the B&C tree they are generated. Another way to see this is by observing that the above cuts will contain the variable ξ_j if so required, even though the variable ξ_j may be fixed to its upper or lower bound in the current LP subproblem. The cut generation algorithm for these cuts does not assume that any binary variable is fixed to a particular bound.

On the other hand, the Gomory cuts and the lift and project cuts are not globally valid. This is because they are derived using *all* the constraints present in the current LP formulation, including the ones that fix the branching variable to its bound. In particular, one may observe that the Gomory cuts generated when a variable ξ_j is fixed (to either its upper or lower bound) will never contain that variable; instead the variable is essentially replaced by the current *value* that it is fixed to and hence any contribution of the variable ξ_j is incorporated into the constant term of the cut. Obviously this cut will not be valid if ξ_j is fixed to a different value. Similar observations can be made for the lift and project cuts.

This means that in practice, all Gomory and lift and project cuts generated at a particular node v of the B&C tree will only remain valid when all the variables that are fixed remain fixed to the same values, i.e., for all the nodes lying on to the sub-tree rooted at node v. The global validity of all cuts generated at the root node is in accordance to the above observation since the entire B&C tree is rooted there.

4.4.2 Cut lifting

The above complications present a drawback to the use of cuts (such as the Gomory and lift and project cuts) whose generation involves the current values of the variable in a Branch and Cut framework. If such cuts are added, they need to be discarded every time we 'backtrack' from the B&C node where they were generated.

A technique called *cut lifting* can be used to remedy this shortfall. For the Gomory cuts this procedure is described in Balas, Ceria, Cornuéjols & Natraj (1996), and it aims to 'lift' a Gomory cut obtained at a particular node of the B&C tree by calculating a coefficient for the variables that are fixed (and thus do not appear in the cut) and adjusting the constant term of the cut as necessary.[29] The new cut is identical to the previous one if any occurrences of the fixed variables in the cut are replaced with their fixed values. The way that these 'lifting coefficients' are calculated for the Gomory cuts is also described in Balas, Ceria, Cornuéjols & Natraj (1996), but the whole method only works when all the integer variables are binary.

Lift and project cuts for 0-1 MIPs can also be lifted to include the variables that are fixed in the current B&C node where they are generated and thus made globally valid. The procedure is explained in Balas, Ceria & Cornuéjols (1996), where it is mentioned that the lifting coefficients can be calculated fairly quickly, in a time proportional to the number of variables that are not fixed to a given bound. The described method, however, needs some quite restrictive requirements regarding the format of the LP relaxation: for instance all fixed variables are required to be set to 0 which means that if there are any variables fixed to $\xi_j = 1$ then the LP must be re-formulated by replacing all occurrences of ξ_j with $1 - \xi_j$ and adjusting the constraint matrix accordingly. This step has to be added to the time taken to generate the lifted cuts. Another drawback is that from the computational results presented by the authors, it is apparent that the benefit of lifting is *very* dependent on the choice of variables that are fixed (in B&C terms, on which node the cut is generated), and moreover, on the number of variables present the problem.

[29] The term *lifting* stems from the fact that adding extra variables to the constraint amounts to lifting the constraint to a higher dimensional space.

Although Gomory and lift and project cuts are generated by the B&C algorithm developed for the URNDP with flows in this chapter, no cut lifting procedure was implemented in order to make these cuts globally valid. This is because of the low impact that these cuts have for the URNDP with flows and also the issues mentioned earlier in this section. This means that whenever such cuts are generated they are removed from the system as soon as the algorithm moves to the parent of the node that generated them.

4.5 Computational implementation of the branch and cut algorithm

4.5.1 Outline of the B&C algorithm

The basic steps of a B&C algorithm were discussed in sections 1.4.4 and 4.2.

The development of a B&C algorithm for this problem is complicated by the fact that not all the reliability constraints are present in the LP relaxation. This implies that an additional *feasibility test* step is required, in particular, whenever an integer feasible solution that is about to replace the incumbent is identified, we apply the separation heuristic for the partition inequalities described in section 3.4.1 to detect any violated cuts. If violated cuts are found, they are added to the formulation and the process continues. Otherwise, this implies that all the reliability constraints are satisfied, therefore this solution is feasible for the URNDP, and hence the incumbent can be updated.

The 'feasibility test' is shown in the diagrammatic description of the complete algorithm for the URNDP with flows in figure 4.1.

4.5.2 The cut generation procedure

The cut generation procedure used during Branch and Cut is summarised below:

Frequency of cut generation:

- Partition/maximum flow/r-cover cuts: we search for these cuts in every node for the first 50 B&C nodes. After that, the following procedure is adopted: if no cuts of a particular category are found in the current node, we postpone the search for those cuts for $K = 30$ nodes. This helps to avoid searching for certain type of cuts at every node if new cuts of certain category cannot be found. Partition inequalities

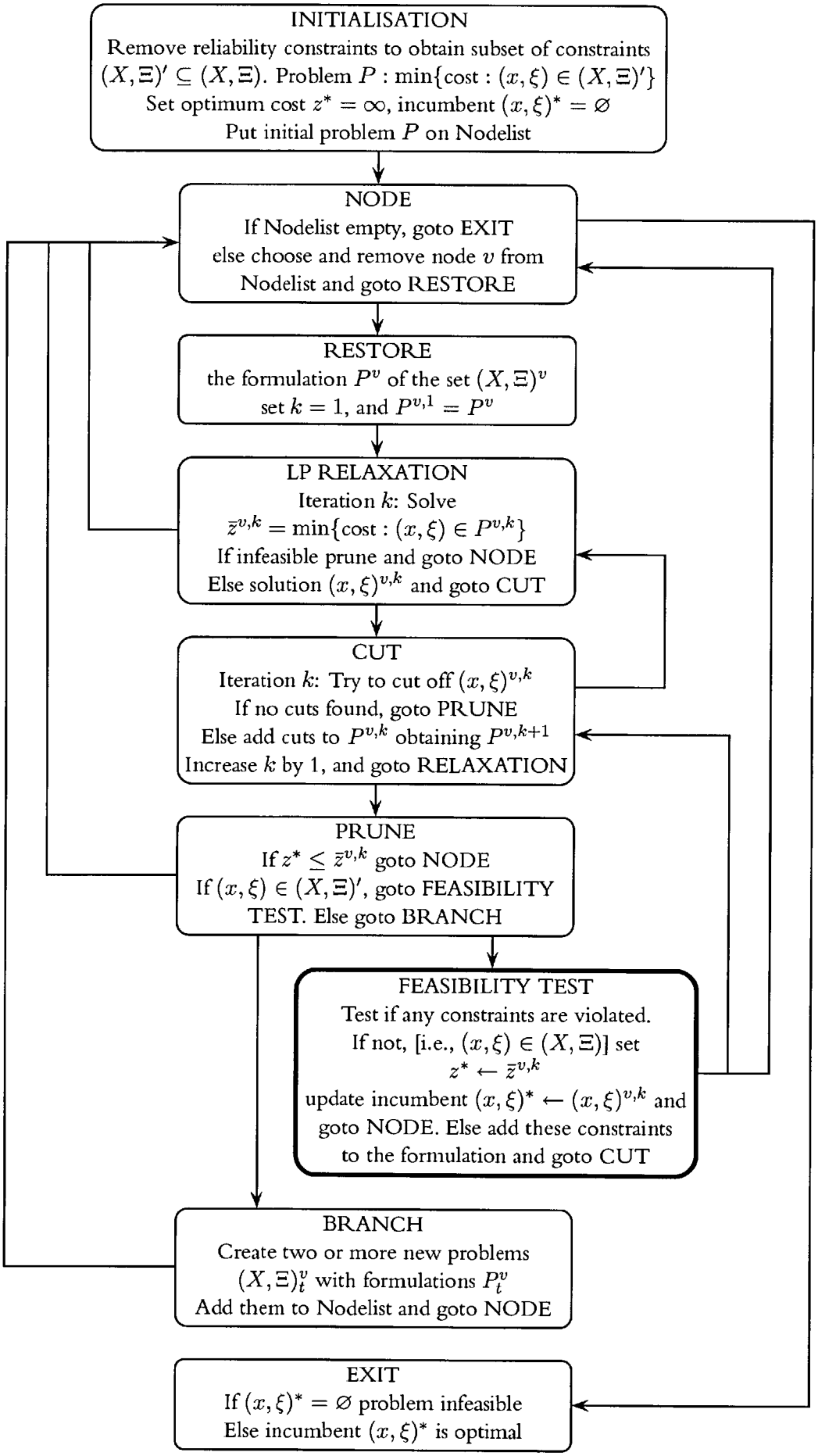


Figure 4.1: Outline of the Branch and Cut algorithm for the URNDP with flows

are also generated whenever an integer LP solution is obtained, regardless of the previous rule, as part of the 'feasibility test' procedure of the Branch and Cut.

- Lift and project cuts: we search for these cuts every $L = 60$ nodes.

Number of cuts generated at a particular node:

- Partition/maximum flow: we generate as many cuts as we can find (i.e., we re-optimise the LP problem using the dual algorithm and apply the cut generation algorithms until no more cuts can be found). This approach is reasonable since, at least for the test instances used, only a few cuts are obtained in every node. Nonetheless, an upper limit of 20 cuts is imposed in the implementation although this value is rarely attained.

- r-cover: we generate as many cuts as we can find but up to a maximum of 20.

- Lift and project cuts: we adopt the same approach as in the root node (section 3.9.2).

Note that no Gomory cuts are generated during the branching part of the algorithm. This is mainly due to the fractional coefficients of the Gomory cuts, and therefore, when a large number of them is added to the formulation, the LP solution process is slowed down considerably.

4.5.3 Evaluation of variable selection strategies

Some of the variable strategies presented in section 4.3 were implemented in the B&C algorithm for the URNDP with flows and the number of nodes and the computational time taken for each strategy was recorded. Branching was performed on the binary variables ξ_{ij}. Table 4.1 presents the performance of the B&C algorithm using various strategies for branching variable selection for some selected test problem instances, with the remaining parameters of the B&C being the same. In the strong branching procedure, five iterations of the dual simplex were computed.[30]

We can observe from the computational results that strong branching tends to produce the smallest B&C trees although the computational time taken does not always scale accordingly, as with strong branching LPs are needed to be solved in order to decide on

[30]This number was taken from the B&C implementation for the fixed charge problem by Ortega & Wolsey (2003), and it worked satisfactory for this problem too.

the branch to move, but this extra time seems to be worthwhile for the most difficult problems. Minimum and maximum infeasibility rules tend to perform consistently worse than strong branching, at least for all but the easiest instances, in terms of the number of B&C nodes generated. This translates to a worse performance than strong branching as the instances become more difficult, because, as the LP relaxation times increase, the comparative disadvantage of strong branching (taking too long to decide on which branch to move next) becomes less significant.

Among the remaining three strategies, where each strategy seems to perform better that the other two for different instances, the following observations can be made. The maximum cost rule performs at its best for the problems with a large value[31] of γ (i.e., when γ is 1 or 2) but these are considered to be 'easy' URNDP instances. The variability in performance of the 'Padberg-Rinaldi' and strong branching techniques for different test problems is not easy to explain, as one algorithm seems to outperform the other for specific instances. Strong branching, however, appears to produce more consistent results than 'Padberg-Rinaldi' (which occasionally tends to take extremely long for certain easy instances) and, therefore, this was the strategy selected for implementation in the B&C algorithm of the URNDP with flows.

[31]Recall that a large γ corresponds to a large variable over fixed cost ratio.

	min infeasibility		*max infeasibility*		*max cost*		*Padberg-Rinaldi*		*strong branching*	
Problem	*nodes*	*time*	*nodes*	*time*	*nodes*	*time*	*nodes*	*time*	*nodes*	*time*
g1n12g001i5	6	2.20	3	2.26	4	2.17	4	2.21	5	2.10
g1n12g01i2	28	5.39	19	2.54	5	2.45	24	2.58	10	2.49
g1n15g01i4	36	3.77	30	3.79	23	3.69	33	3.71	18	4.06
g1n20g01i3	641	55.22	245	29.68	165	31.92	419	43.53	129	40.32
g1n20g005i3	197	25.19	204	28.57	181	30.95	132	26.79	54	28.57
g1n25g01i5	1024	141.32	393	95.10	269	89.80	1161	161.18	224	93.91
g1n25g01i3	7250	1032.87	1069	251.10	688	167.64	2175	352.89	424	207.19
g1n25g01i4	39875	11195.95	8857	1550.81	675	223.56	59496	21889.29	391	214.55
g1n25g2i1	30	18.62	135	31.01	14	17.00	22	17.38	15	17.29
g1n30g1i2	3274	277.37	20934	2058.69	1595	160.12	3926	235.47	111	69.08
g1n30g01i2†	35045†	36000†	13815	11024.17	3015	1637.03	20495	11724.98	3038	2925.94

† the min infeasibility rule failed to reach the optimum within the given time limit. The final lower bound was 2894.51 (99.47%)

Table 4.1: Performance of the algorithm for different variable selection rules

4.5.4 Evaluation of node selection strategies

Computational experimentation (table 4.2) with the various node selection strategies for the URNDP with flows shows that the best-bound strategy works, in general, better than the DFS-b strategy. Using DFS-b, a feasible solution is obtained more quickly than pure best-bound (especially as it is assisted with the branching direction strategy described in the next section), but at the expense of overall large number of B&C nodes and computational time. Moreover, using DFS-b results in many branches being pruned as infeasible and thus there are large periods during the execution of the algorithm where the lower bound does not improve at all.

				DFS-b		*best-bound*	
Problem	*Vertices*	*Edges*	γ	*nodes*	*time*	*nodes*	*time*
g1n12g001i5	12	47	0.01	3	2.14	5	2.10
g1n12g01i2	12	55	0.1	4	2.49	10	2.49
g1n15g01i4	15	85	0.1	17	3.79	18	4.06
g1n20g005i3	20	142	0.05	98	34.67	54	28.57
g1n20g01i3	20	143	0.1	291	57.26	129	40.32
g1n25g01i3	25	222	0.1	1706	431.91	424	207.19
g1n25g01i4	25	230	0.1	1064	39.63	391	214.55
g1n25g01i5	25	224	0.1	678	135.70	224	93.91
g1n25g2i1	25	232	2	18	20.26	15	17.29
g1n30g01i2	30	329	0.1	8135	5461.35	3038	2925.94
g1n30g1i2	30	314	1	1988	190.17	111	69.08

Table 4.2: Implementation of two alternative node selection strategies

4.5.5 Evaluation of branch direction strategies

We finally consider the problem of deciding which child node to visit first: the two alternatives is either to choose the branch with $\xi = 1$ or alternatively the branch with $\xi = 0$. We can expect that the former choice will produce more feasible solutions faster (as it adds edges in the URNDP solution layout), compared with the latter choice which will tend to produce infeasible solutions as it tries to satisfy the integrality constraints by removing edges with fractional ξ values.

Numerical experimentation (table 4.3) verifies that the $\xi = 1$ option performs, in general, better that the alternative $\xi = 0$, as more feasible solutions result in more frequent

incumbent updates, hence more parts of the B&C tree being pruned subsequently.

				Branch to move to first			
				$\xi = 1$		$\xi = 0$	
Problem	*Vertices*	*Edges*	γ	*nodes*	*time*	*nodes*	*time*
g1n12g001i5	12	47	0.01	5	2.10	3	2.14
g1n12g01i2	12	55	0.1	10	2.49	13	2.87
g1n15g01i4	15	85	0.1	18	4.06	13	3.77
g1n20g005i3	20	142	0.05	54	28.57	59	27.77
g1n20g01i3	20	143	0.1	129	40.32	89	33.23
g1n25g01i5	25	224	0.1	224	93.91	216	131.87
g1n25g01i3	25	222	0.1	424	207.19	440	282.02
g1n25g01i4	25	230	0.1	391	214.55	601	300.41
g1n25g2i1	25	232	2	15	17.29	25	25.84
g1n30g01i2	30	329	0.1	3038	2925.94	2835	3585.57
g1n30g1i2	30	314	1	111	69.08	637	120.62

Table 4.3: Implementation of two alternative node selection strategies

4.5.6 Alternative branching strategy

During the implementation of the algorithm, an additional branching strategy was investigated. As explained in section 4.3, branching can be performed on binary variables or on both binary and flow variables. It is, however, possible to combine the two variables in such a way that *three* branches are created from a given node. Each branch contains one of the following (mutually exclusive) inequalities:

$$\xi_j = 0 \tag{4.9}$$

$$\xi_j = 1 \quad \text{and} \quad x_j = 0 \tag{4.10}$$

$$\xi_j = 1 \quad \text{and} \quad x_j \geq 1. \tag{4.11}$$

This covers all the possibilities for the values of the binary and flow variables, namely an edge is either not present (and hence the flow is zero because of the trivial constraints), or a variable is a reliability edge (an edge of zero flow), or it is a flow edge with a non-zero flow.

The advantage of the above strategy is that in the URNDP with flows we know that there will be exactly $N - 1$ flow-carrying edges in the optimal solution, where N is the

number of vertices in the network. (see section 3.3). Therefore, in the 'feasibility test' step of the B&C algorithm it is possible to count the number of flow edges that exist and reject a solution that does not meet the above criterion. Numerical experimentation unfortunately showed that this implementation took longer than the standard two-branch system. This is because the size of the B&C tree grows much faster with three branches than it does with two, meaning that more LPs need to be solved. Nevertheless, it was observed that in some cases one of the three branches generated produces an infeasible LP that gets pruned straight away, and one of the remaining two branches benefits from the existence of the additional restrictions on the x variables (equations (4.10) and (4.11)). In any case however, no literature containing computational results on implementations of B&C algorithms using three branches could be found so this possibility remains to be further evaluated, both theoretically and computationally.

4.6 Generating a feasible solution to the URNDP with flows

Any B&C algorithm requires the construction of an initial feasible solution to the MIP being solved using a heuristic procedure. Ideally, a feasible solution should be created quickly and be of good quality for an efficient implementation.

Several heuristic methods for network design problems can be found in the literature and some of these were discussed in section 2.4.6. Below we describe a heuristic procedure to derive an upper bound on the optimal solution of the URNDP with flows. The proposed algorithm is based on the implementation by Monma & Shallcross (1989) for the design of two-edge connected networks without the presence of flows.

A heuristic algorithm for the URNDP with flows

1. Calculate the minimum cost tree solution using the variable costs (without any reliability considerations).

2. Remove all the edges of the tree obtained and calculate the minimum spanning tree on the *leaves* of the tree (the vertices of degree 1) using the fixed costs as weights.

3. Merge the two trees obtained in the previous steps to obtain a feasible solution.

4. Starting from the most expensive edge added in step 3, test if that edge can be

removed from the layout without violating the reliability requirements. If so, remove that edge from the layout and iterate the process; stop if no more edges can be removed.

4.6.1 An illustrative example

The above procedure is depicted in figures 4.2 to 4.5. The test instance used is a 12 vertex problem, described in appendix A. The total cost of the optimal solution to this problem is 1646.

The cost of the first tree (figure 4.2) is 1784 (sum both fixed and variable costs). The cost of the second tree (figure 4.3) is 253 (fixed costs only), and hence merging the two trees yields a solution of total cost 2037. In the last step we remove the two most 'costly' edges, namely edge 1–11 of cost 71 and edge 1–9 of cost 21 to obtain the final solution of total cost[32] equal to 1945. No more reliability edges can be removed without violating the two-edge connectivity requirements.

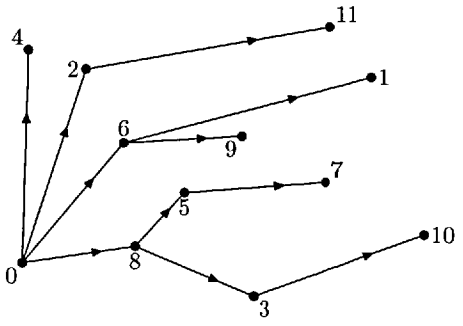


Figure 4.2: Heuristic step 1: The minimum cost tree solution

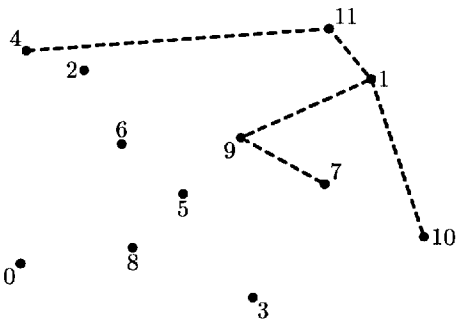


Figure 4.3: Heuristic step 2: The minimum cost tree on the leaves

4.6.2 Computational performance of the heuristic

Computational results obtained using this heuristic are shown in table 4.4, where the total cost of the heuristic solution was compared with the optimal solution values obtained using the B&C algorithm. The interpretation of the notation is presented below:

[32]Note that the (undocumented) problem-independent heuristic for MIP included in the CPLEX package reported a feasible solution cost of 2065 for this example.

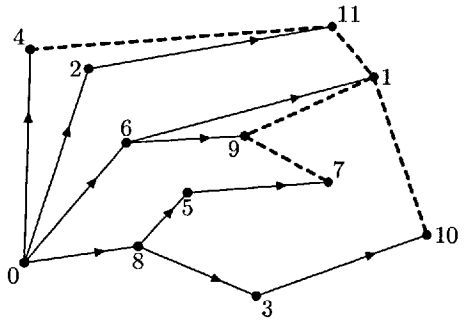


Figure 4.4: Heuristic step 3: Merging the two trees

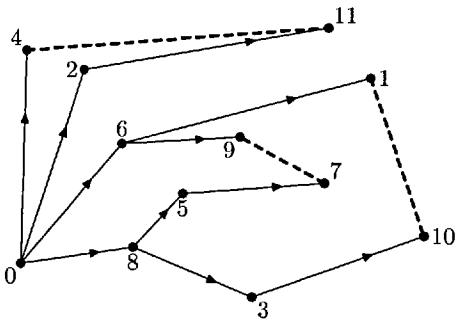


Figure 4.5: Heuristic step 4: The final layout of the solution

UB_{TT} Upper bound obtained by the two-tree heuristic.

z_{opt} The cost of the optimal solution to the URNDP or the cost of the best feasible solution found at the end of the Branch and Cut algorithm.

$\%UB_{TT}$ Percentage ratio $UB_{\text{TT}}/z_{\text{opt}} - 1$.

Problem	*Vertices*	*Edges*	γ	z_{opt}	UB_{TT}	$\%UB_{TT}$
g1n12g1i5	12	51	1	522	557	6.70
g1n12g2i3	12	45	2	335	335	0.00
g1n15g001i4	15	79	0.01	17136	20218	17.99
g1n15g005i3	15	72	0.05	3432	4978	45.07
g1n15g01i4	15	85	0.1	1475	1746	18.37
g1n15g1i3	15	78	1	344	394	14.53
g1n15g2i3	15	82	2	359	365	1.67
g1n15g2i4	15	78	2	320	346	8.13
g1n20g001i2	20	145	0.01	18008	23306	29.42
g1n25g1i5	25	230	1	633	705	11.37
g1n25g2i2	25	229	2	505	528	4.55
g1n30g01i4	30	317	0.1	3275	3999	22.11
g1n30g2i3	30	333	2	559	586	4.83
g1n40g1i2	40	601	1	798	880	10.28
g1n50g1i2	50	955	1	1179	1265	7.29
g1n50g2i5	50	921	2	1021	1048	2.64

Table 4.4: Quality of the heuristic solution at the beginning of the B&C algorithm

With regards to the performance of this heuristic the following observations can be made. The heuristic performs better for the problems corresponding to large values of

γ, i.e., when the variables costs are relatively larger than the fixed costs. (In the case for problem g1n12g2i3, the optimum solution was found.) This is expected, because for problems where the linear costs are relatively larger than the fixed ones, the flow tree of the URNDP optimum solution will be similar to the first tree found by the two-tree heuristic. The computational time required for this algorithm amounts to solving two linear programs (to obtain the two trees) and a sequence of 'feasibility tests' (at most $N-2$, where N is the number of vertices) to verify the feasibility of the layout when edges are removed; it is therefore an extremely fast heuristic procedure.

4.7 A solved example instance

In this section, we present the output of the B&C algorithm for the URNDP with flows produced using an example instance. The test instance has 12 vertices and 64 edges, and its fixed and variable costs, demands and layout are presented in appendix A.

Figure 4.6 shows the 64 edges comprising the layout of the input graph. The solution obtained at the end of the root node is presented in figure 4.7: in this figure the edges with non-zero values for the binary variables ξ_{ij} are drawn and the corresponding values are shown next to each edge. For clarity, if a value is equal to 0.1 or more then the edge appears as a solid line, if it is less than 0.1 the edge is displayed with a dotted line.

Figures 4.8 and 4.9 show the layout of the optimal solution in two different forms. In both figures, the values on the edges represent the flows of each edge (the corresponding binary variables are all equal to 1). Figure 4.8 has been included to facilitate the comparison of the optimal solution with the root node solution of figure 4.7. Figure 4.9 emphasises the tree structure of the solution (the vertices are more spread for clarity); the edges which carry no flow are shown with a dashed line.

The details of the execution of the algorithm for this instance are shown in table 4.5. This table shows the lower bounds obtained at the root node (and the computational time taken, in seconds) from the LP relaxation (LB_0), the maximum flow inequalities (LB_1), the Gomory inequalities (LB_4) and the lift and project inequalities (LB_5). No partition or r-cover were detected at the root node. For the B&C tree the number of cuts and the separation times are shown, for each category of cut.

We observe the special structure of the optimal solution layout for the uncapacitated RNDP: it consists of a flow tree plus some added edges which contain no flow and are included in order to incorporate reliability into the network.

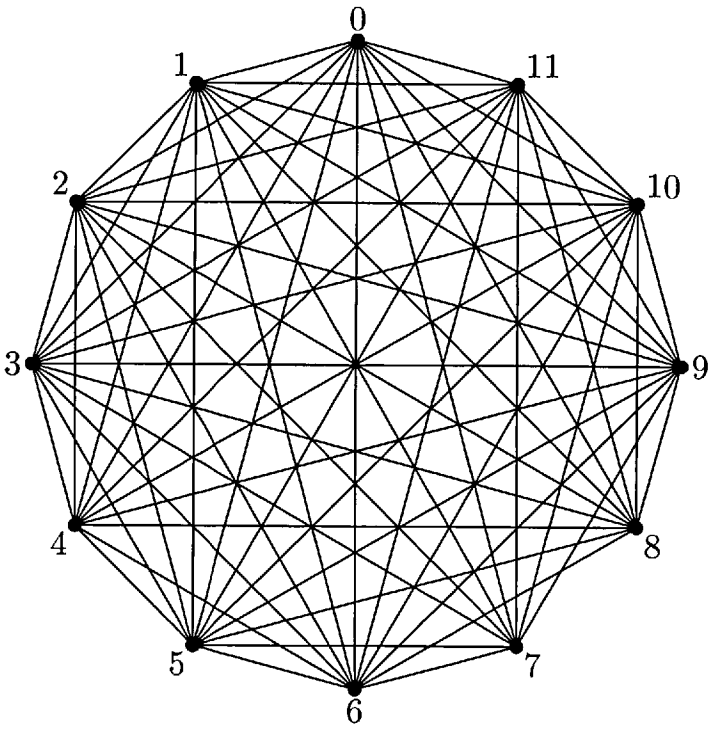


Figure 4.6: The layout of the input graph

		Root node							
Vertices	*Edges*	LB_0 *(%)*	LB_1 *(%)*	t_{mf} *(# cuts)*	LB_4 *(%)*	t_{gom} *(# cuts)*	LB_5 *(%)*	t_{lpr} *(# cuts)*	t_{root}
12	64	1213.04	1424.03	0.04	1424.20	0.55	1424.22	1.68	2.55
		(73.70%)	(86.51%)	(118)	(86.52%)	(19)	(86.52%)	(10)	

B&C tree						
t_{mf} *(# cuts)*	t_{part} *(# cuts)*	t_{rc} *(# cuts)*	$t_{lift\&pro}$ *(# cuts)*	z_{opt}	*B&C nodes*	t_{total}
0.02	0.02	0.61	0.88	1646	175	10.55
(21)	(20)	(13)	(8)			

Table 4.5: Execution of the B&C algorithm for a test instance

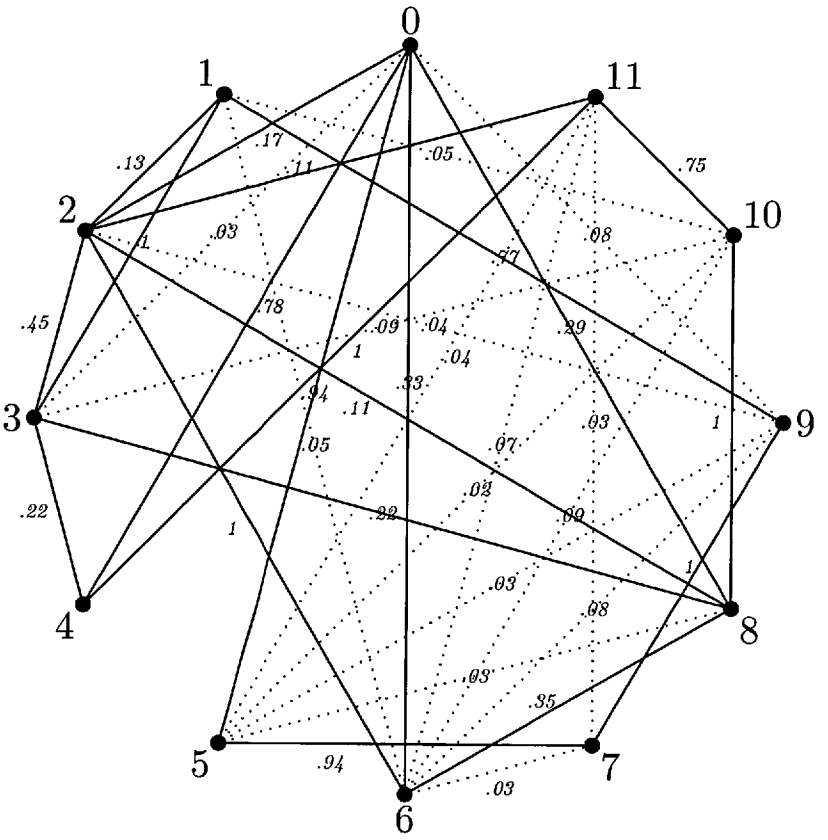


Figure 4.7: The solution layout associated with the best lower bound obtained at the root node

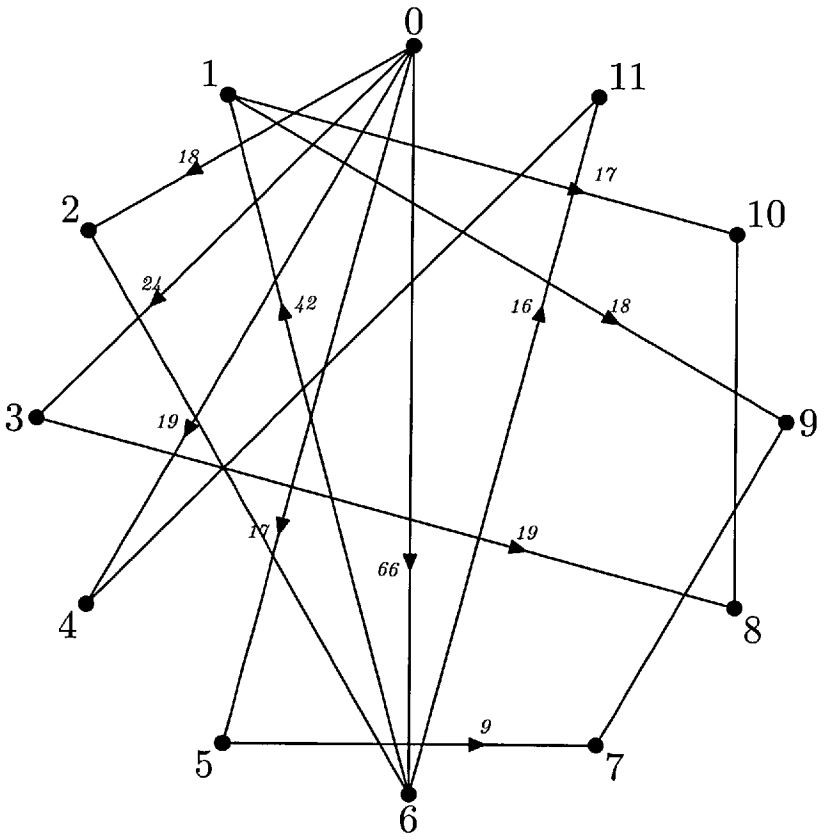


Figure 4.8: The layout of the optimal solution

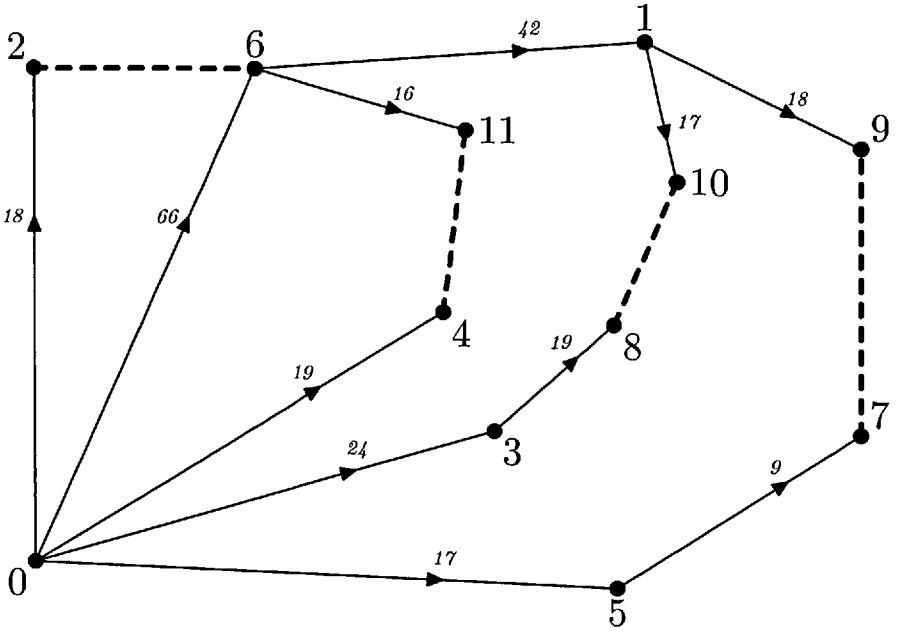


Figure 4.9: The layout of the optimal solution (expanded)

4.8 Computational results

In this section, computational results obtained using the Branch and Cut algorithm for the URNDP with flows are presented. The test instances used were presented in section 2.7 and are the same as those used in the discussion of the lower bound ascent procedure in chapter 3.

As in the previous chapter, the numerical experimentation was performed on a 600MHz *Pentium III* PC, using CPLEX 7.5 as the LP solver. A time limit of 10 hours of CPU time was imposed on all problems.

The notation used in the tables of this section is the following:

LB_{root}	Lower bound obtained at the root node of the B&C algorithm.
z_{opt}	The cost of the optimal solution to the URNDP or the cost of the best feasible solution found at the end of the Branch and Cut algorithm.
$LB_{\text{B\&C}}$	Lower bound obtained at the end of the B&C algorithm. This will be different from z_{opt} if the algorithm did not terminate because of the time limit imposed.
$\%LB_{root}$	Percentage ratio $LB_{\text{root}}/z_{\text{opt}}$.
$\%LB_{B\&C}$	Percentage ratio $LB_{\text{B\&C}}/z_{\text{opt}}$.
t_{root}	The computational time (in seconds) spent at the root node of the B&C algorithm.
t_{total}	The total computational time (in seconds) taken.

Tables 4.6 and 4.7 present the number of cuts generated during the B&C algorithm and the time that was required for their generation. Also reported are the number of B&C nodes and the total time taken, as well as the percentage errors at the root node. Each line reports the average values among the five problems of each class of test problems. In table 4.6 the average values for problems on 12 to 25 vertices are presented, whereas table 4.7 contains the values for the problems with 30 to 50 vertices. Tables 4.8 and 4.9 present similar results for individual test instances. The actual values of the bounds and optimum solutions are included for each individual problem.

			Root node		*Branch and Cut tree*										
					max flow		*partition*		*r-cover*		*lift & project*				
Vertices	*Edges*	γ	$\%LB_{root}$	t_{root}	*# cuts*	*time*	*# cuts*	*time*	*# cuts*	*time*	*# cuts*	*time*	*B&C nodes*	$\%LB_{B\&C}$	t_{total}
12	47.0	0.01	99.86	0.8	0.6	0.00	0.2	0.00	0.0	0.00	0.0	0.00	1.2	100	0.8
12	51.4	0.05	98.27	1.6	2.0	0.00	0.8	0.00	0.8	0.03	0.0	0.00	5.4	100	1.7
12	49.6	0.1	97.28	1.9	6.4	0.00	1.6	0.00	1.0	0.04	0.0	0.00	8.6	100	2.1
12	49.8	1	99.04	0.7	0.0	0.00	0.0	0.00	0.0	0.00	0.0	0.00	1.6	100	0.7
12	47.6	2	99.96	0.3	0.0	0.00	0.0	0.00	0.0	0.00	0.0	0.00	0.0	100	0.3
Average			*98.88*	*1.1*	*1.8*	*0.00*	*0.5*	*0.00*	*0.4*	*0.01*	*0.0*	*0.00*	*3.4*	*100*	*1.1*
15	78.0	0.01	99.36	2.9	2.2	0.00	0.8	0.00	0.2	0.03	0.0	0.00	3.2	100	3.0
15	76.6	0.05	97.40	3.0	10.2	0.01	4.0	0.00	1.0	0.19	0.4	0.09	13.4	100	3.6
15	80.4	0.1	95.57	3.3	13.0	0.01	3.0	0.01	0.8	0.21	0.4	0.07	18.0	100	4.0
15	80.2	1	98.52	2.3	1.4	0.01	2.8	0.01	0.4	0.08	0.4	0.07	9.2	100	2.7
15	78.2	2	99.68	1.2	0.4	0.00	0.0	0.00	0.0	0.01	0.0	0.00	1.2	100	1.2
Average			*98.10*	*2.5*	*5.4*	*0.01*	*2.1*	*0.00*	*0.5*	*0.10*	*0.2*	*0.05*	*9.0*	*100*	*2.9*
20	140.0	0.01	98.97	9.4	12.8	0.02	2.0	0.02	1.6	1.04	0.8	0.54	21.0	100	11.8
20	143.4	0.05	96.74	10.4	66.0	0.18	12.6	0.08	6.2	4.79	4.4	3.79	47.4	100	24.5
20	145.0	0.1	96.13	10.5	57.4	0.14	21.8	0.09	4.8	4.81	5.2	4.05	67.0	100	25.0
20	145.4	1	98.49	5.2	1.0	0.01	0.4	0.01	0.0	0.42	0.0	0.00	13.0	100	5.8
20	140.6	2	98.97	6.3	0.8	0.01	0.4	0.01	0.0	0.31	0.0	0.00	9.4	100	6.9
Average			*97.86*	*8.4*	*27.6*	*0.07*	*7.4*	*0.04*	*2.5*	*2.27*	*2.1*	*1.68*	*31.6*	*100*	*14.8*
25	231.6	0.01	98.12	29.1	76.0	0.30	9.8	0.16	7.8	12.95	4.4	9.01	64.8	100	59.6
25	220.4	0.05	95.93	27.9	177.2	0.77	33.4	0.34	22.0	30.72	28.4	82.27	655.2	100	294.0
25	227.6	0.1	94.12	28.0	221.0	0.89	24.8	0.36	9.4	37.41	26.4	85.37	533.4	100	271.5
25	226.6	1	98.82	15.1	2.2	0.03	0.0	0.01	0.0	0.91	0.0	0.00	9.8	100	15.9
25	224.4	2	99.18	16.5	4.4	0.03	0.0	0.01	0.0	1.25	0.0	0.00	10.4	100	17.7
Average			*97.24*	*23.3*	*96.2*	*0.40*	*13.6*	*0.17*	*7.8*	*16.65*	*11.8*	*35.33*	*254.7*	*100*	*131.8*

Table 4.6: Computational results for the URNDP: 12–25 vertices (averages over five instances)

			Root node		*Branch and Cut tree*										
					max flow		*partition*		*r-cover*		*lift & project*				
Vertices	*Edges*	γ	$\%LB_{root}$	t_{root}	*# cuts*	*time*	*# cuts*	*time*	*# cuts*	*time*	*# cuts*	*time*	*B&C nodes*	$\%LB_{B\&C}$	t_{total}
30	323.2	0.01	97.49	67.9	351.4	5.20	77.0	0.62	37.0	130.86	52.0	481.40	1329.8	100.00	1239.6
30	326.0	0.05	96.16	61.8	356.4	4.98	58.0	0.49	12.4	126.68	49.2	481.21	1097.2	100.00	1087.6
30	330.2	0.1	92.70	75.7	460.2	6.54	78.4	0.75	44.8	179.16	226.8	2712.76	6707.8	100.00	5841.3
30	329.0	1	97.58	44.1	26.6	0.77	11.8	0.04	0.0	18.03	3.6	10.32	75.6	100.00	82.0
30	330.8	2	99.09	46.1	8.0	0.13	0.2	0.01	0.0	5.71	0.8	1.90	21.4	100.00	54.1
Average			*96.60*	*59.1*	*240.5*	*3.52*	*45.1*	*0.38*	*18.8*	*92.09*	*66.5*	*737.52*	1846.4	*100.00*	*1660.9*
40	581.2	0.01	95.97	21.7	1279.6	51.61	253.4	4.88	205.2	1453.48	—	—	22776.0	99.43	33286.0
40	586.4	0.05	92.25	19.2	1399.4	56.68	223.4	5.43	47.4	1381.40	—	—	19738.8	96.88	33741.4
40	579.4	0.1	91.41	15.4	1231.4	49.50	180.2	4.70	40.2	1318.29	—	—	15752.6	97.16	29796.8
40	583.6	1	97.49	10.2	182.4	7.92	2.4	1.55	0.4	192.90	—	—	219.8	100.00	265.8
40	577.6	2	98.66	9.9	23.6	2.25	1.4	0.18	0.0	63.97	—	—	78.0	100.00	90.1
Average			*95.16*	*15.3*	*823.3*	*33.59*	*132.2*	*3.35*	*58.6*	*882.01*	—	—	11713.0	*98.69*	*19436.0*
50	918.6	0.01	94.91	50.0	1452.0	132.18	159.2	12.91	173.4	3751.88	—	—	7045.8	96.39	36000.0
50	917.8	0.05	89.86	45.9	1127.2	98.28	165.2	8.09	131.6	3068.31	—	—	8616.2	92.17	36000.0
50	914.2	0.1	88.32	46.3	1262.0	112.86	165.0	9.86	153.4	3347.45	—	—	8709.8	91.28	36000.0
50	929.8	1	97.09	38.4	238.6	27.03	5.8	2.02	0.0	841.50	—	—	3478.4	100.00	2684.5
50	916.0	2	98.89	38.1	28.6	6.53	2.6	0.61	0.0	226.29	—	—	218.6	100.00	326.9
Average			*93.81*	*43.7*	*821.7*	*75.38*	*99.6*	*6.70*	*91.7*	*2247.08*	—	—	5613.8	*95.97*	*22202.3*

Table 4.7: Computational results for the URNDP: 30–50 vertices (averages over five instances)

				Root node			*Branch and Cut tree*											
							max flow		*partition*		*r-cover*		*lift & project*				B&C	
Problem	*Vertices*	*Edges*	γ	LB_{root}	$\%LB_{root}$	t_{root}	*# cuts*	t_{mf}	*# cuts*	t_{part}	*# cuts*	t_{rc}	*# cuts*	$t_{\text{lift\& pro}}$	z_{opt}	$\%LB_{B\&C}$	*nodes*	t_{total}
g1n12g001i5	12	47	0.01	17315.6	99.32	2.02	3	0.01	1	0.00	0	0.01	0	0.00	17434	100	5	2.1
g1n12g005i3	12	50	0.05	3279.3	97.25	1.76	3	0.01	2	0.01	0	0.05	0	0.00	3372	100	5	1.9
g1n12g01i2	12	55	0.1	2131.0	97.89	2.14	8	0.01	3	0.01	3	0.08	0	0.00	2177	100	10	2.5
g1n12g1i3	12	49	1	360.3	97.37	0.83	0	0.01	0	0.01	0	0.02	0	0.00	370	100	6	0.9
g1n12g2i1	12	49	2	381.0	100.00	0.14	0	0.01	0	0.01	0	0.01	0	0.00	381	100	0	0.1
g1n15g001i4	15	79	0.01	17004.9	99.24	2.51	5	0.01	1	0.02	0	0.06	0	0.00	17136	100	4	2.6
g1n15g005i3	15	72	0.05	3391.9	98.83	2.77	1	0.01	0	0.00	0	0.05	0	0.00	3432	100	7	2.9
g1n15g01i4	15	85	0.1	1390.5	94.27	3.49	11	0.02	1	0.01	0	0.20	0	0.00	1475	100	18	4.1
g1n15g1i3	15	78	1	339.4	98.66	2.16	0	0.01	0	0.01	0	0.02	0	0.00	344	100	4	2.2
g1n15g2i4	15	78	2	318.4	99.49	2.03	0	0.01	0	0.01	0	0.03	0	0.00	320	100	4	2.1
g1n20g001i2	20	145	0.01	17860.6	99.18	9.80	20	0.04	3	0.04	2	1.77	2	1.19	18008	100	29	14.2
g1n20g005i3	20	142	0.05	4223.6	95.43	12.71	98	0.24	9	0.07	8	5.36	6	5.30	4426	100	54	28.6
g1n20g01i3	20	143	0.1	2445.7	94.72	11.26	97	0.20	24	0.14	8	8.41	8	5.76	2582	100	129	40.3
g1n20g1i1	20	144	1	644.1	98.34	5.78	3	0.01	1	0.02	0	0.35	0	0.00	655	100	8	6.4
g1n20g2i4	20	149	2	572.3	98.67	7.61	3	0.01	0	0.02	0	0.69	0	0.00	580	100	16	9.1
g1n25g001i5	25	246	0.01	18976.6	99.24	30.35	36	0.13	5	0.08	1	0.70	2	3.36	19121	100	40	44.5
g1n25g005i4	25	219	0.05	5042.0	94.14	23.24	212	1.00	108	0.63	31	40.98	40	104.65	5356	100	915	407.4
g1n25g01i4	25	230	0.1	2419.6	93.56	33.05	266	1.05	42	0.36	12	43.64	24	74.25	2586	100	391	214.5
g1n25g1i3	25	224	1	638.0	97.55	18.95	1	0.03	0	0.01	0	0.89	0	0.00	654	100	13	20.5
g1n25g2i1	25	232	2	461.8	98.68	14.62	5	0.02	0	0.01	0	1.71	0	0.00	468	100	15	17.3

Table 4.8: Computational results for the URNDP: 12–25 vertices

				Root node			*Branch and Cut tree*											
							max flow		*partition*		*r-cover*		*lift & project*				*B&C*	
Problem	*Vertices*	*Edges*	γ	LB_{root}	$\%LB_{root}$	t_{root}	*# cuts*	t_{mf}	*# cuts*	t_{part}	*# cuts*	t_{rc}	*# cuts*	$t_{\text{lift\& pro}}$	z_{opt}	$\%LB_{B\&C}$	*nodes*	t_{total}
g1n30g001i2	30	328	0.01	21808.4	97.19	73.50	235	3.23	47	1.73	113	112.99	28	260.59	22440	100	700	662.5
g1n30g005i4	30	321	0.05	5433.9	95.75	65.77	339	5.06	32	1.85	15	122.60	60	515.71	5675	100	1541	1216.7
g1n30g01i2	30	329	0.1	2709.0	93.09	73.69	348	4.61	50	2.05	57	141.53	112	1578.84	2910	100	3038	3048.9
g1n30g1i2	30	314	1	836.7	98.43	28.61	20	0.75	5	1.00	0	21.81	4	6.98	850	100	111	68.7
g1n30g2i2	30	337	2	662.6	99.19	52.37	23	0.19	0	0.12	0	12.90	2	5.73	668	100	39	74.9
g1n40g001i3	40	590	0.01	22577.8	96.19	22.82	1601	67.54	320	24.95	266	1685.09	—	—	23473	99.84	32298	36000.0
g1n40g005i5	40	575	0.05	5673.1	91.60	14.28	1465	55.73	166	19.35	20	1297.18	—	—	6193	95.73	18569	36000.0
g1n40g01i2	40	606	0.1	2951.3	90.47	17.83	1394	57.73	244	18.58	10	1501.60	—	—	3262	95.70	16850	36000.0
g1n40g1i4	40	579	1	932.8	97.57	12.28	158	7.19	0	2.48	0	178.60	—	—	956	100	334	275.5
g1n40g2i3	40	572	2	829.5	98.51	12.90	38	2.74	3	1.00	0	110.65	—	—	842	100	125	146.9
g1n50g001i1	50	915	0.01	28207.9	95.08	54.30	1089	104.40	214	66.16	329	3145.56	—	—	29667	96.19	7251	36000.0
g1n50g005i5	50	917	0.05	6172.3	90.09	49.50	1273	116.60	175	36.37	244	3519.40	—	—	6851	92.60	8168	36000.0
g1n50g01i1	50	907	0.1	3387.7	88.36	53.86	1324	114.83	208	45.68	54	3743.30	—	—	3834	92.03	9761	36000.0
g1n50g1i2	50	955	1	1134.6	96.24	34.97	339	33.32	11	8.82	0	1053.11	—	—	1179	100	942	1621.1
g1n50g2i1	50	903	2	870.6	98.82	27.49	26	5.84	1	2.25	0	202.97	—	—	881	100	187	290.4

Table 4.9: Computational results for the URNDP: 30–50 vertices

4.8.1 Discussion

From the computational results of tables 4.6–4.9 the following observations can be made.

For small networks the optimal solution is obtained very quickly, sometimes even at the root node of the B&C tree without the need for branching. As the network size increases, so does the complexity and for problems with 50 vertices the total time required grows to more than 10 hours of CPU time.

With regards to the cut generation procedure during Branch and Cut, the largest number of cuts generated were the maximum flow inequalities, whereas the cuts that took longer to generate were the lift and project. The partition inequalities were generated at smaller numbers, but at a very small computational cost due to the fast separation algorithm used. For these cuts, we note that the ratio #cuts /#B&C nodes seems to decrease as the network size increases. This could be explained by the fact that for larger networks a relatively smaller number of additional partition inequalities is necessary to guarantee feasibility. From table 2.5 we observe that the number of edges required for feasibility grows approximately at a linear rate with network size, whereas on the other hand the number of B&C nodes grows exponentially. Given that the partition cuts impose the addition of 'edges' in the solution, we expect the ratio #cuts /#B&C nodes to be a decreasing function with the number of vertices of the network.

Regarding the r-cover inequalities, we observe that the largest number of cuts is generated for the problem classes $\gamma = 0.01$ and $\gamma = 0.05$, with fewer cuts obtained for the cases $\gamma = 1$ and $\gamma = 2$. In order to evaluate the contribution of the r-cover inequalities to the B&C algorithm, the following investigation was performed: A different version of the B&C algorithm was produced where the generation of the r-cover inequalities was switched off. The computational results obtained with the two versions of the B&C algorithms are shown in table 4.10. We can observe that in many instances the removal of the r-cover inequalities from the B&C algorithm produces an increase in the number of B&C nodes and, in some cases, in computational time.

Finally we investigate the complexity of the problem versus network size and the value of the parameter γ. (The measure of the ratio variable costs over fixed cost.) Figures 4.10 and 4.11 present logarithmic graphs of computational time versus γ or the number of vertices. The (approximately) linear increase of log(time) versus the number of vertices in figure 4.10 shows the exponential correlation in complexity and network size. Figure 4.11 shows that the most difficult instances occur in the region 0.05–0.1, something that is also the case for the fixed charge problem (Hochbaum & Segev (1989)).

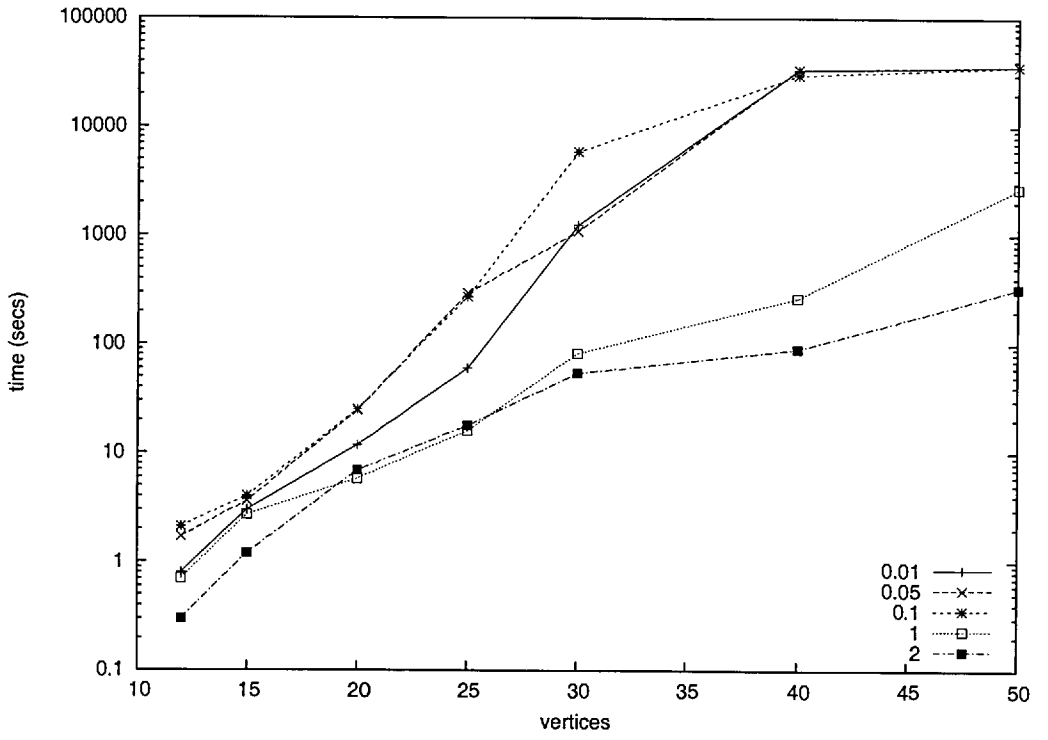


Figure 4.10: Graph of computational time versus network size for different values of γ.

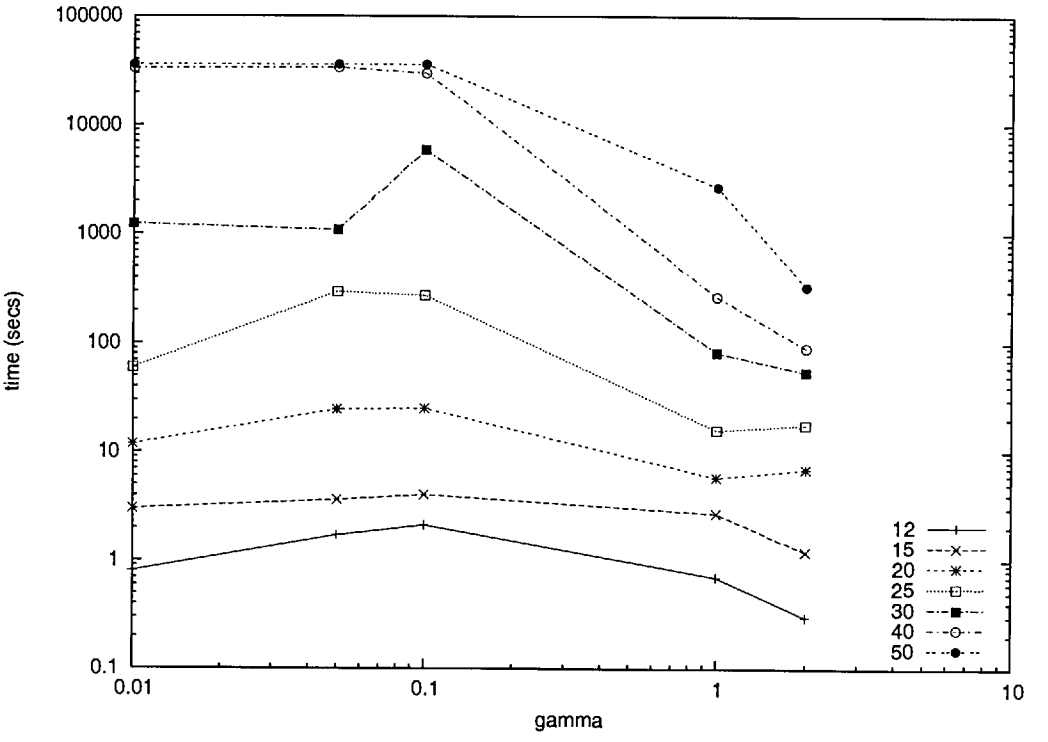


Figure 4.11: Graph of computational time versus γ for different network sizes.

			Without r-cover		*With r-cover*	
Problem	*Vertices*	*Edges*	*B&C nodes*	*time*	*B&C nodes*	*time*
g1n12g005i1	12	50	19	2.2	11	1.8
g1n12g005i4	12	54	11	2.8	8	2.5
g1n12g01i2	12	55	13	2.6	10	2.5
g1n15g001i5	15	76	8	2.2	5	2.1
g1n20g001i4	20	150	51	20.4	18	13.1
g1n20g005i1	20	135	21	8.3	8	7.8
g1n30g005i4	30	321	1965	1291.7	1541	1216.7

Table 4.10: Investigation on the contribution of the r-cover inequalities to the B&C algorithm

4.9 Summary

In this chapter, a Branch and Cut algorithm for solving the URNDP with flows to optimality was presented, and the details of the implementation (cut generation, branching rules etc.) were discussed. The algorithm, assisted by the generation of a set of cuts obtained both at the root node and the B&C tree, solved to optimality several randomly generated test problems. In the next chapter, we focus on a heuristic procedure for solving the URNDP with flows.

An ant colony behaves with an intelligence no particular ant possesses; a brain is conscious although no particular brain cell is; a city develops districts and neighbourhoods no planner could impose. In each case, complex problems are solved by a profusion of relatively simple elements.

Steven Johnson (2001)
Emergence: The connected lives of ants, brains, cities and software

Chapter 5

An ant colony heuristic for the URNDP with flows

5.1 Introduction

Chapters 3 and 4 of this thesis provided a detailed description and implementation of an exact Branch and Cut algorithm for the uncapacitated RNDP with flows. In this chapter a heuristic for this problem will be presented. The algorithm is based on ant colony optimisation which is a recently developed area of metaheuristics and has been successfully used to solve several combinatorial optimisation problems. The chapter initially provides an overview of ant colony optimisation techniques and their application to the solution of operational research problems developed so far. A description of the algorithm developed for the URNDP with flows is then presented, followed by a discussion of implementation issues and computational results.

5.2 An overview of ant colony optimisation

Many recent developments in optimisation algorithms are inspired by processes occurring in Nature. For instance, genetic algorithms are derived from cell reproduction in living organisms, simulated annealing is based on the annealing process of crystals in crystal-

lography and neural algorithms simulate the exchange of information procedure taking place in the neurons of the brain. A general overview on such optimisation algorithms derived from the physical world can be found in Colorni, Dorigo, Maffioli, Maniezzo, Righini & Trubian (1996). Simulating natural processes on a computer is not a new field, in fact it has been done for a long time in the area of Artificial Intelligence, but it was only recently that their potential to solve Operational Research problems became clear.

Ant Colony Optimisation (ACO) was inspired by the behaviour of social insects. The principle of an ant algorithm is based on collaboration between a large number of simple agents: an agent (ant) performs a simple operation on its own, but if many agents are present simultaneously then they can, as a team, perform very complex tasks, e.g., solve a complex optimisation problem. This behaviour is possible due to the exchange of information between the agents, which affects the behaviour of each agent in a predefined way. The interesting observation about ACO algorithms is that, although both the individual behaviour of every agent and the mechanism of information exchange can be very simple, the resulting behaviour of the colony can be extremely complex and can be used to solve difficult problems.

The natural process that inspired ACO is the following: It was observed that real ants, despite being almost blind, can find their way to a food source and back or around an obstacle easily. This behaviour is attributed to the ability of ants to communicate between them chemically using a substance called *pheromone*, and the simultaneous presence of many ants causes an 'intelligent' behaviour to emerge, which is depicted in figure 5.1 and is described below.

First of all, each ant deposits a small amount of pheromone along its path as it moves. Moreover, an ant, having a powerful sense of smell, can detect the pheromone already deposited and instinctively tends to preferentially follow paths of high concentration of the substance. This is the reason why ants seem to follow each other along the same path when travelling to and from a food source (first diagram). Now if the path of the pheromone (the *trail*) suddenly becomes blocked, then the ant that is just before the obstacle cannot move forward to follow the pheromone trail, so it has to move left or right. It is reasonable to assume that there is a 50% chance of moving to either direction. The same applies to the ants on the other side of the obstacle. The picture shown in the second diagram will soon emerge. Since the top path is shorter than the bottom one, after some time it will have a higher concentration of pheromone than the latter, as a higher density of ants will have passed over it.

New ants reaching the obstacle will tend to follow the top path more often than the

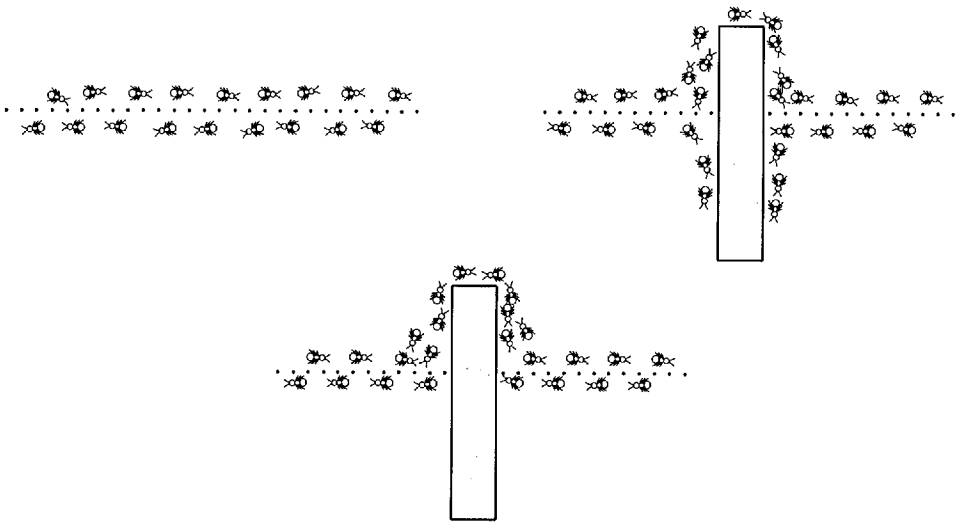


Figure 5.1: How real ants find the shortest way round an obstacle

bottom one since they prefer pheromone-rich paths, thus enhancing the trail of the top path even more. Soon, the top trail will be so much stronger than the bottom one (given that trails also fade out with time as the pheromone evaporates) that all the ants will follow the top route thus establishing a new route to the food source around the obstacle (third diagram).

In this example we can observe that although each ant individually does something relatively simple, by working collectively (with the chemical exchange of 'local knowledge') they can perform complex tasks, as in this case finding the shortest route around an obstacle. This is the principle behind the development of ACO algorithms, which are discussed in the next section.

5.3 Ant colony optimisation algorithms

The first combinatorial optimisation problem for which an ACO algorithm was developed was the *Travelling Salesman Problem (TSP)*, by Dorigo & Gambardella (1996) and Dorigo, Maniezzo & Colorni (1996).

We begin by outlining their approach, which forms the basis for all ACO algorithms.

5.3.1 An ACO algorithm for the TSP

Consider a graph $G(V, E)$, where V is the vertex set with $N = |V|$ and E is the edge set. A group of artificial agents (ants) moves along the edges of a graph, and each ant must visit all the vertices and return to its starting vertex, thus performing a TSP tour. Each edge has a real number T_{ij} attached to it, which corresponds to the amount of pheromone deposited (the trail). Initially all the trails are initialised to the same (small) value. Assume that at time t, an ant is at vertex i. The ant will move to vertex j in the next step $(t + 1)$ according to the following transition probability:

$$\mathbb{P}_t(\text{move from } i \text{ to } j) = p_{ij}(t) = \begin{cases} \frac{[T_{ij}(t)]^\alpha \times [H_{ij}]^\beta}{\sum_{k \in K} [T_{ik}(t)]^\alpha \times [H_{ik}]^\beta}, & \text{if } j \in K \\ 0, & \text{otherwise} \end{cases}, \tag{5.1}$$

where K is the set of 'allowed vertices', i.e., the set of vertices that the ant can move to from vertex i. This set contains all the vertices that are joined to the current vertex i, minus all the vertices that the ant has already visited.[33] The quantity H_{ij} is called the *visibility* and is related to the cost of the edge. In the TSP implementation by Dorigo & Gambardella (1996) it is taken to be the inverse of the edge cost, i.e.,

$$H_{ij} = \frac{1}{\text{cost}(\text{edge}(i, j))} \tag{5.2}$$

Note that the trail function is also indexed by t to stress the fact that the trail values change as ants move on the graph, and that the denominator of (5.1) is a scaling constant so that the probabilities add up to one.

The constant parameters α and β control the relative importance of trail versus visibility. A large value of α means that the ants will tend to follow more closely the edges with high trail, whereas a large value of β will result in ants being very willing to diverge from the "beaten track" in favour of shorter tours. It is obvious that a compromise between the two extremes is necessary: if the agents do not use the trail information then all the benefits of collaboration are lost, whereas if, on the other hand, they follow the trail too strictly then this prevents them from diversifying and discovering better solutions that have not been yet seen.

The trail is updated whenever an ant has finished a TSP tour according to the following rule:

$$T_{ij}(t + N) = \rho T_{ij}(t) + \delta T_{ij} \tag{5.3}$$

[33] unless, of course, if the ant has already visited $N - 1$ vertices in which case it is allowed to return to its starting vertex to close the TSP tour.

where ρ is a coefficient representing the *evaporation* of the trail with time and δT_{ij} is defined by:

$$\delta T_{ij} = \begin{cases} \frac{Q}{L}, & \text{if the ant uses edge } (i,j) \text{ in the tour} \\ 0, & \text{otherwise} \end{cases} \tag{5.4}$$

where Q is a constant and L is the cost (length) of the tour.

The algorithm terminates when a given number of ants have finished their tours, and the output is the best tour performed by an agent. In the implementation by Dorigo, Maniezzo & Colorni (1996), the 'typical' set of parameters used are shown in table 5.1, with the combination producing the best results (averaged over 10 separate computational runs) shown in boldface. The influence of the parameter Q on the quality of the solution was found to be negligible.

Parameter	*Value*
number of ants:	N, the number of TSP cities
Q:	1, 100, 1000
α:	0, 0.5, **1**, 2, 5
β:	0, 1, 2, **5**
ρ:	0.3, **0.5**, 0.7, 0.9, 0.999

Table 5.1: The parameters used in the ACO algorithm for the TSP

The computational results obtained for standard test instances for both the symmetric and the asymmetric TSP are quite good; the ant algorithm always produced a solution within 5% of the best known, and if further assisted by a local optimisation procedure it produced the best known solutions to all the instances tested except one.

A more sophisticated approach found in Dorigo & Gambardella (1996) contains a modified path selection rule. A new constant parameter, q_0, is added, which represents the probability of an ant moving along the edge with the overall strong trail/small cost edge. An ant has a probability of q_0 to move along this 'best' edge, and a probability of $1 - q_0$ to move to any allowed edge as given by rule (5.1). In this approach, a uniform random number q, $0 \leq q \leq 1$ is generated and compared to the parameter q_0. If $q \leq q_0$, an ant currently at vertex i moves to vertex j where

$$j = \arg\max_{k \in K} \left\{ [T_{ik}(t)]^{\alpha} \times [H_{ik}]^{\beta} \right\}. \tag{5.5}$$

On the other hand, if $q > q_0$ the ant is allowed to move to any edge $j \in K$ given by the rule (5.1).

5.3.2 ACO algorithms for other combinatorial optimisation problems

The simplicity and good performance of the ant algorithm for the TSP has spiralled many ACO articles to appear in recent years for many other combinatorial optimisation problems. In the area of transportation, Bullnheimer, Hartl & Strauß (1999) and Gambardella, Taillard & Agazzi (1999) present an ACO algorithm for the vehicle routing problem, whereas Barcos Redín (2003) studies ACO algorithms for point-to-point delivery systems. In these implementations, an ant represents a vehicle which performs a trip from a depot to a customer and back. In Bullnheimer, Hartl & Strauß (1999) the authors report good results for standard VRP test instances, which, in some cases match or are very close to the best published solutions.

An ant colony algorithm for the quadratic assignment problem is presented in Talbi, Roux, Fonlupt & Robillard (2001) and an approach for the set covering problem can be found in Alexandrov & Kochetov (2000). Other ant-based algorithms include the problem of graph coloring (Costa & Hertz (1997)), the sequential ordering problem (hybridised with a local search algorithm) found in Gambardella & Dorigo (2000) and the load balancing problem in telecommunications (Schoonderwoerd, Holland, Bruten & Rothkrantz (1996)). General overviews of ant systems can be found in Bonabeau, Dorigo & Theraulaz (2000) and Glover & Kochenberger (2003).

Finally, in 2000 an entire issue of Future Generation Computer Science[34] was devoted to ACO algorithms and applications. The issues covered include theoretical extensions of ACO (e.g., using alternative trial and path selection strategies) and applications such as the TSP and the frequency assignment problem in radio and mobile phone links.

5.4 An outline of an ant colony heuristic for the URNDP with flows

In this section we develop a new heuristic for solving the uncapacitated RNDP with flows, based on ant colony optimisation.

Although the principles of all ACO algorithms are the same, there are a few important differences between the URNDP and the TSP that affect the design of such algorithms. First of all, in the URNDP with flows there are *two* costs per edge, one linear and one

[34] see *Future Generation Computer Science* (2000) in the bibliographical references.

fixed, and, moreover, the total cost of the solution depends not only on these costs but also on the flows that pass through each edge. These factors have to be incorporated into the algorithm, and this complicates the way that the pheromone trail has to be updated. The level of complexity is further increased by having to determine how to incorporate reliability into the design of the network, i.e., how to force a ACO heuristic to produce a solution that satisfies the two-edge connectivity property. A new approach to include this issues in the ACO procedure was developed, which will be discussed in the next sections.

5.4.1 The ant colony setup

The first key question is how to use ants or agents that somehow move on a graph to represent a solution to the URNDP with flows. An important observation is that, as mentioned in chapter 3, a solution of the URNDP will consist of a set of *flow* edges (which supply the customers) forming a tree, and a set of *reliability* edges which have zero flow and are included in the layout to guarantee the required reliability properties.

In our setup, we are going to distinguish between these two types of edges, and for that purpose there will be *two* types of ant: ants that will seek to find the best location for the flow edges and other ants that concentrate on the reliability edges.

Another key point is that any tree supplying $N-1$ customers from a source can be decomposed into $N-1$ flow paths from the source to the customers, as shown in the example of figure 5.2. Each path will carry the amount of flow that the customer at the end of the path requires. This observation enables us to represent each of these $N-1$ paths from the source to a customer vertex with an ant trip.

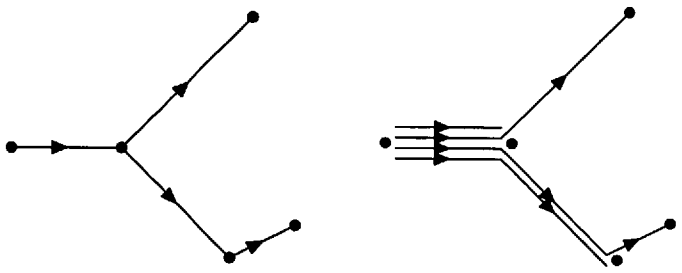


Figure 5.2: A tree with N vertices is equivalent to $N-1$ paths leaving the source

The ACO for the URNDP with flows employs the following setup: We define an *ant group* as a set of artificial agents (ants) which move on the graph using predefined rules, and each ant group represents a feasible solution to the URNDP with flows. In

the implementation, many ant groups can be present simultaneously. We begin by describing the behaviour of an individual ant group, and we shall proceed with the full algorithm where many ant groups are present. The notation used in this section is shown in table 5.2.

Symbol	*Interpretation*
V:	The vertex set, with $N = \|V\|$
V':	$V \backslash \{0\}$, the set of customer vertices
E:	The edge set
b_e:	The fixed cost of edge $e \in E$
c_e:	The linear cost of edge $e \in E$
d_i:	The demand of vertex i, where $i \in V'$

Table 5.2: The notation used for the ACO algorithm for the URNDP with flows

First of all, each edge of the graph has *two* trails attached to it, represented by real numbers, the *edge trail* and the *flow trail*. The edge trails are related to the fixed cost of using the edge, whereas the flow trails reflect the additional cost incurred when the edge has non-zero flow passing through it.

Each ant group consists of two types of agents: *flow ants* and *reliability ants*. When an ant group is first created, it contains the following agents:

- $N - 1$ flow ants which start from the source and have a customer vertex as their destination (each ant having a different customer as its destination)
- $N - 1$ flow ants that perform the reverse trip: from the customer vertices to the source vertex.

Every flow ant also has a integer number associated with it, which is equal to the demand of the customer vertex it is going to or coming from. This number will be referred to as the *demand* of the flow ant. On the other hand, every reliability ant has an edge associated with it, the *tabu edge* which is an edge that it cannot use in its path as this represents the edge that has broken down in the reliability requirement of the URNDP with flows.

Once an ant group has been created, the flow ants of the group start moving along the vertices of the graph using a probabilistic decision rule (which will be discussed later) until they reach their destinations. When all the flow ants have reached their destinations, then reliability ants are added to the ant group. In particular, for every flow ant its path is tracked and a reliability ant is added for every edge visited by that ant. Each reliability ant

will have a particular edge (among the edges that were used by the particular flow ant) as *tabu* meaning that it cannot pass over it. The newly added ants will also have the same source and destination vertices as the corresponding flow ant.

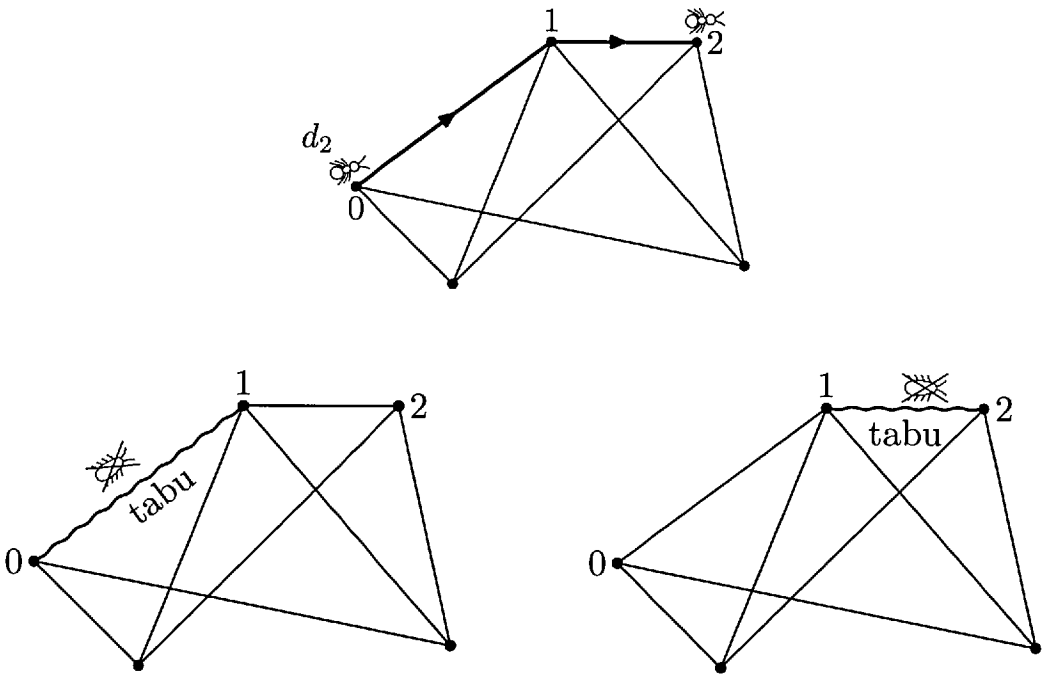


Figure 5.3: Flow ants and reliability ants

As an example, consider the network shown in figure 5.3. A flow ant leaves vertex 0 (the source) and is travelling to vertex 2, carrying a flow of d_2. We assume that the ant followed the path

$$0 \rightarrow 1 \rightarrow 2$$

as shown in the first diagram. For this particular ant two reliability ants will be added since it used two edges in its path. Each of the two reliability ants will have the same source and destination as the original ant (i.e., vertices 0 and 2 respectively), and one of the two edges as their tabu edge, as shown in the last two diagrams of figure 5.3.

After all the reliability ants of the group have finished, we convert the ant paths into a URNDP solution by using a heuristic procedure which will be discussed in section 5.4.6.

5.4.2 Trail deposition

In ACO algorithms, every ant-agent that moves on the network leaves a particular trail on the edges which affects the motion of the other ants. As mentioned earlier, there are

two types of trail and therefore two real numbers associated with each edge—the flow trail T_f and the edge trail T_e. The rule relating agents and trails is the following:

- Flow ants deposit and smell both types of trail.
- Reliability ants deposit and smell the edge trail only.

This is due to the fact that the inclusion of a reliability edge into the layout will only incur the fixed cost, whereas a flow edge will incur both the fixed and the variable cost components.

The contribution to the trails is inversely proportional to the cost incurred. In particular, an ant a moving along edge (i, j) will yield a contribution $\delta T_e(i, j)$ and $\delta T_f(i, j)$ to the edge and flow trails respectively, given by the following rules:

Reliability ants:

$$\delta T_e(i, j) := \delta T_e^a(i, j) = \frac{1}{b_{ij}}, \qquad \delta T_f^a(i, j) = 0 \tag{5.6}$$

Flow ants:

$$\delta T_e(i, j) := \delta T_f^a(i, j) = \frac{1}{b_{ij}}, \qquad \delta T_f^a(i, j) = \frac{1}{c_{ij} d_a}, \tag{5.7}$$

where d_a is the demand of the flow ant.

The reason behind the above rules is that whenever an ant carrying a flow of d_a passes over edge (i, j), it will incur a fixed cost of b_{ij} and a variable cost of $c_{ij} d_a$, represented by the denominators. Reliability ants carry no flow and therefore only the fixed cost is taken into account.

5.4.3 Trail updating

Once the trails have been set up, they need to be periodically updated as the algorithm evolves. The updating of the trails is performed whenever an ant group has finished (i.e., both flow and reliability ants have reached their destinations). The reason for this is that some of the edges may be visited by more that one ant, and for these edges care should be taken in the calculation of the trail corresponding to the fixed costs. This is because in the URNDP solution, the fixed cost b_{ij} is incurred *only once*, no matter how many ants of the group used that particular edge. In order to maintain the balance between the fixed and linear trails, the edge trails must receive only one contribution for any edge visited. The rules for updating the trails are shown below:

Flow trail:

$$T_f(i,j) \leftarrow \rho T_f(i,j) + Q \sum_{a \in \{\text{ants that used edge } (i,j)\}} \delta T_f^a(i,j) \tag{5.8}$$

Edge trail:

$$T_e(i,j) \leftarrow \rho T_e(i,j) + Q \Xi_{ij}\, \delta T_e(i,j), \tag{5.9}$$

where ρ is the evaporation coefficient, Q is a constant and $\delta T_e(i,j)$, $\delta T_f^a(i,j)$ are given by equations (5.6) and (5.7) respectively. The variable Ξ denotes whether an edge has been used by the ant group or not and is defined by

$$\Xi_{ij} = \begin{cases} 1, & \text{if edge } (i,j) \text{ was visited by an ant of the group} \\ 0, & \text{otherwise} \end{cases}. \tag{5.10}$$

5.4.4 Path selection—flow ants

In this section we describe the strategy of selecting where an ant will move to at each step. This strategy is different for each of the two types of ant, so they are best described separately.

Assume that a flow ant a is at vertex i. A random number q is selected from a uniform $(0,1)$ distribution and compared with a fixed parameter q_0 $(0 < q_0 < 1)$.

If $q \leq q_0$ the ant moves to vertex j, where

$$j = \arg\max_{k \in K} \left\{ [F\left(T_f(i,k), T_e(i,k)\right)]^\alpha \, [H(i,k)]^\beta \right\} \tag{5.11}$$

Here $T_f(i,k)$ and $T_e(i,k)$ are the flow and edge trails for edge (i,k), and K is the set of potential vertices, i.e.,

$$K = \{k \in V : (i,k) \in E, \text{ and vertex } k \text{ has not been yet visited}\}, \tag{5.12}$$

representing the edges that the ant is allowed to go to. The function H is a heuristic function representing the *visibility*, and in line with equation (5.2) of the TSP it has been chosen to be equal to the inverse of the total cost of using the edge, i.e.,

$$H(i,k) = \frac{1}{c_{ik} d_a + b_{ik}} \tag{5.13}$$

where d_a is the demand carried by the ant.

The two trails are combined using the function F, called the *trail function*, which was chosen to be

$$F(T_f, T_e) = \frac{1}{\frac{1}{T_f} + \frac{1}{T_e}}. \tag{5.14}$$

The reason for the form (5.14) above is that it best represents the cost incurred by a flow ant when using an edge. In particular, given that the two trails are the inverse of the fixed and variable costs, an equivalent trail combining the two costs will be of the form (5.14) because the total cost is the sum of the two cost components.

Equation (5.11) specifies that the ant should move to the 'best' vertex, i.e., the one that has the highest combined value of 'strong trail' and 'small cost'.

On the other hand, if $q > q_0$, then the ant may move to any allowed vertex using a probability that weights the edges as follows:

$$\mathbb{P}(\text{move to } j \in K) = \frac{[F(T_f(i,j), T_e(i,j))]^\alpha \, [H(i,j)]^\beta}{\sum_{k \in K} [F(T_f(i,k), T_e(i,k))]^\alpha \, [H(i,k)]^\beta} \tag{5.15}$$

and

$$\mathbb{P}(\text{move to } j \notin K) = 0, \tag{5.16}$$

where the interpretation of the variables is the same as in the case $q \leq q_0$.

5.4.5 Path selection—reliability ants

The reliability ants move in a similar way as the flow ants, but because they do not 'smell' the flow trails the corresponding functions are simpler. These ants, however, are not allowed to move along their tabu edge. Assume that a reliability ant a is at vertex i. We select a random variable q as in the previous section, and we compare it with the parameter q_0, just as in the case for flow ants.

If $q \leq q_0$ the ant moves to the 'best' overall vertex j, where

$$j = \arg\max_{k \in K} \left\{ [T_e(i,k)]^\alpha \, [H(i,k)]^\beta \right\}, \tag{5.17}$$

whereas if, on the other hand, $q > q_0$ the ant is allowed to move along any allowed edge using the probability rule:

$$\mathbb{P}(\text{move to } j \in K) = \frac{[T_e(i,j)]^\alpha \, [H(i,j)]^\beta}{\sum_{k \in K} [T_e(i,k)]^\alpha \, [H(i,k)]^\beta} \tag{5.18}$$

and

$$\mathbb{P}(\text{move to } j \notin K) = 0, \tag{5.19}$$

The notation used is the same as in the previous section, with the only difference that the set K contains all the vertices which are joined to the current vertex, unless the move corresponds to the tabu edge, i.e.,

$$K = \{k \in V : (i,k) \in E,\ (i,k) \neq \text{tabu}_a, \text{ and vertex } k \text{ has not been yet visited}\} \tag{5.20}$$

where tabu_a is the tabu edge of ant a.

5.4.6 Obtaining a URNDP solution

A final issue is how to convert an ant group to a solution for the URNDP with flows. In the TSP, each ant is performing a TSP tour so finding a corresponding TSP solution amounts to simply tracking the motion of each ant. In the URNDP, the following procedure is adopted after all the ants in a group have terminated:

Obtaining a solution to the URNDP:

1. Construct a network with the same vertex set as the input network containing the edges that were used in the motion of the flow ants.
2. Find the minimum cost tree solution to the flow problem on this network.
3. Add the edges that were used by the reliability ants.
4. Remove any redundant edges using the following procedure:
   (a) Scan all edges that carry no flow, in descending order of fixed cost (starting with the edge that has the largest fixed cost).
   (b) Test if that edge can be removed without violating the reliability requirements. If so, remove that edge and iterate the process.
   (c) If no more edges can be removed, stop.

5.5 Computational implementation of the ant colony algorithm

In the previous sections the details of the rules governing the behaviour of an ant group were described, such as ant motion, trail deposition and solution extraction.

This section concentrates on the overall design and implementation of the algorithm and will address the issues of ant group collaboration, algorithm initialisation and the handling of 'special' situations that can occur during the execution of the algorithm.

5.5.1 Algorithm initialisation and termination

The algorithm runs until a specified number of ant groups have been generated, which shall be denoted by NG (number of groups). The output of the algorithm is the best solution found by the ant groups during the execution. (As mentioned earlier, every ant group generates one URNDP solution.)

When the algorithm starts, the trails are initialised according to the following rules. The edge trails are all equal and set to a small positive number, ϵ. For the flow trails we solve the flow problem on the input network with the variable costs only (no reliability) and we set the trails on the edges of the flow tree to a positive number, ϵ'. The remaining edges are initialised to ϵ, where $\epsilon' > \epsilon$. This is done in order to assist the flow ants to discover promising paths, since the flow tree above often contains 'good' edges that help the algorithm converge more quickly. In the implementation of the algorithm, these initial values were set to $\epsilon = 10$ and $\epsilon' = 20$.

5.5.2 Ant group generation

As mentioned in the previous section, a total of NG ant groups are generated during the execution of the algorithm. The time taken for each ant group to terminate, however, is not constant because in every group the ants follow different paths, some of which will be longer than others. It is useful to restrict the number of ant groups that are present at the same time on the network. This is beneficial for two reasons: firstly it improves computational performance because otherwise the number of ants present on the network can grow very large, requiring more memory resources and limiting the total number of ant groups that can be used. A second and possibly more important reason is that since each ant groups represents a feasible solution, it is better to wait until a few of them have been produced before creating new ant groups in order to take better advantage of the cooperation between the groups.[35] The maximum number of groups that can be present at any time is denoted by NG_{present}. The algorithm starts by generating NG_{present} ant groups and thereafter, when an ant group has terminated, a new ant group is created until the total number of groups NG is achieved.

[35] Also note that the pheromone trail is updated after an ant group has finished and therefore if all the groups start together they will all share, at least initially, the original trail values. Having some ant groups start later allows the 'experience' of the earlier groups to be propagated.

5.5.3 Global trail updating

The trail updating technique presented in section 5.4.3 describes a sequential trail updating model whereupon each ant group that terminates updates the flow and edge trails to reflect the pheromone deposited by the ants. This procedure needs to be slightly modified in the case where many ant groups are present simultaneously.

The reason for this is that ant groups take different times to terminate, and therefore they will generate trail information at various times during the execution of the algorithm. In the implementation of the algorithm, a global flow trail and a global edge trail is present that is shared among all the ant groups, and the trail updating procedure is performed as follows. Whenever an ant group terminates, it calculates the trail *increments* (i.e., the extra amounts of flow/edge trails[36]) per edge which are added (and adjusted to account for evaporation) to the global trails. This follows from the fact that, between the time an ant group was generated (and obtained its initial trail distribution) and the time it terminated the global trail values may have been modified by another ant group. By using increments instead of the actual values of the trail strengths we ensure that the global trail will reflect the correct contributions from all the ant groups, even if the terminating ant group does not have the most up-to-date version of the trail values.

A final implementation detail regarding the trail updating procedure is *trail enhancement*. Whenever an ant group discovers a solution with a smaller cost that the current incumbent, we reward the trail contributions of this group by multiplying them with the *trail enhancement factor* h. This is done to increase (at least temporarily, since all trails evaporate with time) the number of ants using these edges since it is possible that better solutions may be found in that vicinity.

Figures 5.4, 5.5 and 5.6 show, respectively, the algorithms adopted by the ant agents, the ant groups and the overall ACO algorithm for the URNDP with flows.

5.5.4 The situation where an agent cannot leave a vertex

This section deals with a possible situation that may occur during the execution of the ant colony algorithm. It is a relatively infrequent case, but it can happen especially when the input network is sparse.

During the course of the algorithm, the set of the vertices an ant is allowed to move to may become empty ($K = \emptyset$). This can happen when all the vertices adjacent to the

[36]These increments are given by the quantities $Q \sum_a \delta T_f^a(i,j)$ and $Q \Xi_{ij}\, \delta T_e(i,j)$ of equations (5.8) and (5.9) respectively.

current position of the ant have previously been visited or (in the case of reliability ants) when the only edge that it can follow is tabu. Many techniques to overcome this situation can be devised, and the following four have been tested.

(i) A simple technique is to allow ants with $K = \emptyset$ to move along non-existing or tabu edges of the network but without leaving any trail on these edges. In this case, an ant will always have a vertex to go to and the algorithm can proceed normally.

(ii) Another possibility is to allow ants with $K = \emptyset$ to revisit vertices that they have already been visited. This will always provide them with an alternative vertex to move to.

(iii) A third (and possibly the simplest) strategy is when such a situation occurs to disregard the particular ant path. We do not add any trails corresponding to that particular ant and we remove the ant from our consideration. The algorithm continues as normal, but the particular ant group may not produce a feasible URNDP solution because of the missing ant path(s).

(iv) A final possibility could be to remove both the ant and the corresponding ant group from the system. No trails are left by any ants in the group and no solution is produced, so in this way only 'well-behaved' ant groups are allowed to survive.

With regards to the above strategies, the following observations can be made: Method (i) has the disadvantage that the trails left by the ants may be misleading. An ant moving over a non-existent edge, however rarely it does so, may attract other ants to do the same and this may lead to poor algorithm performance due to the large number of infeasible solutions that will be produced.

Strategy (ii) had the advantage that a feasible solution is always produced, and we can expect that the ant system will eventually eliminate the loops in the paths produced via diversification (other ants will find better solutions that do not involve circling and those that have loops will eventually die out). The disadvantage with this approach is that there is a possibility that an ant may get 'stuck' in a cycle and loop forever (or for a very long period of time). This situation could be overcome by combining this strategy with a more radical one such as (iv) via a rule such as "follow strategy (ii) but if the ant in question has not terminated in a certain number of iterations, follow strategy (iv)."

Using the last two strategies (iii) and (iv) has the disadvantage that a significant number of ants or groups may be removed from the system without having any overall effect.

These techniques do not produce good results for networks where there is a high probability of an ant to get stuck, yielding many groups to die out in this fashion without producing a feasible solution.

Computational results obtained with each of the four strategies are presented in table 5.6 of section 5.6. Note that while implementing the above strategies, the solution obtained from each ant group is tested for feasibility before it is compared to the best incumbent solution. This is because some of the strategies may produce infeasible URNDP solutions.

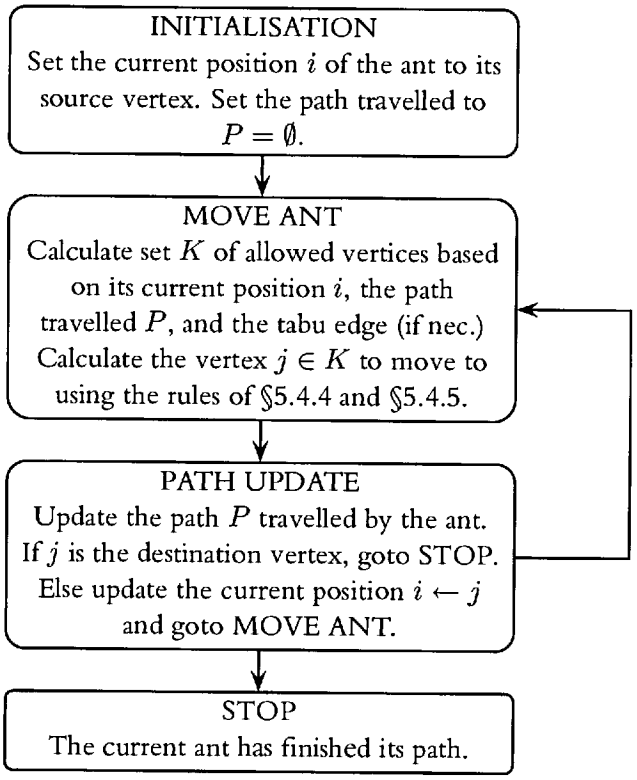


Figure 5.4: Outline of the algorithm used by an ant agent

5.6 Computational results

The heuristic algorithm was implemented using C++ on a *Pentium III* PC with a clock speed of 600MHz. The algorithm was executed twice for each test instance and both the best and the average performance were recorded. (Since the ants use a probabilistic decision rule in their moves, the results obtained by running the algorithm several times can be different.)

We experimented by modifying the values of all the possible parameters of the al-

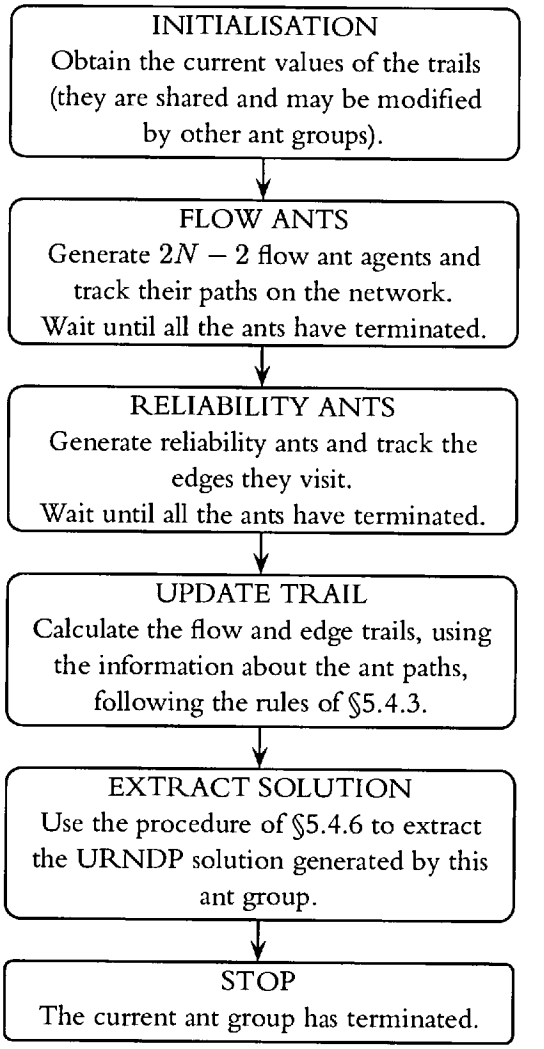


Figure 5.5: Outline of the algorithm used by an ant group

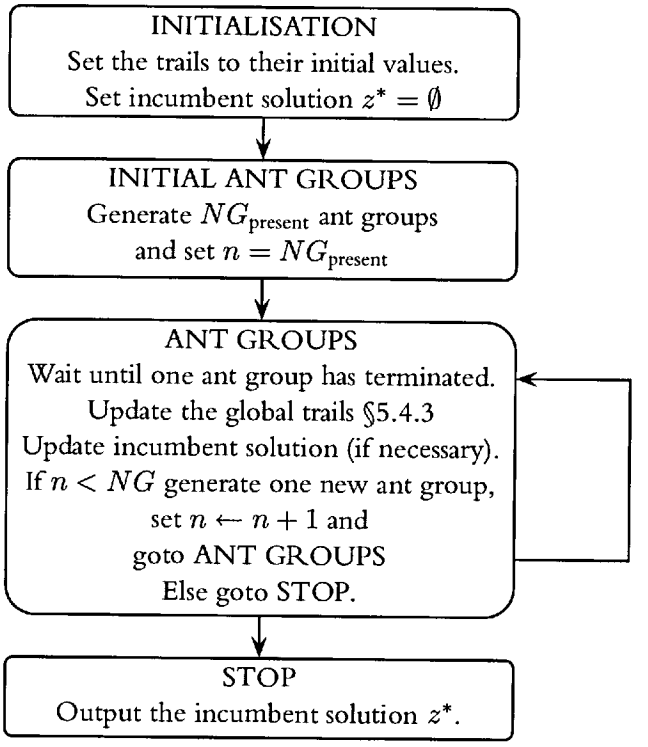


Figure 5.6: Outline of the overall ACO algorithm

gorithm in an effort to find those that produce the best results. A summary of all the algorithmic parameters is presented in table 5.3.

Parameter	*Interpretation*
NG	: Total number of ant groups generated
NG_{present}	: Maximum number of ant groups that can be present simultaneouly
Q	: Amount of trail deposited (§5.4.3)
α, β	: Relative importance of trail strength and small cost (§5.4.4, §5.4.5)
ρ	: Evaporation coefficient (§5.4.3, §5.5.3)
h	: Trail enhancement factor (§5.5.3)
q_0	: Probability of following 'best' edge instead of diversifying (§5.4.4)
S	: The strategy to use (one of four) (§5.5.4)

Table 5.3: The parameters in the ACO algorithm for the URNDP with flows

Recall that the parameters α and β represent the relative weight between the trail strength and visibility (small cost) in the path selection part of the ant algorithm. For this reason, only one of them need to vary and the other one can be fixed. Following Dorigo & Gambardella (1996), we choose to fix α to the value $\alpha = 1$ and vary β in the computational experimentation. The parameter Q can be similarly fixed because it

represents a measure of the quantity of the trail deposited and as such does not have a significant effect on the results.[37] For this reason we fix the value of Q to $Q = 10$.

We investigate first of all the dependence of the algorithm on the parameter β. The results obtained when executing the algorithm with the other parameters set to $NG = 500$, $NG_{\text{present}} = 15$, $\rho = 0.999$, $h = 1.001$, $q_0 = 0.7$ and strategy (ii) are shown in table 5.4. We report the best heuristic value z^* among two program executions of the algorithm, and the optimal value z_{opt} (or the best known solution when the optimal solution is not known). The optimal values were obtained using the Branch and Cut algorithm presented in chapters 3 and 4.

From table 5.4 we can observe that the algorithm performs better for the range between $\beta = 2$ and $\beta = 3$, although in some cases the value $\beta = 5$ produced the same cost as in case $\beta = 3$. By observing the problems with larger objective function values, (for which problems there is more variation among the solution costs) we observe that the value $\beta = 3$ seems to perform better. The last problem was among the only two cases where $\beta = 3$ was outperformed by $\beta = 2$.

				z^* *for different values of* β				
Problem	*Vertices*	*Edges*	z_{opt}	$\beta = 0$	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 5$
g1n12g001i1	12	49	18962	22255	22255	21266	21266	21266
g1n12g005i1	12	50	3230	3765	3713	3673	3673	3860
g1n12g01i1	12	47	2220	2632	2571	2482	2482	2482
g1n15g001i1	15	77	14378	15753	16251	15770	15286	15286
g1n20g001i1	20	137	16964	21267	20267	20267	19781	26767
g1n20g01i1	20	159	1875	1953	1959	1959	1953	1959

Table 5.4: Computational results for different values of β

We next investigate the performance of the algorithm for different values of the probability parameter q_0. The values of the remaining parameters were set to the same values as in the previous investigation and β was set to $\beta = 3$. Among the values tested, the probability $q_0 = 0.7$ appears to perform better for the majority of the test problems.

We finally study the effect of the strategy followed when an ant cannot leave a vertex, in order to decide on the strategy to use in the implementation for the URNDP. Strategy

[37] The results are not affected because if every ant deposits, for example, *double* the amount of trail, the outcome will be the same, since it is the relative strengths of the edge trails that are important. Modifying the parameter Q significantly while keeping the other parameters constant does have an effect on the results, but this is because evaporation and trail weighting (i.e., 'ρ' and 'β') have a different effect depending on whether the values of the edge trails are all very large or very small; this dependence can however be absorbed into these parameters.

				z^* for different values of q_0			
Problem	*Vertices*	*Edges*	z_{opt}	$q_0 = 0.5$	$q_0 = 0.6$	$q_0 = 0.7$	$q_0 = 0.8$
g1n15g001i1	15	77	14378	15770	16251	15753	15770
g1n15g01i1	15	80	2063	2173	3123	2123	2173
g1n20g001i1	20	137	16964	20767	20267	19782	20267
g1n20g01i1	20	159	1875	2009	1959	1953	1959
g1n20g1i1	20	144	655	661	660	657	660

Table 5.5: Computational results for different values of q_0

				Strategy I	*Strategy II*	*Strategy III*	*Strategy IV*
Problem	*Vertices*	*Edges*	z_{opt}	z^*_{I}	z^*_{II}	z^*_{III}	z^*_{IV}
g1n12g005i1	12	50	3230	3597	3768	4460	3673
g1n15g005i1	15	83	3375	3636	3658	3728	3658
g1n15g01i1	15	80	2046	2123	2173	2223	2173
g1n20g001i1	20	137	16964	19782	20267	20282	19782
g1n20g005i1	20	144	4802	5287	5485	5575	5485
g1n20g01i1	20	159	1875	1953	1959	2009	1959

Table 5.6: Comparison of the different strategies used when an ant cannot leave a vertex

(iv) is a weak candidate, because all 'problematic' ant groups are discarded, and, in some cases, many such groups will be generated. Among the strategies (i), (ii), (iii), the strategy (ii) appeared to perform the best. This can be also explained by observing that this strategy is less radical than strategy (i), since the latter strategy introduces infeasibility into the solutions, by allowing jumps over non-existing edges. Strategy (ii) does not suffer from this problem.

The test problems for the URNDP with flows used for the ACO algorithm are the same instances as those used in chapters 3 and 4 (and are described in section 2.7). The total number of groups used for each network size is shown in table 5.7.

Vertices	NG	$NG_{present}$	*Vertices*	NG	$NG_{present}$
12	2000	15	30	300	15
15	2000	15	40	200	10
20	1000	15	50	100	10
25	500	15			

Table 5.7: Parameters used in the ACO algorithm for the various groups of test problems

For each test problem, two runs with the algorithm were performed. Table 5.8 presents the average performance over the five instances of each group. The notation

used is the following:

z_1^*, z_2^* The cost of the solution to the URNDP reported by the ACO algorithm in each of the two executions.

z_{opt} The cost of the optimal solution to the URNDP or the cost of the best feasible solution found at the end of the branch and cut algorithm.

$\%z_1^*$ Percentage ratio $z_1^*/z_{\text{opt}} - 1$.

$\%z_2^*$ Percentage ratio $z_1^*/z_{\text{opt}} - 1$.

$\%z_{best}^*$ The smallest number among $\%z_1^*$ and $\%z_2^*$.

$\%z_{av}^*$ The average value of $\%z_1^*$ and $\%z_2^*$.

t_{av} The computational time (in seconds) taken to execute the algorithm averaged over the two runs.

From the computational results, we observe that the worst performance of the algorithm was for the problem groups with $\gamma = 0.01$ and $\gamma = 0.05$, where the largest percentage errors were produced. For the instances corresponding to $\gamma = 1$ and $\gamma = 2$ the algorithm performed extremely well even for the largest network sizes, with all percentage errors being less than 4%. The most difficult instances detected by the exact Branch and Cut algorithm which correspond to $\gamma = 0.1$ are of average difficulty to the ACO algorithm.

5.7 Summary

In this chapter a heuristic based on ant colony optimisation (ACO) was presented. The algorithm employs the principles of ACO but also incorporates several novel extensions. These include the use of two different types of agents (flow ants and reliability ants), the use of two different types of pheromone trails, requiring new rules to govern the exchange of information between the different agents and the agents' behaviour according to the different types of trail (via the trail function). The computational experience reported is of great interest; the absence, however, of other heuristic or metaheuristic approaches to the URNDP with flows in the literature makes the comparison to the optimal solutions obtained using the exact Branch and Cut algorithm the only possible evaluation of the ACO algorithm at this moment. Moreover, given that ACO is a relatively new area of optimisation, its full potential to solving combinatorial optimisation and, in particular, network design problems has yet to be fully understood.

Vertices	*Edges*	γ	NG	NG_{present}	$\%z^*_{best}$	$\%z^*_{av}$	t_{av}
12	47.0	0.01	2000	15	8.45	9.03	42.1
12	51.4	0.05	2000	15	9.05	9.49	36.0
12	49.6	0.1	2000	15	6.24	7.00	38.4
12	49.8	1	2000	15	0.33	0.44	35.4
12	47.6	2	2000	15	0.83	1.07	31.9
	Average				*4.98*	*5.40*	*36.8*
15	78.0	0.01	2000	15	10.00	10.62	156.6
15	76.6	0.05	2000	15	6.24	6.24	180.0
15	80.4	0.1	2000	15	3.31	4.20	143.5
15	80.2	1	2000	15	1.43	1.76	173.4
15	78.2	2	2000	15	0.55	0.67	126.9
	Average				*4.31*	*4.70*	*156.1*
20	140.0	0.01	1000	15	14.58	14.58	309.5
20	143.4	0.05	1000	15	14.28	14.74	368.4
20	145.0	0.1	1000	15	7.16	7.16	348.1
20	145.4	1	1000	15	0.85	0.91	347.3
20	140.6	2	1000	15	0.86	0.98	325.6
	Average				*7.54*	*7.67*	*339.7*
25	231.6	0.01	500	15	20.76	21.65	560.5
25	220.4	0.05	500	15	21.53	22.91	598.5
25	227.6	0.1	500	15	10.41	11.23	470.8
25	226.6	1	500	15	1.11	1.26	526.9
25	224.4	2	500	15	1.88	1.93	551.5
	Average				*11.14*	*11.80*	*541.6*
30	323.2	0.01	300	15	22.02	22.65	1019.8
30	326.0	0.05	300	15	18.95	19.71	888.5
30	330.2	0.1	300	15	16.71	16.86	950.3
30	329.0	1	300	15	2.49	2.60	844.5
30	330.8	2	300	15	1.87	1.88	859.5
	Average				*12.41*	*12.74*	*912.5*
40	581.2	0.01	200	10	26.46	27.16	3197.0
40	586.4	0.05	200	10	20.62	21.30	3257.6
40	579.4	0.1	200	10	21.27	21.77	3297.2
40	583.6	1	200	10	2.67	2.84	3423.7
40	577.6	2	200	10	2.54	2.56	2912.0
	Average				*14.71*	*15.13*	*3217.5*
50	918.6	0.01	100	10	31.80	33.47	5370.3
50	917.8	0.05	100	10	11.34	11.91	5038.9
50	914.2	0.1	100	10	15.82	16.66	5166.4
50	929.8	1	100	10	2.77	2.93	5876.1
50	916.0	2	100	10	3.27	3.49	5095.2
	Average				*13.00*	*13.69*	*5309.4*

Table 5.8: Computational results: ACO algorithm (averages)

Problem	*Vertices*	*Edges*	γ	NG	NG_{present}	z^*_{best}	z_{opt}	$\%z^*_{best}$	$\%z^*_{av}$	t_{av}
g1n12g001i5	12	47	0.01	2000	15	18287	17434	4.89	7.79	40.6
g1n12g005i3	12	50	0.05	2000	15	3514	3372	4.21	4.21	34.8
g1n12g01i2	12	55	0.1	2000	15	2384	2177	7.44	8.47	38.4
g1n12g1i3	12	49	1	2000	15	370	370	0.00	0.00	30.7
g1n12g2i1	12	49	2	2000	15	381	381	0.00	0.00	26.2
g1n15g001i4	15	79	0.01	2000	15	18718	17136	9.23	9.23	141.0
g1n15g005i3	15	72	0.05	2000	15	3687	3432	7.43	7.43	234.3
g1n15g01i4	15	85	0.1	2000	15	1482	1475	0.47	0.47	159.8
g1n15g1i3	15	78	1	2000	15	354	344	1.45	2.18	138.5
g1n15g2i4	15	78	2	2000	15	322	320	0.63	0.63	135.0
g1n20g001i2	20	145	0.01	1000	15	20816	18008	15.59	15.59	371.2
g1n20g005i3	20	142	0.05	1000	15	4921	4426	11.18	11.18	461.2
g1n20g01i3	20	143	0.1	1000	15	2892	2582	12.01	12.01	484.5
g1n20g1i1	20	144	1	1000	15	657	655	0.03	0.03	300.2
g1n20g2i4	20	149	2	1000	15	584	580	0.69	0.69	314.1
g1n25g001i5	25	246	0.01	500	15	22874	19121	19.63	20.94	668.9
g1n25g005i4	25	219	0.05	500	15	8102	5356	45.74	48.51	861.5
g1n25g01i4	25	230	0.1	500	15	2966	2586	14.69	14.69	405.9
g1n25g1i3	25	224	1	500	15	654	654	0.00	0.38	388.0
g1n25g2i1	25	232	2	500	15	483	468	3.21	3.21	452.5
g1n30g001i2	30	328	0.01	300	15	29677	22440	32.11	32.18	1491.8
g1n30g005i4	30	321	0.05	300	15	6225	5675	9.69	9.69	807.9
g1n30g01i2	30	329	0.1	300	15	3625	2910	24.57	24.60	877.0
g1n30g1i2	30	314	1	300	15	870	850	2.35	2.35	681.0
g1n30g2i2	30	337	2	300	15	683	668	2.25	2.32	841.8
g1n40g001i3	40	590	0.01	200	10	33931	23964	41.58	41.59	2672.1
g1n40g005i5	40	575	0.05	200	10	7997	6193	29.13	29.94	4037.4
g1n40g01i4	40	606	0.1	200	10	5740	4414	28.91	29.47	4302.7
g1n40g1i4	40	579	1	200	10	998	956	4.39	4.39	3199.2
g1n40g2i3	40	572	2	200	10	868	842	3.09	3.09	3239.6
g1n50g001i1	50	915	0.01	100	10	40107	30076	31.69	32.52	5061.4
g1n50g005i5	50	917	0.05	100	10	8262	7925	2.99	3.62	5036.5
g1n50g01i1	50	907	0.1	100	10	4356	3834	13.62	13.65	5659.8
g1n50g1i2	50	955	1	100	10	1213	1179	2.88	2.97	5309.9
g1n50g2i1	50	903	2	100	10	907	881	2.95	3.35	3840.4

Table 5.9: Computational results: ACO algorithm (test instances)

Problems worthy of attack prove
their worth by hitting back.

Problems (1969)
Piet Hein

Chapter 6

A branch and cut algorithm for the capacitated RNDP with flows

6.1 Introduction

In the previous chapters the uncapacitated RNDP with flows was studied, and two algorithms for its solution were proposed. In this chapter we consider the capacitated version of this problem, where capacity restrictions are imposed on the edges of the network. A new MIP formulation for the capacitated RNDP (CRNDP) with flows is presented and solved to optimality using a Branch and Cut algorithm.

6.2 Capacitated network design problems

We begin with an overview of capacitated network design problems with an emphasis to network problems that contain both fixed and variable costs. In the next section, a procedure that can convert a capacitated network design problem (with flows) to an equivalent uncapacitated problem is presented.

6.2.1 Relation to uncapacitated problem

A transformation by Wagner (1959) relates any capacitated network design problem to an uncapacitated one via the insertion of artificial vertices. The procedure can be summarized as follows.

Let $G(V, A)$ be a network with an arc set A and capacities

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \text{for } (i,j) \in C \subseteq A. \tag{6.1}$$

Let the cost of the arc (i, j) be $c_{ij}(x)$, a function of the flow x that passes through the arc. Then the extended network G' is defined as follows:

- the original nodes of G are present in G'
- for each arc (i, j) in C, a new node, k say, is added in G'. Then this node is joined to the nodes i and j with costs
  - $c'_{ki} = 0$
  - $c'_{kj} = c_{ij}(l_{ij} + x_{ij})$
- the node demands are adjusted by
  - $d'_i = d_i + u_{ij}$
  - $d'_j = d_j - l_{ij}$
  - $d'_k = l_{ij} - u_{ij}$

The inserted node k is a source node. The modified uncapacitated network G' has $|V| + |C|$ nodes and $|A| + |C|$ arcs.

The problem in the application of this transformation is that the size of the network increases so much that, except for very small networks, this technique cannot be used in practice. For example, a fully connected network with N customers (i.e., $N+1$ vertices) has $N(N-1) + N = N^2$ arcs, and if all edges are capacitated, then the transformed uncapacitated network will have $N^2 + N + 1$ vertices and $2N^2$ arcs! A 20-customer capacitated problem (which can be solved in less than an hour using the B&C algorithm developed in this chapter) is transformed, using the above procedure, to an uncapacitated problem with 421 vertices and 800 arcs, a manifold increase in complexity.

6.2.2 Overview of capacitated network design problems

In general the capacitated version of a network design problem is much more difficult than its uncapacitated equivalent. For some cases even to decide if a feasible solution exists is $\mathcal{NP}$-hard (Dell'Amico, Maffioli & Martello (1997)). This justifies the relatively less research done in this area compared with the field of uncapacitated network design.

It has been observed, moreover, that flow-based capacitated network design problems lead to large, degenerate problems with a big integrality gap. Moreover, the solution to a capacitated network design problem can be unstable: changing the input network structure slightly yields a significant change in the solution and computational time (Araque, Hall & Magnanti (1990)).

Many different methods to impose capacities on network design problems exist in the literature. For networks without flows, the capacities may restrict the number of node disjoint paths that exist in a network (Ameur (2000)). Another popular group of capacitated network design problems falls in the category of *capacity assignment*, which consists of deciding the values of the capacities that need to be assigned on a network (at a minimum cost) such that some design requirements are satisfied (e.g., Günlük (1999)).

An overview of capacitated network design problems where the capacities on the flows are part of the input can be found in Melkote & Daskin (2001). The capacitated vehicle routing problem is also an extensively studied problem (e.g., see Toth & Vigo (2002) for an overview), which consists of optimally routing a set of vehicles with a given capacity from a central depot to a given set of customers such that the customers' demands are satisfied. Given the various applications of the problem numerous solution techniques have been developed, for example see Achuthan, Caccetta & Hill (1998), Baldacci, Hadjiconstantinou & Mingozzi (2003), Augerat, Belenguer, Benavent, Corberán, Naddef & Rinaldi (1995) and Augerat, Belenguer, Benavent, Corberán & Naddef (1998).

The capacitated fixed charge problem has also been extensively studied, but given its increased complexity most of the research focuses on heuristic procedures. A dual ascent algorithm for finding lower bounds to the optimal solution of the capacitated fixed charge problem can be found in Herrmann, Ioannou, Minis & Proth (1996) (and a corrected version by Gendron (2002)), and a procedure based on Lagrangian relaxations was proposed by Crainic, Frangioni & Gendron (2001). Cutting plane algorithms for capacitated network design with fixed costs have also been developed, e.g., by Magnanti, Mirchandani & Vachani (1995), whereas Sridhar & Park (2000) present a Benders and Cut algorithm for the capacitated fixed charge problem with zero variable costs.

More recently, Ghamlouche, Crainic & Gendreau (2003) proposed a tabu search algorithm for general fixed charge multicommodity network design problems, which performs better than classical heuristics such as greedy descent and resource decomposition.

6.3 A mixed integer programming formulation

In this section we present a new mixed integer programming formulation for the reliable network design problem with flows and capacities.

The reliability requirements of the RNDP were defined in section 2.6, and for the CRNDP they imply that:

(i) The flows of all edges that are included in the design must satisfy given upper bounds, and

(ii) If any edge fails, then the remaining network must able to supply all the customers (satisfying all their demands). This must be done without having to violate any edge capacity constraints (i) above.

This means that the CRNDP with flows cannot be obtained by simply changing the quantity Q that appears the constraints **M7**.5 and **M7**.6 of the MIP formulation of the URNDP (section 3.2). This is due to the fact that although the equations of the form

$$x_{ij} \leq Q\xi_{ij} \tag{6.2}$$

where Q is the upper capacity of the edges are still valid, the reliability constraints

$$\sum_{\substack{i\in S,\, j\in S' \\ (i,j)\in E \\ i<j}} \xi_{ij} + \sum_{\substack{i\in S,\, j\in S' \\ (i,j)\in E \\ i>j}} \xi_{ji} \geq 2, \qquad \text{for all } 0 \in S \subseteq V,\ S' = V\backslash S \tag{6.3}$$

are not enough to guarantee two-edge reliability (which was defined as the ability of the network to survive a single edge failure). This happens because it is possible that, when a cut contains two edges say, and one of them fails, the other may not have enough capacity to supply the customers that the failed edge did and, therefore, no normal network operation is possible following the breakdown of that edge. This observation implies that the capacitated RNDP with is more difficult than the URNDP and requires more constraints to be present in order to ensure that the required level of reliability is achieved. In other words, for the CRNDP two-edge connectivity is not enough.

The Mixed Integer Programming formulation for the *Capacitated Reliable Network Design Problem (CRNDP) with flows* is defined in model **M12**, and the interpretation of the notation used is presented in table 6.1.

MODEL M12

$$\textbf{min} \quad \sum_{\substack{(i,j)\in E \\ i<j}} (b_{ij}\xi_{ij} + c_{ij}x_{ij} + c_{ji}x_{ji}) \qquad \textbf{(M12.1)}$$

subject to

$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = d_i \qquad \text{for all } i \in V' \qquad \textbf{(M12.2)}$$

$$\sum_{j:(0,j)\in E} x_{0j} = \sum_{i\in V'} d_i \qquad \textbf{(M12.3)}$$

For all S with $0 \in S \subset V$, $S' = V \backslash S$:

$$\sum_{\substack{(m,n)\in\delta(S) \\ (m,n)\neq(i,j)}} u_{mn}\xi_{mn} \geq \max\left\{1, \sum_{n\in S'} d_n\right\}, \qquad \forall (i,j) \in \delta(S) \qquad \textbf{(M12.4)}$$

$$x_{ij} \leq \xi_{ij}u_{ij} \qquad \text{for all edges } (i,j),\ i<j \qquad \textbf{(M12.5)}$$

$$x_{ji} \leq \xi_{ij}u_{ij} \qquad \text{for all edges } (i,j),\ i<j \qquad \textbf{(M12.6)}$$

$$x_{ij} \geq 0 \qquad \text{for all edges } (i,j) \qquad \textbf{(M12.7)}$$

$$\xi_{ij} \in \{0,1\} \qquad \text{for all edges } (i,j),\ i<j \qquad \textbf{(M12.8)}$$

THE CAPACITATED RELIABLE NETWORK DESIGN PROBLEM WITH FLOWS

Symbol	*Interpretation*
V':	$V\backslash\{0\}$, the set of vertices excluding the source
d_i:	(integer) demand of vertex i, where $i \in V'$
b_{ij}:	fixed cost, $(i,j) \in E$, $i<j$
c_{ij}:	linear cost per unit flow, $(i,j) \in E$
u_{ij}:	edge capacity, $(i,j) \in E$, $i<j$
ξ_{ij}	are (unknown) binary variables, defined for $(i,j) \in E$, $i<j$, representing whether a given edge is present or not
x_{ij}	are the (unknown) flow variables, defined for $(i,j) \in E$.

Table 6.1: The notation used in the MIP formulation of the CRNDP with flows

The flow balance constraints **M12**.2 and **M12**.3 are the same as in the uncapacitated case and state that the flow in the network is conserved.

The trivial constraints **M12**.5 and **M12**.6 are similar to the trivial constraints of the URNDP but, in the capacitated problem, the upper bounds on the flows are set to the edge capacities u_{ij}.

The reliability constraints, which will be referred to as the *capacitated reliability constraints*, are represented by **M12**.4 and their description is explained below. The reliability constraints used in the uncapacitated problem are not enough to guarantee the same level of reliability in the capacitated case.

These constraints state that if edge (i, j) is removed from the cut, the remaining edges will have enough capacity to supply the demand of the set S' (this is depicted in figure 6.1, where s is the source vertex). This is a necessary and sufficient condition to guarantee the reliability requirements mentioned above. The right hand side of (6.4) must be greater or equal to 1, since if the total demand of S' is zero then we need at least two edges to be present in the cut to ensure two-edge connectivity, just as in the case of the uncapacitated version. If one assumes that there are no transportation vertices (that is, all demands are strictly positive) the 'max' is always more than one and the constraints can be simplified further.

Finally constraints **M12**.7 and **M12**.8 are the non-negativity and integrality constraints imposed on the variables. The objective function **M12**.1 consists of a fixed part and a cost proportional to the flow through the edge, and is of the same form as in the URNDP with flows.

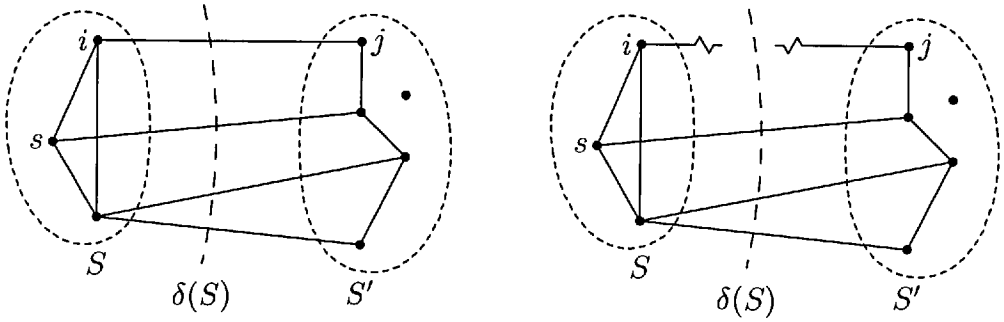


Figure 6.1: Example of an edge failure in the CRNDP with flows

The capacitated reliability constraints **M12**.4 can be re-written as follows: if we add

the term $u_{ij}\xi_{ij}$ to both sides of the inequality we obtain

For all cuts $\delta(S)$ with $0 \in S \subset V$:

$$\sum_{(m,n)\in\delta(S)} u_{mn}\xi_{mn} \geq \max\left\{1, \sum_{n\notin S} d_n\right\} + u_{ij}\xi_{ij}, \quad \text{for all edges } (i,j) \in \delta(S),\ i \in S. \tag{6.4}$$

The equivalent MIP obtained from this more compact formulation is shown in **M13**.

MODEL M13

$$\textbf{min} \quad \sum_{\substack{(i,j)\in E \\ i<j}} (b_{ij}\xi_{ij} + c_{ij}x_{ij} + c_{ji}x_{ji}) \tag{M13.1}$$

subject to

$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = d_i \qquad \text{for all } i \in V' \tag{M13.2}$$

$$\sum_{j:(0,j)\in E} x_{0j} = \sum_{i\in V'} d_i \tag{M13.3}$$

For all S with $0 \in S \subset V$, $S' = V\backslash S$:

$$\sum_{(m,n)\in\delta(S)} u_{mn}\xi_{mn} \geq \max\left\{1, \sum_{n\in S'} d_n\right\} + u_{ij}\xi_{ij}, \quad \forall (i,j) \in \delta(S) \tag{M13.4}$$

$$x_{ij} \leq \xi_{ij}u_{ij} \qquad \text{for all edges } (i,j),\ i<j \tag{M13.5}$$

$$x_{ji} \leq \xi_{ij}u_{ij} \qquad \text{for all edges } (i,j),\ i<j \tag{M13.6}$$

$$x_{ij} \geq 0 \qquad \text{for all edges } (i,j) \tag{M13.7}$$

$$\xi_{ij} \in \{0,1\} \qquad \text{for all edges } (i,j),\ i<j \tag{M13.8}$$

THE CAPACITATED RELIABLE NETWORK DESIGN PROBLEM WITH FLOWS - EQUIVALENT FORMULATION

With respect to the capacitated reliability constraints **M13**.4, the following observations can be made. For a given set S, a group of capacitated reliability constraints will be generated, one for every edge in the cut $\delta(S)$. All these constraints, however, will have the same left-hand side, and will differ only in the right-hand term which will vary according to the edge $(i,j) \in \delta(S)$ selected.

This means that, in the end, only one of constraints of each group will be effective,

namely the one with the largest right-hand side, i.e.,

$$\sum_{(m,n)\in\delta(S)} u_{mn}\xi_{mn} \geq \max\left\{1, \sum_{n\in S'} d_n\right\} + \max_{(i,j)\in\delta(S)}\{u_{ij}\xi_{ij}\} \tag{6.5}$$

which is the constraint that corresponds to the edge (i,j) with the largest capacity among all the edges of the cut that are present in the solution.[38]

The reason why all the constraints are included in formulation **M13** instead of only the 'effective ones' is that we cannot know beforehand which edges will be present in the solution and which will not, and thus we do not know which constraints will be the 'effective' ones until after we have solved the problem. That is, the calculation on the maximum in expression (6.5) includes the values of the decision variables ξ which are part of the problem unknowns.

6.3.1 Layout of the solution

In contrast to the URNDP, the solution of the CRNDP with flows is not a superposition of a tree containing the flows and some additional 'reliability' edges. This is because although cost minimisation will try to push the maximum amount of flow possible through the cheapest edges, the upper capacity values may be reached and hence no more flow can be pushed through this edge, so another edge should be used to carry the remaining flow.

Another interesting observation regarding the layout of the solution is the following: from cost minimisation it appears at first that all the flow arcs incident to a demand vertex will be saturated except one (since the solution will try to push as much flow through the cheapest edges, and only if one edge becomes saturated a new, more costly flow edge will be added). This is shown in the example of figure 6.2 where vertex i has four incident edges of capacity u. However, this is not always true. The reason why this reasoning does not work in general is that the layout of the solution may prohibit some of the "cheaper" edges to become saturated because of a saturated edge further upstream along the path from the source. In other words, it is not always true that (at least) $\lfloor d_i/u \rfloor$ of the edges incident to i will be saturated because other saturated edges in the network can restrict the maximum amount of flow through these edges.

[38]This can be intuitively interpreted as follows: for each cut $\delta(S)$ it is only necessary to consider the worst-case scenario which corresponds to the edge with the largest capacity breaking down; if the network can cope with such a failure, then it can certainly cope with failures of the smaller-capacity edges of the cut.

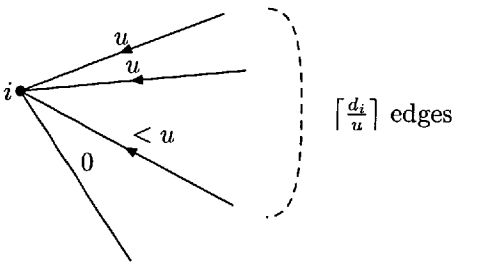


Figure 6.2: The number of saturated edges incident to a vertex

6.4 The initial LP relaxation

This section describes the initial LP relaxation of the capacitated model **M13** which will provide an initial lower bound on the optimal solution of the problem. The first step is to relax the integrality constraints **M13**.8 to their continuous equivalent

$$0 \leq \xi_{ij} \leq 1. \tag{6.6}$$

The capacitated reliability constraints **M13**.4 present the same computational difficulty as in the URNDP, since there is an exponentially large number of them. (Their number is actually much larger than in the URNDP.) This suggests that many of them must be removed from the LP relaxation, and a separation algorithm has to be used to detect any violated ones.

For this reason, we have decided to include the capacitated reliability constraints **M13**.4 that correspond to $|S'| = 1$, i.e., when S' contains only one vertex. The complete initial LP relaxation with this reduced set of constraints is shown in **M14**.

MODEL M14

$$\textbf{min} \quad \sum_{\substack{(i,j)\in E \\ i<j}} (b_{ij}\xi_{ij} + c_{ij}x_{ij} + c_{ji}x_{ji}) \tag{M14.1}$$

subject to

$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = d_i \qquad \text{for all } i \in V' \tag{M14.2}$$

$$\sum_{j:(0,j)\in E} x_{0j} = \sum_{i\in V'} d_i \tag{M14.3}$$

For all $j \neq 0$:

$$\sum_{m:(m,j)\in E} u_{mj}\xi_{mj} \geq \max\{1, d_j\} + u_{ij}\xi_{ij}, \qquad \forall (i,j) \in E,\ i \in S,\ j \in S' \tag{M14.4}$$

$$x_{ij} \leq \xi_{ij}u_{ij} \qquad \text{for all edges } (i,j),\ i<j \tag{M14.5}$$

$$x_{ji} \leq \xi_{ij}u_{ij} \qquad \text{for all edges } (i,j),\ i<j \tag{M14.6}$$

$$x_{ij} \geq 0 \qquad \text{for all edges } (i,j) \tag{M14.7}$$

$$0 \leq \xi_{ij} \leq 1 \qquad \text{for all edges } (i,j),\ i<j \tag{M14.8}$$

THE INITIAL RELAXATION OF THE CRNDP

Having formulated the LP relaxation as shown we need a (preferably fast) algorithm to test if a given solution with binary ξ's violates any capacitated reliability constraints, in other words to check if there are any constraints **M13**.4 that are not satisfied by that solution. If no such constraints exist, then we can conclude that the solution is feasible for the CRNDP, otherwise we can identify them and add them to the formulation. The next section is dedicated to this task.

6.4.1 An exact separation algorithm for the capacitated reliability constraints

We seek an algorithm to separate the capacitated reliability constraints **M13**.4, i.e., given a solution that satisfies constraints **M13**.2, **M13**.3 and **M13**.5–**M13**.8, to identify a constraint **M13**.4 that is violated by this solution or decide that no such constraint exists. Such an algorithm is presented below. We consider a network with N vertices, labelled $0, 1, \ldots, N-1$, and a solution with integer values for the binary variables ξ_{ij}.

Separation algorithm for the reliability constraints (Feasibility Test)

1. **For** all edges (a, b) that are present in the solution:
2. Remove edge (a, b).
3. Give each of the remaining edges (i, j) that are included in the network a weight $w_{ij} = u_{ij}$.
4. Adjust the weights of the edges incident to the source vertex to

$$w_{0j} \leftarrow w_{0j} - d_j \tag{6.7}$$

   creating the edge $(0j)$ if it is not present.

5. Some of the weights will be negative. Covert the network to one with positive weights by adding an artificial vertex, labelled N, and replacing the weights as follows

$$\text{edge } (0, j): \ \max\{0, w_{0j}\}, \tag{6.8}$$

$$\text{edge } (j, N): \ \max\{0, -w_{0j}\}. \tag{6.9}$$

6. Find the minimum cut separating vertices 0 and N (e.g., using a maximum flow algorithm (Ford & Fulkerson (1962)) or the Gomory-Hu algorithm).
7. Remove artificial vertex N and all edges incident to it.
8. Adjust value of the cut

$$\text{cut} \leftarrow \text{cut} - \sum_{j=1}^{N} \max\{0, -w_{0j}\}. \tag{6.10}$$

9. If cut < 0 a violated constraint **M13**.4 is found, the set S corresponds to one shore of the cut and the edge (i, j) of **M13**.4 is the edge (a, b).
10. **EndFor**
11. If cut ≥ 0 for all (a, b) then no violated constraints **M13**.4 exist.

Note the similarities with the separation of the capacity constraints of the Capacitated Vehicle Routing Problem. An example for this procedure is shown in figure 6.3, where the edge $(1, 2)$ is removed. The reason why this algorithm succeeds is that a minimum

cut separating 0 and n in the modified graph corresponds to a cut $\delta(S)$ $(0 \notin S)$ that minimises the quantity

$$\sum_{(i,j)\in\delta(S)} u_{ij}\xi_{ij} - d(S) \tag{6.11}$$

in the original graph. The last term is derived from the weights of the edges $(0j)$, since these edges are present in the cut if and only if $j \notin S$.

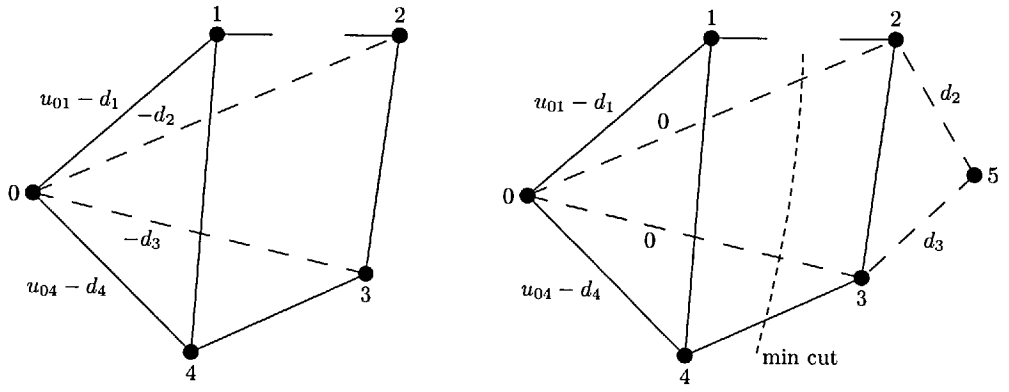


Figure 6.3: The separation of the capacitated reliability constraints

6.5 Valid inequalities for the CRNDP

6.5.1 Valid inequalities derived from the URNDP

Given that the CRNPD is a special case of the URNDP, all the valid inequalities for the URNDP presented in chapter 3 are valid and can be used for the solution the CRNDP as well.

We note however, that some of them can be modified or strengthened in order to perform better in the capacitated version of the problem. The computational performance of the cuts described in chapter 3 will be studied in section 6.8.

6.5.2 The capacitated degree inequalities with equal capacities

The capacitated degree inequalities are similar to the degree constraints **M8**.4 of the URNDP with flows which were presented in section 3.4. In the uncapacitated case, they required that every vertex has at least two edges incident to it, a necessary requirement for a two-edge connected network. These inequalities remain valid for the CRNDP, but

can be further strengthened, to obtain a new set of valid inequalities, the *capacitated degree inequalities.*

Consider the case where all the capacities are equal, i.e., $u_{ij} = u$. Then the reliability constraints corresponding to a single demand vertex **M14**.4 can be written (assuming that all demands are greater than zero) as

$$\sum_{m:(m,j)\in E} \xi_{mj} \geq \frac{d_j}{u} + \xi_{ij}, \quad \text{for all } i \text{ with } (i,j) \in E \tag{6.12}$$

where j is the given demand vertex. Since the LHS is the same for all choices of i, we can see that the strongest inequality is obtained when the RHS is as large as possible, this will correspond to the case where $\xi_{ij} = 1$ since at least one edge containing vertex j will exist. This yields the following inequality which is valid for the CRNDP:

$$\sum_{j:(i,j)\in E} \xi_{ij} \geq \left\lceil \frac{d_i}{u} \right\rceil + 1, \quad \text{for all } i \in V. \tag{6.13}$$

This inequality will be referred to as the *capacitated degree inequality* for vertex j. Note that this inequality is valid for the source vertex too, if we define its demand to be the total demand, i.e., $d_0 := \sum_{k\neq 0} d_k$.

The capacitated degree inequalities are a special case of a more general category of valid inequalities, the capacitated reliability inequalities which are studied in the next section.

6.5.3 The capacitated reliability inequalities

Theorem 18 *For any cut $\delta(S)$ the following inequality is valid for the CRNDP*

$$\xi(\delta(S)) \geq \left\lceil \frac{d(S)}{U} \right\rceil + 1 \tag{6.14}$$

where

$$U = \max_{(i,j)\in\Delta} u_{ij} \tag{6.15}$$

with $\Delta \subset \delta(S)$ is any set of $|\delta(S)| - 1$ edges of the cut, i.e., the cut with one edge removed.

Proof

Let e be the edge not included in Δ, i.e., $\Delta = \delta(S)\backslash e$. Let $e' \in \delta(S)$ be the edge such that

$$u_{e'} = \max_{\substack{(i,j)\in\delta(S) \\ \xi_{ij}=1}} u_{ij}. \tag{6.16}$$

We have the following cases:

(i) *Either* $e' \in \Delta$ and therefore $u_{e'} \leq U$,

(ii) *or* $e' \notin \Delta$ and therefore $e' = e$.

The second case can be further subdivided into two subcases, $u_{e'} \leq U$ and $u_{e'} > U$.

If $u_{e'} \leq U$ (in either of the cases (i) or (ii)) the reliability constraint **M13**.4 becomes

$$\sum_{(i,j)\in\delta(S)} u_{ij}\xi_{ij} \geq d(S) + \xi_{e'}u_{e'} \tag{6.17}$$

but since

$$\sum_{(i,j)\in\delta(S)} u_{ij}\xi_{ij} \leq \xi(\delta(S))u_{e'} \quad \text{and} \quad \xi_{e'} = 1 \tag{6.18}$$

we obtain

$$\xi(\delta(S))u_{e'} \geq d(S) + u_{e'} \tag{6.19}$$

i.e.,

$$\xi(\delta(S)) \geq \frac{d(S)}{u_{e'}} + 1 \geq \frac{d(S)}{U} + 1 \tag{6.20}$$

hence

$$\xi(\delta(S)) \geq \left\lceil \frac{d(S)}{U} \right\rceil + 1 \tag{6.21}$$

since the left hand side is integer.

We only need to consider the alternative $e' = e$ with $u_{e'} > U$. Assume that the cut has only $\lceil d(S)/U \rceil$ edges. We shall reach a contradiction. The sum of the capacities of the edges of the cut is

$$\begin{aligned} C := \sum_{(i,j)\in\delta(S)} u_{ij}\xi_{ij} &= u_e + \sum_{(i,j)\in\Delta} u_{ij}\xi_{ij} \\ &\leq u_e + \sum_{(i,j)\in\Delta} U\,\xi_{ij} \\ &= u_e + \left(\left\lceil \frac{d(S)}{U} \right\rceil - 1 \right) U. \end{aligned} \tag{6.22}$$

The spare capacity of the cut (total capacity − total flow) equals

$$\begin{aligned} \text{spare capacity} &= C - d(S) \\ &= u_e - U + \left(\left\lceil \frac{d(S)}{U} \right\rceil - \frac{d(S)}{U} \right) U \\ &= u_e - U + Uf, \end{aligned} \tag{6.23}$$

where f is defined as

$$f := \left\lceil \frac{d(S)}{U} \right\rceil - \frac{d(S)}{U} \tag{6.24}$$

so

$$0 \leq f < 1. \tag{6.25}$$

If the edge e, normally carrying flow x say, fails, then this leaves an amount of flow equal to x to be re-routed among the remaining edges, and also yields a reduction of the total spare capacity by $u_e - x$, as the spare capacity of that edge cannot be used any more. The total spare capacity of the remaining edges is

$$u_e - U + Uf - (u_e - x) = x - (1 - f)U < x \tag{6.26}$$

and hence there is not enough spare capacity to re-route the x units of flow following the breakdown of that edge. Therefore, in this case too, $\lceil d(S)/U \rceil$ edges are not enough, and so (6.14) must hold. □

6.5.4 The capacitated degree inequalities with different capacities

As mentioned earlier, the capacitated degree inequalities (6.13) of section 6.5.2 are a special case of the capacitated reliability inequalities with $|S| = 1$ and equal capacities. By setting $|S| = 1$ to the capacitated reliability inequalities we can also obtain the corresponding capacitated degree inequalities for the case of general (not equal) capacities. These are the following:

$$\sum_{j:(i,j)\in E} \xi_{ij} \geq \left\lceil \frac{d_i}{U} \right\rceil + 1, \quad \text{for all } i \in V, \tag{6.27}$$

where U is the *second largest* capacity among all edges contained in the cut $\delta(i)$. In other words,

$$U = \max_{\substack{j:(i,j)\in E \\ j \neq j'}} u_{ij} \tag{6.28}$$

where j' is the vertex defined by:

$$j' := \arg \max_{j:(i,j)\in E} u_{ij} \tag{6.29}$$

i.e., it is the vertex such that edge (i, j') has the maximum capacity among all edges containing vertex i.

The capacitated degree constraints also work for the case where i is the source vertex, provided that we assign its demand to be the total demand of the network ($d_0 = \sum_{k \in V'} d_k$), just as in the capacitated degree inequalities with equal capacities.

6.5.5 Strengthening the capacitated degree inequalities using the b-matching problem

The capacitated degree constraints impose a restriction on the minimum number of edges that can meet at any vertex. This is, moreover, the design requirement for the 1-capacitated b-matching problem, and formed the basis for the derivation of the r-cover inequalities for the URNDP with flows in section 3.7.

In the CRNDP with flows we require that at every vertex v at least $\lceil d_v/U \rceil + 1$ are incident, where U is defined in the same way as in the previous section. This implies the following, stronger, b-values for the b-matching problem

$$b_v = |\delta(v)| - \left\lceil \frac{d_v}{U} \right\rceil - 1, \tag{6.30}$$

which translate to the statement that at every vertex v of any feasible CRNDP solution, *at most* this number of edges can be *absent* from the layout.

Following the same procedure as in the derivation of the r-cover inequalities in section 3.7.1.5, we set the b-matching variables to be the complements of the CRNDP binary variables ($\xi'_{ij} = 1 - \xi_{ij}$). Recall that the b-matching inequalities (which are also valid for the CRNDP for the chosen values) are

$$\sum_{i,j \in W} \xi'_{ij} + \sum_{(i,j) \in T'} \xi'_{ij} \leq \frac{1}{2} \left(\sum_{i \in W} b_i + |T'| - 1 \right) \tag{6.31}$$

Using the above substitution, we have that

$$\sum_{v \in H} b_v = \sum_{v \in H} \left(|\delta(v)| - \left\lceil \frac{d_v}{U} \right\rceil - 1 \right) = 2|E(H)| + |\delta(H)| - \sum_{v \in H} \left\lceil \frac{d_v}{U} \right\rceil - |H|, \tag{6.32}$$

$$\xi'(E(H)) = \sum_{e \in E(H)} \xi'_e = |E(H)| - \xi(E(H)) \tag{6.33}$$

and

$$\xi'(T') = \sum_{e \in T'} \xi'_e = |T'| - \xi(T'), \tag{6.34}$$

where $H \subseteq V$, $T' \subseteq \delta(H)$ and $E(H) := \{(i,j) \in E : i \in H, j \in H\}$.

Equation (6.31) becomes

$$|E(H)| - \xi(E(H)) + |T'| - \xi'(T') \leq$$
$$\leq \frac{1}{2} \left(2|E(H)| + |\delta(H)| - \sum_{v \in H} \left\lceil \frac{d_v}{U} \right\rceil - |H| + |T'| - 1 \right) \tag{6.35}$$

and after re-arranging,

$$\xi(E(H)) + \xi(T') \geq \frac{|H| - |\delta(H)| + |T'| + \sum_{v \in H} \left\lceil \frac{d_v}{U} \right\rceil + 1}{2}. \tag{6.36}$$

Setting $T = \delta(H) \backslash T'$ and so $|T'| = |\delta(H)| - |T|$ we obtain

$$\xi(E(H)) + \xi(\delta(H) \backslash T) \geq \frac{\sum_{v \in H} \left\lceil \frac{d_v}{U} \right\rceil + |H| - |T| - 1}{2}. \tag{6.37}$$

Therefore the following new inequalities are valid for the CRNDP with flows:

$$\sum_{i,j \in H} \xi_{ij} + \sum_{(i,j) \notin T} \xi_{ij} \geq \left\lceil \frac{\sum_{v \in H} \left\lceil \frac{d_v}{U} \right\rceil + |H| - |T| - 1}{2} \right\rceil \tag{6.38}$$

where $H \subseteq V$, $T \subseteq \delta(H)$. These inequalities will be referred to as the *strengthened r-cover inequalities.*

We note that if there are no capacity restrictions on the network then the strengthened r-cover inequalities (6.38) reduce to the URNDP r-cover inequalities of section 3.7.

6.6 Description of a branch and cut algorithm

6.6.1 A lower bound ascent procedure

In this section the lower bound ascent procedure used at the root node of the B&C algorithm to solve the CRNDP with flows is described.

The algorithm used for this purpose is summarised below.

The lower bound ascent procedure for the CRNDP with flows

1. Solve the initial LP relaxation (**M14**).
2. Add the capacitated degree inequalities (6.27).
3. Add violated maximum flow inequalities (section 3.6). At each step, the LP is re-optimised and the process is iterated. We stop the cut generation when no more violated maximum flow inequalities can be found.
4. Add violated strengthened r-cover inequalities (6.38). Again, we add as many inequalities as can be detected by the algorithm, re-optimising the LP as necessary.

6.6.2 Cut generation during branch and cut

The implementation aspects of the B&C algorithm for the CRNDP with flows are similar to those for the URNDP as discussed in chapter 4, with the following differences:

- We do not generate partition cuts. There are rarely effective for the problem, as they are often weaker than the required capacitated reliability constraints of the formulation.
- We do not generate r-cover cuts. Instead, we generate the strengthened r-cover cuts, using the same strategy. (i.e., search at every node at the first 50 B&C nodes, and then postpone the search for 30 nodes if none are found.)
- We do not generate Gomory or lift and project cuts. The size of the LP problems involved is very large that the generation of these cuts takes too long for the process to be worthwhile.

Moreover, whenever an integer feasible solution with a smaller cost than the incumbent solution is found, the separation algorithm for the capacitated reliability constraints is applied, to test if the solution satisfies the reliability requirements. If such constraints are detected, then they are added to the model and kept until the end of the optimisation.

6.7 A solved example instance

We demonstrate an example of the output of the algorithm using an example instance. This instance has 12 vertices and 64 edges and is the same as the one used in section 4.7, except that capacities are imposed on the edges. The values of these capacities for each edge can be found in appendix A.

The demands for each vertex are summarised below:

Vertex	1	2	3	4	5	6	7	8	9	10	11
Demand	7	18	5	19	8	8	9	19	18	17	16

The optimal solution to this problem is shown in figure 6.4. In this figure, the edges that do not carry any flow are shown with a dashed line, and values of the flows are shown on every edge. A number appearing in boldface style denotes that this edge is

saturated, i.e., the flow is equal to the maximum capacity of that edge. The total cost of the optimal solution is 2261.

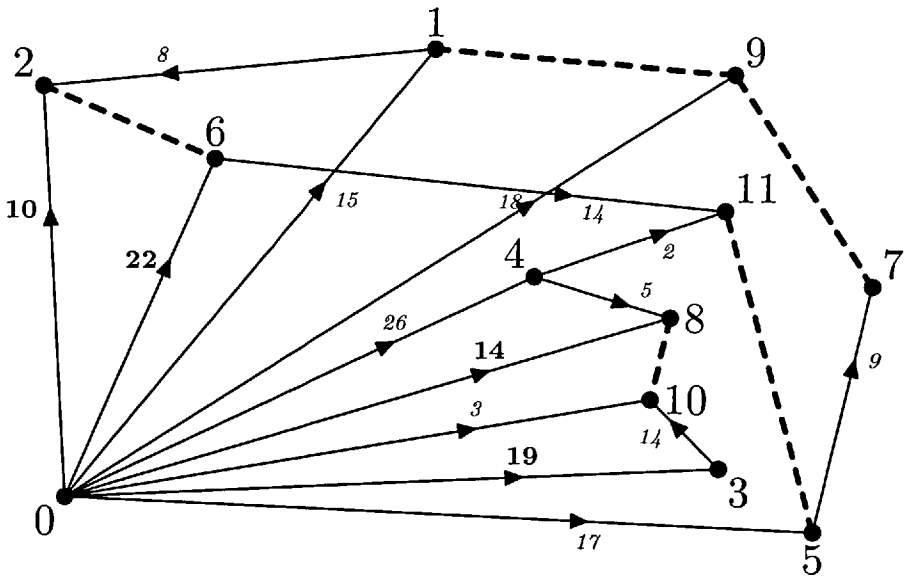


Figure 6.4: The layout of the optimal solution of the capacitated RNDP test instance

Recall that in the uncapacitated RNDP, the layout of the flows in the optimal solution is always a tree. This is no longer true for the case of the CRNDP.

6.8 Computational results

In this section we present computational results obtained using the Branch and Cut algorithm for the capacitated RNDP. The algorithm was implemented using C++ on a *Pentium III* PC with a clock speed of 600MHz. The test instances used were randomly generated, as described in section 2.7, and varied in size from 12 to 30 vertices. For each vertex size, several classes of instances were produced using different values of the parameter γ (the relative ratio of variable over fixed cost). For each class, three instances were generated. A time limit of 10 hours of CPU time was also imposed on the execution of the algorithm.

The notation used in this section is the following:

LB_0	Lower bound obtained at the root node using the initial LP relaxation (**M14**).
LB_1	Lower bound obtained at the root node after adding the capacitated degree inequalities (6.27) to the initial LP relaxation.
LB_2	Lower bound obtained at the root node after adding the capacitated degree and maximum flow inequalities (section 3.6) to the initial LP relaxation.
LB_3	Lower bound obtained at the root node after adding the capacitated degree, maximum flow and strengthened r-cover inequalities (6.38) to the initial LP relaxation.
z_{opt}	The cost of the optimal solution to the CRNDP or the cost of the best feasible solution found at the end of the branch and cut algorithm.
$\%LB_0$	Percentage ratio LB_0/z_{opt}.
$\%LB_1$	Percentage ratio LB_1/z_{opt}.
$\%LB_2$	Percentage ratio LB_2/z_{opt}.
$\%LB_3$	Percentage ratio LB_3/z_{opt}.
t_{mf}	The computational time (in seconds) required for the generation of maximum flow inequalities.
t_{src}	The computational time (in seconds) required for the generation of strengthened r-cover inequalities.
t_{root}	The computational time (in seconds) spent at the root node of the B&C algorithm.
LB_{root}	Lower bound obtained at the root node of the B&C algorithm.
$\%LB_{root}$	Percentage ratio $LB_{\text{root}}/z_{\text{opt}}$.
$LB_{\text{B\&C}}$	Lower bound obtained at the end of the B&C algorithm. This will be different from z_{opt} if the algorithm did not terminate because of the time limit imposed.
$\%LB_{B\&C}$	Percentage ratio $LB_{\text{B\&C}}/z_{\text{opt}}$.
t_{total}	The total computational time (in seconds).

Tables 6.2 and 6.3 present the lower bound ascent algorithm at the root node of the B&C algorithm. Table 6.2 presents the average results over the three instances of each group, whereas table 6.3 presents results for individual instances.

Tables 6.4 and 6.5 similarly contain the computational results obtained during the execution of the B&C algorithm.

					Maximum flow cuts			*Strengthened r-cover cuts*			
Vertices	*Edges*	γ	$\%LB_0$	$\%LB_1$	$\%LB_2$	*# cuts*	t_{mf}	$\%LB_3$	*# cuts*	t_{src}	t_{root}
12	48.0	0.01	34.85	86.68	86.76	87.7	0.03	86.88	0.3	0.07	0.20
12	49.3	0.05	37.95	85.80	86.21	89.3	0.03	86.78	0.7	0.14	0.26
12	48.3	0.1	45.65	82.71	83.15	89.0	0.03	83.35	1.0	0.14	0.25
12	51.7	1	72.87	92.60	95.14	94.3	0.03	95.26	0.7	0.14	0.25
12	52.0	2	81.48	91.18	93.91	97.0	0.03	93.91	0.3	0.14	0.24
	Average		*54.56*	*87.79*	*89.03*	*91.4*	*0.03*	*89.24*	*0.6*	*0.12*	*0.24*
15	78.7	0.01	45.57	80.16	80.23	147.3	0.07	80.28	1.0	0.17	0.46
15	78.3	0.05	44.61	77.29	77.59	146.0	0.07	77.59	0.0	0.16	0.44
15	75.3	0.1	43.32	82.84	83.24	141.0	0.06	83.24	0.0	0.16	0.38
15	83.3	1	80.49	90.50	92.72	155.7	0.05	92.80	0.7	0.17	0.36
15	81.0	2	85.13	92.44	93.59	151.0	0.05	93.59	0.0	0.14	0.32
	Average		*59.83*	*84.64*	*85.47*	*148.2*	*0.06*	*85.50*	*0.3*	*0.16*	*0.39*
20	148.7	0.01	42.68	81.53	81.66	281.7	0.22	81.79	2.0	0.50	1.39
20	142.7	0.05	42.93	84.48	84.69	272.0	0.16	84.70	0.7	0.30	0.93
20	142.3	0.1	53.18	84.23	84.62	271.7	0.13	84.64	1.0	0.30	0.79
20	135.7	1	78.25	91.81	93.40	256.3	0.10	93.43	1.0	0.24	0.62
20	144.7	2	84.77	94.33	96.08	273.7	0.11	96.08	0.7	0.24	0.65
	Average		*60.36*	*87.28*	*88.09*	*271.1*	*0.14*	*88.13*	*1.1*	*0.31*	*0.87*
25	218.0	0.01	50.35	77.55	77.59	418.7	0.40	77.60	0.3	0.59	2.46
25	223.7	0.05	50.43	80.07	80.22	428.0	0.41	80.30	2.7	0.87	2.46
25	226.0	0.1	50.89	82.60	83.36	433.3	0.34	83.39	1.0	0.63	2.12
25	229.3	1	81.19	91.98	94.02	440.7	0.31	94.02	0.3	0.43	1.50
25	223.7	2	88.03	94.76	96.74	430.7	0.28	96.75	0.3	0.40	1.37
	Average		*64.18*	*85.39*	*86.39*	*430.3*	*0.35*	*86.41*	*0.9*	*0.59*	*1.98*
30	326.0	0.01	45.20	76.39	76.45	630.7	1.18	76.58	2.3	1.87	6.17
30	326.0	0.05	51.69	81.10	81.32	629.7	0.75	81.42	1.3	1.09	3.99
30	322.7	0.1	55.96	80.40	80.83	625.0	0.78	80.84	1.0	1.06	3.71
30	333.0	1	81.10	93.08	95.17	642.0	0.53	95.18	1.0	0.78	2.65
30	331.3	2	85.59	94.14	96.59	641.0	0.59	96.59	0.0	0.69	2.46
	Average		*63.91*	*85.02*	*86.07*	*633.7*	*0.77*	*86.12*	*1.1*	*1.10*	*3.79*

Table 6.2: Computational results for the CRNDP: Root node (averages over three instances)

						cap. degree		*max flow*				*strengthened r-cover*				
Problem	*Vertices*	*Edges*	γ	LB_0	$\%LB_0$	LB_1	$\%LB_1$	LB_2	$\%LB_2$	*# cuts*	t_{mf}	$LB3$	$\%LB_3$	*# cuts*	t_{src}	t_{root}
g2n12g001i1	12	50	0.01	9060.0	35.85	21412.4	84.72	21424.0	84.77	93	0.03	21424.0	84.77	0	0.06	0.2
g2n12g005i2	12	49	0.05	1631.0	38.70	3375.7	80.09	3376.7	80.11	89	0.03	3376.7	80.11	0	0.12	0.3
g2n12g01i3	12	50	0.1	1511.4	51.32	2284.1	77.56	2297.9	78.03	91	0.03	2315.7	78.63	1	0.135	0.3
g2n12g1i2	12	54	1	517.3	74.44	617.3	88.83	641.7	92.32	97	0.03	643.2	92.55	1	0.15	0.3
g2n12g2i3	12	55	2	323.7	79.93	361.7	89.31	381.4	94.18	102	0.03	381.4	94.18	0	0.135	0.2
g2n15g001i1	15	83	0.01	14086.3	47.79	24137.3	81.88	24148.8	81.92	156	0.08	24189.4	82.06	2	0.15	0.5
g2n15g005i3	15	75	0.05	2036.8	42.39	3935.5	81.90	3950.6	82.22	138	0.06	3950.6	82.22	0	0.15	0.4
g2n15g01i1	15	77	0.1	807.6	29.89	2347.3	86.87	2361.3	87.39	143	0.06	2361.3	87.39	0	0.15	0.4
g2n15g1i2	15	80	1	560.3	82.28	612.8	89.99	629.4	92.43	149	0.05	630.5	92.59	1	0.165	0.3
g2n15g2i1	15	79	2	450.9	83.34	484.6	89.58	494.0	91.32	147	0.05	494.0	91.32	0	0.15	0.4
g2n20g001i2	20	146	0.01	15653.6	51.22	24109.2	78.88	24114.2	78.90	275	0.14	24157.7	79.04	1	0.33	1.0
g2n20g005i3	20	150	0.05	2193.2	44.32	4236.0	85.59	4249.2	85.86	285	0.17	4249.2	85.86	0	0.2865	0.9
g2n20g01i2	20	144	0.1	1938.4	58.88	2578.9	78.34	2585.3	78.53	273	0.12	2585.3	78.53	0	0.2265	0.7
g2n20g1i3	20	138	1	677.9	83.48	745.9	91.86	752.2	92.63	262	0.11	752.2	92.63	0	0.21	0.6
g2n20g2i1	20	140	2	473.6	83.53	534.6	94.29	546.4	96.36	266	0.09	546.4	96.36	1	0.18	0.5
g2n25g001i3	25	221	0.01	16836.8	47.93	27016.3	76.90	27033.3	76.95	424	0.37	27033.3	76.95	0	0.6518	2.4
g2n25g005i1	25	216	0.05	3456.7	56.72	5222.2	85.69	5231.1	85.84	413	0.38	5242.9	86.03	4	0.9389	2.5
g2n25g01i2	25	221	0.1	2022.4	51.33	3373.9	85.63	3414.4	86.66	423	0.37	3414.4	86.66	0	0.5621	2.1
g2n25g1i1	25	224	1	619.3	78.59	727.4	92.31	747.6	94.87	430	0.28	747.6	94.87	0	0.3708	1.5
g2n25g2i2	25	227	2	1080.0	91.14	1139.4	96.15	1152.9	97.29	438	0.28	1152.9	97.29	0	0.3708	1.4
g2n30g001i3	30	308	0.01	15686.7	45.95	27373.4	80.19	27388.9	80.24	592	0.84	27435.0	80.37	2	1.3993	5.0
g2n30g005i2	30	317	0.05	5519.1	53.75	8070.8	78.60	8103.1	78.92	613	0.84	8104.3	78.93	1	1.1183	4.5
g2n30g01i3	30	325	0.1	4208.2	63.07	5442.3	81.57	5485.5	82.22	629	0.84	5485.6	82.22	1	1.3096	4.3
g2n30g1i1	30	325	1	895.4	81.63	1019.5	92.94	1040.9	94.88	626	0.47	1040.9	94.88	1	0.7475	2.4
g2n30g2i3	30	320	2	820.4	87.93	880.5	94.37	903.4	96.83	620	0.47	903.4	96.83	0	0.6518	2.3

Table 6.3: Computational results for the CRNDP: Root node

			Root node		Branch and Cut tree								
					max flow		*str. r-cover*		*cap.rel.constraints*				
Vertices	*Edges*	γ	$\%LB_{root}$	t_{root}	*# cuts*	*time*	*# cuts*	*time*	*# cuts*	*time*	*B&C nodes*	$\%LB_{B\&C}$	t_{total}
12	48.0	0.01	86.88	0.20	8.0	0.03	23.7	1.11	47.7	0.06	431.0	100.00	12.2
12	49.3	0.05	86.78	0.26	2.7	0.02	32.7	2.09	51.7	0.05	802.3	100.00	9.5
12	48.3	0.1	83.35	0.25	7.7	0.04	58.3	2.68	61.3	0.11	941.0	100.00	11.9
12	51.7	1	95.26	0.25	0.7	0.01	17.3	1.54	33.0	0.07	567.3	100.00	5.8
12	52.0	2	93.91	0.24	0.3	0.02	5.3	0.49	25.7	0.04	133.3	100.00	2.1
Average			*89.24*	*0.24*	*3.9*	*0.02*	*27.5*	*1.58*	*43.9*	*0.06*	*575.0*	*100.00*	*8.3*
15	78.7	0.01	80.28	0.46	13.0	0.23	418.0	62.12	225.3	0.41	11897.0	100.00	419.4
15	78.3	0.05	77.59	0.44	10.3	0.27	271.3	45.50	151.3	0.23	9182.7	100.00	208.3
15	75.3	0.1	83.24	0.38	10.7	0.16	118.7	15.83	81.0	0.17	2903.7	100.00	67.2
15	83.3	1	92.80	0.36	1.7	0.12	15.0	16.17	52.7	0.10	2291.0	100.00	38.6
15	81.0	2	93.59	0.32	0.0	0.11	12.0	12.19	34.3	0.13	2576.3	100.00	27.4
Average			*85.50*	*0.39*	*7.1*	*0.18*	*167.0*	*30.36*	*108.9*	*0.21*	*5770.1*	*100.00*	*152.2*
20	148.7	0.01	81.79	1.39	122.0	9.40	2983.3	4816.47	444.7	1.30	180716.3	98.54	35117.1
20	142.7	0.05	84.70	0.93	29.0	2.90	712.0	1259.29	250.7	0.82	56490.7	100.00	4377.8
20	142.3	0.1	84.64	0.79	21.7	2.84	363.0	1300.63	190.7	0.86	55451.0	100.00	3516.1
20	135.7	1	93.43	0.62	0.0	1.56	160.7	582.35	78.3	0.45	28582.0	100.00	1069.2
20	144.7	2	96.08	0.65	0.0	0.55	40.7	172.50	77.7	0.45	6697.7	100.00	271.2
Average			*88.13*	*0.87*	*34.5*	*3.45*	*851.9*	*1626.25*	*208.4*	*0.77*	*65587.5*	*99.71*	*8870.3*
25	218.0	0.01	77.60	2.46	24.7	32.40	1874.0	6928.73	618.0	6.16	117986.3	93.97	36000.0
25	223.7	0.05	80.30	2.46	17.0	24.25	1729.7	5449.31	561.0	6.43	83387.7	95.12	25985.9
25	226.0	0.1	83.39	2.12	16.3	43.72	1319.7	9644.28	437.0	6.36	162838.0	96.48	36000.0
25	229.3	1	94.02	1.50	0.0	19.43	127.3	3862.82	154.0	2.87	100585.3	100.00	6860.6
25	223.7	2	96.75	1.37	0.0	18.45	100.0	3646.00	95.7	2.46	80211.7	100.00	6235.4
Average			*86.41*	*1.98*	*11.6*	*27.65*	*1030.1*	*5906.23*	*373.1*	*4.86*	*109001.8*	*97.11*	*22216.4*
30	326.0	0.01	76.58	6.17	37.0	32.00	1647.3	6792.73	1126.3	18.90	53193.7	87.32	36000.0
30	326.0	0.05	81.42	3.99	13.0	38.35	1905.3	8438.50	761.0	12.73	66124.3	92.38	36000.0
30	322.7	0.1	80.84	3.71	1.0	47.20	1238.7	10033.42	929.7	19.32	81661.7	91.73	36000.0
30	333.0	1	95.18	2.65	1.7	51.66	483.3	12078.75	165.3	6.29	110788.7	99.66	22297.8
30	331.3	2	96.12	2.46	0.0	73.97	1483.7	14365.14	158.3	6.10	140145.3	99.66	27122.2
Average			*86.12*	*3.79*	*10.5*	*48.64*	*1351.7*	*10341.71*	*628.1*	*12.67*	*90382.7*	*94.15*	*31484.0*

Table 6.4: Computational results for the CRNDP: Branch and Cut (averages over three instances)

				Root node			*Branch and Cut tree*									
							max flow		*strengthened r-cover*		*cap. reliability*					
Problem	*Vertices*	*Edges*	γ	LB_{root}	$\%LB_{root}$	t_{root}	*# cuts*	t_{mf}	*# cuts*	t_{src}	*# cuts*	t_{caprel}	z_{opt}	$\%LB_{B\&C}$	*B&C nodes*	t_{total}
g2n12g001i1	12	50	0.01	21424.0	84.77	0.2	5	0.03	28	1.80	66	0.11	25273	100	696	7.4
g2n12g005i2	12	49	0.05	3376.7	80.11	0.3	1	0.02	64	5.17	118	0.12	4215	100	2022	21.5
g2n12g01i3	12	50	0.1	2315.7	78.63	0.3	2	0.05	19	1.65	85	0.12	2945	100	654	7.7
g2n12g1i2	12	54	1	643.2	92.55	0.3	0	0.02	34	3.79	68	0.14	695	100	1523	14.0
g2n12g2i3	12	55	2	381.4	94.18	0.2	0	0.05	2	0.36	10	0.00	405	100	87	1.6
g2n15g001i1	15	83	0.01	24189.4	82.06	0.5	26	0.36	643	112.94	206	0.41	29478	100	22121	851.1
g2n15g005i3	15	75	0.05	3950.6	82.22	0.4	4	0.12	219	18.66	137	0.23	4805	100	4225	83.0
g2n15g01i1	15	77	0.1	2361.3	87.39	0.4	8	0.11	73	4.21	41	0.09	2702	100	790	18.9
g2n15g1i2	15	80	1	630.5	92.59	0.3	3	0.06	11	10.52	73	0.11	681	100	2051	31.7
g2n15g2i1	15	79	2	494.0	91.32	0.4	0	0.20	10	26.32	11	0.14	541	100	6124	57.9
g2n20g001i2	20	146	0.01	24157.7	79.04	1.0	98	14.87	1454	7124.32	304	1.01	30564	100	307674	33347.7
g2n20g005i3	20	150	0.05	4249.2	85.86	0.9	3	1.88	200	998.98	150	0.47	4949	100	40162	1941.5
g2n20g01i2	20	144	0.1	2585.3	78.53	0.7	60	6.77	512	3197.56	212	1.04	3292	100	137853	8834.3
g2n20g1i3	20	138	1	752.2	92.63	0.6	0	0.60	77	205.87	95	0.35	812	100	8503	342.5
g2n20g2i1	20	140	2	546.4	96.36	0.5	0	0.36	26	75.36	53	0.33	567	100	3047	109.8
g2n25g001i3	25	221	0.01	27033.3	76.95	2.4	39	17.52	3233	3945.00	635	6.24	35130†	93.60	58820	36000.0
g2n25g005i1	25	216	0.05	5242.9	86.03	2.5	8	5.40	1117	1234.39	461	5.52	6094	100	28194	6009.2
g2n25g01i2	25	221	0.1	3414.4	86.66	2.1	27	38.02	781	9413.34	567	7.89	3940†	98.96	177918	36000.0
g2n25g1i1	25	224	1	747.6	94.87	1.5	0	4.38	225	890.40	173	3.74	788	100	20005	1515.6
g2n25g2i2	25	227	2	1152.9	97.29	1.4	0	4.20	24	740.22	127	2.59	1185	100	13919	1090.0
g2n30g001i3	30	308	0.01	27435.0	80.37	5.0	47	34.30	1012	7090.95	1093	18.85	34135†	91.10	60853	36000.0
g2n30g005i2	30	317	0.05	8104.3	78.93	4.5	7	32.55	2037	6723.15	1119	16.71	10268†	89.02	54178	36000.0
g2n30g01i3	30	325	0.1	5485.6	82.22	4.3	1	37.15	735	7925.61	1525	27.90	6672†	91.73	63772	36000.0
g2n30g1i1	30	325	1	1040.9	94.88	2.4	5	85.96	913	18477.36	139	4.10	1097†	98.98	166120	36000.0
g2n30g2i3	30	320	2	903.4	96.83	2.3	0	131.65	1270	22877.06	110	4.01	933†	99.52	230149	36000.0

† this symbol denotes the best solution value found by the algorithm within the time limit.

Table 6.5: Computational results for the CRNDP: Branch and Cut

6.8.1 Discussion

Using the computational results obtained from the execution of the Branch and Cut algorithm for the CRNDP with flows, we note the following:

First of all, it is clear that the CRNDP with flows is much harder than the uncapacitated equivalent. Most capacitated test problems with 30 vertices were not solved to optimality in the 10 hour time limit, whereas in the uncapacitated case this was the case for problems with 50 vertices or more.

The percentage error of the initial LP relaxation bound is large (about 50%), but is rapidly reduced when the capacitated degree inequalities are added (LB_1). With the addition of the maximum flow inequalities, the error is reduced by a further 1–3%, and the addition of the strengthened r-cover inequalities at the end of the lower bound ascent procedure has a further improvement of less that 1%.

In the B&C stage of the algorithm, we observe that a larger number of strengthened r-cover cuts is added than maximum flow cuts. This is in contrast to the URNDP, where the situation was reversed. The effect of the maximum flow inequalities seems to be less important, and this can be justified by the fact that some maximum flow inequalities may be weaker than the trivial constraints of the capacitated LP formulation. The r-cover inequalities, on the other hand, benefit from the strengthening procedure of section 6.5.5, and have a greater impact.

We also report the number of violated capacitated reliability constraints that were detected during the solution process. Their generation occurs when an integer feasible solution is found during the B&C, but the solution layout does not satisfy the reliability requirements (fails the feasibility test). We therefore note that a large number of integer but infeasible solutions is produced during the solution process. The number of the corresponding inequalities for the URNDP—the partition inequalities—was much smaller. This is due to the large number of reliability constraints that have been removed from the capacitated formulation, raising the probability that an integer solution will be infeasible.

Figures 6.5 and 6.6 show graphs of the total CPU time taken versus the number of vertices or the value of γ (in a logarithmic time scale). We observe the near-linear correlation of network size and log(time) (before the time limit of 10 hours is reached) which translates into an exponential relationship between network size and computational time. Figure 6.6 suggests that the most complex capacitated instances correspond to the cases $\gamma = 0.01$ and $\gamma = 0.05$. This is in contrast to the URNDP, where problems with $\gamma = 0.01$ were easier than those with $\gamma = 0.1$.

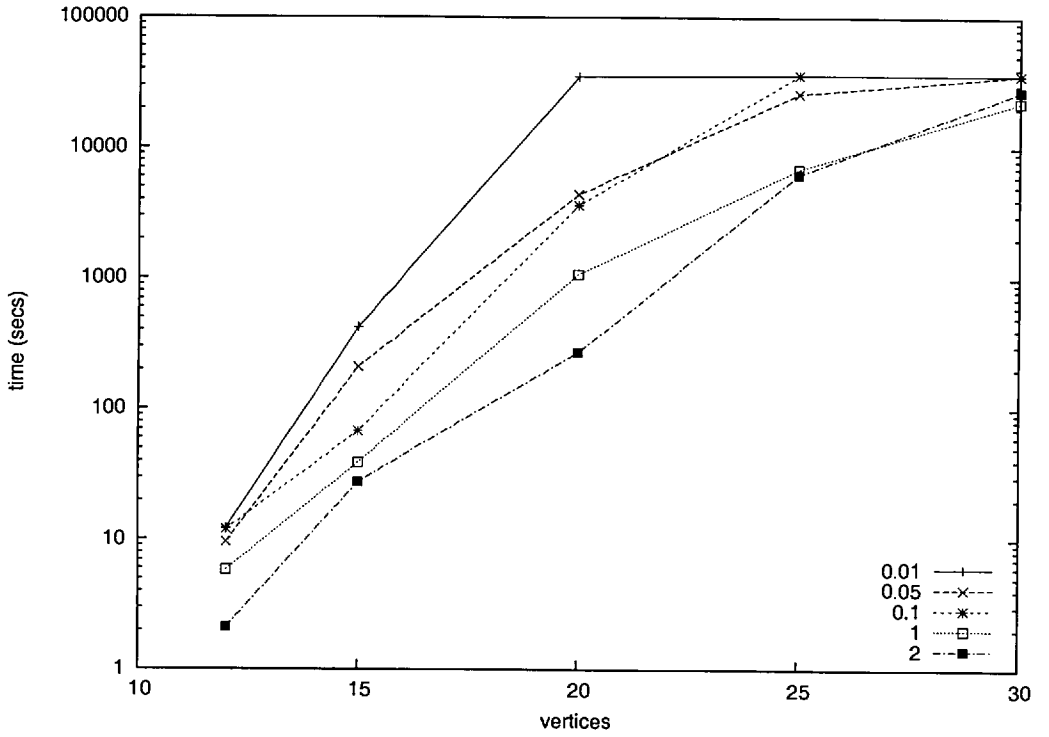


Figure 6.5: Graph of time versus network size for different values of γ.

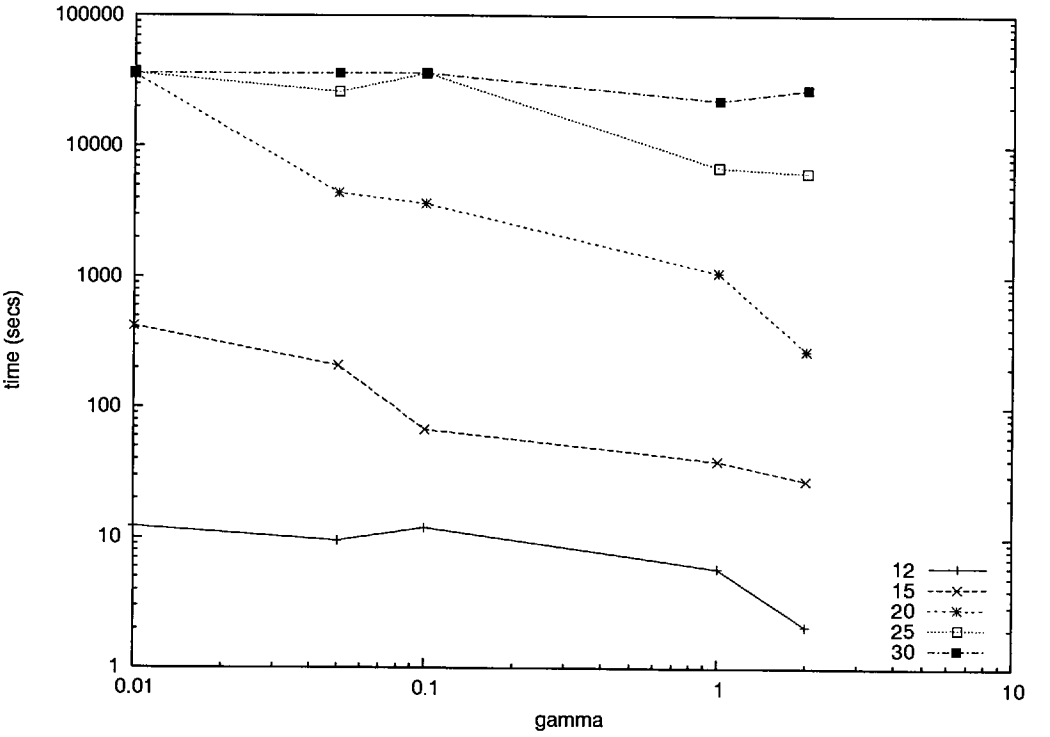


Figure 6.6: Graph of time versus γ for different network sizes.

6.9 Summary

In this chapter a new MIP formulation for the capacitated reliable network design problem with flows was presented. The reliability requirement in the capacitated case is stronger than two-edge connectivity and more constraints are necessary to be included in the problem.

A Branch and Cut algorithm for the solution of the CRNDP with flows was developed. The MIP was relaxed by removing most of the capacitated reliability constraints and an exact separation algorithm for their detection was presented. Valid inequalities for the CRNDP were also developed and included in a Branch and Cut algorithm. The chapter concluded with computational results on randomly generated test problems.

The outcome of any serious research can only be
to make two questions grow where only one grew before.

Thorstein Veblen (1857-1929)
The Place of Science in Modern Civilization
and Other Essays

Chapter 7

Conclusion

7.1 Contributions

In this thesis an operational research network design problem on single commodity flow networks was studied. The research problem included the incorporation of reliability requirements into the design of the layout and the possibility of edge capacities on the values of the network flows.

Starting with an overview of the current literature on combinatorial optimisation, network design problems and network reliability in chapter 2, the research problem was formulated. It consists of a single commodity network design problem, with the reliability constraint that the network should be able to continue to supply the customers if any single edge is removed from the layout.

The uncapacitated version of this Reliable Network Design Problem (RNDP) was modelled using a new mixed integer program. The reliability requirements for this case are equivalent to the solution layout being two-edge connected. An exact Branch and Cut algorithm was developed to solve the problem to optimality using various classes of valid inequalities to strengthen the linear programming relaxation of the model (chapters 3 and 4). These valid inequalities were either inequalities relating to the design of reliable networks without flows that have never before been applied for this problem or new valid inequalities specific to the RNDP with flows, such as the maximum flow inequalities. An

investigation on the performance of the Branch and Cut algorithm on various randomly generated test problems was also carried out and the computational results obtained were reported.

A new heuristic based on ant colony optimisation was further developed for the uncapacitated problem. A new ant colony setup was used, where a group of ant agents represents a feasible URNDP solution. The details of the algorithmic implementation were presented and the computational results obtained were presented in chapter 5. In the last chapter, the capacitated version of the RNDP with flows was examined. For this case, two-edge connectivity is not enough to guarantee the level of reliability required and more constraints (the capacitated reliability constraints) are required. A new MIP formulation was presented for the capacitated RNDP, and an algorithm for the separation of the capacitated reliability constraints was developed. A Branch and Cut algorithm was further produced to solve the problem to optimality, using both valid inequalities derived from the uncapacitated problem of chapter 3 and new inequalities (the capacitated degree and the strengthened r-cover inequalities) that are specific to the capacitated problem. The algorithm was tested using randomly generated test instances.

7.2 Concluding remarks

The Reliable Network Design Problem with flows is the backbone problem for any network design problem that includes a commodity flowing in the network and requires the network to be able to cope with single edge failures. Although it is a very complex problem to solve optimally (belonging to the category of $\mathcal{NP}$-hard problems) its applications cannot be underestimated. Networks of bigger and bigger sizes are used to model many real-life problems and the inclusion of reliability is desired to ensure the level of disruption is kept at a minimum should something go wrong. Indeed, increasing the topological reliability of the layout of an existing network by simply adding an edge at a crucial location may yield significant benefits to the network owner, translating to less disruption to the customers in case of breakdowns, smaller compensation costs and better flexibility when planning for scheduled repairs. Of course, the RNDP can further be extended to take into consideration other design requirements that were not studied in this thesis, possibly specific to the particular type of network considered.

Moreover, the field of computer software for solving mixed integer programs is rapidly expanding (see Bixby, Fenelon, Gu, Rothberg & Wunderling (2000) for an overview of the recent developments in the area). To date, there are five main computer software

designed for implementing Branch and Bound or Branch and Cut algorithms and they widely used both by academics and by private researchers. (CPLEX[39], MINTO[40], ABACUS[41], COIN-OR[42], X-PRESS[43]) This trend, together with the increase in computing power, is bound to make the solution of larger MIP problems easier.

This will undoubtedly make the Reliable Network Design Problem with flows described in the thesis more tractable and therefore more appealing. It will be possible to solve larger instances in smaller amounts of time and will make the task of extending the basic model by including application-specific requirements easier. Nevertheless, this should not make any theoretical research, such as the development of new valid inequalities for exact Branch and Cut methods or other algorithms for the problem, seem redundant. A theoretical understanding of the complexity of the problem is of equal interest, as this problem is a natural extension of the well-studied fixed charge problem and the two-edge connected network design problem.

7.3 Future work

A possible future line of research regarding exact solution algorithms for the Reliable Network Design Problem with flows is to try to derive further valid inequalities or polyhedral results. In general, this is not an easy task but can make a Branch and Cut algorithm significantly faster. For this purpose, a complete description of the polytope bounded by all the feasible solutions for the URNDP with flows for an example input graph is presented in appendix B. This list contains all possible facet defining valid inequalities and was obtained during the course of this research, but the problem in identifying new inequalities lies in the fact that this list can only be produced for the smallest of the networks.

With respect to the algorithmic implementation, it could be of interest to attempt to devise a different algorithm for the RNDP with flows, based on the variations of the Branch and Cut procedure that have recently appeared. These include *branch-and-price* and *branch-and-price-and-cut* (e.g., see Barnhart, Johnson, Nemhauser, Savelsbergh & Vance (1998), Barnhart, Hane & Vance (2000)) which speed up the algorithm by relaxing the problem via the removal of variables and by including a column generation procedure

[39] See ILOG™ S.A. (2001).

[40] "A Mixed Integer Optimizer", see Nemhauser, Savelsbergh & Sigismondi (1994).

[41] "A branch and cut system", see Thienel (1995), Jünger & Thienel (2000).

[42] "Computational Infrastructure for Operations Research", an initiative for open-source software for the operational research community, developed by the *IBM Corporation*.

[43] Commercial software by *Dash Optimization*.

at the nodes of the Branch and Bound tree.

A further possibility, on a more theoretical level, is to investigate the relationship of the input data and the solution layout. For example, it is known that, for the two-edge connected network design problem without flows, if the edge costs satisfy the triangle inequality, then there always exists an optimal solution with every vertex of degree two or three. Similar results could possibly exist for the network design problem with flows.

The relationship between the density of the input network and the complexity of the problem could be investigated. As discussed in section 2.7, the density of the input graph affects the solution. In particular, for sparse initial layouts we expect to have more infeasible solutions than for more dense ones. Moreover, the complexity of the Branch and Cut algorithm is also affected, only few feasible configurations exist for sparse input networks. It could be of interest to study on a more theoretical level the relationship of these two parameters.

The capacitated RNDP is a very complex combinatorial optimisation problem, and possible extensions undoubtedly include the possible derivation of new valid inequalities for a more efficient exact algorithm or the development of faster heuristic methods for its solution.

A simple way that the formulation of the CRNDP could be easily extended, is include *lower capacities* in the model, i.e., lower bounds on the values of the flows for all the edges that are selected to be in the layout. Such a model could be useful for some types of commodity network (e.g., water) where zero flows should be avoided, as standing water can result in contamination of the pipes. A model to include lower capacities l_{ij} per edge can be easily formulated by modifying the capacitated model **M7** of chapter 3, to obtain the MIP shown in **M15**.

MODEL M15

$$\textbf{min} \quad \sum_{\substack{(i,j)\in E \\ i<j}} (b_{ij}\xi_{ij} + c_{ij}x_{ij} + c_{ji}x_{ji}) \tag{M15.1}$$

subject to

$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = d_i \qquad \text{for all } i \in V' \tag{M15.2}$$

$$\sum_{j:(0,j)\in E} x_{0j} = \sum_{i\in V'} d_i \tag{M15.3}$$

$$\sum_{\substack{i\in S, j\in \bar{S} \\ (i,j)\in E \\ i<j}} \xi_{ij} + \sum_{\substack{i\in S, j\in \bar{S} \\ (i,j)\in E \\ i>j}} \xi_{ji} \geq 2 \qquad \text{for all } 0 \in S \subseteq V,\ \bar{S} = V\backslash S \tag{M15.4}$$

$$x_{ij} \leq Q\xi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M15.5}$$

$$x_{ji} \leq Q\xi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M15.6}$$

$$x_{ij} \geq l_{ij}\psi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M15.7}$$

$$x_{ji} \geq l_{ij}(\xi_{ij} - \psi_{ij}) \qquad \text{for all } (i,j) \in E,\ i < j \tag{M15.8}$$

$$x_{ij} \leq Q\psi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M15.9}$$

$$x_{ji} \leq Q(1 - \psi_{ij}) \qquad \text{for all } (i,j) \in E,\ i < j \tag{M15.10}$$

$$\psi_{ij} \leq \xi_{ij} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M15.11}$$

$$x_{ij} \geq 0 \qquad \text{for all } (i,j) \in E \tag{M15.12}$$

$$\psi_{ij}, \xi_{ij} \in \{0,1\} \qquad \text{for all } (i,j) \in E,\ i < j \tag{M15.13}$$

THE RELIABLE NETWORK DESIGN PROBLEM WITH LOWER CAPACITIES

In order to formulate the RNDP with flows and lower capacities it is necessary to introduce an extra binary variable, ψ_{ij}, defined for each edge via $\psi_{ij} = 0$ if $x_{ij} = 0$ and $\psi_{ij} = 1$ if $x_{ij} > 0$. This extra variable makes the distinction of the direction the flow is going through an edge possible, enabling us formulate the lower capacity constraint "*either* $x_{ij} \geq l_{ij}$ *or* $x_{ji} \geq l_{ij}$" by means of the inequalities **M15**.7–**M15**.11.

The ant colony heuristic developed for the URNDP could also be extended. An obvious way to do so, first of all, could be to using parallel processing. This refers to the procedure where a computer algorithm is divided into many separate parts which are solved simultaneously by different processors thus making the solution of very difficult problems possible. The most widely used programming style is the master/worker model,

where all communications between the processors pass through the master processor who collects all the information generated by the different 'workers'. ACO algorithms are ideal candidates for parallelisation, because the communication and exchange of information between the agents is performed via the trail only and, therefore, a set of ant agents can be set to run on a separate computer processor provided that the trail information is correctly maintained. Recently a few attempts to use parallel processing in ACO algorithms have appeared. (Bullnheimer, Kotsis & Strauß (1997) and Talbi, Roux, Fonlupt & Robillard (2001)).

Moreover, the ant colony setup developed for the URNDP with flows, where two types of ant agent were used (flow ants and reliability ants) and the two types of pheromone trail (edge and flow trail) could be extended to other optimisation problems as well, where the single-agent standard approach may not be applicable.

As far as extending the RNDP with flows as a whole, one possibility includes more general reliability requirements. One may be interested to design a network that is *three* edge connected or, in general, k-edge connected. These extensions can be easily modelled by modifying the reliability constraints of formulation **M7**.

A final challenge to the methodology described in this thesis will inevitably be its implementation into real-life network models, where its applicability for the solution of problems containing real data will be evaluated, scrutinised and, hopefully, appreciated.

"It is a capital mistake to theorize before one has data.
Insensibly one begins to twist facts to suit theories,
instead of theories to suit facts."

Sir Arthur Conan Doyle
The Adventures of Sherlock Holmes:
A Scandal in Bohemia (1891)

Appendix A

Sample test instances

In this appendix, we describe the uncapacitated and capacitated test instances used as examples in chapters 3, 4, and 6. Both instances have the same layout, fixed and variable costs and customer demands although the data for the capacitated instance includes edge capacities.

The instances have 12 vertices and 64 edges and the layout, the demands per vertex and the costs per edge are shown in table A.1. Vertex 0 is the source node and vertices 1 to 11 are the demand vertices. The edge capacities for the capacitated instance are shown in table A.2.

For the uncapacitated test instance, the optimal solution has a cost of 1646 and the solution layout is presented in section 4.7. The layout consists of eleven flow edges (a tree) and four reliability edges that carry no flow.

For the capacitated test instance, the optimal solution value is 2261 and this layout is presented in section 6.7. The layout consists of five reliability edges and fifteen flow edges (four of which are saturated, i.e., the flow passing through is equal to the capacity).

Vertex	*Demand*	*Edge*	*Lin. cost*	*Fixed cost*	*Edge*	*Lin. cost*	*Fixed cost*
1	7	0–1	7	65	3–6	6	103
2	18	0–2	4	63	3–7	4	111
3	5	0–3	4	53	3–8	1	136
4	19	0–4	4	77	3–9	9	63
5	8	0–5	8	21	3–10	1	167
6	8	0–6	3	29	3–11	9	160
7	9	0–7	10	172	4–5	5	158
8	19	0–8	2	163	4–6	10	133
9	18	0–9	5	171	4–8	7	79
10	17	0–10	9	158	4–9	8	124
11	16	0–11	6	178	4–10	4	122
		1–2	4	32	4–11	9	27
		1–3	10	59	5–6	6	63
		1–4	8	200	5–7	1	54
		1–5	9	150	5–8	2	178
		1–6	1	70	5–9	2	165
		1–7	4	171	5–10	3	127
		1–8	10	115	5–11	7	69
		1–9	4	21	6–7	3	78
		1–10	2	103	6–8	9	46
		1–11	10	71	6–9	1	151
		2–3	9	62	6–10	1	151
		2–4	10	75	6–11	3	80
		2–5	6	155	7–9	10	31
		2–6	8	23	7–10	10	140
		2–7	8	176	7–11	1	192
		2–8	7	49	8–9	6	160
		2–9	2	182	8–10	6	56
		2–10	9	87	8–11	7	186
		2–11	1	146	9–10	4	138
		3–4	2	86	9–11	4	75
		3–5	4	73	10–11	9	117

Table A.1: Characteristics of the test instance

Edge	*Capacity*	*Edge*	*Capacity*	*Edge*	*Capacity*	*Edge*	*Capacity*
0–1	28	1–7	14	3–6	15	5–10	15
0–2	10	1–8	19	3–7	14	5–11	18
0–3	19	1–9	18	3–8	19	6–7	14
0–4	32	1–10	18	3–9	13	6–8	13
0–5	25	1–11	12	3–10	18	6–9	19
0–6	22	2–3	18	3–11	16	6–10	16
0–7	12	2–4	15	4–5	18	6–11	16
0–8	14	2–5	17	4–6	11	7–9	11
0–9	19	2–6	13	4–8	18	7–10	19
0–10	15	2–7	16	4–9	19	7–11	12
0–11	13	2–8	19	4–10	21	8–9	18
1–2	17	2–9	11	4–11	12	8–10	16
1–3	19	2–10	14	5–6	14	8–11	10
1–4	18	2–11	17	5–7	17	9–10	11
1–5	21	3–4	19	5–8	15	9–11	14
1–6	18	3–5	17	5–9	10	10–11	18

Table A.2: Edge capacities for the CRNDP test instance

Appendix B

A description of the URNDP polytope for K_4

We present in this appendix a complete description of the polytope corresponding to the convex hull of feasible solutions of URNDP with flows. The input network was the complete graph with four vertices (K_4) with unit demands. Figure B.1 shows the interpretation of the variables used, and table B.1 shows the constraints corresponding to the URNDP with flows.

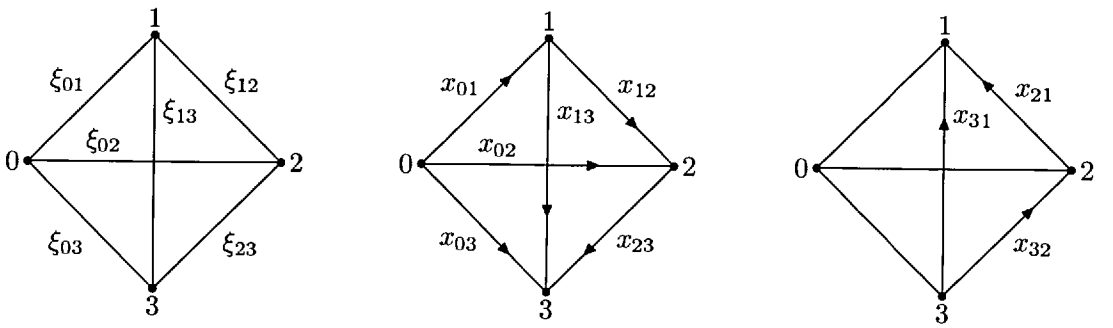


Figure B.1: The variables used in the description of the facets of the URNDP convex hull for K_4

For many combinatorial optimisation problems obtaining the convex hull of all integer solutions and studying similarities in the facets is the first step to trying to obtain new classes of facet-defining valid inequalities that could be invaluable in an exact algorithm. Alternatively, the description of the convex hull can be of assistance in the derivation of

flow balance constraints	*reliability constraints*
$x_{01} + x_{02} + x_{03} = 3$	$\xi_{01} + \xi_{02} + \xi_{03} \geq 2$
$x_{01} + x_{21} + x_{31} - x_{12} - x_{13} = 1$	$\xi_{01} + \xi_{12} + \xi_{13} \geq 2$
$x_{02} + x_{12} + x_{32} - x_{21} - x_{23} = 1$	$\xi_{02} + \xi_{12} + \xi_{23} \geq 2$
$x_{03} + x_{13} + x_{23} - x_{31} - x_{32} = 1$	$\xi_{03} + \xi_{13} + \xi_{23} \geq 2$
trivial constraints	$\xi_{02} + \xi_{03} + \xi_{12} + \xi_{13} \geq 2$
$x_{01} - 3\xi_{01} \leq 0 \quad x_{23} - 3\xi_{23} \leq 0$	$\xi_{01} + \xi_{03} + \xi_{12} + \xi_{23} \geq 2$
$x_{02} - 3\xi_{02} \leq 0 \quad x_{21} - 3\xi_{12} \leq 0$	$\xi_{01} + \xi_{02} + \xi_{13} + \xi_{23} \geq 2$
$x_{03} - 3\xi_{03} \leq 0 \quad x_{31} - 3\xi_{13} \leq 0$	$0 \leq \xi_{ij} \leq 1$
$x_{12} - 3\xi_{12} \leq 0 \quad x_{32} - 3\xi_{23} \leq 0$	$0 \leq x_{ij} \leq 3$
$x_{13} - 3\xi_{13} \leq 0$	ξ_{ij}, x_{ij} integer

Table B.1: The constraints of the URNDP with flows formulation for K_4

more theoretical results, such as the conditions where a given class of valid inequalities defines a facet, or the effect of the input problem to the number of facets that it induces.

For this reason, we present, at the end of this appendix, the complete description of the convex hull of feasible URNDP solutions for the complete graph with four vertices. The difficulty with the URNDP with flows is that there exist a large number of variables which increases quickly as the size of the input network grows. Therefore, the complete set of facets can only be calculated for very small test problems with small values for the vertex demands. This makes the task of spotting similarities among the facets a harder task as many facets may be special cases of more general categories of cuts that only appear for larger networks.

It should also be noted that, although facets represent the *strongest* possible valid inequalities for a problem, not all of them may be beneficial to solving the problem, and this largely depends on the objective function. The reason is that, the facets that lie on 'wrong side' of the polyhedron (with respect to the current LP relaxation solution that we try to 'cut off' and the orientation of the objective function) will be ineffective to the optimisation process.

The convex hull of feasible URNDP for this example consists of 747 facets, which were produced using PORTA (Christof & Löbel (2002)). Given that the constraints of the problem contain equalities, some variables (namely x_{01}, x_{02} and x_{03}) can be eliminated from the formulation using the following equations:

(1)		-x02	+x03	- x12	+ x13	+2x23	+ x21	- x31	-2x32	= 0
(2)	-2x01	+x02	+x03	+3x12	+3x13		-3x21	-3x31		= 0
(3)			+x03		+ x13	+ x23		- x31	- x32	= 1

These variables do not appear in the description of the facets. Note also that the description contains the maximum flow inequalities described in section 3.6, since, as proven in that section, they are facet defining for the complete input graph. (For instance, facet #8 is a maximum flow inequality corresponding to the set $|S| = \{1\}$ and the input arc $(3, 1)$.)

(1)	-2ξ01	-2ξ02	-3ξ03	- 2ξ12	- 3ξ13	- ξ23	- x12	- x13	- x23	+ x21	+ x31	- x32	<=	-9
(2)		-2ξ02	-2ξ03	- 2ξ12	- 2ξ13	- 2ξ23	- x12	- x13	- x23	+ x21		- x32	<=	-7
(3)					+ ξ13								<=	1
(4)							- x12	- x13		+ x21	+ x31		<=	1
(5)	- ξ01	- ξ02	- ξ03										<=	-2
(6)								- x13					<=	0
(7)										- x21			<=	0
(8)					- ξ13		- x12	- x13			+ x31		<=	0
(9)				- 2ξ12	- 3ξ13	- ξ23		- x13			+ x31		<=	-3
(10)				- ξ12	- ξ13	- ξ23							<=	-2
(11)							- x12						<=	0
(12)				- 3ξ12			+ x12						<=	0
(13)				- 3ξ12	- ξ13			- x13		+ x21	+ x31		<=	0
(14)						+ ξ23							<=	1
(15)				- ξ12			- x12	- x13		+ x21			<=	0
(16)				- 3ξ12						+ x21			<=	0
(17)			-2ξ03	- ξ12		- 2ξ23		- x13	- x23			+ x32	<=	-3
(18)	- ξ01			- 3ξ12	- ξ13	- ξ23	+ x12			- x21			<=	-3
(19)			-3ξ03					- x13	- x23		+ x31	+ x32	<=	-1
(20)			-3ξ03			- ξ23	- x12	- x13	- x23		+ x31	+ x32	<=	-2
(21)			-3ξ03	- ξ12		- 2ξ23	- x12	-2x13	- x23	+ x21	+ x31	+ x32	<=	-3
(22)			-3ξ03	- 3ξ12		- ξ23		- x13	- x23	+ x21	+ x31	+ x32	<=	-2
(23)	+ ξ01												<=	1
(24)			-3ξ03		- ξ13			- x13	- x23	- x21	+ x31	+ x32	<=	-2
(25)			-3ξ03		- ξ13	- ξ23	- x12	- x13	- x23	- x21	+ x31	+ x32	<=	-3
(26)			-2ξ03		- ξ13		- x12	- x13	- x23		+ x31		<=	-2
(27)			-2ξ03	- ξ12	- 2ξ13			- x13	- x23		+ x31		<=	-3
(28)			-2ξ03	- 3ξ12	- ξ13			- x13	- x23	+ x21	+ x31		<=	-2
(29)			-3ξ03	- 3ξ12	- ξ13		+ x12	- x13	- x23		+ x31	+ x32	<=	-2
(30)			-3ξ03	- 6ξ12	- ξ13	- ξ23	+ x12	- x13	- x23	+ x21	+ x31	+ x32	<=	-3
(31)				- 3ξ12	- ξ13	- 2ξ23	+ x12			- x21			<=	-3
(32)									- x23				<=	0
(33)			-2ξ03			- ξ23		- x13	- x23	- x21		+ x32	<=	-2
(34)			- ξ03					- x13	- x23				<=	-1
(35)			-2ξ03	- 3ξ12		- ξ23	+ x12	- x13	- x23			+ x32	<=	-2
(36)			-2ξ03	- 3ξ12	- ξ13	- 3ξ23	- x12	- x13	- x23	+ x21		+ x32	<=	-5
(37)							+ x12		- x23	- x21		+ x32	<=	1
(38)						- ξ23			- x23	- x21		+ x32	<=	0
(39)				- 2ξ12	- ξ13	- 3ξ23			- x23			+ x32	<=	-3
(40)				- ξ12			+ x12		- x23	- x21			<=	0
(41)				- 3ξ12		- ξ23	+ x12		- x23			+ x32	<=	0
(42)	- ξ01		- ξ03	- 2ξ12	- ξ13		- x12	- x13	- x23	+ x21			<=	-3
(43)			-3ξ03	- ξ12	- 2ξ13		+ x12	- x13	-2x23	- x21	+ x31	+ x32	<=	-3
(44)	-2ξ01		-2ξ03	- 2ξ12	- 2ξ13		- x12	- x13	- x23	+ x21	+ x31		<=	-5
(45)	- ξ01		-2ξ03	- ξ12	- ξ13		- x12	-2x13	- x23	+ x21	+ x31		<=	-3
(46)			-3ξ03	- 2ξ12		- 2ξ23	- x12	-2x13	-2x23	+ x21		+ x32	<=	-4
(47)			-6ξ03	- 2ξ12		- 4ξ23	- x12	-3x13	-3x23	+ x21	+ x31	+2x32	<=	-7
(48)				- 3ξ12	- 2ξ13	- ξ23	- x12			+ x21			<=	-3
(49)	- ξ01		-3ξ03	- 3ξ12	- 2ξ13	- 3ξ23	- x12	- x13	- x23	+ x21	+ x31	+ x32	<=	-7
(50)			-3ξ03	- 3ξ12	- 2ξ13	- 4ξ23	- x12	- x13	- x23	+ x21	+ x31	+ x32	<=	-7

(51)		-5ξ03	- 3ξ12	- ξ13	- 5ξ23	- x12	-2x13	-2x23	+ x21	+ x31	+2x32	<=	-8
(52)		-2ξ03	- 2ξ12	- 2ξ13	- 2ξ23	- x12	- x13	- x23	+ x21	+ x31		<=	-5
(53)		-2ξ03	- 2ξ12	- ξ13	- 3ξ23	- x12	- x13	- x23	+ x21	+ x31	+ x32	<=	-4
(54)		-3ξ03	- 4ξ12	- 2ξ13	- 6ξ23	- x12	- x13	-2x23	+ x21	+ x31	+2x32	<=	-8
(55)	- ξ01	-3ξ03	- 3ξ12	- 4ξ13	- ξ23	+ x12	- x13	- x23	- x21	+ x31	+ x32	<=	-7
(56)		-3ξ03	- 4ξ12	- 6ξ13	- 2ξ23	+ x12	-2x13	- x23	- x21	+2x31	+ x32	<=	-8
(57)		-3ξ03	- 3ξ12	- 4ξ13	- 2ξ23	+ x12	- x13	- x23	- x21	+ x31	+ x32	<=	-7
(58)		-2ξ03	- 3ξ12	- 3ξ13	- ξ23	+ x12	- x13	- x23	- x21	+ x31		<=	-5
(59)		-5ξ03	- 3ξ12	- 5ξ13	- ξ23	+ x12	-2x13	-2x23	- x21	+2x31	+ x32	<=	-8
(60)	-2ξ01	-2ξ03	- 2ξ12	- 2ξ13		+ x12	- x13	- x23	- x21		+ x32	<=	-5
(61)		-2ξ03	- 2ξ12	- 2ξ13	- 2ξ23	+ x12	- x13	- x23	- x21		+ x32	<=	-5
(62)		-6ξ03	- 2ξ12		- 4ξ23	+ x12	-3x13	-3x23	- x21		+3x32	<=	-7
(63)		-6ξ03	- 2ξ12	- 4ξ13		+ x12	-3x13	-3x23	- x21	+2x31	+ x32	<=	-7
(64)	- ξ01	-2ξ03	- ξ12	- ξ13		+ x12	- x13	-2x23	- x21		+ x32	<=	-3
(65)		-2ξ03	- ξ12	- ξ13	- ξ23	+ x12	- x13	-2x23	- x21		+ x32	<=	-3
(66)		-2ξ03	- 2ξ12	- 3ξ13	- ξ23	+ x12	- x13	- x23	- x21	+ x31	+ x32	<=	-4
(67)		-4ξ03	- ξ12		- 2ξ23	+ x12	-2x13	-3x23	- x21		+2x32	<=	-4
(68)		-2ξ03		- ξ13	- 3ξ23	- x12	- x13			+ x31	+ x32	<=	-2
(69)		-2ξ03	- 3ξ12	- ξ13	- 3ξ23		- x13		+ x21	+ x31	+ x32	<=	-2
(70)		-2ξ03	- ξ12	- 2ξ13	- 3ξ23		- x13			+ x31	+ x32	<=	-3
(71)					- 3ξ23						+ x32	<=	0
(72)		- ξ03			- 3ξ23		- x13				+ x32	<=	-1
(73)			- ξ12		- 3ξ23	+ x12			- x21		+ x32	<=	0
(74)	- ξ01	- ξ03	- 2ξ12	- ξ13	- 3ξ23	- x12	- x13		+ x21		+ x32	<=	-3
(75)		-3ξ03	- ξ12	- 2ξ13	- 3ξ23	+ x12	- x13	- x23	- x21	+ x31	+2x32	<=	-3
(76)	-2ξ01	-2ξ03	- 2ξ12	- 2ξ13	- 3ξ23	- x12	- x13		+ x21	+ x31	+ x32	<=	-5
(77)	- ξ01	-2ξ03	- ξ12	- ξ13	- 3ξ23	- x12	-2x13		+ x21	+ x31	+ x32	<=	-3
(78)		-3ξ03	- 2ξ12		- 5ξ23	- x12	-2x13	- x23	+ x21		+2x32	<=	-4
(79)		-6ξ03	- 2ξ12		- 7ξ23	- x12	-3x13	-2x23	+ x21	+ x31	+3x32	<=	-7
(80)		-2ξ03	- 3ξ12	- 3ξ13	- 4ξ23	+ x12	- x13		- x21	+ x31	+ x32	<=	-5
(81)		-2ξ03	- 2ξ12	- 2ξ13	- 5ξ23	- x12	- x13		+ x21	+ x31	+ x32	<=	-5
(82)		-5ξ03	- 3ξ12	- 5ξ13	- 4ξ23	+ x12	-2x13	- x23	- x21	+2x31	+2x32	<=	-8
(83)		-6ξ03	- 2ξ12	- 4ξ13	- 6ξ23	+ x12	-3x13	- x23	- x21	+2x31	+3x32	<=	-7
(84)	- ξ01	-2ξ03	- ξ12	- ξ13	- 3ξ23	+ x12	- x13	- x23	- x21		+2x32	<=	-3
(85)		-2ξ03	- ξ12	- ξ13	- 4ξ23	+ x12	- x13	- x23	- x21		+2x32	<=	-3
(86)		-4ξ03	- ξ12		- 5ξ23	+ x12	-2x13	-2x23	- x21		+3x32	<=	-4
(87)				- 3ξ13							+ x31	<=	0
(88)			- ξ12	- 3ξ13		- x12			+ x21	+ x31		<=	0
(89)		-2ξ03	- ξ12	- 3ξ13	- 2ξ23			- x23		+ x31	+ x32	<=	-3
(90)		-3ξ03	- ξ12	- 3ξ13	- 2ξ23	- x12	- x13	- x23	+ x21	+2x31	+ x32	<=	-3
(91)		-2ξ03		- 3ξ13	- ξ23			- x23	- x21	+ x31	+ x32	<=	-2
(92)		- ξ03		- 3ξ13				- x23		+ x31		<=	-1
(93)		-2ξ03	- 3ξ12	- 3ξ13	- ξ23	+ x12		- x23		+ x31	+ x32	<=	-2
(94)		-2ξ03	- 3ξ12	- 4ξ13	- 3ξ23	- x12		- x23	+ x21	+ x31	+ x32	<=	-5
(95)	- ξ01	- ξ03	- 2ξ12	- 4ξ13		- x12		- x23	+ x21	+ x31		<=	-3
(96)		-3ξ03	- 2ξ12	- 6ξ13	- 2ξ23	- x12		-2x23	+ x21	+2x31	+ x32	<=	-4
(97)		-6ξ03	- 2ξ12	- 6ξ13	- 4ξ23	- x12	- x13	-3x23	+ x21	+3x31	+2x32	<=	-7
(98)		-5ξ03	- 3ξ12	- 4ξ13	- 5ξ23	- x12	- x13	-2x23	+ x21	+2x31	+2x32	<=	-8
(99)	- ξ01	-2ξ03	- ξ12	- 4ξ13		- x12	- x13	- x23	+ x21	+2x31		<=	-3
(100)	-2ξ01	-2ξ03	- 2ξ12	- 5ξ13		+ x12		- x23	- x21	+ x31	+ x32	<=	-5

(101)			-2ξ03	- 2ξ12	- 5ξ13	- 2ξ23	+ x12		- x23	- x21	+ x31	+ x32	<=	-5
(102)			-6ξ03	- 2ξ12	- 7ξ13		+ x12	-2x13	-3x23	- x21	+3x31	+ x32	<=	-7
(103)			-6ξ03	- 2ξ12	- 9ξ13	- 4ξ23	+ x12		-3x23	- x21	+3x31	+3x32	<=	-7
(104)	- ξ01		-2ξ03	- ξ12	- 4ξ13		+ x12		-2x23	- x21	+ x31	+ x32	<=	-3
(105)			-2ξ03	- ξ12	- 4ξ13	- ξ23	+ x12		-2x23	- x21	+ x31	+ x32	<=	-3
(106)			-4ξ03	- ξ12	- 6ξ13	- 2ξ23	+ x12		-3x23	- x21	+2x31	+2x32	<=	-4
(107)			- ξ03		- 3ξ13	- 3ξ23					+ x31	+ x32	<=	-1
(108)	- ξ01		- ξ03	- 2ξ12	- 4ξ13	- 3ξ23	- x12			+ x21	+ x31	+ x32	<=	-3
(109)	- ξ01		-2ξ03	- ξ12	- 4ξ13	- 3ξ23	- x12	- x13		+ x21	+2x31	+ x32	<=	-3
(110)			-6ξ03	- 2ξ12	- 7ξ13	- 6ξ23	+ x12	-2x13	- x23	- x21	+3x31	+3x32	<=	-7
(111)	- ξ01		-2ξ03	- ξ12	- 4ξ13	- 3ξ23	+ x12		- x23	- x21	+ x31	+2x32	<=	-3
(112)			-2ξ03	- ξ12	- 4ξ13	- 4ξ23	+ x12		- x23	- x21	+ x31	+2x32	<=	-3
(113)			-3ξ03	- 2ξ12	- 6ξ13	- 5ξ23	- x12		- x23	+ x21	+2x31	+2x32	<=	-4
(114)			-6ξ03	- 2ξ12	- 6ξ13	- 7ξ23	- x12	- x13	-2x23	+ x21	+3x31	+3x32	<=	-7
(115)			-4ξ03	- ξ12	- 6ξ13	- 5ξ23	+ x12		-2x23	- x21	+2x31	+3x32	<=	-4
(116)	- ξ01	- ξ02		- ξ12								- x32	<=	-2
(117)		- ξ02					- x12					- x32	<=	-1
(118)		- ξ02		- 3ξ12						+ x21		- x32	<=	-1
(119)												- x32	<=	0
(120)		-2ξ02		- ξ12			- x12	- x13		+ x21		-2x32	<=	-2
(121)	-2ξ01	-2ξ02	-2ξ03	- 2ξ12	- 2ξ13		- x12	- x13	- x23	+ x21	+ x31	-2x32	<=	-7
(122)		-3ξ02		- 2ξ12			- x12			+ x21		-3x32	<=	-3
(123)	- ξ01	-3ξ02	- ξ03	- 2ξ12			- x12	- x13	- x23	+ x21		-3x32	<=	-5
(124)	- ξ01	- ξ02	- ξ03	- 2ξ12			+ x12	- x13	- x23	- x21		- x32	<=	-3
(125)			-4ξ03	- ξ12	- 2ξ13		- x12	-3x13	-2x23	+ x21	+2x31		<=	-4
(126)			-6ξ03	- 2ξ12	- 4ξ13		- x12	-3x13	-3x23	+ x21	+3x31		<=	-7
(127)	- ξ01		- ξ03	- 2ξ12	- ξ13		+ x12	- x13	- x23	- x21			<=	-3
(128)			- ξ03	- 2ξ12	- ξ13	- ξ23	+ x12	- x13	- x23	- x21			<=	-3
(129)			-2ξ03	- ξ12	- ξ13	- ξ23	- x12	-2x13	- x23	+ x21	+ x31		<=	-3
(130)			-3ξ03	- 2ξ12	- 2ξ13		+ x12	-2x13	-2x23	- x21	+ x31		<=	-4
(131)	- ξ01	- ξ02		- ξ12		- 3ξ23			+ x23				<=	-2
(132)		- ξ02				- 3ξ23	- x12		+ x23				<=	-1
(133)		- ξ02		- 3ξ12		- 3ξ23			+ x23	+ x21			<=	-1
(134)						- 3ξ23			+ x23				<=	0
(135)		-2ξ02		- ξ12		- 6ξ23	- x12	- x13	+2x23	+ x21			<=	-2
(136)	-2ξ01	-2ξ02	-2ξ03	- 2ξ12	- 2ξ13	- 9ξ23	- x12	- x13	+2x23	+ x21	+ x31	+ x32	<=	-7
(137)		-3ξ02		- 2ξ12		- 9ξ23	- x12		+3x23	+ x21			<=	-3
(138)	- ξ01	-3ξ02	- ξ03	- 2ξ12		-12ξ23	- x12	- x13	+3x23	+ x21		+ x32	<=	-5
(139)	- ξ01	- ξ02	- ξ03	- 2ξ12		- 6ξ23	+ x12	- x13	+ x23	- x21		+ x32	<=	-3
(140)			-4ξ03	- ξ12	- 2ξ13	- 6ξ23	- x12	-3x13		+ x21	+2x31	+2x32	<=	-4
(141)			-6ξ03	- 2ξ12	- 4ξ13	- 9ξ23	- x12	-3x13		+ x21	+3x31	+3x32	<=	-7
(142)	- ξ01		- ξ03	- 2ξ12	- ξ13	- 3ξ23	+ x12	- x13		- x21		+ x32	<=	-3
(143)			- ξ03	- 2ξ12	- ξ13	- 4ξ23	+ x12	- x13		- x21		+ x32	<=	-3
(144)			-2ξ03	- ξ12	- ξ13	- 4ξ23	- x12	-2x13		+ x21	+ x31	+ x32	<=	-3
(145)			-3ξ03	- 2ξ12	- 2ξ13	- 6ξ23	+ x12	-2x13		- x21	+ x31	+2x32	<=	-4
(146)	- ξ01	- ξ02	- ξ03	- 2ξ12	- 3ξ13		+ x12		- x23	- x21	+ x31	- x32	<=	-3
(147)		-2ξ02		- ξ12	- 3ξ13		- x12			+ x21	+ x31	-2x32	<=	-2
(148)	- ξ01	-3ξ02	- ξ03	- 2ξ12	- 3ξ13		- x12		- x23	+ x21	+ x31	-3x32	<=	-5
(149)			-4ξ03	- ξ12	- 5ξ13		- x12	-2x13	-2x23	+ x21	+3x31		<=	-4
(150)	- ξ01		- ξ03	- 2ξ12	- 4ξ13		+ x12		- x23	- x21	+ x31		<=	-3

(151)			- ξ03	- 2ξ12	- 4ξ13	- ξ23	+ x12		- x23	- x21	+ x31		<=	-3
(152)			-2ξ03	- ξ12	- 4ξ13	- ξ23	- x12	- x13	- x23	+ x21	+2x31		<=	-3
(153)			-3ξ03	- 2ξ12	- 5ξ13		+ x12	- x13	-2x23	- x21	+2x31		<=	-4
(154)	- ξ01	- ξ02	- ξ03	- 2ξ12	- 3ξ13	- 6ξ23	+ x12		+ x23	- x21	+ x31	+ x32	<=	-3
(155)		-2ξ02		- ξ12	- 3ξ13	- 6ξ23	- x12		+2x23	+ x21	+ x31		<=	-2
(156)	- ξ01	-3ξ02	- ξ03	- 2ξ12	- 3ξ13	-12ξ23	- x12		+3x23	+ x21	+ x31	+ x32	<=	-5
(157)			-4ξ03	- ξ12	- 5ξ13	- 6ξ23	- x12	-2x13		+ x21	+3x31	+2x32	<=	-4
(158)	- ξ01		- ξ03	- 2ξ12	- 4ξ13	- 3ξ23	+ x12			- x21	+ x31	+ x32	<=	-3
(159)			- ξ03	- 2ξ12	- 4ξ13	- 4ξ23	+ x12			- x21	+ x31	+ x32	<=	-3
(160)			-2ξ03	- ξ12	- 4ξ13	- 4ξ23	- x12	- x13		+ x21	+2x31	+ x32	<=	-3
(161)			-3ξ03	- 2ξ12	- 5ξ13	- 6ξ23	+ x12	- x13		- x21	+2x31	+2x32	<=	-4
(162)		-3ξ02					- x12		+ x23	+ x21		- x32	<=	-1
(163)	- ξ01	-3ξ02		- ξ12	- ξ13	- ξ23	- x12	- x13	+ x23	+ x21		- x32	<=	-4
(164)	-2ξ01	-3ξ02		- 2ξ12	- 2ξ13	- 2ξ23	- x12		+ x23			- x32	<=	-7
(165)	-2ξ01	-3ξ02		- 2ξ12	- 3ξ13	- 2ξ23	-2x12	- x13	+ x23	+ x21	+ x31	- x32	<=	-7
(166)	-3ξ01	-3ξ02		- 4ξ12	- 5ξ13	- 3ξ23	- x12	- x13	+ x23	+ x21	+ x31	- x32	<=	-11
(167)	-3ξ01	-3ξ02		- 4ξ12	- 2ξ13	- 3ξ23			+ x23			- x32	<=	-9
(168)	-2ξ01	-3ξ02		- 5ξ12	- 2ξ13	- 2ξ23			+ x23	+ x21		- x32	<=	-7
(169)	-2ξ01	-3ξ02		- 5ξ12	- 3ξ13	- 2ξ23	- x12	- x13	+ x23	+2x21	+ x31	- x32	<=	-7
(170)	- ξ01	-2ξ02		- ξ12	- ξ13	- ξ23	- x12		+ x23		- x31	- x32	<=	-4
(171)	-2ξ01	-2ξ02		- 2ξ12		- ξ23			+ x23		- x31	- x32	<=	-4
(172)	- ξ01	-2ξ02		- 4ξ12	- ξ13	- ξ23			+ x23	+ x21	- x31	- x32	<=	-4
(173)		-3ξ02				- ξ23	-2x12	- x13	+ x23	+ x21		- x32	<=	-2
(174)		-3ξ02				- 2ξ23	-3x12		+ x23			- x32	<=	-3
(175)		-3ξ02			- ξ13	- 2ξ23	-4x12	- x13	+ x23	+ x21	+ x31	- x32	<=	-3
(176)		-6ξ02		- ξ12		- 2ξ23	-3x12	-3x13	+2x23	+3x21		-2x32	<=	-4
(177)		-3ξ02		- 3ξ12		- ξ23	- x12	- x13	+ x23	+2x21		- x32	<=	-2
(178)	- ξ01	-3ξ02		- 2ξ12	- 3ξ13	- ξ23	- x12	- x13	+ x23	+ x21	+ x31	- x32	<=	-5
(179)		-3ξ02		- 9ξ12		- 2ξ23			+ x23	+3x21		- x32	<=	-3
(180)		-3ξ02		- 9ξ12	- ξ13	- 2ξ23	- x12	- x13	+ x23	+4x21	+ x31	- x32	<=	-3
(181)		-2ξ02		- 6ξ12		- ξ23			+ x23	+2x21	- x31	- x32	<=	-2
(182)		-2ξ02				- ξ23	-2x12		+ x23		- x31	- x32	<=	-2
(183)	-3ξ01	-3ξ02		- 6ξ12	- 2ξ13	- 4ξ23	+ x12		+ x23	- x21		- x32	<=	-10
(184)	-2ξ01	-3ξ02		- 3ξ12	- ξ13	- 2ξ23			+ x23			- x32	<=	-6
(185)	-4ξ01	-2ξ02		- 5ξ12	- ξ13	- 3ξ23	+ x12		+ x23	- x21	- x31	- x32	<=	-9
(186)		-2ξ02					- x12		+ x23		- x31	- x32	<=	-1
(187)		-2ξ02		- 3ξ12					+ x23	+ x21	- x31	- x32	<=	-1
(188)								+ x13	+ x23		- x31	- x32	<=	1
(189)	-2ξ01	-2ξ02		- 2ξ12	- 2ξ13	- 2ξ23	- x12		+ x23	- x21	- x31	- x32	<=	-7
(190)	-2ξ01			- 2ξ12	- 2ξ13	- 2ξ23		+ x13	+ x23	- x21	- x31	- x32	<=	-5
(191)	-2ξ01	-2ξ02		- 8ξ12	- 2ξ13	- 2ξ23	+ x12		+ x23	+ x21	- x31	- x32	<=	-7
(192)	-3ξ01	-2ξ02		- 3ξ12	- ξ13	- 2ξ23			+ x23	- x21	- x31	- x32	<=	-7
(193)	-3ξ01			- 4ξ12	- 2ξ13	- 3ξ23	+ x12	+ x13	+ x23	- x21	- x31	- x32	<=	-7
(194)	-2ξ01			- 5ξ12	- 2ξ13	- 2ξ23	+ x12	+ x13	+ x23		- x31	- x32	<=	-5
(195)	-3ξ01	-2ξ02		- 6ξ12	- ξ13	- 2ξ23	+ x12		+ x23		- x31	- x32	<=	-7
(196)	-3ξ01			- 6ξ12	- 2ξ13	- 4ξ23	+2x12	+ x13	+ x23	-2x21	- x31	- x32	<=	-8
(197)	-2ξ01			- 3ξ12	- ξ13	- 2ξ23	+ x12	+ x13	+ x23	- x21	- x31	- x32	<=	-4
(198)		-3ξ02		- ξ12			- x12		+ x23	+ x21	- x31	-2x32	<=	-2
(199)						- ξ23			+ x23		- x31	- x32	<=	0
(200)		-6ξ02		- 2ξ12		- ξ23	-2x12		+3x23	+2x21	-3x31	-3x32	<=	-4

(201)			- ξ12	- 2ξ13	- 3ξ23			+ x23			- x32	<=	-3
(202)		-3ξ02	- 2ξ12	- ξ13		- x12	+ x13	+ x23	+ x21	- x31	-4x32	<=	-3
(203)		- ξ02	- ξ12	- ξ13	- 2ξ23			+ x23			- x32	<=	-2
(204)	- ξ01	- ξ02	- 4ξ12	- 2ξ13	- 4ξ23	+ x12		+ x23	- x21		- x32	<=	-6
(205)	-2ξ01		- 3ξ12	- ξ13	- 3ξ23	+ x12		+ x23	- x21	- x31	- x32	<=	-5
(206)		-3ξ02	- ξ12		- 3ξ23	- x12		+2x23	+ x21	- x31	- x32	<=	-2
(207)		-3ξ02	- 2ξ12	- ξ13	- 9ξ23	- x12	+ x13	+4x23	+ x21	- x31	- x32	<=	-3
(208)	-2ξ01	-2ξ02	- 2ξ12	- 3ξ13	- ξ23		+ x13	+ x23			- x32	<=	-4
(209)	- ξ01	-2ξ02	- ξ12	- 4ξ13	- ξ23	- x12	+ x13	+ x23			- x32	<=	-4
(210)	- ξ01	-3ξ02	- ξ12	- 4ξ13	- ξ23	- x12		+ x23	+ x21	+ x31	- x32	<=	-4
(211)	- ξ01	-2ξ02	- 4ξ12	- 4ξ13	- ξ23		+ x13	+ x23	+ x21		- x32	<=	-4
(212)		-3ξ02		- 3ξ13	- ξ23	-2x12		+ x23	+ x21	+ x31	- x32	<=	-2
(213)		-6ξ02	- ξ12	- 9ξ13	- 2ξ23	-3x12		+2x23	+3x21	+3x31	-2x32	<=	-4
(214)		-2ξ02	- 6ξ12	- 3ξ13	- ξ23		+ x13	+ x23	+2x21		- x32	<=	-2
(215)		-2ξ02		- 3ξ13	- ξ23	-2x12	+ x13	+ x23			- x32	<=	-2
(216)		-3ξ02	- 3ξ12	- 3ξ13	- ξ23	- x12		+ x23	+2x21	+ x31	- x32	<=	-2
(217)	-4ξ01	-2ξ02	- 5ξ12	- 4ξ13	- 3ξ23	+ x12	+ x13	+ x23	- x21		- x32	<=	-9
(218)		-2ξ02		- 3ξ13		- x12	+ x13	+ x23			- x32	<=	-1
(219)		-2ξ02	- 3ξ12	- 3ξ13			+ x13	+ x23	+ x21		- x32	<=	-1
(220)	-3ξ01	-2ξ02	- 6ξ12	- 4ξ13	- 2ξ23	+ x12	+ x13	+ x23			- x32	<=	-7
(221)	-3ξ01	-2ξ02	- 3ξ12	- 4ξ13	- 2ξ23		+ x13	+ x23	- x21		- x32	<=	-7
(222)	-2ξ01	-2ξ02	- 8ξ12	- 5ξ13	- 2ξ23	+ x12	+ x13	+ x23	+ x21		- x32	<=	-7
(223)	-2ξ01	-2ξ02	- 2ξ12	- 5ξ13	- 2ξ23	- x12	+ x13	+ x23	- x21		- x32	<=	-7
(224)				- 3ξ13	- ξ23		+ x13	+ x23			- x32	<=	0
(225)		-6ξ02	- 2ξ12	- 9ξ13	- ξ23	-2x12	+3x13	+3x23	+2x21		-3x32	<=	-4
(226)	-2ξ01		- 3ξ12	- 4ξ13	- 3ξ23	+ x12	+ x13	+ x23	- x21		- x32	<=	-5
(227)		-3ξ02	- ξ12	- 3ξ13		- x12	+ x13	+ x23	+ x21		-2x32	<=	-2
(228)		-3ξ02	- ξ12	- 3ξ13	- 3ξ23	- x12	+ x13	+2x23	+ x21		- x32	<=	-2
(229)										- x31		<=	0
(230)	- ξ01								- x21	- x31		<=	-1
(231)	- ξ01			- 3ξ12		+ x12				- x31		<=	-1
(232)	-2ξ01			- 2ξ12	- ξ23	+ x12			- x21	- x31		<=	-3
(233)	-3ξ01			- 2ξ12		+ x12			- x21	- x31		<=	-3
(234)	-2ξ01			- ξ12		+ x12		- x23	- x21	- x31		<=	-2
(235)	-2ξ01		-2ξ03	- 2ξ12	- 2ξ13	- 2ξ23	+ x12	- x13	- x23	- x21	- x31	+ x32	<=	-7
(236)	-2ξ01			- 2ξ12	- 2ξ13	- 2ξ23	+ x12		- x23	- x21	- x31	+ x32	<=	-5
(237)	-2ξ01			- ξ12	- ξ13	- ξ23	+ x12		- x23	-2x21	- x31	+ x32	<=	-3
(238)	-2ξ01			- 4ξ12	- ξ13	- ξ23	+2x12		- x23	- x21	- x31	+ x32	<=	-3
(239)	-6ξ01			- 4ξ12		- 2ξ23	+3x12		- x23	-3x21	-3x31	+ x32	<=	-7
(240)	-4ξ01			- 2ξ12		- ξ23	+2x12		- x23	-3x21	-2x31	+ x32	<=	-4
(241)	-4ξ01			- 5ξ12		- ξ23	+3x12		- x23	-2x21	-2x31	+ x32	<=	-4
(242)	-3ξ01		-2ξ03	- 2ξ12	- 2ξ13	- ξ23	+ x12	- x13	- x23	-2x21	- x31	+ x32	<=	-7
(243)	-2ξ01		-2ξ03	- ξ12	- ξ13	- ξ23	+ x12	- x13	-2x23	-2x21	- x31	+ x32	<=	-5
(244)	-3ξ01		- ξ03	- 2ξ12	- ξ13		+ x12	- x13	- x23	- x21	- x31		<=	-5
(245)	-4ξ01		-2ξ03	- 2ξ12	- 2ξ13	- 2ξ23	+2x12	- x13	-2x23	-4x21	-2x31	+2x32	<=	-8
(246)	-2ξ01		-4ξ03	- 2ξ12	- 2ξ13	- 2ξ23	+2x12	-2x13	-4x23	-2x21	- x31	+2x32	<=	-8
(247)	-3ξ01		-2ξ03	- 5ξ12	- 2ξ13	- ξ23	+2x12	- x13	- x23	- x21	- x31	+ x32	<=	-7
(248)	-2ξ01		-2ξ03	- 4ξ12	- ξ13	- ξ23	+2x12	- x13	-2x23	- x21	- x31	+ x32	<=	-5
(249)	-4ξ01		-2ξ03	- 8ξ12	- 2ξ13	- 2ξ23	+4x12	- x13	-2x23	-2x21	-2x31	+2x32	<=	-8
(250)	-2ξ01			- ξ12		- 3ξ23	+ x12			- x21	- x31	+ x32	<=	-2
(251)	-3ξ01		- ξ03	- 2ξ12	- ξ13	- 3ξ23	+ x12	- x13		- x21	- x31	+ x32	<=	-5
(252)	-2ξ01		-2ξ03	- ξ12	- ξ13	- 4ξ23	+ x12	- x13	- x23	-2x21	- x31	+2x32	<=	-5
(253)	-2ξ01		-4ξ03	- 2ξ12	- 2ξ13	- 8ξ23	+2x12	-2x13	-2x23	-2x21	- x31	+4x32	<=	-8
(254)	-2ξ01		-2ξ03	- 4ξ12	- ξ13	- 4ξ23	+2x12	- x13	- x23	- x21	- x31	+2x32	<=	-5
(255)					- 3ξ13			+ x13					<=	0
(256)	- ξ01				- 3ξ13			+ x13		- x21			<=	-1
(257)	- ξ01			- 3ξ12	- 3ξ13		+ x12	+ x13					<=	-1
(258)	-2ξ01			- 2ξ12	- 3ξ13	- ξ23	+ x12	+ x13		- x21			<=	-3
(259)	-3ξ01			- 2ξ12	- 3ξ13		+ x12	+ x13		- x21			<=	-3
(260)	-2ξ01			- ξ12	- 3ξ13		+ x12	+ x13	- x23	- x21			<=	-2
(261)	-2ξ01		-2ξ03	- 2ξ12	- 8ξ13	- 2ξ23	+ x12	+ x13	- x23	- x21	+ x31	+ x32	<=	-7
(262)	-2ξ01			- 2ξ12	- 5ξ13	- 2ξ23	+ x12	+ x13	- x23	- x21		+ x32	<=	-5
(263)	-2ξ01			- ξ12	- 4ξ13	- ξ23	+ x12	+ x13	- x23	-2x21		+ x32	<=	-3
(264)	-2ξ01			- 4ξ12	- 4ξ13	- ξ23	+2x12	+ x13	- x23	- x21		+ x32	<=	-3
(265)	-6ξ01			- 4ξ12	- 9ξ13	- 2ξ23	+3x12	+3x13	- x23	-3x21		+ x32	<=	-7
(266)	-4ξ01			- 2ξ12	- 6ξ13	- ξ23	+2x12	+2x13	- x23	-3x21		+ x32	<=	-4
(267)	-4ξ01			- 5ξ12	- 6ξ13	- ξ23	+3x12	+2x13	- x23	-2x21		+ x32	<=	-4
(268)	-3ξ01		-2ξ03	- 2ξ12	- 8ξ13	- ξ23	+ x12	+ x13	- x23	-2x21	+ x31	+ x32	<=	-7
(269)	-2ξ01		-2ξ03	- ξ12	- 7ξ13	- ξ23	+ x12	+ x13	-2x23	-2x21	+ x31	+ x32	<=	-5
(270)	-3ξ01		- ξ03	- 2ξ12	- 7ξ13		+ x12	+ x13	- x23	- x21	+ x31		<=	-5
(271)	-4ξ01		-2ξ03	- 2ξ12	-11ξ13	- 2ξ23	+2x12	+2x13	-2x23	-4x21	+ x31	+2x32	<=	-8
(272)	-2ξ01		-4ξ03	- 2ξ12	-11ξ13	- 2ξ23	+2x12	+ x13	-4x23	-2x21	+2x31	+2x32	<=	-8
(273)	-3ξ01		-2ξ03	- 5ξ12	- 8ξ13	- ξ23	+2x12	+ x13	- x23	- x21	+ x31	+ x32	<=	-7
(274)	-2ξ01		-2ξ03	- 4ξ12	- 7ξ13	- ξ23	+2x12	+ x13	-2x23	- x21	+ x31	+ x32	<=	-5
(275)	-4ξ01		-2ξ03	- 8ξ12	-11ξ13	- 2ξ23	+4x12	+2x13	-2x23	-2x21	+ x31	+2x32	<=	-8
(276)	-2ξ01			- ξ12	- 3ξ13	- 3ξ23	+ x12	+ x13		- x21		+ x32	<=	-2
(277)	-3ξ01		- ξ03	- 2ξ12	- 7ξ13	- 3ξ23	+ x12	+ x13		- x21	+ x31	+ x32	<=	-5
(278)	-2ξ01		-2ξ03	- ξ12	- 7ξ13	- 4ξ23	+ x12	+ x13	- x23	-2x21	+ x31	+2x32	<=	-5
(279)	-2ξ01		-4ξ03	- 2ξ12	-11ξ13	- 8ξ23	+2x12	+ x13	-2x23	-2x21	+2x31	+4x32	<=	-8
(280)	-2ξ01		-2ξ03	- 4ξ12	- 7ξ13	- 4ξ23	+2x12	+ x13	- x23	- x21	+ x31	+2x32	<=	-5
(281)	-2ξ01		- ξ03	- 2ξ12	- ξ13	- ξ23	+ x12	- x13	- x23	- x21	- x31		<=	-5
(282)	-3ξ01	- ξ02	- ξ03	- 2ξ12		- 6ξ23	+ x12	- x13	+ x23	- x21	- x31	+ x32	<=	-5
(283)	-3ξ01	- ξ02	- ξ03	- 2ξ12			+ x12	- x13	- x23	- x21	- x31	- x32	<=	-5
(284)	-2ξ01		- ξ03	- 2ξ12	- ξ13	- 4ξ23	+ x12	- x13		- x21	- x31	+ x32	<=	-5
(285)	-2ξ01		- ξ03	- 2ξ12	- 7ξ13	- ξ23	+ x12	+ x13	- x23	- x21	+ x31		<=	-5
(286)	-3ξ01	- ξ02	- ξ03	- 2ξ12	- 6ξ13	- 6ξ23	+ x12	+ x13	+ x23	- x21	+ x31	+ x32	<=	-5
(287)	-3ξ01	- ξ02	- ξ03	- 2ξ12	- 6ξ13		+ x12	+ x13	- x23	- x21	+ x31	- x32	<=	-5
(288)	-2ξ01		- ξ03	- 2ξ12	- 7ξ13	- 4ξ23	+ x12	+ x13		- x21	+ x31	+ x32	<=	-5
(289)	-5ξ01	-2ξ02		- 7ξ12		- 2ξ23	+2x12		+ x23	- x21	-2x31	- x32	<=	-8
(290)	-5ξ01	-2ξ02		- 4ξ12		- 2ξ23	+ x12		+ x23	-2x21	-2x31	- x32	<=	-8
(291)	-3ξ01			- 2ξ12		- ξ23	+ x12	+ x13	+ x23	- x21	-2x31	- x32	<=	-3
(292)	-5ξ01			- 5ξ12	- ξ13	- 3ξ23	+2x12	+ x13	+ x23	-2x21	-2x31	- x32	<=	-8
(293)	-3ξ01			- 2ξ12		- 2ξ23	+ x12		+ x23	-2x21	-2x31	- x32	<=	-4
(294)	-3ξ01			- 5ξ12		- 2ξ23	+2x12		+ x23	- x21	-2x31	- x32	<=	-4
(295)	-6ξ01			- 7ξ12		- 2ξ23	+3x12	+ x13	+ x23	-2x21	-3x31	- x32	<=	-7
(296)	-6ξ01			- 4ξ12		- 2ξ23	+2x12	+ x13	+ x23	-3x21	-3x31	- x32	<=	-7
(297)	-5ξ01	-2ξ02		- 7ξ12	- 6ξ13	- 2ξ23	+2x12	+2x13	+ x23	- x21		- x32	<=	-8
(298)	-5ξ01	-2ξ02		- 4ξ12	- 6ξ13	- 2ξ23	+ x12	+2x13	+ x23	-2x21		- x32	<=	-8
(299)	-3ξ01			- 2ξ12	- 3ξ13	- ξ23	+ x12	+2x13	+ x23	- x21	- x31	- x32	<=	-3
(300)	-5ξ01			- 5ξ12	- 4ξ13	- 3ξ23	+2x12	+2x13	+ x23	-2x21	- x31	- x32	<=	-8

(301)	-3ξ01	- 2ξ12	- 6ξ13	- 2ξ23	+ x12	+2x13	+ x23	-2x21		- x32	<=	-4
(302)	-3ξ01	- 5ξ12	- 6ξ13	- 2ξ23	+2x12	+2x13	+ x23	- x21		- x32	<=	-4
(303)	-6ξ01	- 7ξ12	- 6ξ13	- 2ξ23	+3x12	+3x13	+ x23	-2x21	- x31	- x32	<=	-7
(304)	-6ξ01	- 4ξ12	- 6ξ13	- 2ξ23	+2x12	+3x13	+ x23	-3x21	- x31	- x32	<=	-7
(305)	-2ξ01		- ξ13			+ x13		- x21	- x31	- x32	<=	-2
(306)	-3ξ01	- ξ12			+ x12	+ x13		- x21	- x31	- x32	<=	-2
(307)	-2ξ01	- 3ξ12	- ξ13		+ x12	+ x13			- x31	- x32	<=	-2
(308)	-3ξ01	- ξ12	- ξ13		+ x12	+ x13	- x23	- x21	- x31	- x32	<=	-3
(309)			- ξ13			+ x13			- x31	- x32	<=	0
(310)	-2ξ01		- ξ13	- 3ξ23		+ x13	+ x23	- x21	- x31		<=	-2
(311)	-2ξ01	- 3ξ12	- ξ13	- 3ξ23	+ x12	+ x13	+ x23		- x31		<=	-2
(312)	-3ξ01	- ξ12	- ξ13	- 6ξ23	+ x12	+ x13	+ x23	- x21	- x31	+ x32	<=	-3
(313)	-3ξ01	- ξ12		- 3ξ23	+ x12	+ x13	+ x23	- x21	- x31		<=	-2
(314)			- ξ13	- 3ξ23		+ x13	+ x23		- x31		<=	0
(315)	-3ξ01	- 2ξ12	- 4ξ13	- 3ξ23	+ x12	+ x13	- x23	- x21	- x31	+ x32	<=	-7
(316)	-3ξ01		- 2ξ13	- ξ23	+ x12	+ x13	- x23	-2x21	- x31	+ x32	<=	-3
(317)	-3ξ01				+ x12	+ x13		- x21	- x31		<=	-1
(318)	-3ξ01		- ξ13		+ x12	+ x13	- x23	- x21	- x31		<=	-2
(319)	- ξ01	- ξ12	- 3ξ13	- ξ23		+ x13			- x31		<=	-3
(320)	-3ξ01	- 3ξ12	- 2ξ13	- ξ23	+2x12	+ x13	- x23	- x21	- x31	+ x32	<=	-3
(321)	-3ξ01	- 2ξ12	- 6ξ13	- 4ξ23	+ x12	+2x13	- x23	- x21	-2x31	+ x32	<=	-8
(322)	-2ξ01	- ξ12	- 3ξ13	- 3ξ23		+ x13	- x23	- x21	- x31	+ x32	<=	-5
(323)	-2ξ01	- ξ12	- 3ξ13	- 2ξ23	+ x12	+ x13	- x23	- x21	- x31	+ x32	<=	-4
(324)	-2ξ01	- 4ξ12	- 3ξ13	- 3ξ23	+ x12	+ x13	- x23		- x31	+ x32	<=	-5
(325)	-3ξ01		- 2ξ13			+ x13		- x21	- x31		<=	-3
(326)	-6ξ01		- 4ξ13	- 2ξ23	+ x12	+2x13	- x23	-3x21	-3x31	+ x32	<=	-7
(327)	-5ξ01	- ξ12	- 5ξ13	- 3ξ23	+ x12	+2x13	- x23	-2x21	-2x31	+ x32	<=	-8
(328)	-2ξ01		- 2ξ13	- ξ23		+ x13		- x21	- x31		<=	-3
(329)	-3ξ01	- 3ξ12	- 2ξ13		+ x12	+ x13			- x31		<=	-3
(330)	-6ξ01	- 6ξ12	- 4ξ13	- 2ξ23	+3x12	+2x13	- x23	- x21	-3x31	+ x32	<=	-7
(331)	-5ξ01	- 4ξ12	- 5ξ13	- 3ξ23	+2x12	+2x13	- x23	- x21	-2x31	+ x32	<=	-8
(332)		- ξ12	- 3ξ13	- 2ξ23		+ x13			- x31		<=	-3
(333)	-2ξ01	- 3ξ12	- 2ξ13	- ξ23	+ x12	+ x13			- x31		<=	-3
(334)	-3ξ01		- ξ13	- 3ξ23	+ x12	+ x13		- x21	- x31	+ x32	<=	-2
(335)	-6ξ01	- 6ξ12	- 7ξ13	- 2ξ23	+3x12	+3x13	- x23	- x21	-2x31	+ x32	<=	-7
(336)	-6ξ01		- 7ξ13	- 2ξ23	+ x12	+3x13	- x23	-3x21	-2x31	+ x32	<=	-7
(337)		+ ξ12									<=	1
(338)	+ ξ02										<=	1
(339)	- ξ01 - ξ03		- ξ13								<=	-2
(340)	- ξ03		- ξ13	- ξ23							<=	-2
(341)	- ξ03	- ξ12	- 3ξ13	- ξ23		- x13			+ x31		<=	-3
(342)	- ξ03	- ξ12	- ξ13	- 3ξ23			+ x23			+ x32	<=	-3
(343)	-3ξ03		- 2ξ13			- x13	- x23		+ x31		<=	-3
(344)	-3ξ03			- 2ξ23		- x13	- x23			+ x32	<=	-3
(345)	-3ξ03		- 3ξ13	- 2ξ23			- x23		+ x31	+ x32	<=	-3
(346)	-2ξ01 -2ξ03		- 2ξ13	- 2ξ23	+ x12		- x23	- x21	- x31	+ x32	<=	-5
(347)	- ξ01 -3ξ03		- ξ13	- 2ξ23		- x13	- x23	- x21	- x31	+ x32	<=	-5
(348)	-2ξ01 -3ξ03	- ξ12	- 2ξ13	- 2ξ23	+ x12	- x13	-2x23	- x21	- x31	+ x32	<=	-7
(349)	- ξ01 -3ξ03		- 7ξ13	- 2ξ23		+ x13	- x23	- x21	+ x31	+ x32	<=	-5
(350)	-2ξ01 -2ξ03		- 5ξ13	- 2ξ23	+ x12	+ x13	- x23	- x21		+ x32	<=	-5

(351)	-2ξ01	-3ξ03	- ξ12	- 8ξ13	- 2ξ23	+ x12	+ x13	-2x23	- x21	+ x31	+ x32	<=	-7
(352)	-3ξ01	- ξ03	- ξ12	- 4ξ13	- 3ξ23	+ x12	+ x13	- x23	- x21	- x31	+ x32	<=	-7
(353)		-3ξ03		- 2ξ13	- 3ξ23		- x13			+ x31	+ x32	<=	-3
(354)	-2ξ01	-3ξ03	- ξ12	- 8ξ13	- 5ξ23	+ x12	+ x13	- x23	- x21	+ x31	+2x32	<=	-7
(355)	-2ξ01	-3ξ03	- ξ12	- 2ξ13	- 5ξ23	+ x12	- x13	- x23	- x21	- x31	+2x32	<=	-7
(356)	- ξ01	-3ξ03	- 3ξ12	- 7ξ13	- 2ξ23	+ x12	+ x13	- x23		+ x31	+ x32	<=	-5
(357)	- ξ01	-3ξ03	- 3ξ12	- ξ13	- 2ξ23	+ x12	- x13	- x23		- x31	+ x32	<=	-5
(358)	- ξ02		- 2ξ12	- ξ13		- x12			+ x21		- x32	<=	-2
(359)	- ξ01 - ξ02	-2ξ03	- ξ12	- ξ13		- x12	-2x13	- x23	+ x21	+ x31	- x32	<=	-4
(360)	- ξ01 - ξ02	-2ξ03	- ξ12	- 4ξ13		- x12	- x13	- x23	+ x21	+2x31	- x32	<=	-4
(361)	-2ξ01 -2ξ02	- ξ03	- ξ12	- 2ξ13		- x12	- x13		+ x21	+ x31	- x32	<=	-5
(362)	- ξ01 -2ξ02	- ξ03		- ξ13		-2x12	- x13		+ x21	+ x31	- x32	<=	-3
(363)	-2ξ02					- x12	- x13		+ x21		- x32	<=	-1
(364)	- ξ01 -2ξ02	- ξ03	- ξ12	- ξ13		- x12	- x13		+ x21		- x32	<=	-4
(365)	-2ξ01 -4ξ02	-2ξ03	- ξ12	- 2ξ13		-2x12	-2x13		+2x21	+ x31	-2x32	<=	-7
(366)	-2ξ02	- ξ03	- ξ12	- ξ13	- ξ23	- x12	- x13		+ x21		- x32	<=	-4
(367)	- ξ01 - ξ02	-3ξ03		- 2ξ13		- x12	- x13	- x23		+ x31	- x32	<=	-5
(368)	-3ξ01 -2ξ02	-3ξ03	- 2ξ12	- 3ξ13		- x12	- x13	- x23	+ x21	+ x31	- x32	<=	-9
(369)	- ξ01 -2ξ02	- ξ03	- 2ξ12	- 2ξ13		- x12			+ x21		- x32	<=	-5
(370)	-2ξ01 -2ξ02	-2ξ03	- 2ξ12	- 2ξ13		- x12	- x13	- x23	+ x21		- x32	<=	-7
(371)	-2ξ01 -2ξ02	-3ξ03	- ξ12	- 2ξ13		- x12	-2x13	- x23	+ x21	+ x31	- x32	<=	-7
(372)	-4ξ01 -4ξ02	-4ξ03	- 2ξ12	- 4ξ13		-2x12	-2x13	- x23	+2x21	+2x31	-2x32	<=	-12
(373)	-3ξ01 -3ξ02	-5ξ03	- 2ξ12	- 4ξ13		-2x12	-2x13	-2x23	+ x21	+2x31	-2x32	<=	-12
(374)		+ ξ03										<=	1
(375)	- ξ01		- ξ12	- ξ13								<=	-2
(376)	-2ξ01 -2ξ02	-5ξ03	- 2ξ12	- 4ξ13		- x12	-2x13	-2x23	+ x21	+2x31	- x32	<=	-10
(377)	-2ξ02			- 3ξ13		- x12			+ x21	+ x31	- x32	<=	-1
(378)	- ξ01 -2ξ02	- ξ03	- ξ12	- 4ξ13		- x12			+ x21	+ x31	- x32	<=	-4
(379)	-2ξ01 -2ξ02	-2ξ03	- 2ξ12	- 5ξ13		- x12		- x23	+ x21	+ x31	- x32	<=	-7
(380)	-2ξ02	- ξ03	- ξ12	- 4ξ13	- ξ23	- x12			+ x21	+ x31	- x32	<=	-4
(381)	-2ξ02	-2ξ03	- 2ξ12	- 5ξ13	- 2ξ23	- x12		- x23	+ x21	+ x31	- x32	<=	-7
(382)	-2ξ01 -2ξ02	-3ξ03	- ξ12	- 5ξ13		- x12	- x13	- x23	+ x21	+2x31	- x32	<=	-7
(383)	-2ξ01 -4ξ02	-2ξ03	- ξ12	- 5ξ13		-2x12	- x13		+2x21	+2x31	-2x32	<=	-7
(384)	-2ξ01 -2ξ02	-2ξ03		- 2ξ13		- x12	- x13		+ x21	+ x31	- x32	<=	-5
(385)	- ξ01 - ξ02		- ξ12	- 2ξ13		- x12	- x13			+ x31	- x32	<=	-3
(386)	- ξ01 - ξ02	-2ξ03	- ξ12	- 2ξ13		- x12	- x13	- x23		+ x31	- x32	<=	-5
(387)	-2ξ01 -2ξ02		- 2ξ12	- 2ξ13		- x12	- x13		+ x21	+ x31	-2x32	<=	-5
(388)	-2ξ02		- 2ξ12	- ξ13		- x12			+ x21		-2x32	<=	-3
(389)	-6ξ02		- 4ξ12	- 2ξ13		-3x12	- x13		+3x21	+ x31	-6x32	<=	-7
(390)	- ξ01 -2ξ02	- ξ03	- 2ξ12	- ξ13		- x12	- x13	- x23	+ x21		-2x32	<=	-5
(391)	-2ξ02	- ξ03	- 2ξ12	- ξ13	- ξ23	- x12	- x13	- x23	+ x21		-2x32	<=	-5
(392)	- ξ01 -2ξ02	- ξ03	- 2ξ12	- 4ξ13		- x12		- x23	+ x21	+ x31	-2x32	<=	-5
(393)	-2ξ02	- ξ03	- 2ξ12	- 4ξ13	- ξ23	- x12		- x23	+ x21	+ x31	-2x32	<=	-5
(394)	- ξ02		- 2ξ12	- ξ13	- 3ξ23	- x12		+ x23	+ x21			<=	-2
(395)	- ξ01 - ξ02	-2ξ03	- ξ12	- ξ13	- 6ξ23	- x12	-2x13	+ x23	+ x21	+ x31	+ x32	<=	-4
(396)	- ξ01 - ξ02	-2ξ03	- ξ12	- 4ξ13	- 6ξ23	- x12	- x13	+ x23	+ x21	+2x31	+ x32	<=	-4
(397)	-2ξ01 -2ξ02	- ξ03	- ξ12	- 2ξ13	- 3ξ23	- x12	- x13	+ x23	+ x21	+ x31		<=	-5
(398)	- ξ01 -2ξ02	- ξ03		- ξ13	- 3ξ23	-2x12	- x13	+ x23	+ x21	+ x31		<=	-3
(399)	-2ξ02				- 3ξ23	- x12	- x13	+ x23	+ x21			<=	-1
(400)	- ξ01 -2ξ02	- ξ03	- ξ12	- ξ13	- 3ξ23	- x12	- x13	+ x23	+ x21			<=	-4

(401)	- ξ01	- ξ02	-3ξ03		- 2ξ13	- 6ξ23	- x12	- x13	+ x23		+ x31	+ x32	<=	-5
(402)		-2ξ02	- ξ03	- ξ12	- ξ13	- 4ξ23	- x12	- x13	+ x23	+ x21			<=	-4
(403)	-2ξ01	-4ξ02	-2ξ03	- ξ12	- 2ξ13	- 6ξ23	-2x12	-2x13	+2x23	+2x21	+ x31		<=	-7
(404)	-3ξ01	-2ξ02	-3ξ03	- 2ξ12	- 3ξ13	- 6ξ23	- x12	- x13	+ x23	+ x21	+ x31	+ x32	<=	-9
(405)	- ξ01	-2ξ02	- ξ03	- 2ξ12	- 2ξ13	- 3ξ23	- x12		+ x23	+ x21			<=	-5
(406)	-2ξ01	-2ξ02	-2ξ03	- 2ξ12	- 2ξ13	- 6ξ23	- x12	- x13	+ x23	+ x21		+ x32	<=	-7
(407)	-2ξ01	-2ξ02	-3ξ03	- ξ12	- 2ξ13	- 6ξ23	- x12	-2x13	+ x23	+ x21	+ x31	+ x32	<=	-7
(408)	-3ξ01	-3ξ02	-5ξ03	- 2ξ12	- 4ξ13	-12ξ23	-2x12	-2x13	+2x23	+ x21	+2x31	+2x32	<=	-12
(409)	-2ξ01	-2ξ02	-3ξ03	- 2ξ12	- 3ξ13	- 7ξ23	- x12	- x13	+ x23	+ x21	+ x31	+ x32	<=	-9
(410)		-2ξ02	-2ξ03	- 2ξ12	- 2ξ13	- 8ξ23	- x12	- x13	+ x23	+ x21		+ x32	<=	-7
(411)	-4ξ01	-4ξ02	-4ξ03	- 2ξ12	- 4ξ13	- 9ξ23	-2x12	-2x13	+2x23	+2x21	+2x31	+ x32	<=	-12
(412)	-2ξ01	-2ξ02	-5ξ03	- 2ξ12	- 4ξ13	- 9ξ23	- x12	-2x13	+ x23	+ x21	+2x31	+2x32	<=	-10
(413)		-2ξ02			- 3ξ13	- 3ξ23	- x12		+ x23	+ x21	+ x31		<=	-1
(414)	- ξ01	-2ξ02	- ξ03	- ξ12	- 4ξ13	- 3ξ23	- x12		+ x23	+ x21	+ x31		<=	-4
(415)		-2ξ02	- ξ03	- ξ12	- 4ξ13	- 4ξ23	- x12		+ x23	+ x21	+ x31		<=	-4
(416)		-2ξ02	-2ξ03	- 2ξ12	- 5ξ13	- 8ξ23	- x12		+ x23	+ x21	+ x31	+ x32	<=	-7
(417)	-2ξ01	-2ξ02	-2ξ03	- 2ξ12	- 5ξ13	- 6ξ23	- x12		+ x23	+ x21	+ x31	+ x32	<=	-7
(418)	-2ξ01	-2ξ02	-3ξ03	- ξ12	- 5ξ13	- 6ξ23	- x12	- x13	+ x23	+ x21	+2x31	+ x32	<=	-7
(419)	-2ξ01	-4ξ02	-2ξ03	- ξ12	- 5ξ13	- 6ξ23	-2x12	- x13	+2x23	+2x21	+2x31		<=	-7
(420)	-2ξ01	-2ξ02	-2ξ03		- 2ξ13	- 3ξ23	- x12	- x13	+ x23	+ x21	+ x31		<=	-5
(421)	- ξ01	- ξ02		- ξ12	- 2ξ13	- 3ξ23	- x12	- x13	+ x23		+ x31		<=	-3
(422)	- ξ01	- ξ02	-2ξ03	- ξ12	- 2ξ13	- 6ξ23	- x12	- x13	+ x23		+ x31	+ x32	<=	-5
(423)	-2ξ01	-2ξ02		- 2ξ12	- 2ξ13	- 6ξ23	- x12	- x13	+2x23	+ x21	+ x31		<=	-5
(424)		-2ξ02		- 2ξ12	- ξ13	- 6ξ23	- x12		+2x23	+ x21			<=	-3
(425)		-6ξ02		- 4ξ12	- 2ξ13	-18ξ23	-3x12	- x13	+6x23	+3x21	+ x31		<=	-7
(426)		-2ξ02	- ξ03	- 2ξ12	- ξ13	-10ξ23	- x12	- x13	+2x23	+ x21		+ x32	<=	-5
(427)	- ξ01	-2ξ02	- ξ03	- 2ξ12	- ξ13	- 9ξ23	- x12	- x13	+2x23	+ x21		+ x32	<=	-5
(428)		-2ξ02	- ξ03	- 2ξ12	- 4ξ13	-10ξ23	- x12		+2x23	+ x21	+ x31	+ x32	<=	-5
(429)	- ξ01	-2ξ02	- ξ03	- 2ξ12	- 4ξ13	- 9ξ23	- x12		+2x23	+ x21	+ x31	+ x32	<=	-5
(430)	- ξ01	-2ξ02	- ξ03	- 3ξ12	- ξ13		- x12	- x13		+2x21	+ x31	- x32	<=	-3
(431)	-3ξ01	-3ξ02	-5ξ03	- 5ξ12	- 4ξ13		- x12	-2x13	-2x23	+2x21	+2x31	-2x32	<=	-12
(432)	- ξ01	- ξ02	-3ξ03	- 3ξ12	- 2ξ13			- x13	- x23	+ x21	+ x31	- x32	<=	-5
(433)	- ξ01	- ξ02	-2ξ03	- 4ξ12	- 2ξ13			- x13	- x23	+ x21	+ x31	- x32	<=	-5
(434)	- ξ01	-2ξ02	- ξ03	- 3ξ12	- ξ13	- 3ξ23	- x12	- x13	+ x23	+2x21	+ x31		<=	-3
(435)	- ξ01	- ξ02		- 4ξ12	- 2ξ13			- x13		+ x21	+ x31	- x32	<=	-3
(436)	-3ξ01	-3ξ02	-5ξ03	- 5ξ12	- 4ξ13	-12ξ23	- x12	-2x13	+2x23	+2x21	+2x31	+2x32	<=	-12
(437)	- ξ01	- ξ02	-3ξ03	- 3ξ12	- 2ξ13	- 6ξ23		- x13	+ x23	+ x21	+ x31	+ x32	<=	-5
(438)	- ξ01	- ξ02	-2ξ03	- 4ξ12	- 2ξ13	- 6ξ23		- x13	+ x23	+ x21	+ x31	+ x32	<=	-5
(439)	- ξ01	- ξ02		- 4ξ12	- 2ξ13	- 3ξ23		- x13	+ x23	+ x21	+ x31		<=	-3
(440)		-2ξ02		- ξ12		- ξ23	- x12	- x13	+ x23	+ x21	- x31	- x32	<=	-2
(441)		-2ξ02		- ξ12			- x12		+ x23	+ x21	- x31	- x32	<=	-1
(442)		-2ξ02				- ξ23	- x12	- x13	+ x23	+ x21		- x32	<=	-1
(443)		- ξ02			- ξ13	- 2ξ23	- x12		+ x23			- x32	<=	-2
(444)		-2ξ02		- ξ12	- 6ξ13	- ξ23	- x12	+ x13	+ x23	+ x21	+ x31	- x32	<=	-2
(445)		-2ξ02		- ξ12	- 2ξ13	- ξ23	- x12	- x13	+ x23	+ x21	+ x31	- x32	<=	-2
(446)		-2ξ02		- ξ12	- 3ξ13		- x12	+ x13	+ x23	+ x21		- x32	<=	-1
(447)		-2ξ02			- 3ξ13	- ξ23	- x12		+ x23	+ x21	+ x31	- x32	<=	-1
(448)	- ξ01	-2ξ02	- ξ03		- ξ13		- x12	+ x13	+ x23		- x31	-2x32	<=	-3
(449)	-2ξ01	-4ξ02	-2ξ03		- 2ξ13	- ξ23	-2x12	+ x13	+2x23		-2x31	-2x32	<=	-7
(450)	- ξ01	-3ξ02	- ξ03	- ξ12	- ξ13	- ξ23	- x12	- x13	+ x23	+ x21	- x31	- x32	<=	-5

(451)	- ξ01	-3ξ02	- ξ03	- ξ12	- ξ13		- x12		+ x23	+ x21	- x31	- x32	<=	-4
(452)	- ξ01	-3ξ02	- ξ03		- ξ13	- ξ23	- x12	- x13	+ x23	+ x21		- x32	<=	-4
(453)	- ξ01	-5ξ02	- ξ03	- ξ12	- ξ13	- ξ23	-2x12	-2x13	+ x23	+2x21		-2x32	<=	-6
(454)	- ξ01	-2ξ02	- ξ03		- ξ13	- ξ23	- x12		+ x23		- x31	- x32	<=	-4
(455)		-3ξ02	- ξ03	- ξ12	- ξ13	- ξ23	- x12		+ x23	+ x21	- x31	- x32	<=	-4
(456)	- ξ01	-5ξ02	- ξ03	- ξ12	- ξ13	- ξ23	-2x12		+2x23	+ x21	-2x31	-2x32	<=	-6
(457)	- ξ01	-2ξ02	- ξ03		- 2ξ13	- 2ξ23	- x12		+ x23			- x32	<=	-5
(458)	- ξ01	-3ξ02	- ξ03	- ξ12	- 7ξ13	- ξ23	- x12	+ x13	+ x23	+ x21	+ x31	- x32	<=	-5
(459)	- ξ01	-3ξ02	- ξ03	- ξ12	- 3ξ13	- ξ23	- x12	- x13	+ x23	+ x21	+ x31	- x32	<=	-5
(460)	- ξ01	-2ξ02	- ξ03		- 4ξ13	- ξ23	- x12	+ x13	+ x23			- x32	<=	-4
(461)	- ξ01	-3ξ02	- ξ03	- ξ12	- 4ξ13		- x12	+ x13	+ x23	+ x21		- x32	<=	-4
(462)	- ξ01	-3ξ02	- ξ03		- 4ξ13	- ξ23	- x12		+ x23	+ x21	+ x31	- x32	<=	-4
(463)		-3ξ02	- ξ03	- ξ12	- 4ξ13	- ξ23	- x12	+ x13	+ x23	+ x21		- x32	<=	-4
(464)	- ξ01	-5ξ02	- ξ03	- ξ12	- 7ξ13	- ξ23	-2x12		+ x23	+2x21	+2x31	-2x32	<=	-6
(465)	- ξ01	-5ξ02	- ξ03	- ξ12	- 7ξ13	- ξ23	-2x12	+2x13	+2x23	+ x21		-2x32	<=	-6
(466)	-2ξ01	-4ξ02	-2ξ03		- 5ξ13	- ξ23	-2x12	+2x13	+2x23		- x31	-2x32	<=	-7
(467)	- ξ01	-2ξ02	-2ξ03		- 2ξ13	- ξ23	- x12	+ x13	+ x23		- x31	- x32	<=	-5
(468)	-2ξ01	-2ξ02	-2ξ03		- 2ξ13		- x12	+ x13	+ x23		- x31	- x32	<=	-5
(469)	-2ξ01	-3ξ02	-2ξ03	- 2ξ12	- 3ξ13		- x12	+ x13	+ x23	+ x21	- x31	-2x32	<=	-7
(470)	-2ξ01	-3ξ02	-2ξ03		- 3ξ13	- 2ξ23	-2x12	- x13	+ x23	+ x21	+ x31	- x32	<=	-7
(471)		-3ξ02	-2ξ03	- 2ξ12	- 3ξ13	- 2ξ23	- x12	+ x13	+ x23	+ x21	- x31	-2x32	<=	-7
(472)	-3ξ01	-3ξ02	-3ξ03	- 3ξ12	- 5ξ13	- ξ23	- x12	+ x13	+ x23	+ x21	- x31	- x32	<=	-11
(473)	-3ξ01	-3ξ02	-3ξ03	- ξ12	- 5ξ13	- 3ξ23	- x12	- x13	+ x23	+ x21	+ x31	- x32	<=	-11
(474)		-3ξ02	-3ξ03	- 3ξ12	- 5ξ13	- 4ξ23	- x12	+ x13	+ x23	+ x21	- x31	- x32	<=	-11
(475)	- ξ01	-3ξ02	- ξ03	- ξ12	- 3ξ13	- ξ23	- x12	+ x13	+ x23	+ x21	- x31	- x32	<=	-5
(476)		-3ξ02	- ξ03	- ξ12	- 3ξ13	- 2ξ23	- x12	+ x13	+ x23	+ x21	- x31	- x32	<=	-5
(477)		-2ξ02		- ξ12	- 3ξ13	- ξ23	- x12	- x13	+ x23		+ x31	- x32	<=	-3
(478)		-3ξ02		- 2ξ12	- 4ξ13	- 3ξ23	- x12	- x13	+2x23	+ x21	+ x31	-2x32	<=	-5
(479)	- ξ01	- ξ02	- ξ03		- 2ξ13		- x12	+ x13			- x31	- x32	<=	-3
(480)		- ξ02	- ξ03		- 2ξ13	- ξ23	- x12	+ x13			- x31	- x32	<=	-3
(481)	-2ξ01	-2ξ02	-2ξ03	- 3ξ12	- 5ξ13	- ξ23	- x12	+ x13		+ x21	- x31	- x32	<=	-9
(482)	- ξ01	-2ξ02		- ξ12	- ξ13		- x12	+ x13	+ x23		- x31	-3x32	<=	-3
(483)		-2ξ02		- ξ12	- 3ξ13	- ξ23	- x12	+ x13		+ x21	- x31	- x32	<=	-3
(484)	- ξ01	-2ξ02		- ξ12	- 2ξ13	- 2ξ23	- x12		+ x23			- x32	<=	-5
(485)	- ξ01	-3ξ02		- 2ξ12	- 7ξ13	- ξ23	- x12	+ x13	+ x23	+ x21	+ x31	-2x32	<=	-5
(486)	- ξ01	-3ξ02		- 2ξ12	- ξ13	- ξ23	- x12	- x13	+ x23	+ x21	- x31	-2x32	<=	-5
(487)	-2ξ01	-6ξ02		- 4ξ12	- 2ξ13	- 3ξ23	-2x12	-2x13	+3x23	+2x21	-3x31	-3x32	<=	-10
(488)	-3ξ01	-4ξ02		- 3ξ12	- 4ξ13	- 2ξ23	-2x12	- x13	+ x23	+ x21	+ x31	-2x32	<=	-10
(489)		-2ξ02			- ξ13	- 2ξ23	-2x12		+ x23			- x32	<=	-3
(490)	-2ξ01	-6ξ02		- 4ξ12	-17ξ13	- 3ξ23	-2x12	+3x13	+3x23	+2x21	+2x31	-3x32	<=	-10
(491)	-3ξ01	-6ξ02	-3ξ03	- 2ξ12	- 4ξ13		-3x12	-2x13		+3x21	+2x31	-3x32	<=	-11
(492)	- ξ01	-2ξ02	- ξ03		- ξ13	- 3ξ23	- x12	+ x13	+2x23		- x31	- x32	<=	-3
(493)	- ξ01	-5ξ02	- ξ03	- ξ12	- 7ξ13	- 4ξ23	-2x12		+2x23	+2x21	+2x31	- x32	<=	-6
(494)	- ξ01	-5ξ02	- ξ03	- ξ12	- ξ13	- 4ξ23	-2x12	-2x13	+2x23	+2x21		- x32	<=	-6
(495)	-2ξ01	-3ξ02	-2ξ03	- 2ξ12	- 3ξ13	- 3ξ23	- x12	+ x13	+2x23	+ x21	- x31	- x32	<=	-7
(496)		-3ξ02	-2ξ03	- 2ξ12	- 3ξ13	- 5ξ23	- x12	+ x13	+2x23	+ x21	- x31	- x32	<=	-7
(497)	- ξ01	- ξ02	- ξ03		- 2ξ13	- 3ξ23	- x12	+ x13	+ x23		- x31		<=	-3
(498)		- ξ02	- ξ03		- 2ξ13	- 4ξ23	- x12	+ x13	+ x23		- x31		<=	-3
(499)	-2ξ01	-2ξ02	-2ξ03	- 3ξ12	- 5ξ13	- 4ξ23	- x12	+ x13	+ x23	+ x21	- x31		<=	-9
(500)	- ξ01	-2ξ02		- ξ12	- ξ13	- 6ξ23	- x12	+ x13	+3x23		- x31	- x32	<=	-3

(501)		-2ξ02		- ξ12	- 3ξ13	- 4ξ23	- x12	+ x13	+ x23	+ x21	- x31	<=	-3
(502)	- ξ01	-3ξ02		- 2ξ12	- 7ξ13	- 4ξ23	- x12	+ x13	+2x23	+ x21	+ x31	- x32	<=	-5
(503)	- ξ01	-3ξ02		- 2ξ12	- ξ13	- 4ξ23	- x12	- x13	+2x23	+ x21	- x31	- x32	<=	-5
(504)	-3ξ01	-4ξ02		- 3ξ12	- 4ξ13	- 5ξ23	-2x12	- x13	+2x23	+ x21	+ x31	- x32	<=	-10
(505)	-3ξ01	-6ξ02	-3ξ03	- 2ξ12	- 4ξ13	- 9ξ23	-3x12	-2x13	+3x23	+3x21	+2x31		<=	-11
(506)		- ξ02		- 3ξ12	- ξ13	- 2ξ23			+ x23	+ x21		- x32	<=	-2
(507)	- ξ01	-2ξ02	- ξ03	- 3ξ12	- ξ13			+ x13	+ x23	+ x21	- x31	-2x32	<=	-3
(508)	-2ξ01	-4ξ02	-2ξ03	- 6ξ12	- 2ξ13	- ξ23		+ x13	+2x23	+2x21	-2x31	-2x32	<=	-7
(509)	- ξ01	-2ξ02	- ξ03	- 3ξ12	- ξ13	- ξ23			+ x23	+ x21	- x31	- x32	<=	-4
(510)	- ξ01	-5ξ02	- ξ03	- 4ξ12	- ξ13	- ξ23	- x12		+2x23	+2x21	-2x31	-2x32	<=	-6
(511)	- ξ01	-2ξ02	- ξ03	- 3ξ12	- 2ξ13	- 2ξ23			+ x23	+ x21		- x32	<=	-5
(512)	- ξ01	-5ξ02	- ξ03	- 4ξ12	- 7ξ13	- ξ23	- x12	+2x13	+2x23	+2x21		-2x32	<=	-6
(513)	- ξ01	-2ξ02	- ξ03	- 3ξ12	- 4ξ13	- ξ23		+ x13	+ x23	+ x21		- x32	<=	-4
(514)	-2ξ01	-4ξ02	-2ξ03	- 6ξ12	- 5ξ13	- ξ23		+2x13	+2x23	+2x21	- x31	-2x32	<=	-7
(515)	- ξ01	-2ξ02	-2ξ03	- 3ξ12	- 2ξ13	- ξ23		+ x13	+ x23	+ x21	- x31	- x32	<=	-5
(516)	-2ξ01	-2ξ02	-2ξ03	- 3ξ12	- 2ξ13			+ x13	+ x23	+ x21	- x31	- x32	<=	-5
(517)	-2ξ01	-3ξ02	-2ξ03	- 3ξ12	- 3ξ13	- 2ξ23	- x12	- x13	+ x23	+2x21	+ x31	- x32	<=	-7
(518)	- ξ01	- ξ02	- ξ03	- 3ξ12	- 2ξ13			+ x13		+ x21	- x31	- x32	<=	-3
(519)		- ξ02	- ξ03	- 3ξ12	- 2ξ13	- ξ23		+ x13		+ x21	- x31	- x32	<=	-3
(520)	- ξ01	-2ξ02		- 4ξ12	- ξ13			+ x13	+ x23	+ x21	- x31	-3x32	<=	-3
(521)		-2ξ02		- 4ξ12	- 3ξ13	- ξ23		- x13	+ x23	+ x21	+ x31	- x32	<=	-3
(522)	- ξ01	-2ξ02		- 4ξ12	- 2ξ13	- 2ξ23			+ x23	+ x21		- x32	<=	-5
(523)	-3ξ01	-4ξ02		- 6ξ12	- 4ξ13	- 2ξ23	- x12	- x13	+ x23	+2x21	+ x31	-2x32	<=	-10
(524)		-2ξ02		- 6ξ12	- ξ13	- 2ξ23			+ x23	+2x21		- x32	<=	-3
(525)	- ξ01	- ξ02	- ξ03	- 3ξ12	- 2ξ13	- 3ξ23		+ x13	+ x23	+ x21	- x31		<=	-3
(526)	- ξ01	-2ξ02	- ξ03	- 3ξ12	- ξ13	- 3ξ23		+ x13	+2x23	+ x21	- x31	- x32	<=	-3
(527)	- ξ01	-2ξ02		- 4ξ12	- ξ13	- 6ξ23		+ x13	+3x23	+ x21	- x31	- x32	<=	-3
(528)		- ξ02	- ξ03	- 3ξ12	- 2ξ13	- 4ξ23		+ x13	+ x23	+ x21	- x31		<=	-3
(529)	-3ξ01	-4ξ02		- 6ξ12	- 4ξ13	- 5ξ23	- x12	- x13	+2x23	+2x21	+ x31	- x32	<=	-10
(530)	-2ξ01		- ξ03		- ξ13	- ξ23	+ x12		- x23	-2x21	- x31	+ x32	<=	-3
(531)	-2ξ01		- ξ03	- 3ξ12	- ξ13	- ξ23	+2x12		- x23	- x21	- x31	+ x32	<=	-3
(532)	-2ξ01		- ξ03		- 4ξ13	- ξ23	+ x12	+ x13	- x23	-2x21		+ x32	<=	-3
(533)	-2ξ01		- ξ03	- 3ξ12	- 4ξ13	- ξ23	+2x12	+ x13	- x23	- x21		+ x32	<=	-3
(534)	-3ξ01	- ξ02	- ξ03		- 2ξ13		- x12	+ x13		- x21	- x31	- x32	<=	-5
(535)	- ξ01	- ξ02	- ξ03		- 2ξ13		- x12	- x13			+ x31	- x32	<=	-3
(536)	-3ξ01	- ξ02	- ξ03		- 2ξ13	- 3ξ23	- x12	+ x13	+ x23	- x21	- x31		<=	-5
(537)	- ξ01	- ξ02	- ξ03		- 2ξ13	- 3ξ23	- x12	- x13	+ x23		+ x31		<=	-3
(538)	-3ξ01	- ξ02	- ξ03	- 6ξ12	- 2ξ13		+ x12	+ x13		+ x21	- x31	- x32	<=	-5
(539)	- ξ01	- ξ02	- ξ03	- 3ξ12	- 2ξ13			- x13		+ x21	+ x31	- x32	<=	-3
(540)	-3ξ01	- ξ02	- ξ03	- 6ξ12	- 2ξ13	- 3ξ23	+ x12	+ x13	+ x23	+ x21	- x31		<=	-5
(541)	- ξ01		- ξ03		- ξ13	- 2ξ23			+ x23	- x21	- x31	- x32	<=	-3
(542)	-2ξ01		- ξ03		- ξ13	- ξ23		+ x13	+ x23	- x21	-2x31	- x32	<=	-3
(543)	- ξ01	- ξ02	- ξ03	- 3ξ12	- 2ξ13	- 3ξ23		- x13	+ x23	+ x21	+ x31		<=	-3
(544)	- ξ01		- ξ03		- 4ξ13	- 2ξ23		+ x13	+ x23	- x21		- x32	<=	-3
(545)	-2ξ01		- ξ03		- 4ξ13	- ξ23		+2x13	+ x23	- x21	- x31	- x32	<=	-3
(546)	-3ξ01	-2ξ02	-2ξ03	- ξ12	- 3ξ13	- 2ξ23	- x12	+ x13	+ x23	- x21	- x31	- x32	<=	-9
(547)	-2ξ01	-2ξ02	-2ξ03		- 2ξ13	- 2ξ23	- x12		+ x23	- x21	- x31	- x32	<=	-7
(548)	-2ξ01	-2ξ02	-2ξ03		- 2ξ13	- 2ξ23	-2x12	+ x13	+ x23	- x21	- x31	- x32	<=	-7
(549)	-2ξ01		-2ξ03		- 2ξ13	- 2ξ23		+ x13	+ x23	- x21	- x31	- x32	<=	-5
(550)	-3ξ01	-2ξ02	-2ξ03		- 2ξ13	- ξ23	- x12	+ x13	+ x23	- x21	-2x31	- x32	<=	-7

(551)	-2ξ01	-2ξ02	-2ξ03		- 5ξ13	- 2ξ23	- x12	+ x13	+ x23	- x21		- x32	<=	-7
(552)	-3ξ01	-2ξ02	-2ξ03		- 5ξ13	- ξ23	- x12	+2x13	+ x23	- x21	- x31	- x32	<=	-7
(553)	-4ξ01	-4ξ02	-4ξ03		- 4ξ13	- 2ξ23	-2x12	+2x13	+2x23	- x21	-2x31	-2x32	<=	-12
(554)	-3ξ01	-2ξ02	-3ξ03		- 3ξ13	- 2ξ23	- x12	+ x13	+ x23	- x21	- x31	- x32	<=	-9
(555)	-5ξ01	-2ξ02	-2ξ03		- 4ξ13	- 2ξ23	- x12	+2x13	+ x23	-2x21	-2x31	- x32	<=	-10
(556)	-5ξ01	-3ξ02	-3ξ03		- 4ξ13	- 2ξ23	-2x12	+2x13	+ x23	-2x21	-2x31	-2x32	<=	-12
(557)	-2ξ01	- ξ02	- ξ03		- 2ξ13	- ξ23	- x12	+ x13		- x21	- x31	- x32	<=	-5
(558)	-3ξ01	-2ξ02		- 3ξ12	- 3ξ13	- 2ξ23	- x12	+ x13	+ x23	- x21	- x31	-3x32	<=	-9
(559)	-3ξ01	-2ξ02		- 2ξ12	- 2ξ13	- ξ23	- x12	+ x13	+ x23	- x21	-2x31	-3x32	<=	-7
(560)	-3ξ01	-2ξ02		- 2ξ12	- 5ξ13	- ξ23	- x12	+2x13	+ x23	- x21	- x31	-3x32	<=	-7
(561)	-3ξ01	-6ξ02	-3ξ03		- 4ξ13	- 2ξ23	-3x12	+2x13	+3x23		-2x31	-3x32	<=	-11
(562)		-2ξ02			- ξ13	- ξ23	- x12		+ x23			- x32	<=	-2
(563)		-4ξ02			- ξ13		-2x12	+ x13	+2x23		- x31	-3x32	<=	-2
(564)	-3ξ01	-6ξ02		- 3ξ12	- 5ξ13	- 4ξ23	-3x12	+ x13	+3x23		- x31	-3x32	<=	-13
(565)		-6ξ02			- 2ξ13	- 4ξ23	-6x12	+ x13	+3x23		- x31	-3x32	<=	-7
(566)		-6ξ02			- 2ξ13	- ξ23	-3x12	+ x13	+3x23		- x31	-3x32	<=	-4
(567)		-3ξ02			- 2ξ13		-2x12	- x13	+ x23		+ x31	-2x32	<=	-2
(568)	-5ξ01	-6ξ02		- 5ξ12	- 7ξ13	- 6ξ23	-3x12	+ x13	+3x23	- x21	- x31	-3x32	<=	-19
(569)	-2ξ01	-2ξ02		- 3ξ12	- 5ξ13	- 3ξ23	- x12	- x13	+ x23		+ x31	- x32	<=	-9
(570)	- ξ01			- ξ12	- ξ13	- 2ξ23			+ x23	- x21	- x31	- x32	<=	-3
(571)	-2ξ01			- ξ12	- ξ13	- ξ23		+ x13	+ x23	- x21	-2x31	- x32	<=	-3
(572)	- ξ01			- ξ12	- 4ξ13	- 2ξ23		+ x13	+ x23	- x21		- x32	<=	-3
(573)	-2ξ01			- ξ12	- 4ξ13	- ξ23		+2x13	+ x23	- x21	- x31	- x32	<=	-3
(574)	-6ξ01				- 4ξ13	- 2ξ23		+3x13	+ x23	-3x21	-3x31	- x32	<=	-7
(575)	-4ξ01				- 2ξ13	- ξ23		+2x13	+ x23	-2x21	-3x31	- x32	<=	-4
(576)	-4ξ01				- 5ξ13	- ξ23		+3x13	+ x23	-2x21	-2x31	- x32	<=	-4
(577)	-2ξ01	-2ξ02	-2ξ03	- ξ12	- 5ξ13	- 3ξ23	- x12	- x13	+ x23		+ x31	- x32	<=	-9
(578)	-5ξ01	-3ξ02	-3ξ03		- 4ξ13	- 5ξ23	-2x12	+2x13	+2x23	-2x21	-2x31	- x32	<=	-12
(579)	-2ξ01	- ξ02	- ξ03		- 2ξ13	- 4ξ23	- x12	+ x13	+ x23	- x21	- x31		<=	-5
(580)	-3ξ01	-2ξ02		- 3ξ12	- 3ξ13	- 8ξ23	- x12	+ x13	+3x23	- x21	- x31	- x32	<=	-9
(581)	-3ξ01	-2ξ02		- 2ξ12	- 2ξ13	- 7ξ23	- x12	+ x13	+3x23	- x21	-2x31	- x32	<=	-7
(582)	-3ξ01	-2ξ02		- 2ξ12	- 5ξ13	- 7ξ23	- x12	+2x13	+3x23	- x21	- x31	- x32	<=	-7
(583)		-3ξ02			- 2ξ13	- 3ξ23	-2x12	- x13	+2x23		+ x31	- x32	<=	-2
(584)		-4ξ02			- ξ13	- 3ξ23	-2x12	+ x13	+3x23		- x31	-2x32	<=	-2
(585)	- ξ01		- ξ03	- 3ξ12	- ξ13	- 2ξ23	+ x12		+ x23		- x31	- x32	<=	-3
(586)	-2ξ01		- ξ03	- 3ξ12	- ξ13	- ξ23	+ x12	+ x13	+ x23		-2x31	- x32	<=	-3
(587)	- ξ01		- ξ03	- 3ξ12	- 4ξ13	- 2ξ23	+ x12	+ x13	+ x23			- x32	<=	-3
(588)	-2ξ01		- ξ03	- 3ξ12	- 4ξ13	- ξ23	+ x12	+2x13	+ x23		- x31	- x32	<=	-3
(589)	-3ξ01	-2ξ02	-2ξ03	- 7ξ12	- 3ξ13	- 2ξ23	+ x12	+ x13	+ x23	+ x21	- x31	- x32	<=	-9
(590)	-2ξ01	-2ξ02	-2ξ03	- 9ξ12	- 2ξ13	- 2ξ23	+ x12	+ x13	+ x23	+2x21	- x31	- x32	<=	-7
(591)	-2ξ01		-2ξ03	- 3ξ12	- 2ξ13	- 2ξ23	+ x12	+ x13	+ x23		- x31	- x32	<=	-5
(592)	-2ξ01	-2ξ02	-2ξ03	- 6ξ12	- 2ξ13	- 2ξ23	+ x12		+ x23	+ x21	- x31	- x32	<=	-7
(593)	-3ξ01	-2ξ02	-2ξ03	- 6ξ12	- 2ξ13	- ξ23	+ x12	+ x13	+ x23	+ x21	-2x31	- x32	<=	-7
(594)	-2ξ01	-2ξ02	-2ξ03	- 6ξ12	- 5ξ13	- 2ξ23	+ x12	+ x13	+ x23	+ x21		- x32	<=	-7
(595)	-3ξ01	-2ξ02	-2ξ03	- 6ξ12	- 5ξ13	- ξ23	+ x12	+2x13	+ x23	+ x21	- x31	- x32	<=	-7
(596)	-4ξ01	-4ξ02	-4ξ03	- 9ξ12	- 4ξ13	- 2ξ23	+ x12	+2x13	+2x23	+2x21	-2x31	-2x32	<=	-12
(597)	-3ξ01	-2ξ02	-3ξ03	- 6ξ12	- 3ξ13	- 2ξ23	+ x12	+ x13	+ x23	+ x21	- x31	- x32	<=	-9
(598)	-5ξ01	-2ξ02	-2ξ03	- 9ξ12	- 4ξ13	- 2ξ23	+2x12	+2x13	+ x23	+ x21	-2x31	- x32	<=	-10
(599)	-5ξ01	-3ξ02	-3ξ03	-12ξ12	- 4ξ13	- 2ξ23	+2x12	+2x13	+ x23	+2x21	-2x31	-2x32	<=	-12
(600)	-2ξ01	- ξ02	- ξ03	- 6ξ12	- 2ξ13	- ξ23	+ x12	+ x13		+ x21	- x31	- x32	<=	-5

(601)	-3ξ01	-2ξ02		- 9ξ12	- 3ξ13	- 2ξ23	+ x12	+ x13	+ x23	+ x21	- x31	-3x32	<= -9
(602)	-3ξ01	-2ξ02		- 8ξ12	- 2ξ13	- ξ23	+ x12	+ x13	+ x23	+ x21	-2x31	-3x32	<= -7
(603)	-3ξ01	-2ξ02		- 8ξ12	- 5ξ13	- ξ23	+ x12	+2x13	+ x23	+ x21	- x31	-3x32	<= -7
(604)	-2ξ01	-2ξ02		- 6ξ12	- 5ξ13	- 3ξ23		- x13	+ x23	+ x21	+ x31	- x32	<= -9
(605)	-3ξ01	-6ξ02	-3ξ03	- 9ξ12	- 4ξ13	- 2ξ23		+2x13	+3x23	+3x21	-2x31	-3x32	<= -11
(606)		-2ξ02		- 3ξ12	- ξ13	- ξ23			+ x23	+ x21		- x32	<= -2
(607)		-4ξ02		- 6ξ12	- ξ13			+ x13	+2x23	+2x21	- x31	-3x32	<= -2
(608)	-3ξ01	-6ξ02		-12ξ12	- 5ξ13	- 4ξ23		+ x13	+3x23	+3x21	- x31	-3x32	<= -13
(609)		-6ξ02		-18ξ12	- 2ξ13	- 4ξ23		+ x13	+3x23	+6x21	- x31	-3x32	<= -7
(610)		-6ξ02		- 9ξ12	- 2ξ13	- ξ23		+ x13	+3x23	+3x21	- x31	-3x32	<= -4
(611)		-3ξ02		- 6ξ12	- 2ξ13			- x13	+ x23	+2x21	+ x31	-2x32	<= -2
(612)	-5ξ01	-6ξ02		-17ξ12	- 7ξ13	- 6ξ23	+ x12	+ x13	+3x23	+3x21	- x31	-3x32	<= -19
(613)	- ξ01			- 4ξ12	- ξ13	- 2ξ23	+ x12		+ x23		- x31	- x32	<= -3
(614)	-2ξ01			- 4ξ12	- ξ13	- ξ23	+ x12	+ x13	+ x23		-2x31	- x32	<= -3
(615)	- ξ01			- 4ξ12	- 4ξ13	- 2ξ23	+ x12	+ x13	+ x23			- x32	<= -3
(616)	-2ξ01			- 4ξ12	- 4ξ13	- ξ23	+ x12	+2x13	+ x23		- x31	- x32	<= -3
(617)	-6ξ01			- 9ξ12	- 4ξ13	- 2ξ23	+3x12	+3x13	+ x23		-3x31	- x32	<= -7
(618)	-4ξ01			- 6ξ12	- 2ξ13	- ξ23	+2x12	+2x13	+ x23		-3x31	- x32	<= -4
(619)	-4ξ01			- 6ξ12	- 5ξ13	- ξ23	+2x12	+3x13	+ x23		-2x31	- x32	<= -4
(620)	-2ξ01	-2ξ02	-2ξ03	- 4ξ12	- 5ξ13	- 3ξ23		- x13	+ x23	+ x21	+ x31	- x32	<= -9
(621)	-5ξ01	-3ξ02	-3ξ03	-12ξ12	- 4ξ13	- 5ξ23	+2x12	+2x13	+2x23	+2x21	-2x31	- x32	<= -12
(622)	-2ξ01	- ξ02	- ξ03	- 6ξ12	- 2ξ13	- 4ξ23	+ x12	+ x13	+ x23	+ x21	- x31		<= -5
(623)	-3ξ01	-2ξ02		- 9ξ12	- 3ξ13	- 8ξ23	+ x12	+ x13	+3x23	+ x21	- x31	- x32	<= -9
(624)	-3ξ01	-2ξ02		- 8ξ12	- 2ξ13	- 7ξ23	+ x12	+ x13	+3x23	+ x21	-2x31	- x32	<= -7
(625)	-3ξ01	-2ξ02		- 8ξ12	- 5ξ13	- 7ξ23	+ x12	+2x13	+3x23	+ x21	- x31	- x32	<= -7
(626)		-3ξ02		- 6ξ12	- 2ξ13	- 3ξ23		- x13	+2x23	+2x21	+ x31	- x32	<= -2
(627)		-4ξ02		- 6ξ12	- ξ13	- 3ξ23		+ x13	+3x23	+2x21	- x31	-2x32	<= -2
(628)	-3ξ01		- ξ03	- 3ξ12	- 2ξ13	- 3ξ23	+ x12	+ x13	+ x23	- x21	- x31	- x32	<= -7
(629)		- ξ02	- ξ03			- ξ23	- x12						<= -2
(630)		-2ξ02	- ξ03		- ξ13	- ξ23	-3x12	- x13		+ x21	+ x31	- x32	<= -3
(631)		-2ξ02	-3ξ03	- ξ12	- 2ξ13	- 2ξ23	-3x12	-2x13	- x23	+ x21	+ x31	- x32	<= -7
(632)		-2ξ02	-3ξ03	- 2ξ12	- 3ξ13	- 3ξ23	-3x12	- x13	- x23	+ x21	+ x31	- x32	<= -9
(633)		-2ξ02	- ξ03		- ξ13	- 4ξ23	-3x12	- x13	+ x23	+ x21	+ x31		<= -3
(634)		-2ξ02	-3ξ03	- ξ12	- 2ξ13	- 8ξ23	-3x12	-2x13	+ x23	+ x21	+ x31	+ x32	<= -7
(635)		-2ξ02	-3ξ03	- 2ξ12	- 3ξ13	- 9ξ23	-3x12	- x13	+ x23	+ x21	+ x31	+ x32	<= -9
(636)		-2ξ02	-3ξ03	- ξ12	- 5ξ13	- 8ξ23	-3x12	- x13	+ x23	+ x21	+2x31	+ x32	<= -7
(637)		-2ξ02	-3ξ03	- ξ12	- 5ξ13	- 2ξ23	-3x12	- x13	- x23	+ x21	+2x31	- x32	<= -7
(638)		- ξ02	- ξ03	- 3ξ12		- ξ23				+ x21			<= -2
(639)		-2ξ02	-3ξ03	- 7ξ12	- 2ξ13	- 2ξ23	- x12	-2x13	- x23	+3x21	+ x31	- x32	<= -7
(640)		-2ξ02	-3ξ03	- 8ξ12	- 3ξ13	- 3ξ23	- x12	- x13	- x23	+3x21	+ x31	- x32	<= -9
(641)		-2ξ02	-3ξ03	- 7ξ12	- 2ξ13	- 8ξ23	- x12	-2x13	+ x23	+3x21	+ x31	+ x32	<= -7
(642)		-2ξ02	-3ξ03	- 8ξ12	- 3ξ13	- 9ξ23	- x12	- x13	+ x23	+3x21	+ x31	+ x32	<= -9
(643)		-2ξ02	- ξ03	- 6ξ12	- ξ13	- ξ23	- x12	- x13		+3x21	+ x31	- x32	<= -3
(644)		-2ξ02	- ξ03	- 6ξ12	- ξ13	- 4ξ23	- x12	- x13	+ x23	+3x21	+ x31		<= -3
(645)		-2ξ02	-3ξ03	- 7ξ12	- 5ξ13	- 8ξ23	- x12	- x13	+ x23	+3x21	+2x31	+ x32	<= -7
(646)		-2ξ02	-3ξ03	- 7ξ12	- 5ξ13	- 2ξ23	- x12	- x13	- x23	+3x21	+2x31	- x32	<= -7
(647)		-2ξ02	-2ξ03	- ξ12		- 2ξ23	- x12	- x13		+ x21			<= -4
(648)		-2ξ02	-4ξ03	- 3ξ12	- ξ13	- 5ξ23	- x12	- x13	- x23	+ x21		+ x32	<= -9
(649)		-2ξ02	-5ξ03	- 2ξ12		- 4ξ23	- x12	-2x13	-2x23	+ x21		+ x32	<= -8
(650)		-2ξ02	-3ξ03	- 2ξ12	- ξ13	- 3ξ23	- x12	- x13	- x23	+ x21			<= -7

(651)	-2ξ02	-5ξ03	- 2ξ12		- 7ξ23	- x12	-2x13	- x23	+ x21		+2x32	<= -8
(652)	-2ξ02	-3ξ03	- 2ξ12	- ξ13	- 6ξ23	- x12	- x13		+ x21		+ x32	<= -7
(653)		- ξ03	- 2ξ12	- ξ13	- ξ23	- x12	- x13	- x23	+ x21			<= -3
(654)		- ξ03	- 2ξ12	- ξ13	- 4ξ23	- x12	- x13		+ x21		+ x32	<= -3
(655)	-3ξ02	- ξ03	- ξ12	- ξ13	- 2ξ23	-2x12	- x13	+ x23	+ x21	- x31	- x32	<= -5
(656)	-6ξ02	-2ξ03	- 3ξ12	- 2ξ13	- 4ξ23	-3x12	-3x13	+2x23	+3x21	-2x31	-2x32	<= -10
(657)	-2ξ02	-2ξ03	- ξ12	- 3ξ13	- 2ξ23	- x12			+ x21	+ x31		<= -4
(658)	-2ξ02	-4ξ03	- 3ξ12	- 4ξ13	- 5ξ23	- x12		- x23	+ x21	+ x31	+ x32	<= -9
(659)	-2ξ02	-5ξ03	- 2ξ12	- 6ξ13	- 4ξ23	- x12		-2x23	+ x21	+2x31	+ x32	<= -8
(660)	-2ξ02	-3ξ03	- 2ξ12	- 4ξ13	- 6ξ23	- x12			+ x21	+ x31	+ x32	<= -7
(661)	-2ξ02	-3ξ03	- 2ξ12	- 4ξ13	- 3ξ23	- x12		- x23	+ x21	+ x31		<= -7
(662)	-2ξ02	-5ξ03	- 2ξ12	- 6ξ13	- 7ξ23	- x12		- x23	+ x21	+2x31	+2x32	<= -8
(663)		- ξ03	- 2ξ12	- 4ξ13	- ξ23	- x12		- x23	+ x21	+ x31		<= -3
(664)		- ξ03	- 2ξ12	- 4ξ13	- 4ξ23	- x12			+ x21	+ x31	+ x32	<= -3
(665)	-6ξ02	-2ξ03	- 3ξ12	-17ξ13	- 4ξ23	-3x12	+2x13	+2x23	+3x21	+3x31	-2x32	<= -10
(666)	-3ξ02	- ξ03	- ξ12	- 7ξ13	- 2ξ23	-2x12	+ x13	+ x23	+ x21	+ x31	- x32	<= -5
(667)	-3ξ02	- ξ03	- 4ξ12	- 7ξ13	- 2ξ23	- x12	+ x13	+ x23	+2x21	+ x31	- x32	<= -5
(668)	-3ξ02	- ξ03	- 4ξ12	- ξ13	- 2ξ23	- x12	- x13	+ x23	+2x21	- x31	- x32	<= -5
(669)	- ξ02		- 2ξ12	- ξ13	- ξ23	- x12			+ x21			<= -2
(670)	- ξ02	- ξ03	- 4ξ12	- 2ξ13	- 4ξ23	- x12		- x23	+ x21		+ x32	<= -6
(671)	-3ξ02	-2ξ03	- 2ξ12	- 2ξ13	- 2ξ23	- x12			+ x21		- x32	<= -7
(672)	-3ξ02	-2ξ03	- 2ξ12	- 2ξ13	- 5ξ23	- x12		+ x23	+ x21			<= -7
(673)	-3ξ02	-3ξ03	- 3ξ12	- 2ξ13	- 4ξ23	- x12			+ x21			<= -9
(674)	-3ξ02	-2ξ03	- 2ξ12	- ξ13	- 3ξ23	- x12			+ x21			<= -6
(675)	-3ξ02	-3ξ03	- 4ξ12	- 2ξ13	- 6ξ23	- x12		- x23	+ x21		+ x32	<= -10
(676)	-4ξ02			- ξ13	- 6ξ23	-3x12	- x13	+2x23	+2x21	+ x31		<= -2
(677)	-4ξ02			- ξ13		-3x12	- x13		+2x21	+ x31	-2x32	<= -2
(678)	-6ξ02	-2ξ03	- 6ξ12	- 4ξ13	-23ξ23	-3x12	- x13	+6x23	+3x21	+ x31	+ x32	<= -13
(679)	-6ξ02	-2ξ03	- 6ξ12	- 4ξ13	- 2ξ23	-3x12	- x13	- x23	+3x21	+ x31	-6x32	<= -13
(680)	-2ξ02	- ξ03	- 2ξ12	- 2ξ13	- ξ23	- x12			+ x21		- x32	<= -5
(681)	-2ξ02		- ξ12	- ξ13		- x12			+ x21		- x32	<= -2
(682)	-6ξ02	-5ξ03	- 6ξ12	- 7ξ13	- 5ξ23	-3x12	- x13	- x23	+3x21	+ x31	-3x32	<= -19
(683)	-2ξ02	- ξ03	- 2ξ12	- 2ξ13	- 4ξ23	- x12		+ x23	+ x21			<= -5
(684)	-2ξ02		- ξ12	- ξ13	- 3ξ23	- x12		+ x23	+ x21			<= -2
(685)	-6ξ02	-5ξ03	- 6ξ12	- 7ξ13	-17ξ23	-3x12	- x13	+3x23	+3x21	+ x31	+ x32	<= -19
(686)	-2ξ02	-2ξ03	- 3ξ12	- 5ξ13	- 6ξ23	- x12	+ x13	+ x23	+ x21	- x31		<= -9
(687)	-2ξ02	-2ξ03	- 3ξ12	- 5ξ13	- 3ξ23	- x12	+ x13		+ x21	- x31	- x32	<= -9
(688)	-4ξ02	-3ξ03	- 2ξ12	- 4ξ13	- 6ξ23	-2x12	+ x13	+2x23	+ x21	- x31	- x32	<= -10
(689)	-4ξ02	-3ξ03	- 2ξ12	- 4ξ13	- 3ξ23	-2x12	+ x13	+ x23	+ x21	- x31	-2x32	<= -10
(690)	-2ξ02	-2ξ03		- 2ξ13	- 2ξ23	-2x12	+ x13	+ x23		- x31	- x32	<= -5
(691)	-3ξ02		- 3ξ12	- 4ξ13	- 2ξ23	-2x12	+ x13	+ x23	+2x21	- x31	- x32	<= -5
(692)	-2ξ02		- ξ12	- 2ξ13	- ξ23	- x12	+ x13	+ x23	+ x21	- x31	- x32	<= -2
(693)	-3ξ02			- 2ξ13		-2x12	+ x13		+ x21	- x31	-2x32	<= -2
(694)	-6ξ02	-3ξ03	- 4ξ12	- 5ξ13	- 3ξ23	-3x12	- x13		+3x21	+ x31	-3x32	<= -13
(695)	-6ξ02		- ξ12	- 2ξ13		-3x12	- x13		+3x21	+ x31	-3x32	<= -4
(696)	-3ξ02			- 2ξ13	- 6ξ23	-2x12	+ x13	+2x23	+ x21	- x31		<= -2
(697)	-6ξ02	-3ξ03	- 4ξ12	- 5ξ13	-12ξ23	-3x12	- x13	+3x23	+3x21	+ x31		<= -13
(698)	-6ξ02		- ξ12	- 2ξ13	- 9ξ23	-3x12	- x13	+3x23	+3x21	+ x31		<= -4
(699)	-4ξ02		- 3ξ12	- ξ13	- 6ξ23	-2x12	- x13	+2x23	+3x21	+ x31		<= -2
(700)	-4ξ02		- 3ξ12	- ξ13		-2x12	- x13		+3x21	+ x31	-2x32	<= -2

(701)		-2ξ02	-2ξ03	- 6ξ12	- 2ξ13	- 2ξ23		+ x13	+ x23	+2x21	- x31	- x32	<=	-5
(702)		-4ξ02	-3ξ03	- 5ξ12	- 4ξ13	- 6ξ23	- x12	+ x13	+2x23	+2x21	- x31	- x32	<=	-10
(703)		-4ξ02	-3ξ03	- 5ξ12	- 4ξ13	- 3ξ23	- x12	+ x13	+ x23	+2x21	- x31	-2x32	<=	-10
(704)		-3ξ02		- 3ξ12	- 2ξ13	- 6ξ23	- x12	+ x13	+2x23	+2x21	- x31		<=	-2
(705)		-3ξ02		- 3ξ12	- 2ξ13		- x12	+ x13		+2x21	- x31	-2x32	<=	-2
(706)	- ξ01	- ξ02	-3ξ03			- 2ξ23	- x12	- x13	- x23	- x21	- x31	+ x32	<=	-5
(707)	- ξ01	-3ξ02	- ξ03			- 2ξ23	-3x12		+ x23	- x21	- x31	- x32	<=	-5
(708)	- ξ01	- ξ02	- ξ03			- 2ξ23	- x12		- x23	- x21	- x31	+ x32	<=	-3
(709)	- ξ01		-2ξ03	- ξ12	- ξ13	- 2ξ23		- x13	- x23	- x21	- x31	+ x32	<=	-5
(710)	- ξ01	-2ξ02		- ξ12	- ξ13	- 2ξ23	-2x12		+ x23	- x21	- x31	- x32	<=	-5
(711)	- ξ01		- ξ03		- ξ13	- 2ξ23			- x23	- x21	- x31	+ x32	<=	-3
(712)	- ξ01			- ξ12	- ξ13	- 2ξ23			- x23	- x21	- x31	+ x32	<=	-3
(713)	- ξ01	-2ξ02	- ξ03		- ξ13	- 2ξ23	-2x12		+ x23	- x21	- x31	- x32	<=	-5
(714)	- ξ01	- ξ02	-3ξ03		- 6ξ13	- 2ξ23	- x12	+ x13	- x23	- x21	+ x31	+ x32	<=	-5
(715)	- ξ01	-3ξ02	- ξ03		- 3ξ13	- 2ξ23	-3x12	+ x13	+ x23	- x21		- x32	<=	-5
(716)	- ξ01	- ξ02	- ξ03		- 3ξ13	- 2ξ23	- x12	+ x13	- x23	- x21		+ x32	<=	-3
(717)	- ξ01	-2ξ02		- ξ12	- 4ξ13	- 2ξ23	-2x12	+ x13	+ x23	- x21		- x32	<=	-5
(718)	- ξ01			- ξ12	- 4ξ13	- 2ξ23		+ x13	- x23	- x21		+ x32	<=	-3
(719)	- ξ01		-2ξ03	- ξ12	- 7ξ13	- 2ξ23		+ x13	- x23	- x21	+ x31	+ x32	<=	-5
(720)	- ξ01		- ξ03		- 4ξ13	- 2ξ23		+ x13	- x23	- x21		+ x32	<=	-3
(721)	- ξ01	-3ξ02	- ξ03	-12ξ12		- 2ξ23	+ x12		+ x23	+3x21	- x31	- x32	<=	-5
(722)	- ξ01	- ξ02	-3ξ03	- 6ξ12		- 2ξ23	+ x12	- x13	- x23	+ x21	- x31	+ x32	<=	-5
(723)	- ξ01	-2ξ02	- ξ03		- 4ξ13	- 2ξ23	-2x12	+ x13	+ x23	- x21		- x32	<=	-5
(724)	- ξ01	- ξ02	- ξ03	- 6ξ12		- 2ξ23	+ x12		- x23	+ x21	- x31	+ x32	<=	-3
(725)	- ξ01	-2ξ02		-10ξ12	- ξ13	- 2ξ23	+ x12		+ x23	+2x21	- x31	- x32	<=	-5
(726)	- ξ01		-2ξ03	- 4ξ12	- ξ13	- 2ξ23	+ x12	- x13	- x23		- x31	+ x32	<=	-5
(727)	- ξ01			- 4ξ12	- ξ13	- 2ξ23	+ x12		- x23		- x31	+ x32	<=	-3
(728)	- ξ01	-2ξ02	- ξ03	- 9ξ12	- ξ13	- 2ξ23	+ x12		+ x23	+2x21	- x31	- x32	<=	-5
(729)	- ξ01		- ξ03	- 3ξ12	- ξ13	- 2ξ23	+ x12		- x23		- x31	+ x32	<=	-3
(730)	- ξ01	- ξ02	-3ξ03	- 6ξ12	- 6ξ13	- 2ξ23	+ x12	+ x13	- x23	+ x21	+ x31	+ x32	<=	-5
(731)	- ξ01	-3ξ02	- ξ03	-12ξ12	- 3ξ13	- 2ξ23	+ x12	+ x13	+ x23	+3x21		- x32	<=	-5
(732)	- ξ01	- ξ02	- ξ03	- 6ξ12	- 3ξ13	- 2ξ23	+ x12	+ x13	- x23	+ x21		+ x32	<=	-3
(733)	- ξ01	-2ξ02		-10ξ12	- 4ξ13	- 2ξ23	+ x12	+ x13	+ x23	+2x21		- x32	<=	-5
(734)	- ξ01		-2ξ03	- 4ξ12	- 7ξ13	- 2ξ23	+ x12	+ x13	- x23		+ x31	+ x32	<=	-5
(735)	- ξ01		- ξ03	- 3ξ12	- 4ξ13	- 2ξ23	+ x12	+ x13	- x23			+ x32	<=	-3
(736)	- ξ01			- 4ξ12	- 4ξ13	- 2ξ23	+ x12	+ x13	- x23			+ x32	<=	-3
(737)	- ξ01	-2ξ02	- ξ03	- 9ξ12	- 4ξ13	- 2ξ23	+ x12	+ x13	+ x23	+2x21		- x32	<=	-5
(738)	-2ξ01	- ξ02	- ξ03		- ξ13	- ξ23	- x12	+ x13	+ x23	- x21	-2x31	- x32	<=	-4
(739)	-2ξ01	-6ξ02		- 2ξ12	- 4ξ13	- 6ξ23	-6x12	+ x13	+3x23	- x21	- x31	-3x32	<=	-13
(740)	-2ξ01	- ξ02	- ξ03	- 6ξ12	- ξ13	- ξ23	+ x12	+ x13	+ x23	+ x21	-2x31	- x32	<=	-4
(741)	-3ξ01				- 2ξ13	- 2ξ23		+ x13	- x23	-2x21	-2x31	+ x32	<=	-4
(742)	-2ξ01	- ξ02	- ξ03		- 4ξ13	- ξ23	- x12	+2x13	+ x23	- x21	- x31	- x32	<=	-4
(743)	-3ξ01				- 5ξ13	- 2ξ23		+2x13	- x23	-2x21	- x31	+ x32	<=	-4
(744)	-2ξ01	-6ξ02		-23ξ12	- 4ξ13	- 6ξ23	+ x12	+ x13	+3x23	+6x21	- x31	-3x32	<=	-13
(745)	-3ξ01			- 6ξ12	- 2ξ13	- 2ξ23	+2x12	+ x13	- x23		-2x31	+ x32	<=	-4
(746)	-2ξ01	- ξ02	- ξ03	- 6ξ12	- 4ξ13	- ξ23	+ x12	+2x13	+ x23	+ x21	- x31	- x32	<=	-4
(747)	-3ξ01			- 6ξ12	- 5ξ13	- 2ξ23	+2x12	+2x13	- x23		- x31	+ x32	<=	-4

Knowledge is of two kinds:
We know a subject ourselves or we know
where we can find information upon it.

Samuel Johnson (1709–1784)

References

Achuthan, N. R., Caccetta, L. & Hill, S. P. (1998), 'Capacitated vehicle routing problem: some new cutting planes', *Asia-Pacific Journal of Operational Research* **15**, 109–123.

Adlakha, V. & Kowalski, K. (1999), 'On the fixed-charge transportation problem', *Omega, International Journal of Management Science* **27**, 381–388.

Adlakha, V. & Kowalski, K. (2003), 'A simple heuristic for solving small fixed-charge transportation problems', *Omega, International Journal of Management Science* **31**, 205–211.

Agrawal, A., Klein, P. & Ravi, R. (1995), 'When trees collide: An approximation algorithm for the generalized Steiner tree problem on networks', *SIAM Journal on Computing* **24**, 440–456.

Alexandrov, D. A. & Kochetov, Y. A. (2000), The behavior of the ant colony algorithm for the set covering problem, *in* K. Inderfurth, G. Schwödiauer, W. Damschke, F. Juhnke, P. Kleinschmidt & G. Wäscher, eds, 'Operations Research Proceedings 1999', Springer-Verlag, pp. 255–260.

Alperovits, E. & Shamir, U. (1977), 'Design of optimal water distribution systems', *Water Resources Research* **13**(6), 885–890.

Altinkemer, K. & Yu, Z. (1992), 'Topological design of wide area communication networks', *Annals of Operations Research* **36**, 365–382.

Ameur, W. B. (2000), 'Constrained length connectivity and survivable networks', *Networks* **36**(1), 17–33.

Araque, J. R., Hall, L. & Magnanti, T. L. (1990), Capacitated trees, capacitated routing and associated polyhedra, CORE Discussion Paper 9061, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

Augerat, P., Belenguer, J. M., Benavent, E., Corberán, A. & Naddef, D. (1998), 'Separating capacity constraints in the CVRP using tabu search', *European Journal of Operational Research* **106**, 546–557.

Augerat, P., Belenguer, J. M., Benavent, E., Corberán, A., Naddef, D. & Rinaldi, G. (1995), Computational results with a branch and cut code for the capacitated vehicle routing problem, Rapport de recherche RR949-M, ARTEMIS-IMAG, Grenoble, France.

Awumah, K., Goulter, I. & Bhatt, S. K. (1991), 'Entropy-based redundancy measures in water-distribution systems', *Journal of Hydraulic Engineering* **117**(5), 595–614.

Balakrishnan, A., Magnanti, T. L. & Mirchandani, P. (1997), Network design, *in* M. Dell'Amico, F. Maffioli & S. Martello, eds, 'Annotated bibliographies in combinatorial optimization', Wiley, pp. 311–334.

Balas, E. (1997), 'A modified lift-and-project procedure', *Mathematical Programming (Series B)* **79**, 19–31.

Balas, E., Ceria, S. & Cornuéjols, G. (1993), 'A lift-and-project cutting plane algorithm for mixed 0-1 programs', *Mathematical Programming* **58**, 295–324.

Balas, E., Ceria, S. & Cornuéjols, G. (1996), 'Mixed 0-1 programming by lift-and-project in a branch-and-cut framework', *Management Science* **42**, 1229–1246.

Balas, E., Ceria, S., Cornuéjols, G. & Natraj, N. (1996), 'Gomory cuts revisited', *Operations Research Letters* **19**, 1–9.

Baldacci, R., Hadjiconstantinou, E. & Mingozzi, A. (2003), An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. Scheduled to appear in *Operations Research*.

Ball, M. O. (1986), 'Computational complexity of network reliability analysis: an overview', *IEEE Transactions on Reliability* **R-35**(3), 230–239.

Ball, M. O., Colbourn, C. J. & Scott Provan, J. (1995), Network reliability, *in* M. O. Ball, T. L. Magnanti, C. L. Monma & G. L. Nemhauser, eds, 'Network Models',

Vol. 7 of *Handbooks in Operations Research and Management Science*, Elsevier Science B.V., pp. 673–762.

Barahona, F. (1996), 'Network design using cut inequalities', *SIAM Journal on Optimization* **6**, 823–837.

Barahona, F. & Mahjoub, A. R. (1995), 'On two-connected subgraph polytopes', *Discrete Mathematics* **147**(1–3), 19–34.

Barcos Redín, L. (2003), Optimización de rutas de transporte de carga de muchos orígines a muchos destinos mediante colonias de hormigas, Ph.D thesis, TECNUN, Universidad de Navarra, Spain.

Barnhart, C., Hane, C. A. & Vance, P. H. (2000), 'Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems', *Operations Research* **48**(2), 318–326.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P. & Vance, P. H. (1998), 'Branch-and-price: column generation for solving huge integer programs', *Operations Research* **46**(3), 316–329.

Beasley, J. E., ed. (1996), *Advances in Linear and Integer Programming*, Clarendon Press, Oxford.

Bienstock, D., Brickell, E. F. & Monma, C. L. (1990), 'On the structure of minimum-weight k-connected spanning networks', *SIAM Journal on Discrete Mathematics* **3**(3), 320–329.

Bienstock, D. & Günlük, O. (1996), 'Capacitated network design-polyhedral structure and computation', *INFORMS Journal on Computing* **8**, 243–259.

Bixby, R. E., Fenelon, M., Gu, Z., Rothberg, E. & Wunderling, R. (2000), MIP: theory and practice—closing the gap, *in* M. J. D. Powell & S. Scholtes, eds, 'System Modelling and Optimization: Methods, Theory and Applications', Kluwer Academic Publishers.

Bollobás, B. (1978), *Extremal Graph Theory*, Academic Press, New York.

Bollobás, B. (2001), *Random graphs*, Cambridge University Press.

Bonabeau, E., Dorigo, M. & Theraulaz, G. (2000), 'Inspiration for optimization from social insect behavior', *Nature* **406**, 39–42.

Borůvka, O. (1926), 'O jistém problému minimálním', *Práce Moravské Přírodovědecké Společnosti (Acta Societ. Scient. Natur. Moravicae)* **3**, 37–58.

Boyd, S. C. & Hao, T. (1993), 'An integer polytope related to the design of survivable communication networks', *SIAM Journal on Discrete Mathematics* **6**(4), 612–630.

Brouwer, A. E. & Mudler, H. M. (1997), 'The vertex connectivity of a $\{0, 2\}$-regular graph equals its degree', *Discrete Mathematics* **169**, 153–155.

Bullnheimer, B., Hartl, R. F. & Strauß, C. (1999), Applying the ant system to the vehicle routing problem, *in* S. Voss, S. Martello, I. H. Osman & C. Roucairol, eds, 'Meta-Heuristics: Advances and Trends in local search paradigms for optimization', Kluwer Academic Publishers, pp. 285–296.

Bullnheimer, B., Kotsis, G. & Strauß, C. (1997), Parallelization strategies for the ant system, Report Series No. 8, Vienna University of Economics and Business Administration, Austria.

Capara, A., Fischetti, M. & Letchford, A. N. (2000), 'On the separation of maximally violated mod-k cuts.', *Mathematical Programming (Series A)* **87**, 37–56.

Cardwell, R. H., Monma, C. L. & Wu, T. H. (1989), 'Computer-aided design procedures for fiber optic networks', *IEEE Journal on Selected Areas in Communications* **7**, 1188–1197.

Ceria, S., Cordier, C., Marchand, H. & Wolsey, L. A. (1998), 'Cutting planes for integer programs with general integer variables', *Mathematical Programming (Series B)* **81**, 201–214.

Chartrand, G. (1966), 'A graph-theoretical approach to communication problems', *SIAM Journal on Applied Mathematics* **14**, 778–781.

Chartrand, G. & Harary, F. (1968), Graphs with described connectivities, *in* P. Erdős & G. Katona, eds, 'Theory of Graphs', Academic Press, New York, pp. 61–63.

Chekuri, C. S., Goldberg, A. V., Karger, D. R., Levine, M. S. & Stein, C. (1996), 'Experimental study of minimum cut algorithms'. Technical report. *An expanded version of Chekuri, Goldberg, Karger, Levine & Stein (1997).*

Chekuri, C. S., Goldberg, A. V., Karger, D. R., Levine, M. S. & Stein, C. (1997), Experimental study of minimum cut algorithms, *in* 'Proceeedings of the 8th Annnual ACM-SIAM Symposium on Discrete Algorithms'.

Chopra, S. (1994), 'The k-edge-connected spanning subgraph polyhedron', *SIAM Journal on Discrete Mathematics* **7**(2), 245–259.

Christof, T. & Löbel, A. (2002), 'PORTA—a polyhedron representation transformation algorithm', ZIB: Konrad-Zuse-Zentrum für Informationstechnik Berlin (Zuse Institute Berlin), Germany. Version 1.4.0α, http://www.zib.de/Optimization/Software/Porta/.

Christofides, N. (1976), Worst-case analysis of a new heuristic for the travelling salesman problem, Management Sciences Research Report 388, Carnegie-Mellon University, Pennsylvania.

Chvátal, V. (1973), 'Edmonds polytopes and a hierarchy of combinatorial problems', *Discrete Mathematics* **4**, 305–337.

Colbourn, C. J. (1987), *The Combinatorics of Network Reliability*, International Series of Monographs in Computer Science, Oxford University Press.

Colorni, A., Dorigo, M., Maffioli, F., Maniezzo, V., Righini, G. & Trubian, M. (1996), 'Heuristics from nature for hard combinatorial optimization problems', *International Transactions in Operational Research* **3**(1), 1–21.

Cornuéjols, G. & Li, Y. (2001), 'Elementary closures for integer programmes', *Operational Research Letters* **28**(1), 1–8.

Costa, D. & Hertz, A. (1997), 'Ants can colour graphs', *Journal of the Operational Research Sociaty* **48**, 295–305.

Crainic, T. G., Frangioni, A. & Gendron, B. (2001), 'Bundle-based relaxation methods for multicommodity capacitated fixed charge network design', *Discrete Applied Mathematics* **112**, 73–99.

Cullinane, M. J., Lansey, K. E. & Mays, L. W. (1992), 'Optimization-availability-based design of water-distribution networks', *Journal of Hydraulic Engineering* **118**(3), 420–441.

Current, J. R., ReVelle, C. S. & Cohon, J. L. (1986), 'The hierarchical network design problem', *European Journal of Operational Research* **27**, 57–66.

da Silva, A. (2001), 'Combining constraint programming and linear programming on an example of bus driver scheduling', *Annals of Operations Research* **108**, 277–764.

Dantzig, G. B. & Thapa, M. N. (1997), *Linear Programming*, Vol. I of *Springer Series in Operations Research*, Springer-Verlag.

Darby-Dowman, K. & Wilson, J. M. (2002), 'Developments in linear and integer programming', *Journal of the Operational Research Society* **53**(10), 1065–1071.

Deeter, D. L. & Smith, A. E. (1998), 'Economic design of reliable networks', *IIE Transactions* **30**(12), 1161–1174.

Dell'Amico, M., Maffioli, F. & Martello, S., eds (1997), *Annotated bibliographies in combinatorial optimization*, Wiley.

Dorigo, M. & Gambardella, L. M. (1996), Ant colonies for the traveling salesman problem, Technical Report TR/IRIDIA/1996-3, Université Libre de Bruxelles, Belgium.

Dorigo, M., Maniezzo, V. & Colorni, A. (1996), 'The ant system: optimization by a colony of cooperating agents', *IEEE Transactions on Systems, Man and Cybernetics (Part B)* **26**(1), 29–41.

Duin, C. & Volgenant, A. (1989), 'Reducing the hierarchical network design problem', *European Journal of Operational Research* **39**, 332–344.

Edmonds, J. (1971), 'Matroids and the greedy algorithm', *Mathematical Programming* **1**, 127–136.

Edmonds, J. & Johnson, E. L. (1970), Matching: a well solved class of integer programming, *in* R. Guy, ed., 'Combinatorial Structure and their applications', Gordon and Breach, New York.

Eiselt, H. A. & Sandblom, C.-L. (2000), *Integer programming and network models*, Springer-Verlag.

Eswaran, K. P. & Tarjan, R. E. (1976), 'Augmentation problems', *SIAM Journal on Computing* **5**, 653–665.

Evans, G. W. & Fairbairn, R. (1989), 'Selection and scheduling of advanced missions for NASA using 0-1 integer linear programming', *Journal of the Operational Research Society* **40**(11), 971–981.

Even, S. (1977), 'Algorithm for determining whether the connectivity of a graph is at least k', *SIAM Journal on Computing* **6**, 393–396.

Even, S. & Tarjan, R. E. (1975), 'Network flow and testing graph connectivity', *SIAM Journal on Computing* **4**, 507–512.

Farber, M., Richter, B. & Shank, H. (1985), 'Edge-disjoint spanning trees: a connectedness theorem', *Journal of Graph Theory* **8**, 319–324.

Fonlupt, J. & Mahjoub, A. R. (1999), Critical extreme points of the 2-edge connected spanning subgraph polytope, *in* G. Cornuéjols, R. E. Burkard & G. Woeginger, eds, 'Integer Programming and Combinatorial Optimization', 7th International IPCO Conference, Graz, Austria, Springer-Verlag, pp. 166–182.

Fontes, D. B. M. M. (2000), Optimal network design using nonlinear cost flows, Ph.D thesis, The Management School, Imperial College of Science, Technology and Medicine, London.

Ford, Jr., L. R. & Fulkerson, D. R. (1962), *Flows in networks*, Princeton University Press, New Jersey.

Frank, A. (1992), 'Augmenting graphs to meet edge-connectivity requirements', *SIAM Journal on Discrete Mathematics* **5**, 25–53.

Frank, A. (1994), Connectivity augmentation problems in network design, *in* J. R. Birge & K. G. Murty, eds, 'Mathematical programming: State of the art 1994', The University of Michigan, Ann Arbor, Michigan, pp. 34–63.

Frank, A. (1995), Connectivity and network flows, *in* R. Graham, M. Grötschel & L. Lovász, eds, 'Handbook of Combinatorics', Elsevier Science B.V.

Frederickson, G. N. & Ja'ja', J. (1981), 'Approximation algorithms for several graph augmentation problems', *SIAM Journal on Computing* **10**(2), 270–283.

Frederickson, G. N. & Ja'ja', J. (1982), 'On the relationship between the biconnectivity augmentation problem and travelling salesman problem', *Theoretical Computer Science* **19**, 189–201.

Fujiwara, O. & Khang, D. B. (1990), 'A two-phase decomposition model for optimal design of looped water distribution networks', *Water Resources Research* **26**(4), 539–549.

Future Generation Computer Science (2000). Volume **16**, issue 8.

Gabow, H. N., Goemans, M. X. & Willamson, D. P. (1998), 'An efficient approximation algorithm for the survivable network design problem', *Mathematical Programming (Series B)* **82**(1–2), 13–40.

Galil, Z. (1980), 'Finding the vertex connectivity of graphs', *SIAM Journal on Computing* **9**, 197–199.

Gambardella, L. M. & Dorigo, M. (2000), 'Ant colony system hybridized with a new local search for the sequential ordering problem', *INFORMS Journal on Computing* **12**(3), 237–255.

Gambardella, L. M., Taillard, É. & Agazzi, G. (1999), MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows, *in* D. Corne, M. Dorigo & F. Glover, eds, 'New ideas in optimization', McGraw-Hill, London, pp. 63–76.

Garey, M. R. & Johnson, D. S. (1979), *Computers and Intractability: A Guide to the theory of NP-completeness*, Freeman, San Francisco, California.

Garg, N., Santosh, V. & Singla, A. (1993), 'Improved approximation algorithms for biconnected subgraphs via better lower bound techniques', *Proceedings of the 4th Annual ACM-SIAM Symposium of Discrete Algorithms* pp. 103–111.

Gavish, B. (1992), 'Topological design of computer communication networks—the overall design problem', *European Journal of Operational Research* **58**, 149–172.

Gavish, B. & Neuman, I. (1992), 'Routing in a network with unreliable components', *IEEE Transactions on Communications* **40**(7), 1248–1258.

Gendron, B. (2002), 'A note on "a dual-ascent approach to the fixed-charge capacitated network design problem"', *European Journal of Operational Research* **138**, 671–675.

Ghamlouche, I., Crainic, T. G. & Gendreau, M. (2003), 'Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design', *Operations Research* **51**(4), 655–667.

Glover, F. & Kochenberger, G. A., eds (2003), *Handbook of Metaheuristics*, Kluwer Academic Publishers.

Goemans, M. X. & Bertsimas, D. J. (1993), 'Survivable networks, linear programming relaxations and the parsimonious property', *Mathematical Programming* **60**(2), 145–166.

Gomory, R. E. (1960), An algorithm for the mixed integer problem, Technical Report RM-2597, The Rand Corporation, Santa Monica, California.

Gomory, R. E. & Hu, T. C. (1961), 'Multi-terminal network flows', *SIAM Journal on Applied Mathematics* **9**, 551–570.

Goulter, I. (1995), Analytical and simulation models for reliability analysis in water distribution systems, *in* E. Cabrera & A. F. Vela, eds, 'Improving Efficiency and Reliability in Water Distribution Systems', Kluwer Academic Publishers, pp. 235–266.

Goulter, I. C. & Morgan, D. R. (1985), 'An integrated approach to the layout and design of water distribution systems', *Civil Engineering Systems* **2**(2), 104–113.

Gross, D. & Saccoman, J. T. (1998), 'Uniformly optimally reliable graphs', *Networks* **31**, 217–225.

Grötschel, M. & Monma, C. L. (1990), 'Integer polyhedra associated with certain network design problems with connectivity constraints', *SIAM Journal on Discrete Mathematics* **3**, 502–523.

Grötschel, M., Monma, C. L. & Stoer, M. (1992*a*), 'Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints', *Operations Research* **40**(2), 309–330.

Grötschel, M., Monma, C. L. & Stoer, M. (1992*b*), 'Facets and polyhedra arising in the design of communications networks with low-connectivity constraints', *SIAM Journal on Optimization* **2**, 474–504.

Grötschel, M., Monma, C. L. & Stoer, M. (1995*a*), Design of survivable networks, *in* M. O. Ball, T. L. Magnanti, C. L. Monma & G. L. Nemhauser, eds, 'Network Models', Vol. 7 of *Handbooks in Operations Research and Management Science*, Elsevier Science B.V., pp. 617–672.

Grötschel, M., Monma, C. L. & Stoer, M. (1995*b*), 'Polyhedral and computational investigations for designing communication networks with high survivability requirements', *Operations Research* **43**(6), 1012–1024.

Günlük, O. (1999), 'A branch-and-cut algorithm for capacitated network design problems', *Mathematical Programming (Series A)* **86**, 17–39.

Gusfield, D. (1990), 'Very simple methods for all pairs network flow analysis', *SIAM Journal on Computing* **19**(1), 143–155.

Hadlock, F. (1975), 'Finding a maximum cut of a planar graph in polynomial time', *SIAM Journal on Computing* **4**, 221–225.

Halin, R. (1971), Studies on minimally n-connected graphs, *in* D. J. A. Welsh, ed., 'Combinatorial Mathematics and its Applications', Academic Press, New York, pp. 129–136.

Hao, J. & Orlin, J. B. (1994), 'A faster algorithm for finding the minimum cut in a directed graph', *J. Algorithms* **17**(3), 424–446.

Harary, F. (1962), 'The maximum connectivity of a graph', *Proceedings of the National Academy of Science USA* **48**, 1142–1146.

Hellstrand, J., Larson, T. & Migdalas, A. (1992), 'A characterization of the uncapacitated network design problem', *Operations Research Letters* **12**, 159–163.

Hemaspaandra, L. A. & Ogihara, M. (2002), *The complexity theory companion*, Springer-Verlag.

Herer, Y. T., Rosenblatt, M. J. & Hefter, I. (1996), 'Fast algorithms for single-sink fixed charge transportation problems with applications to manufacturing and transportation', *Transportation Science* **30**(4), 276–290.

Herrmann, J. W., Ioannou, G., Minis, I. & Proth, J. M. (1996), 'A dual ascent approach to the fixed-charge capacitated network design problem', *European Journal of Operational Research* **95**, 476–490.

Hochbaum, D. S. & Segev, A. (1989), 'Analysis of a flow problem with fixed charges', *Networks* **19**, 291–312.

Hopcroft, J. & Tarjan, R. E. (1973), 'Dividing a graph into triconnected components', *SIAM Journal on Computing* **2**, 135–158.

Huck, A. (1995), 'Disproof of a conjecture about independent spanning trees in k-connected directed graphs', *Journal of Graph Theory* **20**, 235–239.

Hwang, F. W. & Richards, D. S. (1992), 'Steiner tree problems', *Networks* **22**, 55–90.

IEEE Journal on Selected Areas in Communications (1994). Volume **12**, issue 1.

ILOG™ S.A. (2001), *ILOG CPLEX 7.1: User's Manual*, France.

Itai, A. & Rodeh, M. (1984), The multi-tree approach to reliability in distributed networks, *in* 'Proceedings of the 25th Symposium of Foundations of Computer Science', pp. 137–147.

Johnson, E. L., Nemhauser, G. L. & Savelsbergh, M. W. P. (2000), 'Progress in linear programming-based algorithms for integer programming: an exposition', *INFORMS Journal on Computing* **12**(1), 2–23.

Jünger, M., Reinelt, G. & Thienel, S. (1995), Practical problem solving with cutting plane algorithms in combinatorial optimization, *in* W. Cook, L. Lovász & P. Seymour, eds, 'Combinatorial Optimization', Vol. 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, pp. 111–152.

Jünger, M. & Thienel, S. (2000), 'The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization', *Software—Practice and Experience* **30**(11), 1325–1352.

Jungnickel, D. (1999), *Graphs, networks and algorithms*, Vol. 5 of *Algorithms and computation mathematics*, Springer-Verlag.

Kapoor, S. & Ramesh, H. (1995), 'Algorithms for generating all spanning trees of undirected and weighted graphs', *SIAM Journal on Computing* **24**(2), 247–265.

Kapoor, S. & Ramesh, H. (2000), 'An algorithm for enumerating all spanning trees of a directed graph', *Algorithmica* **27**, 120–130.

Karger, D. R. & Stein, C. (1993), An $\tilde{O}(n^2)$ algorithm for minimum cuts, *in* 'Proceeedings of STOC 25', pp. 757–765.

Kel'mans, A. K. (1979), 'The graph with the maximum probability of remaining connected depends on the edge-removal probability', *Graph Theory Newsletter* **9**, 2–3.

Kessler, A., Ormsbee, L. & Shamir, U. (1990), 'A methodology for least-cost design of invulnerable water distribution networks', *Civil Engineering Systems* **1**, 20–28.

Kessler, A. & Shamir, U. (1991), 'Decomposition technique for optimal design of water supply networks', *Engineering Optimization* **17**, 1–19.

Khachiĭan, L. G. (1979), 'Polinomial'nyĭ algoritm v lineĭnom programmirovanii (A polynomial algorithm in linear programming)', *Doklady Akademii Nauk SSSR* **244**(5), 1093–1096. (Translated in: *Soviet Mathematics Doklady* **20**, 191–194, 1979).

Khomsi, D. (1994), Optimal Design of Water Distribution Networks with Reliability Considerations, Ph.D thesis, City University, London.

Khuller, S. & Vishkin, U. (1994), 'Biconnectivity approximations and graph carvings', *Journal of the ACM* **41**, 214–235.

Kim, D. & Pardalos, P. M. (1999), 'A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure', *Operations Research Letters* **24**, 195–203.

Kim, H.-J. & Hooker, J. N. (2002), 'Solving fixed-charge network flow problems with a hybrid optimization and constraint programming approach', *Annals of Operations Research* **115**, 94–124.

Kirchhoff, G. (1847), 'Über die Auflösungen der gleichungen, auf die man bei der Untersuchung der Verteilung galvanischer Ströme geführt wird', *Ann. Phys. Chem.* **72**, 497–508.

Korte, B. & Vyger, J. (2000), *Combinatorial Optimization: theory and algorithms*, Vol. 21 of *Algoriths and combinatorics*, Springer-Verlag.

Kruskal, J. B. (1956), 'On the shortest spanning subtree of a graph and the travelling salesman problem', *Proceedings of the American Mathematical Society* **7**, 48–50.

Land, A. H. & Doig, A. G. (1960), 'An automatic method for solving discrete programming problem', *Econometrica* **28**, 497–520.

Letchford, A. N. & Lodi, A. (2002), 'Strengthening Chvátal-Gomory cuts and Gomory fractional cuts', *Operations Research Letters* **30**, 74–82.

Li, X. (1998), 'Edge-disjoint minimum-weight connected spanning k-edge subgraphs in a weighted graph: A connectedness theorem', *Discrete Mathematics* **188**, 175–182.

Little, J. D. C., Murty, K. G., Sweeney, D. W. & Karel, C. (1963), 'An algorithm for the traveling salesman problem', *Operations Research* **11**, 972–989.

Loganathan, G. V., Greene, J. J. & Ahn, T. J. (1995), 'Design heuristics for globally minimum cost water-distribution systems', *Journal of Water Resources Planning and Management* **121**(2), 182–192.

Loganathan, G. V., Sherali, H. D. & Shah, M. P. (1990), 'A two-phase network design heuristic for minimum cost water distribution systems under a reliability constraint', *Engineering Optimization* **15**, 311–336.

Lovász, L. (1979), *Combinatorial Problems and Exercises*, North-Holland, Amsterdam.

Lucet, C., Manouvrier, J.-F. & Carlier, J. (2000), 'Evaluating network reliability and 2-edge-connected reliability in linear time for bounded pathwidth graphs', *Algorithmica* **27**, 316–336.

Maculan, N. (1987), 'The Steiner problem in graphs', *Annals of Discrete Mathematics* **31**, 185–212.

Mader, W. (1978), 'A reduction method for edge-connectivity in graphs', *Annals of Discrete Mathematics* **3**, 145–164.

Mader, W. (1979), Connectivity and edge-connectivity in finite graphs, *in* B. Bollobás, ed., 'Surveys on Combinatorics', London Mathematical Society, pp. 293–309.

Magnanti, T. L. & Mirchandani, P. (1993), 'Shortest paths, single origin-destination network design, and associated polyhedra', *Networks* **23**, 103–121.

Magnanti, T. L., Mirchandani, P. & Vachani, R. (1995), 'Modeling and solving the two facility capacitated network loading problem', *Operations Research* **43**, 142–157.

Magnanti, T. L. & Wolsey, L. A. (1995), Optimal trees, *in* M. O. Ball, T. L. Magnanti, C. L. Monma & G. L. Nemhauser, eds, 'Network Models', Vol. 7 of *Handbooks in Operations Research and Management Science*, Elsevier Science B.V., pp. 503–615.

Mahjoub, A. R. (1994), 'Two-edge connected spanning subgraphs and polyhedra', *Mathematical Programming (Series A)* **64**(2), 199–208.

Mahjoub, A. R. (1997), 'On perfectly two-edge connected graphs', *Discrete Mathematics* **170**, 153–172.

Mansour, Y. & Schieber, B. (1989), 'Finding the edge connectivity of directed graphs', *Journal of Algorithms* **10**, 76–85.

Marchand, H., Martin, A., Weismantel, R. & Wolsey, L. (1999), Cutting planes in integer and mixed integer programming, CORE discussion paper 9953, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

Matula, D. W. (1987), Determining the edge connectivity in $O(mn)$, *in* 'Proceedings of the 28th Symposium of Foundations of Computer Science', pp. 249–251.

Melkote, S. & Daskin, M. S. (2001), 'Capacitated facility location/network design problems', *European Journal of Operational Research* **129**, 481–495.

Menger, K. (1927), 'Zur allgemeinen Kurventheorie', *Fundamenta Mathematicae* **10**, 96–115.

Monma, C. L., Munson, B. & Pulleyblank, W. R. (1990), 'Minimum-weight two-connected spanning networks', *Mathematical Programming* **46**(2), 153–171.

Monma, C. L. & Shallcross, D. F. (1989), 'Methods for designing communications networks with certain two-connected survivability constraints', *Operations Research* **37**(4), 531–541.

Moore, E. F. & Shannon, C. E. (1956), 'Reliable circuits using less reliable relays', *Journal of the Franklin Institute* **262**, 191–208.

Myrvold, W. J., Cheung, K. H., Page, L. & Perry, J. (1991), 'Uniformly reliable graphs do not always exit', *Networks* **21**, 417–419.

Myung, Y.-S., Kim, H.-J. & Tcha, D.-W. (1999), 'Design of communication networks with survivability constraints', *Management Science* **45**(2), 238–252.

Nagamochi, H. & Ibaraki, T. (1992), 'Computing edge-connectivity in multigraphs and capacitated graphs', *SIAM Journal on Discrete Mathematics* **5**, 54–66.

Nagamochi, H., Ono, T. & Ibaraki, T. (1994), 'Implementing an efficient minimum capacity cut algorithm', *Mathematical Programming* **67**, 297–324.

Naor, D., Gusfield, D. & Martel, C. (1997), 'A fast algorithm for optimally increasing the edge connectivity', *SIAM Journal on Computing* **26**(4), 1139–1165.

Nemhauser, G. H. & Wolsey, L. A. (1988), *Integer and Combinatorial Optimization*, Wiley, New York.

Nemhauser, G. L., Savelsbergh, M. W. P. & Sigismondi, G. C. (1994), 'MINTO, a mixed integer optimizer', *Operations Research Letters* **15**(1), 47–58.

Ng, P. H. & Rardin, R. L. (1997), 'Commodity family extended formulations of uncapacitated fixed charge network flow problems', *Networks* **30**(1), 57–71.

Ormsbee, L. & Kessler, A. (1990), 'Optimal upgrading of hydraulic-network reliability', *Journal of Water Resources Planning and Management* **116**(6), 784–801.

Ortega, F. & Wolsey, L. A. (2003), 'A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem', *Networks* **41**(3), 143–158.

Ostfeld, A. & Shamir, U. (1993), 'Incorporating reliability in optimal design of water distribution networks—review and new concepts', *Reliability Engineering and System Safety* **42**, 5–11.

Ostfeld, A. & Shamir, U. (1996), 'Design of optimal reliable multiquality water-supply systems', *Journal of Water Resources Planning and Management* **122**(5), 322–333.

Padberg, M. (2001), 'Classical cuts for mixed-integer programming and branch-and-cut', *Mathematics of Operations Research* **53**, 173–203.

Padberg, M. & Rinaldi, G. (1990), 'An efficient algorithm for the minimum capacity cut problem', *Mathematical Programming* **47**, 19–36.

Padberg, M. & Rinaldi, G. (1991), 'A branch-and-cut algorithm for the resolution of large-scale symmetric travelling salesman problems', *SIAM Review* **33**, 60–100.

Padberg, M. W. & Rao, M. R. (1982), 'Odd minimum cut-sets and b-matchings', *Mathematics of Operations Research* **7**(1), 67–80.

Pirkul, H., Current, J. R. & Nagarajan, V. (1991), 'The hierarchical network design problem: A new formulation and solution procedures', *Transportation Science* **25**.

Pittel, B. (1982), 'On the probable behaviour of some algorithms for finding the stability number of a graph', *Mathematical Proceedings of the Cambridge Philosophical Society* **92**, 511–526.

PORTA 1.4.0α (2002). see Christof & Löbel (2002).

Powell, S. (1999), 'Using linear programming to simulate service engineers', *Journal of the Operational Research Society* **50**(12), 1252–1255.

Prim, R. C. (1957), 'Shortest connection networks and some generalizations', *The Bell System Technical Journal* **36**, 1389–1401.

Raghavan, S. & Magnanti, T. L. (1997), Network connectivity, *in* M. Dell'Amico, F. Maffioli & S. Martello, eds, 'Annotated bibliographies in combinatorial optimization', Wiley, pp. 335–354.

Raidl, G. R. & Ljubić, I. (2002), 'Evolutionary local search for the edge-biconnectivity augmentation problem', *Information Processing Letters* **82**, 39–45.

Rardin, R. L. & Choe, U. (1979), Tighter relaxations of fixed charge network flow problems, Technical Report J-79-18, Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

Rardin, R. L. & Wolsey, L. A. (1993), 'Valid inequalities and projecting the multicommodity extended formulation for uncapacitated fixed charge network flow problems', *European Journal of Operational Research* **71**, 95–109.

Richards, A., Schouwenaars, T., How, J. P. & Feron, E. (2002), 'Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming', *Journal of Guidance Control and Dynamics* **25**(4), 755–764.

Schoonderwoerd, R., Holland, O., Bruten, J. & Rothkrantz, L. (1996), 'Ant-based load balancing in telecommunication networks', *Adaptive Behavior* **5**(2), 169–207.

Schrijver, A. (1986), *Theory of linear and integer programming*, Wiley.

Sherali, H. D., Ozbay, K. & Subramanian, S. (1998), 'The time-dependent shortest pair of disjoint paths problem: Complexity, models and algorithms', *Networks* **31**, 259–272.

Sridhar, V. & Park, J. S. (2000), 'Benders-and-cut algorithm for fixed-charge capacitated network design problem', *European Journal of Operational Research* **125**, 622–632.

Stallaert, J. (2000), 'Valid inequalities and separation for capacitated fixed charge flow problems', *Discrete Applied Mathematics* **98**, 265–274.

Sun, M., Aronson, J. E., McKeown, P. G. & Drinka, D. (1998), 'A tabu search heuristic procedure for the fixed charge transportation problem', *European Journal of Operational Research* **106**, 441–456.

Talbi, E.-G., Roux, O., Fonlupt, C. & Robillard, D. (2001), 'Parallel and colonies for the quadratic assignment problem', *Future Generation Computer Systems* **17**(4), 441–449.

Tarjan, R. E. (1972), 'Depth first search and linear graph algorithms', *SIAM Journal on Computing* **1**, 146–160.

Thienel, S. (1995), ABACUS—A Branch-And-CUt System, doctoral thesis, Universität zu Köln.

Toth, P. & Vigo, D. (2002), *The Vehicle Routing Problem*, SIAM.

Tripathy, A. (1984), 'School timetabling—a case study in large binary integer linear programming', *Management Science* **30**(12), 1473—1489.

Tutte, W. T. (1966), *Connectivity in graphs*, University of Toronto Press, Toronto, Canada.

Tutte, W. T. (1984), *Graph Theory*, Addison-Wesley, Reading, MA.

Vazirani, V. V. (2003), *Approximation Algorithms*, 2nd edn, Springer-Verlag.

Wagner, H. M. (1959), 'On a class of capacitated transportation problems', *Management Science* **5**(1), 304–318.

Wagner, J. M., Shamir, U. & Marks, D. H. (1988*a*), 'Water distribution reliability—analytical methods', *Journal of Water Resources Planning and Management* **114**(3), 253–275.

Wagner, J. M., Shamir, U. & Marks, D. H. (1988*b*), 'Water distribution reliability—simulation methods', *Journal of Water Resources Planning and Management* **114**(3), 276–294.

Walters, G. A. & Smith, D. K. (1995), 'Evolutionary design algorithm for optimal layout of tree networks', *Engineering Optimization* **24**, 261–281.

Watanabe, T. & Nakamura, A. (1987), 'Edge-connectivity augmentation problems', *Journal of Computer and System Sciences* **35**, 96–144.

Weber, R. (1998), 'Optimization', Lecture Notes, Mathematical Tripos Part IB, University of Cambridge. http://www.statslab.cam.ac.uk/~rrw1/opt/.

Whitney, H. (1932*a*), 'Congruent graphs and the connectivity of graphs', *American Journal of Mathematics* **54**, 150–168.

Whitney, H. (1932*b*), 'Non-separable and planar graphs', *Transactions of the American Mathematical Society* **34**, 339–362.

Whitty, R. (1987), 'Vertex-disjoint paths and edge-disjoint branchings in directed graphs', *Graph Theory* **11**, 349–358.

Williamson, D. P., Goemans, M. X., Mihail, M. & Vazirani, V. V. (1995), 'A primal-dual approximation algorithm for generalized Steiner network problems', *Combinatorica* **15**, 435–454.

Winter, P. (1985), 'Generalized Steiner tree problem in Halin networks', *Proceedings of the 12th International Symposium on Mathematical Programming, MIT, Massachusetts* .

Winter, P. (1986), 'Generalized Steiner tree problem in series-parallel networks', *Journal of Algorithms* **7**, 549–566.

Wolsey, L. A. (1998), *Integer Programming*, Wiley Interscience.

Wright, E. M. (1972), 'The probability of connectedness of an unlabelled graph can be less for more edges', *Proceedings of the American Mathematical Society* **35**, 21–25.

Xue, G. (1997), 'Linear time algorithms for computing the most reliable source on an unreliable tree network', *Networks* **30**, 37–45.

Zangwill, W. I. (1968), 'Minimum concave cost flows in certain networks', *Management Science* **14**(7), 429–450.

Zehavi, A. & Itai, A. (1989), 'Three tree-paths', *Journal of Graph Theory* **13**, 175–188.

Zemel, E. (1982), 'Polynomial algorithms for estimating network reliability', *Networks* **12**, 439–452.

A heavy weight is now to be removed from my conscience.
So essential did I consider an Index to be to every book,
that I proposed to bring a Bill into Parliament
to deprive an author who publishes a book without and Index
of the privilege of copyright; and, moreover,
to subject him, for his offence, to a pecuniary penalty.
Yet, from difficulties started by my printers,
my own books have hitherto been without an Index.

Lord Campbell
Lives of the Chief Justices of England, vol. 3 (1857)

Index

capacitated reliable network design problem with flows, → CRNDP
uncapacitated reliable network design problem with flows, → URNDP

A

ant colony optimisation
  flow ant, 163
  for the TSP, 159
  for the URNDP, 161–172
  overview, 156
  reliability ant, 163
ant group, 162, 169
ant path selection, 166
ant trail deposition, 164
ant trail enhancement, 170
ant trail updating, 165, 170

B

the b-matching Problem, 100
best bound node search, 131
binary enumeration tree, 29
branch and bound algorithms, 29
branch and cut algorithms, 29, 127
branch variable selection, 129

C

the Capacity Assignment Problem, 182
computational results, 116–125, 148–153, 172–177, 198–204
connectivity, *see* network connectivity
CRNDP
  capacitated degree inequalities, 192
  capacitated reliability constraints, 185
  capacitated reliability cuts, 191–196
  cut generation, 197
  definition, 60
  example, 197
  lower bound ascent, 196
  MIP formulation, 183
  relaxation, 188
  separation of reliability constraints, 189
  solution layout, 187
  strengthened r-cover inequalities, 196
cut lifting (during branch and cut), 133
cut validity (during branch and cut), 132
cutting plane algorithms, 25–28

D

depth first node search, 131

F

the Fixed Charge Problem, 37–42

G

Gomory fractional cut, 107
Gomory-Chvátal cut, 30–33
Gomory-Hu algorithm, 79
graph (definition), 19

H

Halin graphs, 52

Hao-Orlin algorithm, 87

K

k-terminal reliability, 55

L

lift and project cut, 108

M

matching inequalities, 101
maximum cost branching rule, 129
maximum cut (in a network), 80
maximum infeasibility branching rule, 129
Menger's theorem, 44
minimum cut (in a network), 78–80
minimum directed cut (in a network), 87
minimum infeasibility branching rule, 130
minimum odd cut (in a network), 102
the Minimum Spanning Tree Problem, 37
mixed integer program (MIP), 24
the Multi-Level Network Design Problem, 35

N

network (definition), 20
network augmentation problems, 54
network connectivity
    edge, 43
    node, 43
    perfectly-two-edge, 52
    strong, 47
    tree, 44
network design problem
    capacitated, 180–183
    classification, 34–36
    definition, 20
    overview, 18
the Network Loading Problem, 35
the Network Restoration Problem, 36

P

Padberg-Rao algorithm, 102
parallel processing, 211
perfectly two-edge connected graphs, 52
Petersen graph, 76

R

random graphs, 62
reliability, 22
    k-terminal, 55
    s-t connectedness, 55
    reachability, 55
    topological, 42–44
reliability polynomial, 56

S

series-parallel graphs, 19, 41, 52
the Steiner Tree Problem, 35
strong branching, 130
survivability, *see* reliability

T

test problems
    generation, 60–68
    sample instances, 213

U

unlabelled random graphs, 63
URNDP
    algorithm outline, 135
    branch selection, 139
    cut generation, 134
    definition, 60
    example, 73–76, 142, 144
    Gomory inequalities, 107
    heuristic, 141
    lift and project, 108
    lower bound ascent, 115–125
    maximum flow inequalities, 84
    MIP formulation, 71
    node selection, 139
    partition inequalities, 80
    polytope description for K_4, 216
    proof of facet defining cuts, 88–96
    r-cover inequalities, 99
    ralaxation, 76
    reliability constraint separation, 78
    solution layout, 74
    variable selection, 136

V

vertex degree, 19