Analysis of Large Scale Linear Programming Problems with Embedded Network Structures: Detection and Solution Algorithms

A thesis submitted for the degree of Doctor of Philosophy

by

Nalân Gülpınar

Department of Mathematics and Statistics, Brunel University

December 1998

Abstract

Linear programming (LP) models that contain a (substantial) network structure frequently arise in many real life applications. In this thesis, we investigate two main questions; i) how an embedded network structure can be detected, ii) how the network structure can be exploited to create improved sparse simplex solution algorithms. In order to extract an embedded pure network structure from a general LP problem we develop two new heuristics. The first heuristic is an alternative multi-stage generalised upper bounds (GUB) based approach which finds as many GUB subsets as possible. In order to identify a GUB subset two different approaches are introduced; the first is based on the notion of Markowitz merit count and the second exploits an independent set in the corresponding graph. The second heuristic is based on the generalised signed graph of the coefficient matrix. This heuristic determines whether the given LP problem is an entirely pure network; this is in contrast to all previously known heuristics. Using generalised signed graphs, we prove that the problem of detecting the maximum size embedded network structure within an LP problem is NP-hard. The two detection algorithms perform very well computationally and make positive contributions to the known body of results for the embedded network detection. For computational solution a decomposition based approach is presented which solves a network problem with side constraints. In this approach, the original coefficient matrix is partitioned into the network and the non-network parts. For the partitioned problem, we investigate two alternative decomposition techniques namely, Lagrangean relaxation and Benders decomposition. Active variables identified by these procedures are then used to create an advanced basis for the original problem. The computational results of applying these techniques to a selection of Netlib models are encouraging. The development and computational investigation of this solution algorithm constitute further contribution made by the research reported in this thesis.

Dedicated to my family

Acknowledgements

I would like to thank to my supervisor Professor Gautam Mitra for his continuous and enthusiastic support, guidance and many discussions of research topics.

I wish to also thank to Professor Istvan Maros for his generous advice, guidance and careful comments on my analytical as well as written contribution.

I would like to sincerely thank to Doctor Gregory Gutin for helpful advices, guidance and friendly encouragement throughout years.

I also would like to thank to all members of mathematical programming research group who have always been friendly and kind to me and made my experience of research in the Department of Mathematics and Statistics enjoyable one.

I am also deeply grateful to Turkish Educational Council and Muğla University for the financial support to complete this research.

Contents

List of Figures v

List of Tables vi

List of Symbols ix

List of Abbreviations ix

Terminology and Notation xii

1 Structure Analysis in Linear Programming: Identification and Solution Methods 1

1.1 Background to Optimisation . . . 1

1.2 The Linear Programming Problem . . . 3

1.3 Network Flow Problem . . . 4

1.4 Special Structures in LP Constraints . . . 6

1.5 Exploiting Structures in LPs . . . 8

1.6 A Need For an Automatic Structure Extraction Method . . . 10

1.7 An LP Problem with Embedded Pure Network Structure (LPEN) . . . 11

1.8 Analysis of LP Models . . . 14

1.9 Structure of the Thesis . . . 21

PART 1: Detection of Embedded Pure Network Structures 21

2 Review of Algorithms for Network Extraction 22

2.1 Introduction . . . 22

2.2 Problem Statement 23
2.3 Preprocessing 24
2.4 Deletion and Addition Based Methods for Embedded Network Detection 28
2.4.1 Terminology 30
2.4.2 The Row Scanning Deletion Algorithm 31
2.4.3 The Column Scanning Deletion Algorithm 33
2.4.4 The Row Scanning Addition Algorithm 34
2.4.5 An Example (Row Scanning Deletion Algorithm) 36
2.5 GUB Based Methods for Embedded Network Detection 37

3 A New GUB Based Algorithm for Network Extraction 39
3.1 Introduction 39
3.2 Terminology 40
3.3 A Multi-stage GUB Based Algorithm (M-GUB) 41
3.4 Using the Merit Counts For Detecting GUB 42
3.4.1 An Example (M-GUB with Merit Count) 44
3.5 An Independent Set Algorithm For Detecting GUB 46
3.5.1 An Example (M-GUB with an Independent Set) 48
3.6 Computational Results 50
3.7 Discussion 55

4 A Network Extraction Algorithm By Using Generalised Signed Graphs 56
4.1 Introduction 56
4.2 Generalised Signed Graphs 57
4.3 Balanced Signed Graphs 58
4.4 Embedded Networks and Generalised Signed Graphs 59
4.5 Pure Network Graphs 61
4.6 A Network Extraction Algorithm: GSG 63
4.7 Complexity of the Algorithm 65
4.8 An Example with the GSG Algorithm 66
4.9 Computational Results 68
4.10 Discussion 71

PART 2: Computational Solution of Embedded Network Problems Using Lagrangean Relaxation and Benders Decomposition **71**

5 Alternative Approaches For Solving LPEN Problems **72**

5.1 Introduction 72
5.2 Problem Statement 73
5.3 Specialised Simplex Methods 74
5.4 Decomposition Algorithms 80
5.5 An Advanced Basis Method For Solving LPEN Problems 82

6 Lagrangean Relaxation and Benders Decomposition **87**

6.1 Introduction 87
6.2 Lagrangean Relaxation 88
6.2.1 Basic Methodology 88
6.2.2 A Lagrangean Relaxation of the LPEN Problem 89
6.2.3 Determination of the Lagrangean Multipliers 91
6.2.4 Solving Lagrangean Relaxation of the LPEN Problem 94
6.3 Aggregation of Side Constraints 96
6.4 Benders Decomposition 99
6.4.1 Theoretical Framework 99
6.4.2 Benders Decomposition For the LPEN Problem 102

7 Solving an LP with an Embedded Network Structure: Advanced Basis **104**

7.1 Introduction 104
7.2 Unit Starting Basis and Crash Procedures 105
7.2.1 The Crash CLTSF Procedure 109
7.3 Constructing An Advanced Basis with Lagrangean Relaxation 111
7.4 Constructing An Advanced Basis with Benders Decomposition 112
7.5 Computational Results 113
7.5.1 Results with Lagrangean Relaxation 114
7.5.2 Results with Benders Decomposition 116
7.5.3 Consolidated Results 119

7.6 Discussion 121

8 Summary Conclusions and Future Directions 122

8.1 Summary of Contributions 122

8.2 Suggestions for Further Work 124

References 125

List of Figures

3.4.1 The network detected within the LP problem. 45

3.5.2 The undirected graph of the LP problem. 49

4.3.1 Balanced and unbalanced signed graphs. 59

4.8.2 The generalised signed graph of the LP coefficient matrix. 67

4.8.3 The generalised signed graph after the W-reflection. 67

5.5.1 Schematic diagram of LPEN solution method using advanced NSSX basis. 85

List of Tables

1.8.1 The characteristics of models, the embedded network and GUB structures. 16

1.8.2 The characteristics of models, the embedded network and GUB structures. 17

1.8.3 The characteristics of models, the embedded network and GUB structures. 18

1.8.4 The characteristics of models, the embedded network and GUB structures. 19

1.8.5 The characteristics of models, the embedded network and GUB structures for supply chain problems. 19

1.8.6 The qualitative analysis of industrial models 20

2.3.1 The number of inessential rows and inessential columns. 26

2.3.2 The number of essential rows and essential columns. 29

3.6.1 The number of network rows detected by the two-stage GUB based algorithms. 52

4.9.1 The number of network rows and network extraction time for Netlib models. 69

4.9.2 The number of network rows for supply chain models. 71

7.2.1 Row and column priorities. 110

7.5.2 Results with the Lagrangean relaxation method. 115

7.5.3 Results with Benders decomposition with one cut. 117

7.5.4 Results with Benders decomposition with no cut 118

7.5.5 The consolidated results. 120

List of Symbols

$\forall$	For all
$\cup$	Union
$\cap$	Intersection
$\subseteq$	Subset
$\subset$	Proper subset
$\emptyset$	Empty set
z^T	Transpose of z
$\in$	Element of a set
x, y	Decision variables
g_i	Gradient of row i
P_i	Penalty of row i
G_i	The ith GUB subset
$\Re$	The set of real numbers
N	Set of network row indices
$\mathcal{N}$	Node-arc incidence matrix
E	Set of eligible (essential) rows
C	Set of eligible (essential) columns
P	Feasible point set, constraint set
λ	The vector of Lagrangean multipliers
z^*	Optimum value of a given linear programming problem
A	Coefficient matrix of a linear programming problem
a_{ij}	A non-zero element of the coefficient matrix in row i and column j
$\tilde{A}$	Ternary matrix
$\tilde{a}_{ij}$	A non-zero element of the ternary matrix in row i and column j

v_i	A vertex i in a graph
W	A subset of vertices
$\overline{W}$	The complement set of W
T	Tree subgraph for a graph
S	An independent set of a graph
$\mathcal{V}$	Set of vertices in a graph
$\mathcal{A}$	Set of arcs in a directed graph
$\mathcal{E}$	Set of edges in an undirected graph
ω_k	Maximum merit count for row k
RP_i	The reflected row penalty of row i
$p := q$	p is "assigned the value" q
$c(v_i v_k)$	Weight of an edge $v_i v_k$
$\mathcal{G}, \mathcal{H}$	Alternative symbols to represent a graph
$\mathcal{G}[W]$	Subgraph of $\mathcal{G}$ induced by W
$e_j = v_i v_k$	An edge j linking vertices v_i and v_k in an undirected graph
$\mathrm{a}_j = v_i v_k$	An arc j linking vertices v_i and v_k in a directed graph
m_{ij}	The merit count of non-zero element in row i and column j
$\alpha(\mathcal{G})$	Cardinality of a maximum independent set of graph $\mathcal{G}$
$\nu(A)$	Maximum number of network rows in the coefficient matrix A
$\mu(\mathcal{G})$	Number of edges with negative weights in graph $\mathcal{G}$
$I_i(N)$	Set of rows in which each variable is a network variable
$I_i(\overline{N})$	Set of rows in which each variable is a non-network variable
RC_i	The number of non-zero elements in row i, row count
CC_j	The number of non-zero elements in column j, column count

List of Abbreviations

LP	Linear Programming
ER	The number of Essential Rows
EC	The number of Essential Columns
GUB	Generalised Upper Bound
SUB	Simple Upper Bound
VUB	Variable Upper Bound
ERS	Exclusive Row Structure
PNF	Pure Network Flow problem
EPN	Embedded Pure Network
SSX	Sparse SimpleX method
GSG	Generalised Signed Graph
PNET	Pure NETwork structure
GNET	Generalised NETwork structure
GVUB	Generalised Variable Upper Bound
TIME	CPU time (seconds)
D-GUB	Double GUB algorithm
ITER	The number of iterations
NETR	Number of NETwork Rows detected
NETC	Number of NETwork Columns detected
GUBR	Number of GUB Rows detected
GUBC	Number of GUB Columns detected
M-GUB	Multi-stage GUB based algorithm
NSSX	Network based advanced basis
CNET	NETwork based Crash procedures
CLTSF	Lower Triangular Symbolic Crash for Feasibility
LPEN	Linear Programming problem with Embedded Network structure
DMEPN	Problem of Detecting Maximum Embedded Pure Network structure

Terminology and Notation

Graph

A graph $\mathcal{G}$ with m vertices (also called nodes) and n edges consists of a vertex set $\mathcal{V}(\mathcal{G}) = \{v_1, v_2, \cdots, v_m\}$ and an edge set $\mathcal{E}(\mathcal{G}) = \{e_1, e_2, \cdots, e_n\}$ where each edge is an unordered pair of vertices. We denote the edge e_i in alternative ways as $v_k v_j$, (k, j) or (v_k, v_j) where $k, j \in \{1, \cdots, m\}$ and $i \in \{1, \cdots, n\}$. If $v_k v_j \in \mathcal{E}(\mathcal{G})$, then v_k and v_j are said to be adjacent. If a graph does not have any parallel edges or loops where the endpoints coincide, then it is called a simple graph; otherwise it is a multigraph.

Directed Graph

A directed graph or digraph is a graph where each edge is an ordered pair of vertices. In other words, if there is a direction specified for the edges connecting the vertices, then a graph is called a directed graph; otherwise, a graph is called an undirected graph. In a digraph $\mathcal{G}$, the link $\mathrm{a}_i = (v_k, v_j)$ is called an arc; the arc set is denoted by $\mathcal{A}$. The vertex v_k is called the tail of a_i and the vertex v_j is the head of a_i, and the direction is denoted as $v_k \rightarrow v_j$ meaning "there is a link from vertex v_k to vertex v_j".

Network

A network is a directed graph whose arcs have associated numerical values of costs, capacities and nodes are suply, demand and as well as transhipment.

Node-arc Incidence Matrix

The node-arc incidence matrix can be portrayed pictorially as a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$ where $\mathcal{V} = \{v_1, \cdots, v_m\}$ is the set of nodes corresponding to rows of the matrix $\mathcal{N}$ and $\mathcal{A} = \{\mathrm{a}_1, \cdots, \mathrm{a}_n\}$ is the set of arcs representing columns of the matrix $\mathcal{N}$. In the matrix $\mathcal{N} = [n_{ij}]$, $i \in \{1, \cdots, m\}$ and $j \in \{1, \cdots, n\}$, each entry is either

(a) $n_{ij} = +1$, if v_i is the tail (initial) vertex of arc a_j,

(b) $n_{ij} = -1$, if v_i is the head (final) vertex of arc a_j, or

(c) $n_{ij} = 0$, if v_i is not connected by arc a_j.

Walk, Path, Cycle

In an undirected graph, a walk of length k is a sequence $v_0, e_1, v_1, e_2, \cdots, e_k, v_k$ of vertices and edges such that $e_i = v_{i-1}v_i$ for all i. A path is a walk with no repeated vertex. A walk with the first vertex v_k and the last vertex v_j is called closed if $v_k = v_j$. A cycle is a closed walk of length at least one in which the first vertex is the same as the last vertex and this is the only vertex repetition. A loop is a cycle of length one.

Complete Graph

A complete graph or a clique is a simple graph in which for every pair of vertices there exists an edge between them.

Connected Graph

A graph $\mathcal{G}$ is connected if for each pair $v_i, v_k \in \mathcal{V}(\mathcal{G})$ there exists a path, v_i, v_k-path, connecting vertex v_i to vertex v_k. Otherwise, it is a disconnected graph.

Independent Set

A subset S of the vertex set $\mathcal{V}(\mathcal{G})$ is called an independent set of the graph $\mathcal{G}$ if no two vertices of S are adjacent in $\mathcal{G}$. An independent set with maximum cardinality is said a maximum independent set. The number of vertices in a maximum independent set of $\mathcal{G}$ is called the independence number of $\mathcal{G}$ and is denoted by $\alpha(\mathcal{G})$.

Isomorphic

An isomorphism from $\mathcal{G}$ to $\mathcal{H}$ is a bijection $f : \mathcal{V}(\mathcal{G}) \to \mathcal{V}(\mathcal{H})$ such that $v_iv_k \in \mathcal{E}(\mathcal{G})$ if and only if $f(v_i)f(v_k) \in \mathcal{E}(\mathcal{H})$. The graph $\mathcal{G}$ is said to be isomorphic to $\mathcal{H}$, and written $\mathcal{G} \cong \mathcal{H}$.

Subgraph, Spanning Subgraph

A subgraph of a graph $\mathcal{G}$ is a graph $\mathcal{H}$ such that $\mathcal{V}(\mathcal{H}) \subseteq \mathcal{V}(\mathcal{G})$, $\mathcal{E}(\mathcal{H}) \subseteq \mathcal{E}(\mathcal{G})$ and every endpoint of an edge in $\mathcal{E}(\mathcal{H})$ is in $\mathcal{V}(\mathcal{H})$. This is denoted as $\mathcal{H} \subseteq \mathcal{G}$ and means that "$\mathcal{G}$ contains $\mathcal{H}$". If the graph $\mathcal{H}$ is a subgraph of $\mathcal{G}$, then $\mathcal{G}$ is a supergraph of $\mathcal{H}$.

A spanning subgraph (or spanning supergraph) of $\mathcal{G}$ is a subgraph (or supergraph) $\mathcal{H}$ with $\mathcal{V}(\mathcal{H}) = \mathcal{V}(\mathcal{G})$.

Tree, Spanning Tree

A tree is a connected graph which contains no cycles. A spanning tree is a spanning graph that is a tree.

Forest

A forest is a graph G with no cycles, that G is not necessarily connected. Every component of a forest is a tree.

Degree

The degree of a vertex v_i in graph $\mathcal{G}$ is the number of edges of $\mathcal{G}$ incident with v_i.

Induced Subgraph

Suppose that $\mathcal{V}'$ is a nonempty subset of $\mathcal{V}$. The subgraph of $\mathcal{G}$ whose vertex set is $\mathcal{V}'$ and whose edge set is the set of those edges of $\mathcal{G}$ that have both ends in $\mathcal{V}'$ is called the subgraph of $\mathcal{G}$ induced by $\mathcal{V}'$ and is denoted by $\mathcal{G}[\mathcal{V}']$.

Chapter 1

Structure Analysis in Linear Programming: Identification and Solution Methods

1.1 Background to Optimisation

Optimisation is broadly defined as the mathematical problem of finding the minimum or maximum of a function $f(x)$ and the corresponding set of values of the vector $x = (x_1, x_2, ..., x_n)$. The variables x_j may be allowed to take on any value in $\Re^n$, that is $-\infty < x_j < \infty$ for $j = 1, \cdots, n$, whereupon the problem is called an unconstrained optimisation. The general unconstrained optimisation problem may be stated as

$$\begin{gathered}\text{Minimise (Maximise) } f(x)\\ \text{subject to}\\ x \in \Re^n\end{gathered} \tag{1.1}$$

Alternatively, if the variables take values which are restricted to a point set P, that is $x_j \in \Re^n \cap P$, then the problem is called a constrained optimisation problem. The general constrained optimisation problem may be stated as

$$\begin{gathered}\text{Minimise (Maximise) } f(x)\\ \text{subject to}\\ x \in \Re^n \cap P\end{gathered} \tag{1.2}$$

In (1.2), the point set P specifies restrictions on x, that is $P = \{x \mid g_i(x) = b_i,\ i = 1, \cdots, l \text{ and } g_i(x) \geq b_i,\ i = l+1, ..., m\}$; the restrictions include both equality and inequality constraints. The function $f(x)$ is called an objective function. The problem is that of finding a point $x^\star$, out of all points which satisfy the constraints $g_i(x^\star) = b_i$ for $i = 1, ..., l$ and $g_i(x^\star) \geq b_i$ for $i = l+1, ..., m$ which minimises (maximises) the objective function, that is $z^\star = f(x^\star)$. Any point $\bar{x}$ which satisfies all constraints ($\bar{x} \in P$) is said to be feasible. If the set P is empty, that is $P = \emptyset$, then it is not possible to find an x such that $x \in P$ and the given problem has no feasible solution. A maximisation problem can be easily converted to a minimisation problem using the simple transformation

$$\text{Maximum } f(x) = -[\text{Minimum} - f(x)] \tag{1.3}$$

The constrained optimisation problems are traditionally divided into two categories. The first is the ubiquitous linear programming (LP) in which the constraints and the objective function are linear. The second category includes non-linear optimisation in which at least one constraint and/or the objective function is non-linear. In this study, we only consider the linear case.

From a practical and computational point of view, optimisation problems are often loosely categorised into three groups: small, medium, and large scale. Optimisation methods which are the most appropriate for small and medium scale problems are not necessarily appropriate for large scale problems. A particular characteristic of most large scale problems is that the constraints are usually sparse, that is most of the coefficients associated with a given variable are zero. Thus, algorithms for solving large scale optimisation problems need to exploit this sparsity and any recognisable structure within the problem. The solution of such problems presents a real challenge to practitioners who need to develop efficient and practical computational methods.

This study addresses large scale LP problems with embedded pure network structures and focuses on the detection of the embedded network structure and its solution methods. In this chapter, we progressively introduce LP problems with embedded pure network structures. The rest of this chapter is organised in the following way. In section

1.2, we introduce an LP problem stated in a general form. In section 1.3, a network flow problem is defined and its properties are described. In section 1.4, we consider structure constraints as they arise in LPs. In section 1.5, the relevance of exploiting network structures in LPs is emphasised. In section 1.6, we explain the necessity of an automatic network extraction algorithm. In section 1.7, we define LP problems with embedded pure network structures (LPEN) and two different problem statements are given. Section 1.8 presents an analysis of a selection of applied models which appear in the literature. In section 1.9, we provide an outline of the thesis.

1.2 The Linear Programming Problem

The linear programming problem forms an important part of optimisation problems in which all functions f, g_i in (1.2) are linear. LPs arise in many real life problems such as scheduling of work and transportation, maximising profits, minimising costs and network optimisation. A wide range of representative applications is discussed in [96]. The general LP problem can be stated as follows;

$$\begin{gathered}
\text{Minimise } f(x) \\
\text{subject to} \\
\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, \cdots, l \\
\sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = l+1, \cdots, m \\
-\infty < x_j < +\infty, \quad j = 1, \cdots, k \\
x_j \ge 0, \quad j = k+1, \cdots, n.
\end{gathered}$$

The LP problem may be presented in several equivalent forms [32]. The most common is the standard form in which all constraints are stated as equalities:

$$\begin{gathered}
\text{Minimise } f(x) \\
\text{subject to} \\
Ax = b \\
x \ge 0
\end{gathered}$$

where the matrix A is called a constraint or coefficient matrix.

1.3 Network Flow Problem

A mathematical model of a network describes a system where the flow of some resource is conserved; the system is organised into a set of sites called nodes where a resource may be distributed or accumulated [8]. The resource may be transferred within the system from one site to another following a set of directed arcs. If for every arc, one unit of flow on the arc decreases the amount of resource at the origin node by one unit of resource and increases the amount of resource at the destination node by one unit, then the network is called a *pure network*. In a less restrictive form of the network called a *generalised network*, there need not be a one-for-one transfer of resource from one node to another on every arc. That is, every column in the constraint matrix has at most two non-zero elements, but the values of these two elements are not necessarily unit but finite. Such a network is also known as a graph with gains.

Consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ which consists of a set of nodes $\mathcal{V} = \{1, \cdots, m\}$ and a set of arcs $\mathcal{A} = \{(i,j) : i,j \in \mathcal{V}\}$ joining pairs of nodes in $\mathcal{V}$. The pure network flow (PNF) problem or minimum cost network flow problem is stated as follows

$$\begin{gathered}\text{Minimise } z = \sum_{(i,j)\in\mathcal{A}} p_{ij} y_{ij} \\ \text{subject to} \\ \sum_{i\in I(k)} y_{ik} - \sum_{j\in O(k)} y_{kj} = b_k, \ \ \forall \ k \in \mathcal{V} \\ l_{ij} \le y_{ij} \le u_{ij}, \ \ \forall \ (i,j) \in \mathcal{A}\end{gathered} \tag{1.4}$$

where $I(k) = \{i \in \mathcal{V} : (i,k) \in \mathcal{A}\}$ and $O(k) = \{j \in \mathcal{V} : (k,j) \in \mathcal{A}\}$.

An arc $a_k = (i,j)$ is said to be directed from node i and directed to node j. In this sense, the set of tail nodes of arcs that are directed (in) to node k is $I(k)$ and the set of head nodes of arcs that are directed (out of) from node k is $O(k)$. The cardinality of $\mathcal{V}$ is denoted by m and that of $\mathcal{A}$ is denoted by n. There are altogether n decision variables y_{ij} (in alternative notation this can be represented as x_j) which represent the number of units of flow in the arc (i,j). For each node $k \in \mathcal{V}$, a constant b_k represents the requirement at that node; a node for which $b_k > 0$ is called a supply node, a node for which $b_k < 0$ is called a demand node, and a node for which $b_k = 0$ is called a transhipment node.

The constraints in (1.4) are called *flow conservation* (*nodal balance* or *Kirchoff*) equations which indicate that the flow may be neither created nor destroyed in the network. These equations require that the net flow out of node k, that is the total flow into the node k minus the total flow out of node k, should balance the supply or demand, b_k, of node k. In a network, total supply is assumed to equal to total demand, that is $\sum_{k \in \mathcal{V}} b_k = 0$. The vector p represents the costs for per unit flow along the corresponding arcs. The last inequality in (1.4) is called the capacity restriction for each arc.

The PNF problem is a special class of the LP problem with a constraint matrix made up of the node-arc incidence matrix (see the definition). The PNF problem can be stated in more compact form as follows;

$$\text{Minimise } \{cx, \ \text{ subject to : } \ \mathcal{N}x = r, \ \ l \leq x \leq u\},$$

and has the following special properties (see [105]);

- each column in the coefficient matrix $\mathcal{N}$ has at most two unit entries which are of opposite sign, that is at most one $+1$ and at most one -1 entry,
- a non-singular basis corresponds, in a natural way, to a spanning tree, therefore, it has a triangular structure,
- a non-singular basis is unimodular; that is the determinant of the basis matrix is either $+1$ or -1,
- there is an optimal integral flow if the flow capacities and node requirements are integral due to the total modularity of the constraint matrix.

These properties lead to some special additive algorithms which have low order polynomial complexity [1]. We consider the following example of the PNF problem.

Minimise $z = \sum_{j=1}^{7} x_j$

subject to

$x_1 + x_2 = 4$

$x_3 + x_4 = 8$

$-x_1 - x_4 - x_5 + x_7 = -6$

$-x_2 - x_3 - x_6 - x_7 = -16$

$x_5 + x_6 = 10$

$0 \le x_j \le 8 \;\; j = 1, \cdots, 7$

The LP representation of the above PNF problem in terms of the node-arc incidence matrix $\mathcal{N}$ is set out as follows.

$$\text{Minimise} \quad \overset{\text{c x}}{\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}} \; x$$

$$\text{subject to} \quad \overset{\mathcal{N}}{\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & -1 & -1 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}} \overset{x}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}} = \overset{b}{\begin{bmatrix} 4 \\ 8 \\ -6 \\ -16 \\ 10 \end{bmatrix}}$$

$$0 \le x^T = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)^T \le 8$$

1.4 Special Structures in LP Constraints

Large scale LP models which arise in many practical applications have sparse coefficient matrices and display invariably special structure(s). Over the years, a few of these special structures have proven to be desirable from a computational stand point. The most well known constraint structures (see Gunawardane et al. [54]) are set out below.

1. Simple Upper Bound (SUB),

2. Variable Upper Bound (VUB),
3. Generalised Variable Upper Bound (GVUB),
4. Generalised Upper Bound (GUB),
5. Exclusive Row Structure (ERS),
6. Generalised Network Structure (GNET),
7. Pure Network Structure (PNET).

Simple Upper Bound

Simple upper bounds are a set of rows in the LP coefficient matrix A for which each constraint i has only one non-zero coefficient. For example, $l_j \leq x_j \leq u_j$ where l_j and u_j are lower and upper bounds of the variable x_j. Some or all components of l_j and u_j can be $-\infty$ or ∞, respectively. Particularly, if $l_j = -\infty$ and $u_j = \infty$, then the variable x_j is called a free variable.

Variable Upper Bound

A set of constraints which are of the form $x_j \leq x_k$, $j \neq k$ where x_j may not appear in any other special constraints is called VUBs. However, the variable x_k may appear in some constraints. In this case, the variable x_k is said to be the variable upper bound of x_j.

Generalised Variable Upper Bound

If a set of constraints has a non-negative right hand side value, and every variable with a strictly positive coefficient in each constraint does not have a non-zero coefficient in any other constraint of this set, then the constraint is called a Generalised Variable Upper Bound (GVUB).

Generalised Upper Bound and Exclusive Row Structure

A GUB structure for the coefficient matrix A of an LP problem refers to a subset of rows such that every column of A has at most one unit entry within the subset of rows.

A given GUB row taken from such a subset defines a collection of GUB columns which have unit entries in this row and zero in all the remaining GUB rows in this set. The exclusive row structure is a generalisation of the GUB structure. In this case, we can consider any non-zero entry in these rows instead of $+1$ and -1 entries, for more details see [17].

Generalised Network Structure and Pure Network Structure

Consider the coefficient matrix A of an LP problem and also a subset of constraints. If each column of A in the restricted subset of constraints has at most two non-zero entries of arbitrary sign and value, then the columns of A and the subset of rows comprise a generalised network structure. If these non-zero elements at each column are unit and having of opposite sign, that is each column has at most one $+1$ and at most one -1 non-zero entry in these rows, then the columns of A and the subset of rows comprise a pure network structure. In the rest of this thesis, we consider only pure network structures and often refer to a pure network structure as a network structure.

1.5 Exploiting Structures in LPs

The invention of the simplex method for LP problems by Dantzig in 1947 has ushered in the era of optimisation. Since then, operations researchers have made considerable improvements in methodologies for solving LPs. And yet practitioners have outpaced these developments, as there are many LP models of real problems whose solutions are beyond the capability of the current technology. The challenge today in linear programming remains to scale up and to solve larger and larger models [79].

Large scale LP problems which arise in many practical applications have a sparse constraint matrix and often possess alternative structures such as those described in the previous section. For very large LP problems whose coefficient matrices are made up of a high proportion of rows and columns of a special structure, the direct solution using the classical simplex method is generally computationally expensive and may be impractical on a restricted computer platform. If properly identified (extracted), these structures may be then exploited by an appropriate optimisation method. As many researchers such as Brown and Olson [18] reported, exploiting a special structure within an LP problem can lead to remarkable improvements in the computational solution time. These methods take advantage of the special structure when updating the basis by using either special simplex algorithms or the basis factorisations.

Apart from improvements in the computational solution time, the resulting advantages over the standard simplex method are several. The special operations related to the structure within the LP reduces both the amount of work needed to perform the algorithmic steps and the amount of computer memory to store the essential data. In addition, these special operations can be executed by making extensive use of linked list structures, pointers or logical operations in the iterative steps of LP solution algorithms. For instance, the graph based operations for network structures are easy to implement and perform much faster than other methods. The matrix operations of finding the entering vector and determining the updated dual variable values are performed by tracing paths within the basis graph. Since the graph contains only non-zero entries in the problem (basis), the list procedures eliminate checking or performing unnecessary arithmetic operations on zero elements. In addition, by exploiting the triangular basis properties of such problems, the basis inverse is stored implicitly as a graph. This graph is updated during basis exchange steps by simply changing a few pointers in the list structures.

In the last decade, the growing success of solving pure network flow problems and generalised network flow problems have given motivation to consider methods for solving more general LP problems with embedded network structures. The embedded networks are of interest for the following main reasons. First, the presence of network constraints can suggest a useful interpretation of the LP in terms of some network in the underlying applications. For instance, Greenberg [51] provides a paradigm for explaining the flow variables and dual values in the solution analysis of such problems. Secondly, for very large LP problems whose coefficient matrices are made up of a high proportion of rows and columns of embedded network structures, the direct solution using the classical simplex method may be generally expensive and may be impractical on a restricted platform. These classes of problems can be solved by exploiting the underlying network structure. The idea of exploiting the network structure within LP has a growing importance in mathematical programming since network flow problems can be solved much faster using specialised additive algorithms such as network simplex rather than the state-of-the-art LP codes [1].

1.6 A Need For an Automatic Structure Extraction Method

An embedded structure must be identified before developing a methodology to exploit it in the solution algorithm. We set out the rationale for underlying procedures which can automatically identify the embedded structure. When a human analyst is modelling a linear programming application, he or she generally knows which portion of the LP constraint matrix contains the embedded structure. In this case, the problem still remains as how to specify the portion of the embedded structure to the solver.

One way is for the modeller to explicitly specify which constraints and variables constitute the structure. Another way could be to require the modeller to arrange the LP problem so that the embedded structure is easy to recognise; for instance, the structure could be put at the top of the constraint matrix. As a last way, the modeller can identify the structure through an appropriate row and column ordering. In this case, the modeller can analyse the situation beforehand and specify what kind of ordering will produce the desired structure.

Even though these ways are possible, in many cases it may not be easy to transfer this information to the machine readable format for the data of the optimisation solver. Therefore, there is a need to extend the concept of algebraic modelling languages to allow transformation of the information about the structure. It is unfortunate that no algebraic modelling language is equipped with this extension. The languages which are available in the public domain follow some internal and systematic rules to generate the input data file. The resulting ordering of the structure of rows and columns is very unlikely to display any exploitable structure. Recently, Fragniere et al. [42] proposed such an extension which attempts to create a general scheme for passing a model structure from algebraic modelling to the solver.

Assuming that such tools are not readily available and we have to deal with many legacy models, we can see why an automated structure extraction procedure is a necessity. Firstly, it prevents the burden on the modeller of specifying the location of the structure. Secondly, arranging the LP in a specific way or ordering the rows and columns may not be possible for every LP model, and may sometimes decrease the size of the structure. As suggested by Bixby and Fourer [13], an automated identification of the network structure might be faster and more reliable than the human analyst. In addition, a larger network than the modeller can detect by inspection might be found. Finally, the user should have a tool for transmission of the information about the structure. As a result, in the context of exploiting the structure in a general purpose solution methodology, an automated extraction procedure is necessary since a user might not be familiar with the structure in the LP model.

1.7 An LP Problem with Embedded Pure Network Structure (LPEN)

If an LP model includes a subset of constraints and variables which together define a pure network flow, such a network structure appearing within the LP problem is called an *embedded pure network* (EPN) structure and the remaining constraints are called *side (or coupling) constraints.* The variables that do not represent a flow in the embedded network are called *side variables.* Detecting an EPN structure within the LP problem with constraint matrix A is to find a submatrix $\mathcal{N}$ where $\mathcal{N}$ comprises some rows of A and the reflections of some other rows of A such that every column of $\mathcal{N}$ has at most one $+1$ entry and one -1 entry. The reflection of a row i is the row obtained from i after multiplying every non-zero entry of row i and the right hand side value b_i by -1. We refer to the submatrix consisting of pure network structure as *an embedded network matrix.* The problem of detecting a maximum embedded pure network structure (DMEPN) is to find the maximum number of pure network rows in the coefficient matrix of the LP problem. Consider an LP problem with simple upper bounds in the standard form stated as

$$\begin{gathered}\text{Minimise } c^T x \\ \text{subject to} \\ Ax = b \\ l \leq x \leq u\end{gathered} \tag{1.5}$$

where $A \in \Re^{m \times n}$ and c, x, l, $u \in \Re^n$ and $b \in \Re^m$.

Problem Statement 1 (PS1)

Bixby and Fourer [13] defined an EPN structure within the LP problem as a subset of rows of the coefficient matrix A such that each column intersecting with these rows contains at most two unit entries and are of opposite signs; that is one $+1$ and one -1. Let $\mathcal{N}$ be the matrix representation of the network subset and Q be the submatrix of the other constraints that might have also non-zero elements apart from $+1$ and -1. The EPN problem for the given LP problem may be stated as
PS1:

$$\begin{gathered}\text{Minimise } c^T x \\ \text{subject to} \\ \mathcal{N} x = b' \\ Qx = b'' \\ l \leq x \leq u\end{gathered} \tag{1.6}$$

Problem Statement 2 (PS2)

Glover and Klingman [47] defined the EPN structure within an LP problem in an alternative form. They viewed the EPN problem as an LP problem that has a network structure with additional (side) constraints, which are not network and additional (side) variables, which are not network variables. Glover and Klingman's EPN problem statement is set out as

PS2:

$$\begin{aligned} &\text{Minimise } {c_1}^T x' + {c_2}^T x'' \\ &\text{subject to} \\ &A_{11}x' + A_{12}x'' = b_1 \\ &A_{21}x' + A_{22}x'' = b_2 \\ &l_1 \leq x' \leq u_1 \\ &l_2 \leq x'' \leq u_2 \end{aligned} \tag{1.7}$$

where $A_{11} \in \Re^{p \times q}$, $A_{12} \in \Re^{p \times r}$, $A_{21} \in \Re^{s \times q}$ and $A_{22} \in \Re^{s \times r}$. The remaining vectors are of appropriate dimensions. It is seen that the coefficient matrix A of the given LP problem is partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

where A_{11} displays a pure network structure. A major portion of the LP literature has been devoted to special classes of LP embedded network problems such as

- if $p = q = 0$, then the problem is a standard LP problem,
- if $r = 0$, then it is a pure network flow problem with side constraints (in PS1 form),
- if $r = 0$ and the submatrix A_{11} contains only one non-zero entry in each column, it is a GUB problem with side constraints,
- if $r = s = 0$, then it is a pure network flow problem.

Equivalence of the two Problem Statements

It can be shown that the two different problem definitions of embedded pure networks are equivalent; one can be reformulated as the other by simple algebraic reformulation steps. Consider Glover and Klingman's formulation of the EPN problem PS2. By introducing a vector of free variables, $y = (y_1, y_2, \ldots, y_p)^T$ with zero cost, the above representation may be restated as

$$\begin{gathered}
\text{Minimise } {c_1}^T x' + 0y + {c_2}^T x'' \\
\text{subject to} \\
A_{11} x' + Iy = b_1 \\
A_{12} x'' - Iy = 0 \\
A_{21} x' + A_{22} x'' = b_2 \\
l_1 \le x' \le u_1 \\
l_2 \le x'' \le u_2 \\
-\infty < y < +\infty
\end{gathered}$$

where I is the identity matrix of dimension $(p \times p)$. In this formulation, the columns corresponding to the non-network variables become linear constraints while the network structure is kept. As a result, the above formulation can be stated as Bixby and Fourer's problem definition PS1 where

$$\mathcal{N} = \begin{bmatrix} A_{11} & I & 0 \end{bmatrix}, \quad Q = \begin{bmatrix} A_{21} & 0 & A_{22} \\ 0 & -I & A_{12} \end{bmatrix}, \quad x = \begin{bmatrix} x' \\ y \\ x'' \end{bmatrix},$$

$$b'' = \begin{bmatrix} b_2 \\ 0 \end{bmatrix}, \quad c^T = \begin{bmatrix} c_1 & 0 & c_2 \end{bmatrix}, \quad \text{and } b' = b_1.$$

1.8 Analysis of LP Models

An embedded network structure arises in many applications including production scheduling, distribution, facility location and personnel assignment and is of importance to many industries including transportation, energy and distribution. In order to demonstrate the benefits of exploiting the EPN structure within LP problems, we analyse models that are widely accessible and represent a diverse set of applications. Therefore, many of our test problems have been drawn from the Netlib library [43] which contains LP problems that are generally accepted as benchmarks for the development of LP solution methods. The characteristics of the test problems are summarised in terms of the number of constraints (under the heading ROWS), the number of variables (under the heading COLUMNS) and the number of non-zero entries (under the heading NONZEROS) in Tables 1.8.1, 1.8.2, 1.8.3 and 1.8.4.

We also consider a set of three instances of a large scale industrial model taken from the domain of supply chain planning [80]. These models are instances of a multi-stage multi-period production and distribution model and are known to posses a large proportion of embedded network rows. The problem statistics of these models are presented in Table 1.8.5.

For the purpose of this thesis, we restrict our attention to the exploitation of only EPN structure within LP models. However, we can also easily extract a GUB structure since some of our network extraction methods are based on the identification of GUB structures. Therefore, we analyse models in the literature in terms of not only the network structure but also the GUB structure. The results are presented in terms of the number of network rows (under the heading NETR) and the number of network columns (under the heading NETC), namely those that have at least one non-zero entry in the network rows, as well as the number of GUB rows (under the heading GUBR) and the number of GUB columns (under the heading GUBC) in Tables 1.8.1, 1.8.2, 1.8.3, 1.8.4 and 1.8.5. These results show that many real life LP problems include a certain number of network rows and network columns. These results vindicate that it is worthwhile to investigate special algorithms which detect and exploit the EPN structure.

MODELS	ROWS	COLUMNS	NONZEROS	NETR	NETC	GUBR	GUBC
25fv47	822	1571	11127	199	781	141	542
adlittle	57	97	465	29	81	28	80
afiro	28	32	88	15	28	14	26
agg2	517	302	4515	62	119	36	89
agg3	517	302	4531	62	119	36	89
bandm	306	472	2659	74	204	56	169
beaconfd	174	262	3476	88	206	83	204
bgprob	1650	1425	8952	652	1089	375	821
blend	75	83	521	19	59	14	56
bnl1	644	1175	6129	255	676	184	630
bnl2	2325	3489	16124	1284	2459	820	2148
boeing1	351	384	3865	95	268	70	197
boeing2	167	143	1339	38	131	31	123
bore3d	234	315	1525	78	139	57	122
brandy	221	249	2150	39	147	31	121
capri	272	353	1786	70	210	47	175
cre-a	3517	4067	19054	803	3291	675	3173
cre-c	3069	3678	16922	718	3224	607	3147
cycle	1904	2857	21322	505	2349	392	1295
czprob	930	3523	14173	718	3101	702	3086
d2q06c	2172	5167	35674	758	2904	567	2315
d6cube	416	6184	43888	38	781	27	566
degen2	445	534	4449	189	489	180	480
degen3	1504	1818	26230	620	1665	579	1654
dfl001	6072	12230	41873	2922	9659	2073	8114
disp4l	2004	1747	5904	725	1305	632	1252
e226	224	282	2767	76	196	60	164

Table 1.8.1: The characteristics of models, the embedded network and GUB structures.

MODELS	ROWS	COLUMNS	NONZEROS	NETR	NETC	GUBR	GUBC
energy	2263	9799	29063	1486	7643	1086	4735
etamacro	401	688	2489	98	416	68	391
fffff800	525	854	6235	97	755	83	688
finnis	498	614	2714	199	348	136	272
forplan	162	421	4916	30	240	20	240
ganges	1310	1681	7021	526	1299	293	921
gfrd-pnc	617	1092	3467	459	1047	276	918
greenbea	2393	5405	31499	881	4363	766	2574
greenbeb	2393	5405	31499	880	4253	767	2582
grow15	301	645	5665	15	75	8	47
grow22	441	946	8318	22	110	11	65
grow7	141	301	2633	7	35	4	23
israel	175	142	2358	18	28	13	26
kb2	44	41	291	11	33	8	30
ken7	2426	3602	11981	1186	2559	788	2258
kl02	71	36699	212536	33	36699	17	36699
l30	2701	15380	51169	226	4490	226	4490
lotfi	154	308	1086	72	232	50	224
maros	847	1443	10006	286	1264	199	797
modszk1	688	1620	4158	113	537	104	505
nesm	663	2923	13988	190	1635	161	1512
notch	217	204	636	108	204	102	204
pds-2	2953	7535	16390	2148	7155	1254	5687
perold	626	1376	6026	139	578	100	525
pilot4	411	1000	5154	105	409	101	404
pilot87	2031	4883	73804	304	905	265	875

Table 1.8.2: The characteristics of models, the embedded network and GUB structures.

MODELS	ROWS	COLUMNS	NONZEROS	NETR	NETC	GUBR	GUBC
pilot	1442	3652	43220	253	833	193	736
pilot.ja	941	1988	14706	192	754	152	702
pilot.we	723	2789	9218	201	1044	154	988
pilotnov	976	2172	13129	198	828	153	718
recipe	92	180	752	44	112	30	96
sc105	106	103	281	41	74	33	70
sc205	206	203	552	77	141	64	136
scagr25	472	500	2029	270	375	213	305
scagr7	130	140	553	72	105	60	89
scfxm1	331	457	2612	104	341	91	300
scfxm2	661	914	5229	208	682	182	600
scfxm3	991	1371	7846	312	1023	273	900
scorpion	389	358	1708	164	260	107	192
scrs8	491	1169	4029	212	950	132	860
scsd1	78	760	3148	39	640	7	224
scsd6	148	1350	5666	74	1140	15	456
scsd8	398	2750	11334	199	2360	40	944
sctap1	301	480	2052	120	360	120	360
sctap2	1091	1880	8124	470	1410	470	1410
sctap3	1481	2480	10734	620	1860	620	1860
seba	516	1028	4874	134	355	103	285
share1b	118	225	1182	37	144	31	134
share2b	97	79	730	23	64	18	43
shell	537	1775	4900	479	1475	238	972
ship04l	403	2118	8450	312	2082	280	1872
ship04s	403	1458	5810	224	1334	192	1128

Table 1.8.3: The characteristics of models, the embedded network and GUB structures.

MODELS	ROWS	COLUMNS	NONZEROS	NETR	NETC	GUBR	GUBC
ship08l	779	4283	17085	608	4259	544	3760
ship08s	779	2387	9501	336	2091	272	1600
ship12l	1152	5427	21597	732	5223	636	4500
ship12s	1152	2763	10941	360	2187	264	1464
sierra	1228	2036	9252	672	2016	650	2014
stair	357	467	3857	153	313	87	231
standata	360	1075	3038	165	681	110	528
stocfor1	118	111	474	47	82	43	78
stocfor2	2158	2031	9492	914	1674	824	1636
stocfor3	16676	15695	74004	7028	12998	6354	12682
storm5	3530	6900	20205	1639	4975	1125	4253
truss	1001	8806	36642	498	6864	182	4714
tuff	334	587	4523	99	353	91	331
voids	160	132	446	75	132	64	128
vtp.base	199	203	914	38	89	24	88
wood1p	245	2594	70216	77	885	77	885
woodw	1099	8405	37478	301	3545	293	2281

Table 1.8.4: The characteristics of models, the embedded network and GUB structures.

MODELS	ROWS	COLUMNS	NONZEROS	NETR	NETC	GUBR	GUBC
model1	4740	36277	105875	3755	36166	2580	30191
model2	4450	62366	204954	3306	58836	2154	41724
model3	3692	59907	158650	3306	58836	2154	41724

Table 1.8.5: The characteristics of models, the embedded network and GUB structures for supply chain problems.

We have also carried out a qualitative analysis of industrial LP models which have been reported in the literature. For our analysis, we have considered a representative sample reported by Sharda [96]. This sample includes recently reported applications of LP, mostly since 1980. The results of our analysis are shown in Table 1.8.6.

APPLICATION AREAS	NUMBER OF MODELS CONSIDERED	EPN STRUCTURE DEDUCED
Airlines industry	17	8
Chemical industry	7	6
Coal industry	3	2
Communications, and computer industry	16	6
Iron and steel industry	6	4
Paper and publication industry	4	3
Petroleum industry	8	6
Textile industry	4	2
Transportation	5	5
Production scheduling, inventory control, and planning	8	6
Forestry	5	3
Energy and natural resources	10	6
Health care	7	3
Other industries	8	2
Economic analysis, banking, and finance	19	5
Miscellaneous	18	7

Table 1.8.6: The qualitative analysis of industrial models

1.9 Structure of the Thesis

This thesis is organised in two parts. The first part contains a description of methods used to detect network structures in general LP problems. The second part describes solution techniques which exploit the underlying network structure. In **chapter 2**, we review network extraction algorithms discussed in the research literature. In **chapter 3**, we investigate a multi-stage GUB based algorithm which includes two different approaches for detecting a GUB subset efficiently. The first approach uses the merit count concept and the second one exploits an independent set in the corresponding graph of the coefficient matrix. The computational experiments are reported in this chapter. In **chapter 4**, an EPN detection heuristic is described which uses a generalised signed graph of the corresponding coefficient matrix. We prove that the detection of a maximum EPN structure in an LP problem is NP-hard, even if we restrict ourselves to special classes of the LP problems. The computational results are reported and the algorithm is also compared with a well established network detection algorithm in this chapter. In **chapter 5**, we review alternative solution methods for the LPEN problem and describe the algorithmic framework of our procedure. In this approach, we exploit the EPN structure by first partitioning the problem into a network and a non-network part and then applying a decomposition technique. We consider Lagrangean relaxation and Benders decomposition procedures. In **chapter 6**, we explain the theoretical aspects of these decomposition procedures and show how to apply them to the LPEN problem. The solution of decomposed problems is used to create an advanced starting point for the LPEN problem. This concept of creating an advanced basis is the subject of **chapter 7**. We introduce a network based advanced basis procedure in this chapter and present our computational results of two decomposition methods with different advanced bases. The computational results of applying these techniques to a selection of Netlib models are also reported in this chapter. **Chapter 8** summarises the research results reported in this thesis and concludes with suggestions for further work.

Chapter 2

Review of Algorithms for Network Extraction

2.1 Introduction

A number of researchers have reported alternative algorithms for detecting EPN structures within an LP problem. These range from simple permutations of rows and columns to full (linear) transformations of the coefficient matrix. Generally, entire transformation methods are used to convert the complete coefficient matrix to a node-arc incidence matrix for the pure network; such a network structure is called a *hidden structure.*

Hidden structures were investigated by Schrage [97] and Bixby [15]. Bixby and Cunningham [16] described an algorithm which either converts an LP problem to a PNF problem or shows that such a conversion is impossible. Their algorithm which is based on matroids uses elementary row operations and non-zero variable scaling. Recently, Baston, Rahmouni and Williams [7] introduced another algorithm which exploits the representation of a network by a polygon matrix. In this representation, polygons of a network are given in terms of a spanning tree. The theory of conversion of an entire LP problem to a PNF problem has been underpinned by graph theoretic approaches. However, when the entire conversion fails, few practical results have been achieved to reliably identify a subset of rows which form the network structure. An efficient algorithm for doing so is of considerable value because real life LP models are not usually converted to a PNF problem completely.

Partial transformation methods are designed to look for a large subset of the coefficient matrix which exhibits the network structure, with the presumption that large subsets are more efficiently exploited than small subsets. The problem of detecting an EPN structure of maximum size was shown by Bartholdi [6] to be NP-hard. Therefore, heuristic approaches have in practice been the most effective in finding large embedded network structures although they are not guaranteed to find the largest one. These heuristics are based on simple deletion or addition operations or finding other special structures. We classify them into two categories; the first category includes deletion and addition based methods and the algorithms in the second category are based on the GUB structures (see section 1.4).

The rest of this chapter is organised in the following way. In section 2.2, we introduce the problem statement. In section 2.3, the main preprocessing procedures such as reduction and scaling are introduced. In section 2.4, deletion and addition based algorithms are described. In section 2.5, we review the GUB based methods.

2.2 Problem Statement

We are concerned with an LP problem with m constraints and n bounded variables as set out in (1.5). Certain rows and columns of the sparse coefficient matrix A have no effect on whether any subset is an embedded pure network. Moreover, rows that contain entries other than $+1$, -1 and 0 cannot be considered as network rows. Preprocessing procedures remove and set aside certain number of rows and columns in the coefficient matrix. Having applied the preprocessing procedures, the coefficient matrix A is partitioned into two submatrices. We introduce a submatrix $\tilde{A}$ which has only $+1, -1$ non-zero entries and call this a *ternary matrix*. The matrix formed by the rest of the rows of A which are not included in $\tilde{A}$ (that is rows that consist of non-unit entries) is denoted as $\mathcal{R}$. The LP problem (1.5) may then be restated as

$$\begin{aligned} &\text{Minimise } c^T x \\ &\text{subject to} \\ &\tilde{A}x = b' \\ &\mathcal{R}x = b'' \\ &l \leq x \leq u. \end{aligned} \tag{2.1}$$

Rows and columns of the submatrix $\tilde{A}$ are called *eligible rows* and *eligible columns*, respectively. Network extraction algorithms are applied to only eligible rows and columns. Having applied any network extraction method, the LP problem given in (1.5) or in (2.1) may be stated as

$$\begin{aligned} &\text{Minimise } c^T x \\ &\text{subject to} \\ &\mathcal{N}x = b_1' \\ &\mathcal{S}x = b_2' \\ &\mathcal{R}x = b'' \\ &l \leq x \leq u. \end{aligned} \tag{2.2}$$

It is easily seen that $\tilde{A}$ and b' in (2.1) are partitioned as $\tilde{A} = \begin{bmatrix} \mathcal{N} \\ \mathcal{S} \end{bmatrix}$ and $b' = \begin{bmatrix} b_1' \\ b_2' \end{bmatrix}$, respectively. The matrix $\mathcal{N}$ is an embedded network which is formed by EPN rows and $\mathcal{S}$ is the submatrix formed by the rest of the rows of $\tilde{A}$ which are not included in the network structure.

2.3 Preprocessing

A preprocessing procedure is applied to the coefficient matrix of the given LP problem to make the number of rows and columns of the ternary matrix as large as possible. In this section, we describe the main steps of reduction and scaling procedures which have been used by most researchers as an initial step of the EPN extraction algorithm.

Reduction Procedure

A reduction procedure is designed to reduce the dimension of the coefficient matrix yet maintaining an equivalent LP model. The main idea is to look for many simple instances of inessential rows and columns which need not appear explicitly in the constraints. Reduction methods were originally introduced by Brearley, Mitra and Williams [17].

Simple reduction

A given LP problem may have different types of inessential rows and columns which are listed below. Where appropriate, we assume that MPS format [37] is the method of specifying the LP model.

1. Rows that have only zero elements (ZR),
2. Columns that have only zero elements (ZC),
3. Rows specified as free (type N) in the rows section of the MPS input (FR),
4. Columns that have elements only in the free rows (FC),
5. Columns specified as fixed (type FX) in the bounds section of the MPS input (FXC),
6. Rows that have elements only in the fixed columns (FXR).

A further reduction is based on the observation that if a row has only one non-zero entry and is defined as an equality (type E) in MPS format, then the row only serves to fix a variable. This is denoted as 1-ER. Thus, the row and its one intersecting column, 1-EC, are inessential (ineligible). Table 2.3.1 contains the number of inessential rows and inessential columns computed in this way for a subset of Netlib models and a set of supply chain models.

Reduction with computed bounds

It is also possible to compute lower and upper bounds on the linear forms and use these bounds to fix variables or free rows. Such steps are fully discussed in [17].

	INESSENTIAL ROWS				INESSENTIAL COLUMNS			
Model Names	ZR	FR	FXR	1-ER	ZC	FC	FXC	1-EC
25fv47	—	—	—	27	—	—	—	25
bnl2	45	—	—	15	—	—	—	14
cre-a	89	1	—	6	—	—	—	3
cre-c	83	1	—	15	—	—	—	11
cycle	19	330	—	123	—	—	—	107
czprob	3	—	—	190	—	—	229	190
d2q06c	1	—	—	10	—	—	—	10
d6cube	12	—	—	1	—	—	—	1
energy	26	—	2	—	127	—	391	—
ganges	1	—	—	184	—	—	—	184
greenbea	4	—	—	73	—	—	103	73
greenbeb	4	—	—	75	—	—	115	75
ken7	1	—	—	989	—	—	—	989
pilot	1	—	1	1	—	—	203	1
scfxm3	1	—	—	24	—	—	—	24
sctap2	1	—	—	—	—	—	—	—
sctap	1	—	—	—	—	—	—	—
scrs8	1	—	—	25	—	—	—	25
ship12l	109	—	—	204	—	—	—	204
sierra	—	1	5	—	—	—	20	-
stocfor2	—	—	—	16	—	—	—	16
woodw	1	1	—	—	—	4	—	—
model1	1	—	—	5	—	—	—	5
model2	1	—	290	1	—	—	2459	1
model3	1	—	—	1	—	—	—	1

Table 2.3.1: The number of inessential rows and inessential columns.

These reductions are applied repeatedly to the given model, revealing at each iteration more rows which can be removed or made free, and columns which can be fixed. Thus, the cyclic application of reduction continues until a minimal model results.

Scaling Procedure

Scaling is another preprocessing step which multiplies the rows and columns by scalar weights. Normally, scaling procedures are introduced to improve the numerical stability of the solution procedure. However, the scaling procedure is adopted here to increase the number of eligible rows and eligible columns which have only $+1$, -1 non-zero entries, that is the dimension of the ternary matrix is increased. After scaling, we can set aside the remaining rows which cannot be in the network.

Bixby and Fourer [13] suggested a myopic algorithm to check whether a $+1, -1$ scaling exists for all rows within the coefficient matrix. The algorithm first fixes the scale on any row or column. By repeatedly scanning rows and columns, the other scales for each row and column implied by the requirement that each element be $+1, -1$ are found. If this procedure leads to a contradiction, then no $+1, -1$ scaling for all rows exists. If the procedure terminates before the whole matrix is scanned, then the matrix is not connected. We have developed the following scaling algorithm which is based on the Bixby and Fourer's approach [13]; for more detail see [55] and [58].

Scaling Algorithm

Step 1 *Find a set of essential rows and columns*

Set aside the empty and single element columns. Let E' and C' be sets of rows and columns, respectively, such that for each $j \in C'$ there is $a_{ij} \neq 0$ for at least two rows $i \in E'$.

Step 2 *Scale rows*

For each row $i \in E'$, identify the most frequently occurring row non-zero value of magnitude, α_i such that $\alpha_i \neq 1$. Scale row i by $1/\alpha_i$ to maximise the number of columns $j \in C'$ such that $|a_{ij}| = 1$

Step 3 *Scale columns*

Find a column $j \in C'$ in which multiple non-zero entries have the same magnitude, α_j. Scale column j by $1/\alpha_j$.

Step 4 *Improve the scaling*

For each $i \in E'$, let p_i be the number of columns $j \in C'$ such that $|a_{ij}| \neq 0$ and the smallest p_i for any intersecting row be $p_{min} = \min\{p_i : i \in E'\}$ such that $|a_{ij}| \neq 0$. Find the magnitude $\beta_j \neq 1$ for column j by using the following criterias;

1. the set of row indices $\{i \in E' : \ p_i = p_{min}, |a_{ij}| = \beta_j\}$ is as large as possible,

2. the set of row indices $\{i \in E' : \ p_i = p_{min}, \ |a_{ij}| = \beta_j\}$ has more members than the set $\{i \in E' : \ p_i = p_{min}, \ |a_{ij}| = 1\}$.

Scale column j by $1/\beta_j$.

The results of applying this scaling algorithm to a subset of Netlib models and a set of supply chain models are presented in Table 2.3.2.

2.4 Deletion and Addition Based Methods for Embedded Network Detection

Deletion and addition based methods are designed to extract an EPN structure by deletion or addition of rows or columns. These algorithms are summarised as

1. the row scanning deletion,

2. the column scanning deletion,

3. the row scanning addition.

Both the row scanning deletion and column scanning deletion algorithms start with a full network initialisation, then scan each row and column to delete rows and columns which violate the network structure. The row scanning addition algorithm starts with

	BEFORE SCALING		AFTER SCALING	
Model Names	Essential Rows	Essential Columns	Essential Rows	Essential Columns
25fv47	186	745	224	911
bnl2	1314	2054	1418	2483
cre-a	554	2687	1247	3825
cre-c	573	2433	997	3626
cycle	312	2100	507	2350
czprob	718	3102	719	3102
d2q06c	642	2373	844	3177
d6cube	48	905	87	1733
energy	1427	7489	1531	7693
ganges	581	1456	631	1481
greenbea	443	3994	916	4660
greenbeb	443	3991	914	4650
ken7	1437	2613	1437	2613
pilot	165	548	276	907
scfxm3	360	1080	423	1176
sctap2	470	1410	470	1410
sctap3	620	1860	620	1860
scrs8	39	601	214	955
ship12l	828	5197	828	5207
sierra	1155	2016	1155	2016
stocfor2	1262	765	1262	1698
woodw	228	3151	301	3545
model1	4153	36134	4291	36272
model2	3690	58836	3690	58836
model3	3690	58836	3690	58836

Table 2.3.2: The number of essential rows and essential columns.

an empty network subset, then adds rows into the network subset without destroying the structure until the maximal number of network rows is found.

2.4.1 Terminology

Let E and C denote the sets of essential rows and columns, respectively, which define the ternary matrix $\tilde{A} = [\tilde{a}_{ij}]$ such that $\tilde{a}_{ij} \in \{-1, 1, 0\}$, $i \in E$, $j \in C$, $E \subseteq \{1, 2, ..., m\}$, $C \subseteq \{1, 2, ..., n\}$ and $\tilde{A} \in \Re^{|E| \times |C|}$.

Conflict

Two rows in $\tilde{A}$ are said to be in conflict if there is at least one column of $\tilde{A}$ with non-zero entries of the same sign in both rows. Conflicts prevent the appearance of certain pairs of rows to create an embedded network.

Row penalty

The penalty of row i of the matrix $\tilde{A}$ is defined as the number of conflicts in which it participates. If $c_j{}^+$ and $c_j{}^-$ are respectively the numbers of $+1$'s and -1's in column j, the penalty of row i, P_i, may be formalised as

$$P_i = \sum_{\tilde{a}_{ij}>0} (c_j{}^+ - 1) + \sum_{\tilde{a}_{ij}<0} (c_j{}^- - 1), \quad \forall\, i \in E \tag{2.3}$$

where

$$c_j{}^+ = \sum_{\tilde{a}_{ij}>0} \tilde{a}_{ij}, \quad \text{and } c_j{}^- = \sum_{\tilde{a}_{ij}<0} -\tilde{a}_{ij}, \quad \forall\, j \in C. \tag{2.4}$$

A matrix penalty, h, is the sum of the individual row penalties and is computed as

$$h = \sum_{i \in E} P_i. \tag{2.5}$$

Row reflection

The reflection of row i of the matrix $\tilde{A}$ refers to the multiplication of each element in row i by -1. Row reflection can be seen as a special case of scaling with -1; each non-zero entry in row i is $\tilde{a}_{ij} := -\tilde{a}_{ij}$, for $j \in C$ and the right hand side value is $b_i := -b_i$.

Penalty of row reflection

The penalty for the reflection of row i, RP_i, can be written as

$$RP_i = \sum_{\tilde{a}_{ij}>0} c_j^- + \sum_{\tilde{a}_{ij}<0} c_j^+, \quad \forall j \in C. \tag{2.6}$$

2.4.2 The Row Scanning Deletion Algorithm

The row scanning deletion heuristic was first developed by Brown and Wright [20], [22]. The general idea is to initially consider all rows in the subset of the essential rows as potentially in the embedded network, then to delete one row at a time until the remaining subset is a feasible network.

The algorithm consists of two phases. Phase I attempts to delete rows in order to obtain a feasible set. The measure of infeasibility at any point is either the row penalty or the matrix penalty. The row penalty represents how severely a row conflicts with other rows in the subset. The algorithm iterates for each row which has the penalty such that $P_i > 0$, $i \in E$. The row is deleted or reflected to reduce the penalty at each iteration. When $P_i = 0$, the row i is a network row. The algorithm terminates with zero penalty for all rows in a network subset N. Thus, N becomes the set of row indices of the network matrix $\mathcal{N}$ introduced in (1.5). In Phase II, the deleted rows which do not cause any row conflicts are reconsidered. This leads to a reinsertion algorithm which increases the number of network rows in the set N. We outline the basic steps of this algorithm below.

The Row Scanning Deletion Algorithm

Phase I - Deletion of Infeasible Rows

Step 1 *Initialisation*

Compute c_j^+ and c_j^- from (2.4) for all $j \in C$. Initialise the rows of the network matrix N as the essential rows of $\tilde{A}$, $N := E$.

Step 2 *Compute row penalties*

Compute row penalties from (2.3) for all $i \in N$. If $P_i = 0$ for all $i \in N$, then terminate phase I and go to step 7; otherwise, go to step 3.

Step 3 *Select a row*

Select any $t \in N$ such that $P_t > 0$. Compute the reflected row penalty for each $j \in C$ by using formula (2.6).

Step 4 *Delete or reflect the row*

If $RP_t < P_t$, then reflect the row t and go to step 5. Otherwise, delete row t, then set $N := N \backslash \{t\}$, and go to step 6.

Step 5 *Update c_j^+ and c_j^- for each column*

Update number of non-zero entries for each column as follows and go to step 2.

For each $\tilde{a}_{tj} = 1$ and $j \in C$, set $c_j^+ := c_j^+ - 1$ and $c_j^- := c_j^- + 1$.

For each $\tilde{a}_{tj} = -1$ and $j \in C$, set $c_j^- := c_j^- - 1$ and $c_j^+ := c_j^+ + 1$.

Step 6 *Reduce the number of non-zero entries in each column*

For each $j \in C$, reduce the non-zero counts as follows, and then go to step 2.

If $\tilde{a}_{tj} = 1$, then set $c_j^+ := c_j^+ - 1$.

If $\tilde{a}_{tj} = -1$, then set $c_j^- := c_j^- - 1$.

Phase II - Reinsertion

Step 7 *Select a row for reinclusion*

For row $t \in E \backslash N$;

if $c_j^+ = 0, \forall\, \tilde{a}_{tj} = 1, j \in C$, or $c_j^- = 0, \forall\, \tilde{a}_{tj} = -1, j \in C$, go to step 8,

if $c_j^+ = 0, \forall\, \tilde{a}_{tj} = -1, j \in C$, or $c_j^- = 0, \forall\, \tilde{a}_{tj} = +1, j \in C$, reflect the row t and then go to step 8.

Step 8 *Restore the row*

Set $N := N \bigcup \{t\}$ and $E \backslash N := (E \backslash N) \backslash \{t\}$.

Let $c_j^+ := 1, \ \forall\ \tilde{a}_{tj} = 1$ and $j \in C$, $c_j^- := 1, \ \forall\ \tilde{a}_{tj} = -1$ and $j \in C$.

Step 9 *Terminate the algorithm*

If $E \backslash N = \emptyset$, then terminate the algorithm. Otherwise, go to step 7.

Brown and Wright used the matrix penalty instead of the row penalty and chose the row with maximal penalty, $P_t = \max\{P_i : i \in N\}$, in their algorithm (see [20] and [22]). In addition, they obtained the following sharp upper bound on the network set size for a coefficient matrix with m essential rows: $u = m - \max(c_j^+ + c_j^-)$, $\forall\, j \in C$. Bixby and Fourer [13] reported some modifications of this algorithm to increase the efficiency and effectiveness. One of them is introduced to reduce the cost of a pass in the row scanning algorithm. Rows are considered for deletion or reflection in decreasing order of their original penalties and row penalties are not updated entirely. This ordering remains unchanged throughout the algorithm except to accommodate the reflected rows.

2.4.3 The Column Scanning Deletion Algorithm

This algorithm was first studied by Ahn (see [13]). The idea is that for any column of the coefficient matrix $\tilde{A}$, only two rows from N which have a common non-zero entry in this column may be in the embedded network; the other rows which have a non-zero entry in this column have to be deleted. The deletion algorithm based on scanning columns operates quite differently from the previously described algorithm based on the row scanning.

The column scanning deletion algorithm proceeds as follows. The essential rows and columns (formed after reduction and scaling procedures) are considered. At the beginning, all essential rows are initialised as a network row, $N := E$. Each column is examined once and all but one or two rows intersecting with (having a non-zero entry in) this column are deleted. In this way, the network set N is reduced until all columns are examined or there is no essential row in the set E. After applying the column scanning deletion algorithm, some rows which are already deleted can be restored to increase the number of network rows. This is achieved using the Phase II procedure as in the row scanning deletion algorithm. In this case, c_j^+ and c_j^- are determined from (2.4).

The important aspect of this algorithm is that the algorithm does not reflect rows in the course of scanning a column, since a reflection may create conflicts with columns which have already been scanned. In order to deal with this problem, Bixby and Fourer [13] distinguished new and old rows in the set N. Initially, all rows are new but whenever a column is scanned and two intersecting rows remain undeleted, both rows become old if they are not labelled old already, see [13] for more details. We outline the basic steps of the algorithm. Let L_i be the label of row i as new or old and I_j is the set of all intersecting rows when column j is scanned.

The Column Scanning Deletion Algorithm

Step 1 *Initialise*

Set $N := E$ and label all rows as a new row $L_i := new$, $\forall i \in N$.

Step 2 *Find a set of rows intersecting with the current column*

Consider column $j \in C$ and find a set of rows intersecting with column j as $I_j = \{i \in N : \tilde{a}_{ij} \neq 0\}$.

Step 3 *Choose rows*

If $|I_j| \geq 2$, then choose p, $q \in I_j$ such that $L_p = new$ or $L_p = L_q = old$ and $\tilde{a}_{pj} = 1$, $\tilde{a}_{qj} = -1$. Otherwise choose some $p \in I_j$ and delete all other rows, then set $N := N \backslash \{i \in I_j : i \neq p\}$.

Step 4 *Reflect and delete*

If $\tilde{a}_{pj} = \tilde{a}_{qj}$, then reflect row p and delete other rows. Set $N := N \backslash \{i \in I_j : i \neq p, i \neq q\}$ and $L_p := old$ and $L_q := old$ and then go to step 2 to scan another column.

2.4.4 The Row Scanning Addition Algorithm

An addition algorithm is obviously to add rows to an empty subset which is trivially a network. The algorithm starts with an empty network set and increases the number of network rows without creating conflicts with rows already inserted, so that each column restricted to these rows has only one $+1$ and one -1 non-zero entries. An addition algorithm was first introduced by Brearley, Mitra, and Williams [17] for finding embedded GUB structures within LP problems. Brown, McBride and Wood [21] presented an addition algorithm for finding an embedded generalised network structure. Bixby and Fourer [13] reported some implementation issues to make the algorithm more efficient. Conceptually, the addition algorithm is similar to the reinsertion steps introduced in Phase II of the previous algorithm. In this case, the algorithm starts with an empty set of network rows.

The Row Scanning Addition Algorithm

Step 1 *Initialisation*

Initialise the network set as $N := \emptyset$. For each $j \in C$, set $c_j^+ := 0$ and $c_j^- := 0$.

Step 2 *Add a row*

Choose a row $i \in E$. If $c_j^+ = 0$, for all $\tilde{a}_{ij} = +1$, $j \in C$; and $c_j^- = 0$, for all $\tilde{a}_{ij} = -1$, $j \in C$, then add row i and set $N := N \cup \{i\}$ and go to step 3.

Step 3 *Update c_j^+ and c_j^-*

Update the number of $+1$'s and -1's in each column:

for each $\tilde{a}_{ij} = +1$, $j \in C$, set $c_j^+ := 1$ and

for each $\tilde{a}_{ij} = -1$, $j \in C$, set $c_j^- := 1$.

Step 4 *Reflect and add*

If $c_j^+ = 0$, for all $\tilde{a}_{ij} = -1$, $j \in C$; and if $c_j^- = 0$, for all $\tilde{a}_{ij} = +1$, $j \in C$, then reflect row i and set $N := N \cup \{i\}$. Update c_j^+ and c_j^- like in step 3 and go to step 2.

Recently, Hsu and Fourer [73] have introduced a variation of the addition algorithm which uses the scale factors for each row and column in the network structure. In this approach, the scaling and the detection steps are applied together.

The Augmentation heuristics

Bixby and Fourer [14] introduced heuristics that try to enlarge a network structure that have been found by any of the deletion and addition based methods. They called them *augmentation heuristics*, which are summarised as

- the deletion-driven exchange,
- the insertion-driven exchange,
- the hybrid exchange.

The deletion-driven exchange approach picks a network row that might be deleted, then determines whether one or more non-network rows might be inserted to the network set as a result. The insertion-driven exchange approach first picks a non-network row that might be inserted then determines whether its insertion can be made possible by the deletion of at most one network row. The last approach is the hybridisation of these two approaches and attempts to combine the best features of the preceding two procedures. Note that although augmentation heuristics might improve the results of any extraction heuristic, they might be, however, time consuming.

2.4.5 An Example (Row Scanning Deletion Algorithm)

Consider the following constraints of an LP problem given.

$$\begin{aligned}
r_1 &: \ x_1 + x_2 \geq 8 \\
r_2 &: \ x_2 + x_3 \geq 8 \\
r_3 &: \ x_3 + x_4 \geq 6 \\
r_4 &: \ x_1 + x_4 + x_5 \geq 4 \\
r_5 &: \ x_1 + x_2 + x_5 + x_6 \geq 9 \\
r_6 &: \ x_2 + x_3 + x_6 + x_7 \geq 4 \\
r_7 &: \ x_3 + x_4 + x_7 \geq 7 \\
r_8 &: \ x_4 + x_5 \geq 3 \\
r_9 &: \ x_5 + x_6 \geq 3 \\
r_{10} &: \ x_6 + x_7 \geq 3
\end{aligned}$$

An initial simplex tableau is constructed by adding slack variables $s_1, s_2, \cdots, s_{10}$ and artificial variables $a_1, a_2, \cdots, a_{10}$. The columns associated with the slack variables and the artificial variables are set aside since they can be included at any time to the network matrix. The set of the essential rows is $E = \{r_1, r_2, \cdots, r_{10}\}$. For each $i \in \{1, \cdots, 10\}$, the initial row penalties are $P_1 = 5$, $P_2 = 6$, $P_3 = 6$, $P_4 = 8$, $P_5 = 11$, $P_6 = 11$, $P_7 = 8$, $P_8 = 6$, $P_9 = 6$, $P_{10} = 5$ and the network subset N is initialised as $N = \{r_1, r_2, \cdots, r_{10}\}$. At each iteration, a row is chosen to reflect or delete with the maximum row penalty as $P_t = \max\{P_i : i \in \{1, \cdots, 10\}\}$. In the case of a tie, an arbitrary choice is made. At the first iteration, row $t = 6$ is selected. The reflected row penalty is $RP_6 = 0$. Since $RP_6 < P_6$, the sixth row is reflected, then all row penalties are reduced. By continuing to iterate, the fifth, fourth, and seventh rows are reflected.

Thus, the current row penalties for each row are obtained as $P_1 = 1,\ P_2 = 2,\ P_3 = 2,\ P_4 = 3,\ P_5 = 4, P_6 = 4,\ P_7 = 3,\ P_8 = 2,\ P_9 = 2,\ P_{10} = 1$. Row 5 with the maximum penalty is then chosen. Since $RP_5 = 7$ and $RP_5 > P_5$ the fifth row is deleted. The same procedure is applied to the remaining rows. At the end of Phase I, the network set is obtained as $N = \{r_1, r_3, -r_4, -r_6, r_9\}$. In Phase II, no row deleted in Phase I is reinserted to the network set since all rows cause the conflict with rows in N. Thus, the algorithm terminates with network structure N.

2.5 GUB Based Methods for Embedded Network Detection

The concept of GUB was introduced by Dantzig and Van Slyke [31]. Since then, much work has been done on this subject. The identification of a GUB structure within an LP problem was investigated by Brearley, Mitra and Williams [17]. The automatic identification of a GUB structure in the LP problem was also developed by Brown and Thomen [23]. In this section, we concentrate on detection of embedded pure network rows by making use of the GUB structure.

The Brown and Wright's Approach

Brown and Thomen [23] first proposed a bipartite network flow factorisation by finding two GUB subsets which typically appear in a transportation problem and in an assignment problem. They assumed that rows of the coefficient matrix of a PNF problem are partitioned into two subsets such that each column has only one non-zero entry in a subset and the other entry is in another subset; but additionally the entries are of opposite sign. Clearly, each subset corresponds to a GUB structure. As a result, the EPN structure can be considered to be a paired combination of GUB subsets. This relationship between network and GUB structures motivated Brown and Wright [20] to introduce an automatic EPN identification heuristic which is based on double GUB factorisation called D-GUB.

The algorithm D-GUB proceeds as follows. The first eligible row subset is determined by scaling after the reduction process. The first GUB subset G_1 is found by applying a GUB detection heuristic to a subset of essential rows E_1. Rows in G_1 are then removed from the set E_1. For the second eligible set E_2, each row that is not involved in G_1 is scanned and checked for columns in which the row has non-zero entries. Row reflection is also carried out if necessary in order to create as many essential rows as possible to obtain opposite sign entries. If the set G_1 has no non-zero entries or has one non-zero entry of opposite sign, then the row is an essential row. If G_1 has no non-zero entry or one non-zero entry with the same sign in each column, then the row is a candidate to create the second GUB set in the reflected form. Otherwise, the row is not eligible and can be deleted. The GUB heuristic is then applied to the eligible set E_2 and the second GUB set G_2 is obtained. The network structure is then constructed by considering the rows in G_1 and in G_2. The main steps of the D-GUB algorithm are set out below.

The D-GUB Algorithm

Step 1 Determine a set of essential rows E_1 for the selection as network rows.

Step 2 Apply a GUB heuristic to the eligible set E_1. Find the first GUB subset G_1.

Step 3 Determine the second set of essential rows E_2 from the remaining rows which are not involved in G_1. If $E_2 = \emptyset$, then go to step 6.

Step 4 Reapply the GUB heuristic to rows of E_2. Identify the second GUB subset G_2.

Step 5 Construct the network structure N by combining rows of G_1 and G_2.

Step 6 Terminate the algorithm.

Brown and Wright applied a GUB extraction heuristic which consists of two phases. Phase I attempts to delete as few rows as possible in order to produce a feasible GUB set. Phase II examines rows deleted in phase I and reincludes rows to find as many GUB rows as possible. They implemented the D-GUB algorithm and compared it with the row scanning deletion algorithm. Their computational results showed that D-GUB is inferior for a set of models in terms of the number of network rows detected.

Chapter 3

A New GUB Based Algorithm for Network Extraction

3.1 Introduction

In this chapter, we present a new GUB based algorithm for detecting an EPN structure within the LPEN problem. This heuristic is based on pairing as many GUB subsets as possible. Therefore, we call this a *multi-stage GUB based algorithm*, M-GUB. In order to detect a GUB subset, two different procedures are introduced. The first procedure is based on the notion of the Markowitz merit count concept to exploit the matrix non-zero structures. The second procedure considers the relationship between the GUB structures in an LP problem and the independent sets in the corresponding graph. The GUB structures in this case are detected by using some known independent set heuristics.

The rest of this chapter is organised as follows. In section 3.2, we define the basic terminology. In section 3.3, the multi-stage GUB based algorithm is introduced. In section 3.4, we describe the use of merit count criterion for detecting GUB structures in M-GUB algorithm. In section 3.5, the independent set algorithm which is applied to detect GUB structures is explained. The results of computational experiments are reported in section 3.6. A brief discussion of results is given in section 3.7.

3.2 Terminology

In this section, we give definitions of some terminology which are used in the description of our algorithm. For more detail, the reader is referred to [59].

PN-conflict

Consider the ternary matrix $\tilde{A}$ defined in section 2.2 and recall that $\tilde{a}_{ij} \in \{-1, 0, 1\}$. We say that a pair of rows i and k of the coefficient matrix $\tilde{A}$ is in *PN-conflict* if there exists at least one column j such that the following property holds;

$$\tilde{a}_{ij} = \tilde{a}_{kj} \text{ and } |\tilde{a}_{ij}| = |\tilde{a}_{kj}| = 1.$$

The following structures for $\tilde{A} \in \Re^{2\times 2}$ are examples of two rows in PN-conflict.

$$s_1 : \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad s_2 : \begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix}, \quad s_3 : \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}, \quad s_4 : \begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix}$$

$$s_5 : \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad s_6 : \begin{bmatrix} -1 & 0 \\ -1 & -1 \end{bmatrix}$$

Clearly, an embedded network structure in the coefficient matrix $\tilde{A}$ is a set of rows such that no row is in PN-conflict with any other row in this set. It can be easily shown that a matrix is a pure network matrix if and only if no pair of rows in the matrix is in PN-conflict. To establish this, it suffices to show that if no pair of rows of the matrix is in PN-conflict, then it is a network matrix. Assume that for a network matrix it is not so. Then there is a column with non-zeros of the same sign. The corresponding rows are clearly in PN-conflict which contradicts the assumption of a network matrix.

GUB-conflict

We say that two rows of the coefficient matrix $\tilde{A}$ are in *GUB-conflict* if they have two non-zero entries in the same column. Clearly, a GUB structure is a set of rows, such that no pair of rows from this set is in GUB-conflict.

Merit Count

Let $\tilde{a}_{ij} \neq 0$ be an arbitrary non-zero element of the coefficient matrix $\tilde{A}$. The merit count of this element, m_{ij}, is defined as

$$m_{ij} = (RC_i - 1)(CC_j - 1), \tag{3.1}$$

where RC_i and CC_j are the number of non-zero elements in row i and in column j, respectively. The numbers RC_i and CC_j are often referred to as row counts and column counts.

3.3 A Multi-stage GUB Based Algorithm (M-GUB)

We extend the D-GUB algorithm considered in [20] to more than two, that is multiple GUB structures, and call this a multi-stage GUB based algorithm, M-GUB. This algorithm finds as many GUB subsets as possible to create a large number of network rows [59]. In this section, we describe the M-GUB algorithm; the two procedures for finding GUB structures are discussed in section 3.4 and section 3.5.

The M-GUB algorithm proceeds as follows. Initially, the network subset is assumed to be empty. At each stage, the current network set is constructed by adding the new GUB structure to the previous network subset. A GUB set G_1 of rows is extracted from E_1. The first network structure N_1 coincides with G_1. The second eligible set E_2 consists of rows r_i (or their reflections) in E_1 but not in N_1, such that either r_i or its reflected form $-r_i$ is not in PN-conflict with any row in N_1. If r_i is in PN-conflict with a row in N_1 but $-r_i$ is not in PN-conflict with any row in N_1, then $-r_i$ rather than r_i belongs to E_2.

The second GUB set G_2 is extracted from E_2. The second network structure N_2 is set to be $N_1 \cup G_2$. Subsequently, the third eligible set E_3 is constructed similarly to E_2 and the third GUB subset G_3 is extracted from E_3. The third network structure N_3 is set to be $N_2 \cup G_3$. The algorithm repeats the above operations until the current E_i is empty. Then the corresponding network structure N_{i-1} includes the maximum number of pure network rows within the scope of the M-GUB algorithm. If the algorithm is restricted to obtaining the network set N_2 only, then this becomes equivalent to the D-GUB algorithm (see section 2.5). Our M-GUB algorithm is a generalisation of the D-GUB algorithm and is stated below.

The Multi-stage GUB Algorithm: M-GUB

```
Step 1 Determine the initial eligible row set $E_1$ and network set $N_0 = \phi$.
Step 2 Extract the initial GUB subset $G_1 \subseteq E_1$, and the first network
subset is $N_1 = G_1$.
Repeat for $k = 1, 2, 3, \cdots$
    Step 3 Determine the set $E_{k+1}$ of the remaining eligible rows not in
    PN-conflict with $G_1 \cup \ldots \cup G_k$ using the row reflection if necessary.
    Step 4 If $E_{k+1}$ is empty, then go to step 7.
    Step 5 Extract a GUB subset $G_{k+1} \subseteq E_{k+1}$.
    Step 6 Construct the pure network set $N_{k+1}$ by appending the rows of
    $G_{k+1}$ to the previous network set $N_k$, as $N_{k+1} = N_k \cup G_{k+1}$.
Step 7 Terminate the algorithm.
```

3.4 Using the Merit Counts For Detecting GUB

It is now well established that the concept of merit count as introduced by Markowitz [81] (also see Duff et al. [36]) plays an important role in identifying sparse matrix structures. Within LP, the merit count has been used to reduce the non-zero growth during basis factorisation [90].

In the M-GUB algorithm, we use the merit count concept to establish possible row and column interactions and to make a choice for the current GUB subset. The main concern is to choose a GUB row out of the rows in the current eligible subset such that the chosen row does not prevent a large number of other candidate rows from inclusion into subsequent GUB sets. We illustrate the usefulness of our merit count approach with an example in the next section.

The first GUB subset G_1 is detected from the first eligible set E_1 by using the following procedure. The first row in G_1 is the one with the minimum row count (that is the minimum number of non-zero entries). The kth row in G_1 is the one with the minimum row count among remaining rows of E_1, such that the kth row is not in GUB-conflict with the rest of the rows in G_1.

In order to construct the ith GUB subset G_i, we introduce the following procedure. For each column j, we compute the *congestion* T_j (restricted non-zero count) which is the number of non-zero entries in column j in the network subset N_{i-1}. For a given column s, let K_s be the set of row indices of the non-zero coefficients, that is, $K_s = \{k | \tilde{a}_{ks} \neq 0\}$. The merit counts m_{ks} of all non-zero entries in column s are calculated by the formula

$$m_{ks} = (RC_k - 1)(CC_s - T_s - 1), \; k \in K_s. \tag{3.2}$$

Let L_k denote the set of column indices that have non-zero coefficients in row k. Then, the maximum merit count for row k is defined as

$$\omega_k = \max_{s \in L_k}\{m_{ks}\}. \tag{3.3}$$

For stage i ($i \geq 2$), the GUB set G_i is constructed in the following way. We consider all columns j (in order) with congestion values $T_j = 0$ or 1 such that at least one row from E_i has a non-zero entry in this column. We compute ω_k for each row k having a non-zero entry in the intersection with column j; we then choose the row with minimum ω_k and add it to the set G_i.

This heuristic is adopted to increase the number of GUB sets found in this way. The congestion values of columns which appear in the chosen GUB row are then updated and the algorithm steps set out above are repeated. We terminate the algorithm at stage i when every column has been considered.

3.4.1 An Example (M-GUB with Merit Count)

Consider the following LP constraints. We apply the M-GUB algorithm with the merit count procedure explained above. Columns corresponding to slack variables and artificial variables need not be considered since they can be included as network columns at any time.

$$\begin{aligned}
r_1 &: \ x_1 + x_2 \geq 8 \\
r_2 &: \ x_2 + x_7 + x_{10} \geq 8 \\
r_3 &: \ x_3 + x_4 \geq 6 \\
r_4 &: \ x_1 + x_4 + x_9 \geq 4 \\
r_5 &: \ x_1 + x_2 + x_5 + x_6 \geq 9 \\
r_6 &: \ x_2 + x_3 + x_6 + x_7 \geq 4 \\
r_7 &: \ x_3 + x_4 + x_7 \geq 7 \\
r_8 &: \ x_4 + x_5 \geq 3 \\
r_9 &: \ x_5 + x_6 \geq 3 \\
r_{10} &: \ -x_6 + x_8 + x_{10} \geq 3 \\
r_{11} &: \ -x_3 + x_7 + x_9 = 0 \\
r_{12} &: \ x_5 + x_8 \geq 0
\end{aligned}$$

Thus, the eligible subset of rows $E_1 = \{r_1, r_2, \cdots, r_{12}\}$ is considered. The sequence of row counts is $\{RC_1, RC_2, ..., RC_{12}\} = \{2,3,2,3,4,4,3,2,2,2,3,3,2\}$. If we apply the minimum row count procedure in order to construct the first GUB subset, the first row is chosen automatically as an element of the first GUB subset. The third and ninth rows are then included as rows of G_1. So the first GUB subset is $G_1 = \{r_1, r_3, r_9\}$ which is also the first network set. The congestion of columns are $T_j = 1$ for $j = 1, 2, ..., 6$ and $T_j = 0$ for $j = 7, ..., 10$.

We proceed with the second eligible set $E_2 = \{-r_2, -r_4, -r_5, -r_6, -r_7, -r_8, r_{10}, r_{11}, -r_{12}\}$. The first column with congestion one is considered. There are two candidate rows which have a non-zero entry in this column, that is $K_1 = \{4, 5\}$. Thus, one of them can be chosen as the first row of G_2. The sequence of merit counts corresponding to each non-zero entry in rows 4 and 5 are $\{m_{4j} | \tilde{a}_{4j} \neq 0, j = 1, 4, 9\} = \{2, 4, 2\}$ and $\{m_{5j} | \ \tilde{a}_{5j} \neq 0,$

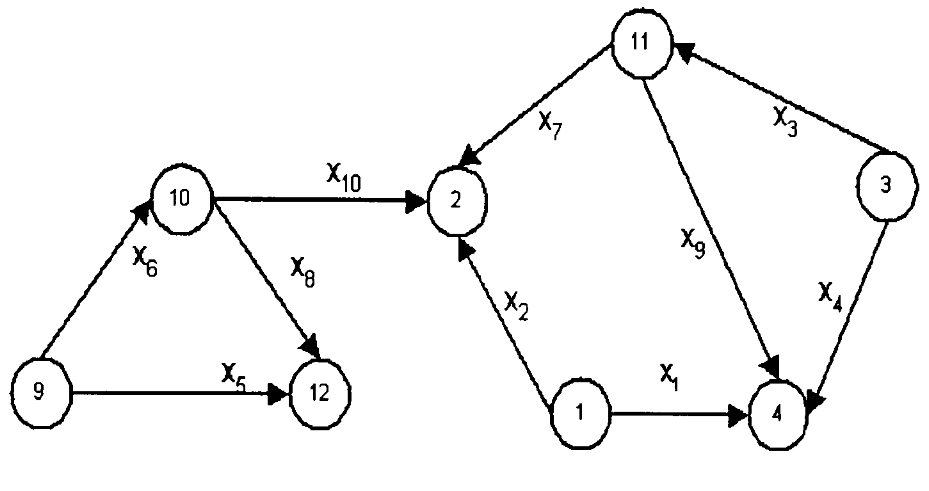

Figure 3.4.1: The network detected within the LP problem.

$j = 1, 2, 5, 6\} = \{3, 6, 6, 6\}$, respectively. The maximum merit count of row 4, $\omega_4 = 4$ is smaller than that of row 5, $\omega_5 = 6$. Thus, row 4 in the reflected form is included in G_2. If we did not use the merit count heuristic and chose row 5 which has a higher merit count this would prevent a number of rows from entering the network structure. In the same manner, rows 2 and 6 are considered for the second column and row 2 in the reflected form is included to the set G_2. For the third column, neither row 7 nor row 11 is chosen, because row 7 has a non-zero entry in column 4 whose congestion is two and row 11 is in GUB-conflict with rows in G_2. By repeating the same procedure, the second GUB subset G_2 is obtained as $G_2 = \{-r_4, -r_2, -r_{12}\}$. In this stage, the network structure is constructed as $N_2 = N_1 \cup G_2 = \{r_1, r_3, r_9, -r_4, -r_2, -r_{12}\}$.

The same procedure is carried out for detecting the third GUB set G_3 out of rows in the third eligible set $E_3 = \{r_{10}, r_{11}\}$. The set E_3 is obtained from the remaining rows that are not involved in neither G_1 nor G_2 and not in PN-conflict with N_2. The third GUB set $G_3 = \{r_{11}, r_{10}\}$ is detected in the same manner. The network structure N_3 is obtained by adding rows in G_3 to the network subset N_2 such that $N_3 = \{r_1, r_3, r_9, -r_4, -r_2, -r_{12}, r_{11}, r_{10}\}$. Since it is not possible to find any more eligible rows, the algorithm terminates. The network structure N_3 consisting of eight nodes and ten arcs is displayed in Figure 3.4.1 and the node-arc incidence matrix $\mathcal{N}$ is as follows.

$$\mathcal{N} = \begin{array}{r|rrrrrrrrrr|} r_1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ r_3 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ r_9 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ -r_4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ -r_{12} & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ -r_2 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 \\ r_{11} & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ r_{10} & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 1 \end{array}$$

3.5 An Independent Set Algorithm For Detecting GUB

A GUB subset in an LP problem corresponds to an independent set in a graph corresponding to the coefficient matrix of the LP problem [23]. This motivated us to consider another approach to detect each GUB subset in our network detection algorithm [56]. The relationship between GUB structures and independent sets in the corresponding graphs was used to prove that the problem of detecting a maximum GUB structure is NP-hard in [23].

For a ternary matrix $\tilde{A}$, the corresponding graph $\mathcal{G}(\tilde{A})$ has vertex set $\mathcal{V} = \{v_1, v_2, \cdots, v_{m'}\}$ where m' is the number of rows in $\tilde{A}$. Vertices v_i and v_k in $\mathcal{G}(\tilde{A})$ are adjacent, that is they are linked by an edge, if and only if row i and row k are in GUB-conflict. A set of vertices S in $\mathcal{G}(\tilde{A})$ is called independent if no two vertices in S are adjacent. Clearly, every GUB structure in $\tilde{A}$ corresponds to an independent set in $\mathcal{G}(\tilde{A})$ and vice versa.

Most algorithms for constructing an independent set in a graph can be divided into two categories. In the first category, we have algorithms that construct the independent set directly such as a *greedy* algorithm and a *matching* algorithm. The algorithms which improve (enlarge) the existing independent set belong to the second category and are called *improvement* algorithms. We describe algorithms from both categories which we use in our network detection procedure.

Greedy and Matching Algorithms

The greedy algorithm picks up a vertex of minimum degree, adds it to the current (initially empty) independent set and deletes it together with its neighbours from the graph. This procedure is repeated until all vertices are deleted. A set of edges of $\mathcal{G}$ is called a matching, if the edges in the set have no common vertices. The matching algorithm builds a maximal matching in graph $\mathcal{G}$. A maximal matching is constructed by choosing edges of $\mathcal{G}$ one by one and discarding those which have common vertices with already chosen ones. The vertices of the maximal matching are deleted from $\mathcal{G}$; the remaining vertices constitute an independent set. Descriptions of the greedy and matching algorithms can be found in numerous articles in the literature; for example, see [92] for a recent reference.

Improvement Algorithms

Improvement algorithms such as *local search* and *k-change* are designed to improve the current feasible solution. Local search procedure starts at an initial feasible solution and searches for a better solution in its neighbourhood [93]. If there exists an improved solution, then the search is repeated from the new solution. The k-change procedure removes k elements from a neighbourhood and replaces them with the new solution in order to achieve a better feasible solution.

The 2-opt algorithm is a k-change algorithm. Given an independent set S in graph $\mathcal{G}$, the algorithm tries to find a pair of non-adjacent vertices v_i, v_j not in S with only one neighbour, v_k, in S (v_i and v_j has only one edge to S). If the algorithm succeeds it replaces v_k by v_i and v_j in S. The search for an improving pair of vertices continues until no improvement is possible. A description of the 2-opt algorithm can be found in [76].

M-GUB with an Independent Set

We consider an undirected graph $\mathcal{G}$ induced by the nodes corresponding to the rows of the eligible set E_1. The first independent set S_1 is obtained by applying a greedy algorithm to this graph. The nodes in S_1 also coincide to the rows of the first network set N_1, that is $N_1 = S_1$. The second independent set S_2 is found by reapplying the same algorithm to the subgraph induced by the nodes which are not included in S_1. If the set S_2 is empty, then the algorithm terminates with the network structure N_1 detected. Otherwise, all rows (both in the original and in the reflected forms) corresponding to nodes of the set S_2 are then scanned one by one in respect of PN-conflict with rows of the set N_1. If row r_i or $-r_i$ is not in PN-conflict with rows in N_1, then r_i or $-r_i$ is included to the network set. In this way, the second network set N_2 is obtained. The third independent set S_3 is obtained from the subgraph induced by the remaining nodes which are not included in N_2. If $S_3 \neq \emptyset$, then rows which correspond to nodes in S_3 are scanned to check whether they are in PN-conflict with any row in the network set N_2. The same procedure is repeated until no more independent set is found or no more row can be added to the previous network structure.

3.5.1 An Example (M-GUB with an Independent Set)

Consider the following LP problem constraints. We apply the M-GUB algorithm with the independent set heuristic consisting of the greedy algorithm.

$$\begin{aligned}
r_1 &: \; x_1 + x_2 = 1 \\
r_2 &: \; x_3 + x_4 = 2 \\
r_3 &: \; x_5 + x_6 = 3 \\
r_4 &: \; x_3 + x_7 + x_{13} = 4 \\
r_5 &: \; x_6 + x_7 + x_8 + x_9 + x_{15} = 5 \\
r_6 &: \; x_2 + x_9 + x_{10} + x_{14} = 6 \\
r_7 &: \; x_4 + x_{10} + x_{11} = 7 \\
r_8 &: \; x_5 + x_8 + x_{11} + x_{12} = 8 \\
r_9 &: \; x_1 + x_{12} + x_{13} + x_{14} + x_{15} = 9
\end{aligned}$$

The corresponding undirected graph of the coefficient matrix is depicted in Figure 3.5.2. The degrees of nodes are $d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 3, d_5 = 5, d_6 = 4, d_7 = 3, d_8 = 4, d_9 = 5$. We detect the first independent set $S_1 = \{v_1, v_2, v_3\}$ which is also the first network set, $N_1 = \{r_1, r_2, r_3\}$. By reapplying the greedy algorithm to the subgraph induced by nodes not included in S_1, the second independent set $S_2 = \{v_4, v_6, v_8\}$ is obtained. Then all rows corresponding to nodes of the set S_2 are scanned and the network structure $N_2 = \{r_1, r_2, r_3, -r_4, -r_6, -r_8\}$ is obtained. The third independent set is found from the subgraph induced by the remaining nodes which are not included in N_2; $S_3 = \{v_5, v_7\}$. However, the network set N_2 cannot be improved since rows 5 and 7 are in PN-conflict, even in the reflected forms, with rows of N_2. Therefore, the algorithm terminates.

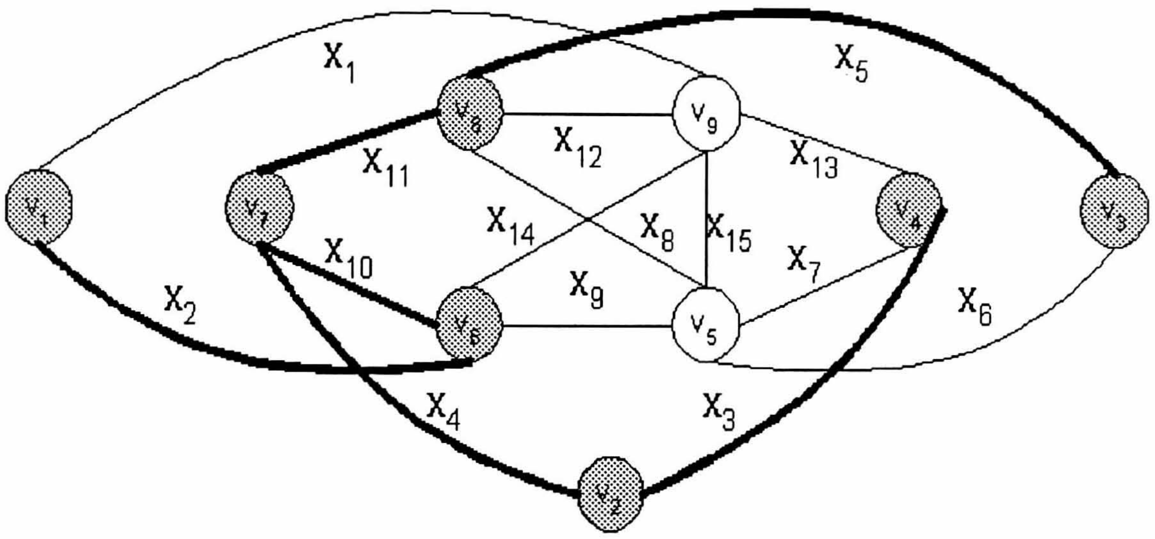

Figure 3.5.2: The undirected graph of the LP problem.

We apply the 2-opt algorithm after the first independent set S_1 is found. It can be seen from Figure 3.5.2 that S_1 can be improved only by exchanging node v_2 with nodes v_4 and v_7. Therefore, the first independent set is enlarged to $S'_1 = \{v_1, v_3, v_4, v_7\}$. When applying the greedy algorithm to the subgraph consisting of the remaining nodes, the second independent set $S'_2 = \{v_2, v_6, v_8\}$ is obtained. After scanning nodes in S'_2 in respect of PN-conflict with the rows of the first independent set, the network subset is obtained by adding the reflected form of the rows in S'_2 to the first independent set S'_1; $N'_2 = \{r_1, r_3, r_4, r_7, -r_2, -r_6, -r_8\}$.

The same procedure is reapplied to the subgraph consisting of the nodes, not included in the network N'_2, so the third independent set is found $S'_3 = \{v_5\}$. Since row 5, in the original and in the reflected form, is in PN-conflict with rows in N'_2, the network structure detected already is not changed. The algorithm stops with the network structure N'_2 whose node-arc incidence matrix $\mathcal{N}$ is as follows. The rows and columns of the network matrix detected are depicted with the bold nodes and arcs in Figure 3.5.2. Columns which have only one non-zero entry in the matrix $\mathcal{N}$ correspond to a self-loop.

$$\mathcal{N} = \begin{array}{r} r_1 \\ r_3 \\ r_4 \\ r_7 \\ -r_2 \\ -r_6 \\ -r_8 \end{array} \left[\begin{array}{cccccccccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & -1 & 0 & 0 \end{array} \right]$$

3.6 Computational Results

In the computational experiments, we evaluate the performance of our network extraction algorithms for both *efficiency* and *effectiveness.* Efficiency of an algorithm refers to the CPU time taken to find an EPN structure while effectiveness of the algorithm is measured by the size of the network structure detected.

We computationally investigate the performance of the M-GUB algorithm for a set of real-world problems which are readily available to the scientific community. Therefore, we chose a set of Netlib models [43] whose characteristics in terms of the number of rows, columns and non-zeros are summarised in section 1.8. The number of eligible rows of the coefficient matrix A is presented under the heading EROW in Table 3.6.1.

All algorithms used in the experimental investigation are implemented in FORTRAN, compiled and run on a DEC ALPHA 3000/600 computer with 96 MB memory. We investigate altogether five algorithms which are

D-GUB:	Two-stage (double) GUB without merit count,
D-GUBMC:	Two-stage GUB with merit count,
M-GUBMC:	Multi-stage GUB with merit count,
D-INDSET:	Two-stage GUB with independent set,
M-INDSET:	Multi-stage GUB with independent set.

In order to evaluate the performance of the merit count concept, we consider only two-stage of the M-GUB algorithm; and apply it with merit count and without merit count. The results obtained by D-GUB and D-GUBMC are set out in Table 3.6.1 in terms of the number of network rows. In the D-GUB algorithm, the rows are considered in non-decreasing order of row counts. The results in Table 3.6.1 reveal that the number of network rows detected by D-GUBMC are more than those detected by D-GUB for all models except `ship12l` and `25fv47`. Therefore, we may claim that our use of the merit count concept is clearly vindicated.

We carry out extensive computational experiments to check which of the above independent set algorithms is suitable for the multi-stage GUB algorithm. The obtained results show that the matching algorithm is always inferior to the greedy one. For this reason, an independent set heuristic consisting of the greedy algorithm is used to detect each GUB subset in the M-GUB algorithm. We also attempt to enlarge results of the GUB structure by using an improvement algorithm, namely the 2-opt algorithm. The 2-opt heuristic applied to the independent sets constructed by the greedy algorithm gives very marginal improvements or no improvements at all. We observe that the 2-opt heuristic unlike the greedy algorithm is time consuming. For these reasons, we use only the greedy algorithm for constructing GUB structures in our computational experiments.

In Table 3.6.2, we set out the results for the M-GUB algorithm with two different GUB detection procedures in terms of the number of network rows, the number of network columns, namely, those that have at least one non-zero entry within the network rows, and the computational time taken to detect network structures. In order to

Model Names	EROW	D-GUB	D-GUBMC
25fv47	224	149	146
agg3	141	46	46
cre_a	1247	659	688
cycle	507	373	403
czprob	719	640	708
energy	1531	633	1237
greenbea	916	743	775
nesm	190	161	170
osa007	1048	1036	1043
pilot87	341	266	270
scagr25	301	213	235
scrs8	214	132	146
scfxm3	423	279	280
sctap3	620	620	620
sierra	1155	641	666
ship12l	828	732	666
stocfor2	1262	812	898

Table 3.6.1: The number of network rows detected by the two-stage GUB based algorithms.

show the performance improvement of the multi-stage compared to the two-stage algorithm, we juxtapose the results obtained using two different GUB detection procedures.

For the two-stage case, we terminate the M-GUB algorithm after finding the second GUB set G_2. From the results set out in Table 3.6.2, we may conclude that the multi-stage GUB algorithm in the majority of the cases performs better than the two-stage GUB algorithm; thus indicating the improvement achieved by the extension introduced by us over the double GUB algorithm of Brown and Wright.

We further wish to consider the relative performances of the M-INDSET and M-GUBMC algorithms. An analysis of the results in Table 3.6.2 also reveals that there are seven winners for the M-INDSET algorithm and seven winners for M-GUBMC while there are three ties. Thus, the difference in performance taking into consideration the number of network rows is not significant. As a result, we cannot conclude that any-one of these GUB detection procedures dominates the other. We therefore extend the experiments to a set of three instances of a large scale industrial model taken from the domain of supply chain planning [80]. These models are known to possess a large proportion of embedded network rows. For these three models, the problem statistics are given in Table 1.8.5. The number of essential rows (under the heading EROW) and the corresponding network rows detected by M-GUBMC and M-INDSET are presented in Table 3.6.3.

Model Names	EROW	M-GUBMC	M-INDSET
model1	4291	3656	3755
model2	3690	3060	3306
model3	3690	3060	3306

Table 3.6.3: The number of network rows for supply chain models.

The results in Table 3.6.3 encourage us to conclude that perhaps for certain classes of constraints M-INDSET is better than M-GUBMC in identifying the EPN structure.

Algorithms	D-GUBMC			M-GUBMC			D-INDSET			M-INDSET		
Model Names	NETR	NETC	TIME	NETR	NETC	TIME	NETR	NETC	TIME	NETR	NETC	TIME
25fv47	146	507	0.01	198	768	0.03	187	746	0.01	199	781	0.01
agg3	46	91	0.01	59	106	0.01	62	119	0.01	62	119	0.01
cre_a	688	3131	0.20	823	3474	1.03	799	3283	0.12	803	3291	0.17
cycle	403	1264	0.04	506	2350	0.09	500	2345	0.01	505	2349	0.02
czprob	708	3093	0.05	717	3102	0.02	718	3101	0.02	718	3101	0.02
energy	1237	5899	0.13	1504	7547	0.95	1475	7623	0.05	1486	7643	0.06
greenbea	775	2587	0.09	877	4344	0.21	859	4062	0.03	881	4363	0.04
nesm	170	1538	0.03	188	1630	0.04	178	1585	0.01	190	1635	0.01
osa007	1043	23949	1.06	1043	23949	1.18	1043	23949	0.08	1043	23949	0.10
pilot87	270	868	0.02	303	905	0.04	302	905	0.02	304	905	0.03
scagr25	235	321	0.01	271	375	0.02	270	375	0.01	270	375	0.01
scrs8	146	852	0.07	213	952	0.14	212	950	0.01	212	950	0.01
scfxm3	280	806	0.02	355	1034	0.05	309	1020	0.02	312	1023	0.02
sctap3	620	1860	0.01	620	1860	0.01	620	1860	0.01	620	1860	0.01
sierra	666	2016	0.06	671	2016	0.07	671	2016	0.05	672	2016	0.07
ship12l	666	4704	0.83	732	5222	1.96	732	5223	0.10	732	5223	0.10
stocfor2	898	1659	0.21	947	1698	0.25	901	1666	0.06	914	1674	0.09
Total	8997	55145	2.85	10027	61332	6.10	9838	60828	0.62	9923	61257	0.78

NETR: The number of network rows, **NETC:** The number of network columns, **TIME:** CPU time(seconds).

Table 3.6.2: The performance of four selected algorithms.

If we consider the structure detection time, for these three models, the time taken is between five and eight seconds for the M-INDSET algorithm which is again well within 5% of the sparse simplex solution time. However, the algorithm M-GUBMC is slower than M-INDSET. As a result we may claim that while the M-INDSET algorithm is marginally better than the M-GUBMC algorithm for Netlib models, the M-INDSET algorithm is dominant for supply chain models.

3.7 Discussion

In this chapter, we have considered a GUB based algorithm for detecting EPN structure in an LP problem and have shown that our approach of extending the two-stage heuristic to a multi-stage heuristic leads to improved performance. We have applied the Markowitz merit count concept in a novel way and improved the GUB detection heuristic. We have introduced the independent set algorithm as an alternative approach to finding GUB sets.

All the algorithms have been tested on a range of Netlib models and we have not been able to identify a GUB detection heuristic in M-GUB which is the best in all cases for the Netlib models. However, we can easily conclude that the algorithm M-INDSET outperforms all the others for supply chain models. Taking into consideration the computing time, we have observed that the structure detection time is usually less than 5% of the sparse simplex solution time.

Chapter 4

A Network Extraction Algorithm By Using Generalised Signed Graphs

4.1 Introduction

Signed graphs are often used in social psychology as a mathematical model to investigate relationships among people within a group [27]. Consider a group of people such that every two individuals are either friendly, unfriendly, or indifferent toward each other. Such a group of people with such kind of relationships between each individual is referred to as a social system which is represented by a signed graph. Each vertex represents the individual within the group. A positive edge joins two vertices if there is a positive relation between two people, that is they are friendly toward each other. A negative edge joins two vertices if two corresponding people have a negative relation, that is they are unfriendly with each other. Indifference between two individuals is indicated by the lack of any edge joining the corresponding individuals. A signed graph is a special case of a generalised signed graph. For the purpose of this thesis, we use generalised signed graphs to represent the relationships among rows of the coefficient matrix of an LP problem with respect to the pure network structure. This relationship leads to a heuristic for extracting such an EPN structure within the LP problem. The importance of this heuristic is that, in contrast to the previously known network extraction procedures, the heuristic determines whether a given LP problem is an entirely pure network.

The rest of this chapter is organised in the following way. In section 4.2, we broadly define the generalised signed graphs. In section 4.3, balanced signed graphs are described. In section 4.4, by exploiting the relationship between the generalised signed graphs and embedded pure networks, it is proved that the problem of detecting the maximum size EPN structure in an LP problem is NP-hard, even for very special families of matrices. In section 4.5, the pure network graphs are summarised. In section 4.6, we introduce a new network extraction algorithm based on generalised signed graphs: we call this the GSG algorithm. In section 4.7, the complexity issues of the GSG algorithm are explained. In section 4.8, the GSG algorithm is explained using an illustrative example. The computational results are presented in section 4.9 followed by a discussion of the results in section 4.10.

4.2 Generalised Signed Graphs

A *generalised signed (GS) graph* is defined as an undirected graph in which the weight of any edge is $+1, -1$ or 0. If we consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the set of vertices $\mathcal{V}$ and the set of edges $\mathcal{E}$, a GS graph is represented with a function $c : \mathcal{E} \rightarrow \{-1, 0, +1\}$. A GS graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c)$ is called a *signed graph* if $c(\mathcal{E}) \subseteq \{-1, +1\}$. Since a positive or a negative sign is attached to every edge of the signed graph, it is natural to refer to each edge as a positive or a negative edge with respect to their weights of $+1$ or -1, respectively.

We construct a generalised signed graph corresponding to the coefficient matrix of an LP problem as follows. A ternary submatrix $\tilde{A} = [\tilde{a_{ij}}]$ (see section 2.2) of the LP coefficient matrix A is obtained after the preprocessing procedure described in section 2.3. Assume that $\tilde{A}$ is an $m' \times n'$ matrix such that $m' \leq m$ and $n' \leq n$. The vertex set of $\mathcal{G}(\tilde{A})$ is $\{v_1, ..., v_{m'}\}$; $v_i v_k$ is an edge of $\mathcal{G}(\tilde{A})$ if and only if there exists a column j of

$\tilde{A}$ such that both $\tilde{a_{ij}}$ and $\tilde{a_{kj}}$ are non-zero. For an edge v_iv_k, the weight $c(v_iv_k)$ equals to **either** $(+1)$ **or** (-1) if for every column j of $\tilde{A}$ **either** (at least one of $\tilde{a_{ij}}$ and $\tilde{a_{kj}}$ is zero or both of them are non-zero and of the opposite signs) **or** (of the same sign), respectively. Otherwise, the weight $c(v_iv_k)$ is zero. In other words, if two rows i and k of matrix $\tilde{A}$ comprise a network structure (not in PN-conflict), then the corresponding vertices v_i and v_k are adjacent with a weight of $+1$. Otherwise, these rows cannot be together in the network structure, therefore they are joined by an edge with a weight of 0. If two rows become network rows only after reflecting one of them, then the two adjacent vertices are joined by an edge with a weight of -1.

We restrict ourselves (as it is done in most of the papers on the topic, for example see [3, 13, 20, 56]) to only the row reflection operation. We define the reflection operation in a GS graph as follows. For a vertex v_i in the GS graph $\mathcal{G}$, the *v_i-reflection* of $\mathcal{G}$ is the GS graph $\mathcal{G}_{v_i}$ which is obtained from $\mathcal{G}$ by changing the signs of the weights of edges incident to the vertex v_i. For a non-empty subset $W = \{w_1, \cdots, w_l\}$ of $\mathcal{V}$, the *W-reflection* of $\mathcal{G}$ is a GS graph $\mathcal{G}_W = (\cdots((\mathcal{G}_{w_1})_{w_2})\cdots)_{w_l}$. Clearly, $\mathcal{G}_W$ can be obtained from $\mathcal{G}$ by changing the signs of weights of the edges between the set of vertices W and the set of remaining vertices which do not belong to W, $\overline{W}$.

4.3 Balanced Signed Graphs

Consider a signed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The graph $\mathcal{G}$ is *balanced* if its vertex set can be partitioned into two subsets W and $\overline{W} = \mathcal{V} \setminus W$ (one of which may be empty) such that the two sets satisfy the following conditions:

- each edge joining two vertices in the same subset is positive,
- each edge joining vertices in the different subsets is negative.

For a graph $\mathcal{H}$, let $\mu(\mathcal{H})$ be the number of negative edges e in $\mathcal{H}$ (that is, $c(e) = -1$). Clearly, $\mu(\mathcal{G}_W) = 0$ and $\mu(\mathcal{G}_{\overline{W}}) = 0$. When the set W equals $\mathcal{V}$, the set $\overline{W}$ is obviously empty.

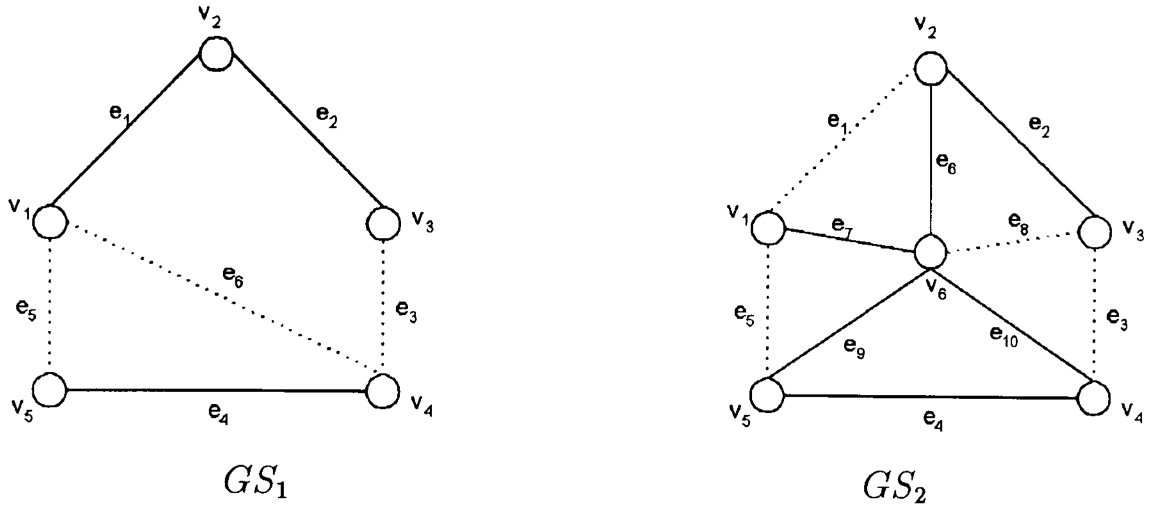

Figure 4.3.1: Balanced and unbalanced signed graphs.

In Figure 4.3.1, we consider two signed graphs GS_1 and GS_2 in which the negative edges are denoted by dashed lines and the remaining edges are positive. It can be easily shown that the signed graph GS_1 is balanced while the GS_2 is unbalanced. Assume that the vertex set in GS_1 is partitioned as $W = \{v_1, v_2, v_3\}$ and $\overline{W} = \{v_4, v_5\}$. Clearly, this partition satisfies the two conditions for being a balanced signed graph.

In a signed graph, a cycle is called *positive* if it has an even number of negative edges, and is called *negative* otherwise. We observe that a cycle with no negative edge is positive. It can be easily shown that by vertex reflections a cycle can be turned into one whose edges are all positive if and only if it is a positive cycle. Therefore, the reflections of vertices in cycles in a signed graph do not have any effect on whether the cycle is positive or negative. By using this definition, we can now give the main characteristic of a balanced signed graph: a signed graph is balanced if and only if every cycle of the signed graph is positive [65].

4.4 Embedded Networks and Generalised Signed Graphs

Consider the DMEPN problem defined in section 1.7. Let $\nu(\tilde{A})$ denote the maximum number of pure network rows. In order to show that the DMEPN problem is NP-hard, we first introduce the following parameter for the GS graph $\mathcal{G} = \mathcal{G}(\tilde{A}) = (\mathcal{V}, \mathcal{E})$:

$$\eta(\mathcal{G}) = \max\{\alpha((\mathcal{G}_W)^{(0)} \cup (\mathcal{G}_W)^{(-1)}) : \ W \subseteq \mathcal{V}\}. \tag{4.1}$$

For a graph $\mathcal{G}$, $(\mathcal{G})^{(-1)}$ ($(\mathcal{G})^{(0)}$ and $(\mathcal{G})^{(+1)}$, respectively) denotes the spanning subgraph of $\mathcal{G}$ whose edges are of negative (of zero and positive weight, respectively). The cardinality of a maximum independent set of vertices in $\mathcal{G}$ is denoted as $\alpha(\mathcal{G})$, and is called the independence number of $\mathcal{G}$. The parameter in (4.1) is of importance due to the following easily verifiable claim.

Proposition 4.4.1 *For a ternary matrix $\tilde{A}$, we have $\nu(\tilde{A}) = \eta(\mathcal{G}(\tilde{A}))$.*

Proposition 4.4.1 allows us to study $\eta(\mathcal{G})$ for GS graphs $\mathcal{G}$ rather than $\nu(\tilde{A})$ for the ternary matrix $\tilde{A}$. However, we cannot restrict ourselves to any special class of GS graphs due to the following claim:

Proposition 4.4.2 *For a GS graph $\mathcal{H}$, there exists a ternary matrix $\tilde{A}$ such that $\mathcal{G}(\tilde{A})$ is isomorphic to $\mathcal{H}$.*

Proof: Let $\{v_1, ..., v_{m'}\}$ and $\{e_1, ..., e_{n'}\}$ be the vertex set and the edge set, respectively, of $\mathcal{H}$. Construct a matrix $\tilde{A}$ of dimension $m' \times 2n'$ as follows. For an index $j \in \{1, ..., n'\}$, if the weight $c(e_j) = 1$ ($c(e_j) = -1$), $e_j = v_i v_k$, then columns $2j - 1$ and $2j$ consist of zero entries apart from $a_{i,2j} = 1$, $a_{k,2j} = -1$ ($a_{i,2j} = a_{k,2j} = 1$); if the weight $c(e_j) = 0$, $e_j = v_i v_k$, then columns $2j - 1$ and $2j$ consist of zero entries apart from $a_{i,2j} = 1$, $a_{k,2j} = -1$, $a_{i,2j-1} = a_{k,2j-1} = 1$. It is easy to check that $\mathcal{H}$ is isomorphic to $\mathcal{G}(\tilde{A})$.

The two above propositions and the well-known fact that the independence number problem is NP-hard [92] imply that the DMEPN problem is NP-hard as well. Here, we can restrict ourselves to GS graphs $\mathcal{G}$ all of whose weights are zero. When all weights are $+1$, the problem to compute $\eta(\mathcal{G})$ becomes trivial; the case of all weights equal to -1 is treated in the following theorem.

Theorem 4.4.3 *The problem of computing $\eta(\mathcal{G})$ for signed graphs $\mathcal{G}$ all of whose weights are -1 is NP-hard.*

Proof: Let $\mathcal{H}$ be a graph and $\mathcal{H}'$ be a vertex disjoint copy of $\mathcal{H}$. To obtain the graph $\mathcal{G} = \mathcal{G}(\mathcal{H})$ add to $\mathcal{H} \cup \mathcal{H}'$ all edges between $\mathcal{H}$ and $\mathcal{H}'$. Assign to every edge of $\mathcal{G}$ the weight -1. This theorem follows from the fact that $\eta(\mathcal{G}) = 2\alpha(\mathcal{H})$ whose proof is given below. Let W be a set of vertices in $\mathcal{G}$. As can be seen from (4.1), $\eta(\mathcal{G}) = \max\alpha((\mathcal{G}_W)^{(-1)})$. Then,

$$\begin{aligned}\alpha((\mathcal{G}_W)^{(-1)}) &= \max\{\alpha(\mathcal{G}[\mathcal{V}(\mathcal{H}) \cap W]), \alpha(\mathcal{G}[\mathcal{V}(\mathcal{H}') \cap W])\} \\ &+ \max\{\alpha(\mathcal{G}[\mathcal{V}(\mathcal{H}) \cap \overline{W}]), \alpha(\mathcal{G}[\mathcal{V}(\mathcal{H}') \cap \overline{W}])\} \le \alpha(\mathcal{H}) + \alpha(\mathcal{H}') = 2\alpha(\mathcal{H}).\end{aligned}$$

Thus $\eta(\mathcal{G}) \le 2\alpha(\mathcal{H})$. In addition, we have $\eta(\mathcal{G}) \ge 2\alpha(\mathcal{H})$ since $\alpha((\mathcal{G}_{V(\mathcal{H})})^{(-1)}) = 2\alpha(\mathcal{H})$. Therefore, the proof is completed.

Here, for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $W \subseteq \mathcal{V}$, $\mathcal{G}[W]$ stands for the subgraph of $\mathcal{G}$ induced by W; $\overline{W} = \mathcal{V} \backslash W$. In the construction of the proof of the last theorem, let us add edges of weight 1 between the pairs of non-adjacent vertices in $\mathcal{G}$. We obtain a signed complete graph K. As K is a spanning supergraph of $\mathcal{G}$, $\eta(K) \le \eta(\mathcal{G})$. However, $\alpha((K_{V(\mathcal{H})})^{(-1)}) = 2\alpha(\mathcal{H})$. Thus, $\eta(K) = 2\alpha(\mathcal{H})$. We derive the following corollary.

Corollary 4.4.4 *The problem of computing $\eta(K)$ for signed complete graphs K is NP-hard.*

4.5 Pure Network Graphs

In this section, we characterise signed graphs $\mathcal{G}(\tilde{A})$ whose matrices $\tilde{A}$ are pure network matrices and call such graphs *pure network graphs.* In other words, a signed graph $\mathcal{G}$ is a *pure network graph* if and only if there exists a set W of vertices in $\mathcal{G}$ such that $\mu(\mathcal{G}_W) = 0$. In this section, a few results which are directly related to pure network matrices are presented since they are used in the description of our EPN extraction algorithm GSG. The assertions stated here are scattered in the literature, and are rather unknown to nonexperts in the area of signed graphs. We were not be able to find any publications in which all the results are covered. Therefore, we refer to the reader to a number of publications which 'cover' the results of this section; for example, see

[66, 64, 67, 107, 108]. We provide complete proofs of the following lemmas and theorem as they are short, possibly, original and (especially that of Lemma 4.5.2) useful from an algorithmic point of view. We consider only connected graphs; disconnected graphs can be treated by studying their components one by one.

Lemma 4.5.1 *Every signed tree T can be turned into an all plus tree.*

Proof: We prove the lemma by induction of the number of edges in T. The lemma is true when the number of edges is one. Let x be a vertex of T of degree one. By the induction hypothesis, there is a set $W \subseteq \mathcal{V}(T) - x$ such that $\mu((T-x)_W) = 0$. In the tree T_W, the edge e incident to x is positive or negative. In the first case, let $W' = W$ and the second case, let $W' = W \cup \{x\}$. Then, $\mu(T_{W'}) = 0$.

Lemma 4.5.2 *Every signed tree T is a pure network graph.*

Proof: From lemma 4.5.1, every signed tree of a signed graph can be turned into an all positive tree. Therefore, each tree corresponds to a pure network graph.

By considering these lemmas, we can summarise the relationship between the corresponding signed graphs of an LP coefficient matrix and pure network graphs in the following theorem.

Theorem 4.5.3 *For a connected signed graph $\mathcal{G}$ and a spanning tree T of $\mathcal{G}$, the following assertions are equivalent:*

1. *$\mathcal{G}$ is a pure network graph;*
2. *$\mathcal{G}$ is a balanced graph;*
3. *$\mathcal{G}$ does not have a negative cycle;*
4. *If $W \subseteq \mathcal{V}(\mathcal{G})$ such that $\mu(T_W) = 0$, then $\mu(\mathcal{G}_W) = 0$.*

Proof: The equivalence of assertions 1 and 2 is obvious and follows from the definitions of pure network graphs and balanced signed graphs. By the latter definition, assertion 2 implies assertion 3. By considering the chords of T one by one, we see that assertion 4 follows from assertion 3. Clearly, assertion 4 and Lemma 4.5.2 imply assertion 1.

4.6 A Network Extraction Algorithm: GSG

Theorem 4.5.3 leads to a new algorithm which can be applied to extract an EPN structure within an LP problem [61]. The algorithm is essentially based on the GS graph of the corresponding coefficient matrix of the LP problem and we call it the GSG algorithm. The GSG algorithm has two important properties that distinguish it from other network extraction algorithms in the literature. The first property is that the GSG algorithm determines whether the given LP problem is an entirely pure network. The second feature is that instead of any addition or deletion strategies or any other special structure detection heuristics to construct an EPN structure, the GSG algorithm employs two graph theoretic algorithms; namely finding a spanning tree and detecting an independent set in the GS graph.

The algorithm GSG proceeds as follows. The GS graph of the corresponding ternary matrix of the given LP problem is constructed by scanning either rows or columns of the submatrix $\tilde{A}$. Then a spanning forest on the subgraph $\mathcal{H} = \mathcal{G}^{(+1)} \cup \mathcal{G}^{(-1)}$ is found. Since the spanning forest can be converted to one with all positive signed edges (see Lemma 4.5.1), by using a recursive algorithm we can compute $W \subseteq \mathcal{V}$ such that $\mu(T_W) = 0$. At this stage, the W-reflection procedure is applied to nodes of W and as many negative edges as possible are turned into positive edges in the subgraph; we denote the GS graph as $\mathcal{H}' = \mathcal{G}_W$. If all edges in the graph $\mathcal{H}'$ have weights of $+1$, then the algorithm terminates by concluding that the GS graph is a pure network graph. Otherwise, we obtain a spanning subgraph $\mathcal{H}'' = (\mathcal{G}_W)^{(0)} \cup (\mathcal{G}_W)^{(-1)}$ by deleting all edges with positive weights. A maximal independent set S is then found by applying a minimum-degree greedy heuristic to the subgraph $\mathcal{H}''$. The minimum-degree greedy algorithm proceeds as follows; starting from the empty set S, it appends to S a vertex of $\mathcal{H}''$ of minimum degree, deletes this vertex together with its neighbours from $\mathcal{H}''$, and the above procedure is repeated until $\mathcal{H}''$ has no more vertex (for more detail, see [92]). It can be easily seen that the vertices of the maximal independent set S correspond to rows of $\tilde{A}$ that form a pure network [62]. The main steps of the algorithm are set out as follows.

The GSG Algorithm

Step 1 Construct the GS graph $\mathcal{G} = \mathcal{G}(\tilde{A}) = (\mathcal{V}, \mathcal{E}, c)$;

Step 2 Find a spanning forest T of $\mathcal{H} = \mathcal{G}^{(+1)} \cup \mathcal{G}^{(-1)}$;

Step 3 Using a recursive algorithm based on the proof of Lemma 4.5.2, compute $W \subseteq \mathcal{V}$ such that $\mu(T_W) = 0$. Find the graph $\mathcal{H}' = \mathcal{G}_W$.

Step 4 Using the minimum-degree greedy heuristic, find a maximal independent set S in the graph $\mathcal{H}'' = (\mathcal{G}_W)^{(0)} \cup (\mathcal{G}_W)^{(-1)}$.

Step 5 Terminate the algorithm with network matrix $N = S$.

Note that if $\mathcal{G}$ is a pure network graph, our heuristic's output is $S = \mathcal{V}$. We observe that constructing a generalised signed graph by scanning columns is faster than rows; this assertion is discussed in section 4.7. Therefore, the following pseudo code of the GSG algorithm is based on this observation.

Pseudo Code of the GSG Heuristic

```
Arguments:
$\tilde{A} = [\tilde{a}_{ij}]$ is a ternary matrix of dimension $m' \times n'$
$N =: \emptyset$, $W := \emptyset$ and $S := \emptyset$.
begin
$\mathcal{V} := \{v_1, \cdots, v_{m'}\}$
$\mathcal{E} := \emptyset$
(*Construct the signed graph $\mathcal{G} = \mathcal{G}(\tilde{A}) = (\mathcal{V}, \mathcal{E})$ with the function c.*)
for $j := 1, \cdots, n'$ do
    for all pairs of rows $i$, $k$ such that $\tilde{a}_{ij} \neq 0$, $\tilde{a}_{kj} \neq 0$ and $1 \leq i < k \leq m'$ do
        if $\tilde{a}_{ij} = \tilde{a}_{kj}$, then
            $w := -1$                (* $\tilde{a}_{ij}$ and $\tilde{a}_{kj}$ are the same sign*)
        else
            $w := +1$                (* $\tilde{a}_{ij}$ and $\tilde{a}_{kj}$ are of opposite sign*)
        endif
```

```
            if $v_iv_k \notin \mathcal{E}$ then
                $\mathcal{E} := \mathcal{E} \bigcup \{v_iv_k\}$
                $c(v_iv_k) := w$
            elseif $c(v_iv_k) \neq 0$ and $c(v_iv_k) \neq w$ then
                $c(v_iv_k) := 0$
            endif
        endfor
    endfor
    (*Find a spanning forest $T$ of $\mathcal{H} = \mathcal{G}^{(+1)} \cup \mathcal{G}^{(-1)}$. Compute $W \subseteq \mathcal{V}$ for $\mu(T_W) = 0$.*)
    while $W \neq \mathcal{V}$ do
    if there is an edge $e = v_iv_k \in \mathcal{E}$ such that $v_i \in W$, $v_k \in \mathcal{V} \setminus W$ and $c(v_iv_k) \neq 0$, then
        set $W := W \bigcup \{v_k\}$.
        if $c(v_iv_k) = -1$ then
            for all $v_s \in \mathcal{V}$ and $v_sv_k \in \mathcal{E}$ do
                $c(v_sv_k) := -c(v_sv_k)$
            endfor
        endif
    else
        let $v_s$ be any vertex of $\mathcal{V} \setminus W$; set $W := W \bigcup \{v_s\}$.
    endif
    endwhile
    (*Find the maximal independent set $S$ on subgraph $\mathcal{H}'' = (\mathcal{G}_W)^{(0)} \cup (\mathcal{G}_W)^{(-1)}$.*)
    (*Terminate the algorithm.*)
    end
```

4.7 Complexity of the Algorithm

Clearly, time complexity in step 1 is $O(n'm'^2)$. However, in practice, most matrices of LP problems are sparse and, therefore, represented in computer memory by special data structures [5] (linked lists [29]) which keep only non-zero entries.

In order to analyse what time is required when the matrix $\tilde{A}$ is sparse, we, for simplicity, assume that every column of $\tilde{A}$ has exactly k_c non-zero entries. Then, step 1 runs in time $O(n'k_c^2)$. The above implementation of step 1 is based on the column scanning. Similarly, we can construct a signed graph $\mathcal{G}$ by scanning rows. However, assuming that every row of $\tilde{A}$ has exactly k_r non-zero entries, we obtain that the implementation of step 1 by scanning rows requires time $O(k_r m'^2)$. For real values of m', n', k_r and k_c, $n'k_c^2 \ll k_r m'^2$. This fact leads us to the column scanning implementation of step 1.

Using breadth-first search or depth-first search, the spanning forest T in step 2 can be computed in time $O(|\mathcal{E}|)$. The proof of Lemma 4.5.2 implies an $O(|\mathcal{E}|)$-time recursive algorithm. It is known that a degree-greedy algorithm has complexity of $O(|\mathcal{E}|)$ [92].

4.8 An Example with the GSG Algorithm

Consider the following constraint set of an LP problem.

		x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
$r_1 : x_1+x_2=1$	$v_1 :$	1	1										
$r_2 : -x_1-x_2+x_3+x_4=2$	$v_2 :$	-1	-1	1	1								
$r_3 : -x_3+x_5+x_6=3$	$v_3 :$			-1		1	1						
$r_4 : x_3+x_7=4$	$v_4 :$			1				1					
$r_5 : -x_6-x_7-x_8-x_9=5$	$v_5 :$						-1	-1	-1	-1			
$r_6 : x_2+x_9+x_{10}=6$	$v_6 :$		1							1	1		
$r_7 : -x_1+x_2+x_4+x_{10}-x_{11}=7$	$v_7 :$	-1	1		1						1	-1	
$r_8 : -x_5-x_8+x_{11}+x_{12}=8$	$v_8 :$					-1			-1			1	1

The GS graph $\mathcal{G}$ corresponding to the coefficient matrix of the LP problem is constructed, and is depicted in Figure 4.8.2. A spanning tree of the graph $\mathcal{G}$ is displayed with bold lines on the signed subgraph.

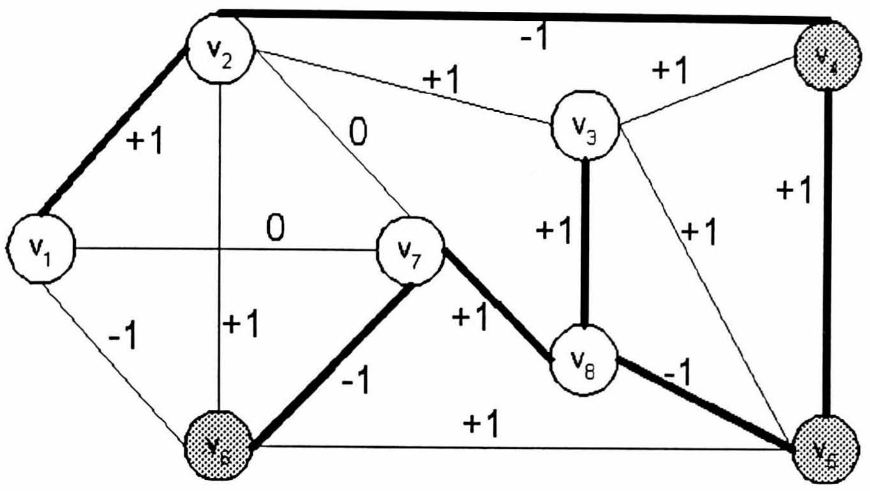

Figure 4.8.2: The generalised signed graph of the LP coefficient matrix.

With the recursion procedure, the vertex subset W is obtained as $W = \{v_6, v_5, v_4\}$ in where each vertex has a negative edge incident on. Applying the W-reflection procedure to the set W, all negative edges on the spanning tree are turned into all positive edges. This step aims to reduce the number of negative edges on the GS graph. The GS graph $\mathcal{H}'$ after the W-reflection operation is displayed in Figure 4.8.3.

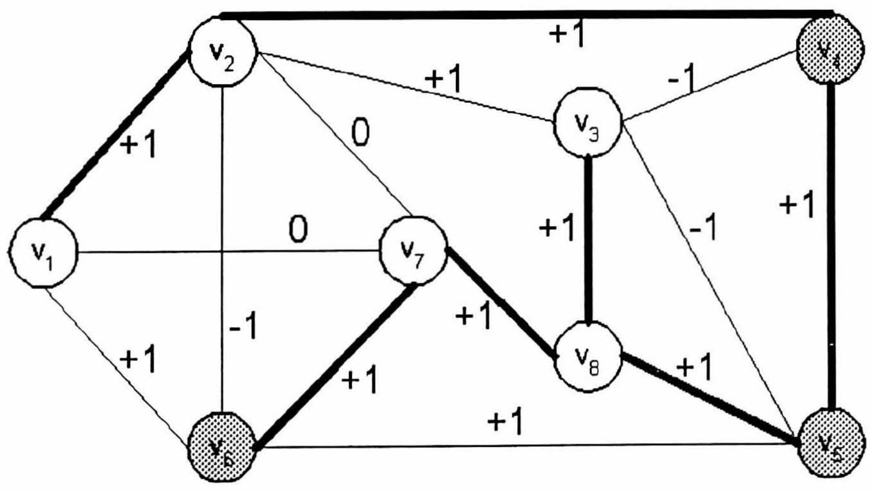

Figure 4.8.3: The generalised signed graph after the W-reflection.

By taking out the all +1 edges, we obtain the new GS graph $\mathcal{H}''$ and the maximal independent set $S = \{v_1, v_2, v_4, v_5, v_8\}$ is then obtained by using the degree-greedy algorithm. Thus, the set of network rows in terms of the original row indices is $N = \{r_1, r_2, -r_4, -r_5, r_8\}$.

4.9 Computational Results

The GSG algorithm is implemented in C++ and computational experiments are carried out on a Pentium-II 266MHz computer with 128 MB of RAM. The performance of the GSG algorithm is tested for a set of Netlib models as well as supply chain models used in the previous chapter. The effectiveness and efficiency of the GSG algorithm is compared with the well established Row Scanning Deletion algorithm, RSD (see section 2.4.2) and the M-GUB algorithm with two GUB detection procedures (see section 3.3).

Of these algorithms RSD is implemented using two different rules for the selection of rows. The first rule selects rows in the original order of their row penalties and the second one considers the row with the maximum row penalty first. In both cases, the row penalties are updated as the heuristic proceeds. We observe that ordering rows according to their row penalties and choosing the one with the maximum penalty always performs better than the first rule.

In Table 4.9.1, we present the results of the RSD algorithm combined with the second rule which gives consistently better performance. Since the performance of the multi-stage algorithm has been found to dominate those of two-stage algorithms, we restrict ourselves to the multi-stage algorithms; M-GUBMC (multi-stage GUB with merit count) and M-INDSET (multi-stage GUB with independent set). We therefore set out the results of these algorithms in Table 4.9.1 in terms of the number of network rows (under the heading NETR) and the CPU time in seconds (under the heading TIME). These results include only the time spent on the detection of EPN structures. The time taken to input the initial data and to apply preprocessing and scaling procedures is excluded since it is the same for all algorithms.

Algorithms	RSD		M-GUBMC		M-INDSET		GSG	
Model Names	NETR	TIME	NETR	TIME	NETR	TIME	NETR	TIME
25fv47	207	0.05	198	0.02	199	0.01	204	0.01
agg3	61	0.07	59	0.00	62	0.01	58	0.01
cre_a	809	1.32	823	0.29	803	0.07	828	0.20
cycle	505	0.04	506	0.04	505	0.01	505	0.04
czprob	718	0.93	717	0.03	718	0.01	718	0.06
energy	1497	0.11	1504	0.30	1486	0.02	1497	0.32
greenbea	881	0.25	877	0.08	881	0.02	877	0.07
nesm	190	0.03	188	0.02	190	0.02	190	0.01
osa007	1043	0.19	1043	0.59	1043	0.06	1043	0.11
pilot87	300	0.10	303	0.01	304	0.01	303	0.03
scagr25	275	0.03	271	0.01	270	0.01	300	0.01
scrs8	213	0.03	213	0.03	212	0.01	213	0.01
scfxm3	366	0.11	355	0.01	312	0.01	375	0.02
sctap3	620	0.03	620	0.01	620	0.01	620	0.04
sierra	790	0.66	671	0.03	672	0.02	762	0.12
ship12l	732	0.46	732	0.36	732	0.04	732	0.28
stocfor2	947	0.41	947	0.08	914	0.03	1042	0.14
Total	10154	4.82	10027	1.91	9923	0.37	10267	1.48

NETR: The number of network rows, **TIME**: CPU time (seconds).

Table 4.9.1: The number of network rows and network extraction time for Netlib models.

The results in Table 4.9.1 show that the GSG heuristic broadly speaking outperforms all other algorithms. Although for some models the difference between the number of network rows detected by different algorithms is not significant, GSG finds significantly more network rows than others for some models such as `stocfor2`, `scagr25` and `scfxm3`. `Sierra` is the only model where GSG detects a significantly smaller number of pure network rows compared with RSD (the best heuristic for this model). The GSG algorithm still significantly outperforms the other two heuristics, namely M-GUBMC and M-INDSET for this model.

Overall, we can conclude that the heuristic GSG dominates other algorithms for the chosen subset of Netlib models. When we consider the efficiency issue, we may conclude that the GSG algorithm is faster than the RSD and the M-GUBMC algorithms, however, it is slower than the M-INDSET algorithm. In the aggregated form, looking at the total number of network rows, RSD performs close to GSG. Taking into consideration the effectiveness as well as the efficiency of algorithms GSG performs better than RSD.

We also wish to test the performance of these heuristics for a set of three instances of a large scale industrial model taken from the domain of supply chain planning [80]. As mentioned before, these models are instances of a multi-stage multi-period production and distribution model and are known to posses a large proportion of embedded network rows. The problem statistics of these models are presented in Table 1.8.5. Since the M-INDSET algorithm dominates the M-GUBMC algorithm (see Table 3.6.3) for these models, we only present the results obtained by the M-INDSET algorithm as well as the algorithms RSD and GSG in Table 4.9.2. In these cases, while time taken is between two and three seconds for M-INDSET, the GSG takes around eleven seconds. Moreover, RSD is slower than these two algorithms.

We observe from the results in Table 4.9.2 that the algorithm GSG clearly outperforms the other two heuristics in terms of the number of network rows. In addition, the GSG algorithm terminates by concluding that the ternary matrix $\tilde{A}$ is a pure network itself

Model Names	ALGORITHMS		
	RSD	M-INDSET	GSG
modell	3509	3755	4139
model2	3060	3306	3690
model3	3060	3306	3690

Table 4.9.2: The number of network rows for supply chain models.

for `model 2` and `model 3`. The GSG detects more network rows than RSD and M-INDSET for `model 1`. Regarding the computing time, the network extraction time for the algorithm GSG is still less than 5% of SSX solution time.

4.10 Discussion

In this chapter, we have shown that the problem of detecting a maximum EPN structure in an LP problem is related to balancing of generalised signed graphs. We have presented a network extraction algorithm which is based on the generalised signed graph of the corresponding coefficient matrix of the given LP problem. The computational results showed that this algorithm performs well compared to other algorithms.

We have observed that the quality of the solutions found by the GSG heuristic can be improved if one is ready to spend more time on detecting an embedded pure network. Indeed, our computational experiments have shown that the number of rows in the network found depends (sometimes, significantly) on the spanning tree. To enhance the results, one can build several spanning trees rather than just one in which case parallel algorithms may be considered. Another issue which helps to improve the performance of this heuristic is to use a local search improvement algorithm. In order to obtain an independent set S, we have used the degree-greedy algorithm. The set S, in many cases, can be enlarged by using local search improvement algorithms (for instance, see [95]). However, the improvement algorithms are normally time demanding.

Chapter 5

Alternative Approaches For Solving LPEN Problems

5.1 Introduction

In this work our aim is not only to detect EPN structures in LPEN problems, but also to exploit the network structures to find improved computational solutions to LPEN models. Three classes of solution methods which exploit embedded network structures have been discussed in the literature. In essence, the first group of methods is made up of specialised simplex algorithms for solving embedded network linear programs. These methods are based on the partitioning of the basis. The second group consists of the methods for solving network flow problems where the side constraints have a special (non-network) structure. These are singly constrained network problems and multicommodity flow problems. The methods in this group are specialised and have the drawback in that they do not solve an arbitrary network flow problem with side constraints. The third group consists of methods which are based on the strategy of problem decomposition using Lagrangean relaxation.

In this chapter, we first describe these three classes of solution methods and then explain our approach which involves creating an advanced starting point for the LPEN problem. The rest of this chapter is organised in the following way. In section 5.2, the problem statement is recast in a convenient form. In section 5.3, we review the specialised simplex algorithms which are based on factorisation of the basis matrix. In section 5.4, the partitioning algorithms that are based on the decomposition of the LP problem are described. In section 5.5, our network based advanced basis procedure is introduced in a summary form.

5.2 Problem Statement

We consider the LP problem with simple upper bounds in the standard form given in (1.5). Assume that an EPN structure within an LPEN problem is detected. After such a subset of EPN rows has been identified, the LP problem can be interpreted as satisfying the conservation of flow at the nodes defined by network rows and side conditions on flows specified by the non-network rows. Using the node-arc incidence matrix $\mathcal{N}$, the given LP problem (1.5) can be restated as an LPEN problem in a decomposed form shown below.

P_0:

$$\begin{aligned} \text{Minimise}\, z_0 = c'^T x' + c''^T x'' \\ \text{subject to} \\ \mathcal{N}x' = b' \\ Ux' + Vx'' = b'' \\ l' \leq x' \leq u' \\ l'' \leq x'' \leq u''. \end{aligned} \tag{5.1}$$

In (5.1), the subsets of rows and columns, submatrices and vectors are respectively defined as

$$\begin{aligned} & m = m_1 + m_2,\ n = n_1 + n_2, \\ & \mathcal{N} = [n_{ij}] \in \Re^{m_1 \times n_1},\ U = [u_{ij}] \in \Re^{m_2 \times n_1},\ V = [v_{ij}] \in \Re^{m_2 \times n_2}, \\ & c', x', l', u' \in \Re^{n_1},\ c'', x'', l'', u'' \in \Re^{n_2},\ b' \in \Re^{m_1} \text{ and } b'' \in \Re^{m_2}. \end{aligned}$$

In general l' and l'' may be $-\infty$ or finite and similarly u' and u'', $+\infty$ or finite. In many real life problems, l' and l'' are usually zero. For finite values of u', u'' inequalities $x' \leq u'$, $x'' \leq u''$ are turned into the equations $x' + s' = u'$, $x'' + s'' = u''$, where $s', s'' \geq 0$

and $s' \in \Re^{n_1}$, $s'' \in \Re^{n_2}$. The vector x' represents the network variables that have at least one non-zero element in the corresponding column of $\mathcal{N}$. The second constraint set is referred to as the side constraints and the vector x'' defines the non-network variables.

5.3 Specialised Simplex Methods

In the past specialised simplex methods have been developed to process difficult classes of LP problems. By and large these methods aim to reduce

- storage requirement of the original data and intermediate computed data (transformation matrices),
- computational time for each SSX iteration.

Two main approaches called *inverse compactification* and *mechanised pricing* have been developed by researchers [19]. Inverse compactification schemes involve maintaining the basis inverse matrix or an operationally sufficient substitute as the basis factors in a more advantageous form than the explicit one. One of the earliest and the most significant examples is the product form of the inverse given by Dantzig and Orchard-Hays [33] which takes advantage of the sparseness of most large matrices arising in practical applications. Subsequent and much improved schemes involve triangular factorisation, partitioning or use of a working basis that is more tractable than the original one, [72], [35].

Mechanised pricing, sometimes called column generation, involves the use of a subsidiary optimisation algorithm instead of direct enumeration to find the best nonbasic variable to enter the basis when there are many variables. The first contribution was given by Ford and Fulkerson [41], in which columns were generated by a network flow algorithm. Since then column generation has been applied to many mathematical programming problems. In this section, we review the specialised simplex algorithms which have been developed to solve structured LP problems using the inverse compactification techniques.

Structured LP problems can be partitioned such that the basis factorisation which is used in each iterative step of the SSX algorithm can be sped up. In the literature, some factorisation techniques which take advantage of special structures within the problem have been developed. These algorithms first partition the constraints of the LP problem into two classes: those that have the special structure (factored) and those that do not (explicit). Then the special structure is induced in the basis as well as the LP tableau. If the dimension of the basis submatrix consisting of rows in the special structure are allowed to vary (or even fail to be present) as the solution progresses, then it is called a *dynamic row factorisation*, otherwise it is called a *static row factorisation.*

The earliest example of factorisation was given by Dantzig [32] for simple upper bounds. Dantzig and Van Slyke [31] extended this approach to the GUB structure which is an example of static row factorisation. The GUB/SSX algorithm is based on the revised simplex method which uses a working basis of constraints which are not GUB rows. This basis representation is used for pivoting, pricing, and inversion. A variable in the basis which corresponds to a GUB row is called a *key variable.* This algorithm was known to reduce substantially computational time for problems which have a large number of GUB rows. In the 1970's, IBM, SCICONIC and other optimisation software developers had GUB based SSX.

Hartman and Lasdon [68] specialised this GUB approach to the multicommodity capacitated transhipment problems. They considered this problem as a block diagonal linear program with coupling rows and described a compact inverse version of the SSX method. In this case, the structure of the basic pure network columns introduces additional structure into the working basis, allowing further simplifications in the basis representation and update techniques. The only nongraph theoretic or nonadditive operations required are updating of the working basis inverse and multiplication by this inverse. Hence all the nonunimodular aspects of the problem are condensed into a minimal size single matrix. However, they did not implement their procedure. Graves and McBride [50] subsequently formalised and also generalised this factorisation approach.

Schrage [99] extended the succession of SUBs and GUBs by introducing VUB constraints. He used a compact or implicit scheme for storing the VUB constraints by expressing them in terms of the other variables. In this way he was able to solve the LP as if the special structure constraints did not exist. This permits the basis representation to be treated in two parts; one a large matrix which changes infrequently and thus needs occasional update, and the other a small working basis which requires regular update.

In addition, he applied this idea to GVUB constraints which arise frequently in models with fixed charges [98]. He demonstrated the manner in which the GVUB structure can be used to advantage in accelerating the computations within the simplex solution algorithm. It was claimed that implicit representation of GVUB constraints results in computational savings in the process of the revised simplex method. Daniel [30] presented a generalisation of VUB and GUB constraints and pointed out some issues of computational implementation of implicit representation.

Todd [103] developed a geometric interpretation of factorisation and showed that an extreme point of the feasible region of an LP problem lies in a face of the polyhedron of a simple structure. Then he defined directions of motion from the extreme point that either are in this face or move into a face of dimension one higher. Such directions form direction matrices which help to check the optimality of the current extreme point, and if not attained, proceed to an adjacent extreme point with strictly improved objective function value. Some special cases corresponding to variable and generalised upper bounds and a network polyhedron where the direction matrices can be obtained explicitly were examined.

A unifying mathematical framework for dynamic row factorisation was presented by Brown and Olson [18]. They reported three algorithms which are based on different LP model row structures: GUB, pure network rows, and generalised network rows. They reported a distinguishing feature of their dynamic factorisation is that it limits attention to binding constraints, handling binding factored constraints with great efficiency, and working with a relatively small number of binding explicit constraints. They implemented these algorithms and compared their performance with two well known commercial solvers OSL and CPLEX. Their computational results show that each of these algorithms is superior to the traditional solvers. Based on their experience they claimed that the efficiency of any particular factorised approach is influenced by the relative number of special constraints and their influence on the algorithm: size and quality of the special structure determines the influence of any particular factorisation applied to any particular LP.

Specialised simplex methods have been developed for problems having a special structure with a single side constraint. These methods exploit the near triangularity of the basis; in other words the factored structure completely dominates. Glover et al. [49] reported an implementation of Klingman and Russell design (see [78]) for solving singly constrained transhipment problems and reported computational results of the algorithm. Their limited computational results from a set of randomly generated test problems show that singly constrained transhipment network flow problems can be solved $25 - 30$ times faster than the state-of-the-art LP code (at the time of publication of their paper) APEX-III. They also developed a single pivot procedure for determining near optimal integer solutions when the optimal solution to this problem is not integer.

Generalised networks with a side constraint were addressed by Hultz and Klingman [74] who presented details for simplex priceout, column generation, and basis update. They also reported an implementation that solves the singularly constrained generalised network problem. They claimed that their code was between 6 to 28 times faster than APEX-III for only three randomly generated test problems. For the same problem, a cyclic method was developed by Mash [89]. It was claimed that this method provides a more efficient computational scheme than known adaptations of the simplex method since the codes for the pure network problem can be easily altered to accommodate the cyclic method. The method produces integral solutions at all iterations (except possibly the last one).

For large LP problems with pure network structures, researchers have developed specialised simplex methods in which the basis is considered in two parts: one corresponding to a rooted spanning tree defined on the underlying graph, and the other a general working basis. Efficient graph theoretic labelling and traversal algorithms are applied for pricing, basis representation and basis update. Klingman and Russell [78] sketched a factorisation method for solving transportation problems with side constraints. The method is basically the primal simplex method, specialised to exploit fully the topological structure embedded in the problem. The steps of updating costs and finding representations in the simplex procedure reduce to a sequence of simpler operations that utilise fully the triangularity of the spanning tree.

Chen and Saigal [28] presented a similar approach for solving capacitated network flow problems with additional linear constraints. While Klingman and Russell did not provide any computational results for transportation problems with side constraints, Chen and Saigal presented only four problems which have at most twenty side constraints. They compared their results with the out-of-kilter procedure and the MPSX solver. From the limited results, they conjectured that their procedure is roughly twice as fast as MPSX on constrained capacitated network flow problems.

Barr et al. [4] exploited the network structure in LP problems by applying the primal simplex algorithm in which the inverse of the working basis is maintained as an LU factorisation. They implemented these procedure called NETSIDE and compared it with general in-core LP systems; XMP, MINOS and LISS and a special system MCNF for multicommodity network flow problems. They tested the performance of their procedure on a set of randomly generated models and two real life problems. They reported that NETSIDE is approximately twice as fast as XMP and MINOS.

The presence of very efficient generalised network solvers motivated McBride [91] to develop a specialised simplex method for solving embedded generalised network problems with additional side constraints and additional variables. He presented methods for pricing, column generation, basis representation and basis update. He also described data structures representing LPs with embedded generalised network structures. He implemented these procedures and called it EMNET. EMNET was tested for a number of generated problems which had very few side constraints and side variables. For this class of models EMNET was found to be five times faster than MINOS.

The factorisation approach has been extended to embedded pure network structures by Glover and Klingman [47]. They introduced a special partitioning procedure for the LPEN problem and called it the Simplex Special Ordered Network (SON) procedure. This procedure is based on the steps of the primal simplex algorithm for the general case of EPN structures (see problem statement PS2 given in section 1.7) with side constraints and side variables.

The SON procedure is derived from a theoretical characterisation of the network topology of the basis embodied in the master basis tree. The basic variables related to the network portion are stored in a specially constructed graph called the *master basis tree.* The rules characterising the conditions for adding and deleting arcs, and specifying the appropriate restructuring of the master basis tree are summarised using fundamental exchange rules. The exchange rules and accelerated labelling algorithms for modifying the master basis tree in an efficient manner to replace arithmetic operations are the main features of this procedure. The operations normally performed by using the full basis inverse was replaced by special labelling and graph traversal techniques [5] to the master basis tree and its interface with the working basis.

They also showed that the topology of the master basis tree and exchange rules to reconstruct the master basis can be characterised by seven mutually exclusive and collectively exhaustive basis exchange cases in [48]. The organisation of the simplex SON method maintains the network portion of the basis as large as possible at each iteration of the simplex algorithm, thereby enabling these labelling and list procedures to operate on a maximally dimensioned part of the basis.

5.4 Decomposition Algorithms

LP problems of practical interest have the property that they may be described, in part, as composed of separate LP models tied together by a number of constraints considerably smaller than the total number imposed on the original problem. From a computational point of view, decomposition is a well established approach for solving large scale LP problems and especially those that contain constraints of special structures.

The decomposition techniques first manipulate the given LP problem and then uses one of solution procedures. Problem manipulation is a device for restating the given problem in an alternative form that is apt to be more amenable to solution. This leads to a residual problem which is called the master problem. On the other hand, solution strategies reduce an optimisation problem to a related sequence of simpler optimisation problems. This leads to subproblems amenable to solution by specialised algorithms.

The key problem manipulations are dualisation, projection, inner linearisation and outer linearisation while the key solution strategies are feasible directions, piecewise, restriction and relaxation. The definitions of these terms are expressed in more detail by Geoffrion in [45] and [46]. Many existing computational methods for large scale programming can be formulated as particular patterns of problem manipulations and solution strategies applied to a particular structure.

The first decomposition method was developed by Dantzig and Wolfe [34] to process LP problems with block angular structures. The block angular structure involves a set of disjoint block diagonal submatrices as well as a set of coupling constraints. If the adjacent block structure of the matrix is joined by a few columns from the beginning of the first block and the end columns of the neighbouring block, then the matrix has a staircase structure. These problems are often called multi-stage problems since each major block corresponds to a stage. If the number of blocks are limited to only two, then such a problem is called a two-stage problem.

It is well known that Benders decomposition is suited for two-stage (mixed) integer programming problems [11] and stochastic programming with two-stage and multi-stage problems [75]. In Benders decomposition, the problem defined by the binding constraints is called the *master problem* and the problems defined by the side constraints are called *subproblems*. The strategy of the decomposition procedure is to operate on two separate problems. Information between the two problems is passed from one problem to another until a point is reached where the solution to the original problem is achieved. These steps are interpreted as projection followed by outer linearisation and relaxation.

The other well known decomposition method is Lagrangean relaxation which can be applied to any problem and does not presuppose any special structure. A large scale mathematical programming problem is decomposed into simpler problems where constraints are partitioned into two categories. Constraints in the first category are retained as binding constraints and constraints in the second category are removed (relaxed), grouped together as side constraints and penalised for the constraint violation. Relaxation of these side constraints makes the corresponding sub-problem easier to solve than the original problem and this approach has become well known as Lagrangean relaxation. This procedure has been applied to many linear, non-linear, integer and mixed integer problems. We observe that the Lagrangean relaxation procedure can also be classified as dualising the side constraints, then relaxing the side constraints and solving a piecewise linear or non-linear Lagrangean dual problem by finding feasible directions.

The Lagrangean relaxation procedure has been used as an alternative approach to exploit EPN structures in LP problems. Venkataraman et al. [106] introduced a surrogate constraint and a Lagrangean approach to solve constrained network problems. The surrogate constraint approach is used to generate a singly constrained network problem which is solved using the algorithm of Glover et al. [49]. They compared their results with a subgradient optimisation approach.

Belling-Seib et al. [10] considered three alternative solution procedures for network flow problems with one side constraint. These methods are a specialised primal simplex algorithm, a straightforward dual method, and Lagrangean relaxation. Their computational results indicated that the specialised primal simplex algorithm is superior to other approaches for all but very small problems.

Bryson [24, 25] applied a parametric programming method to solve the Lagrangean dual problem obtained by dualising the single and multiple side constraints. Shetty [102] applied the Lagrangean relaxation method for a network flow problem with variable upper bounds. Recently, Hsu and Fourer [73] have also investigated this approach for solving the LPEN problem. In our investigation, we have chosen Lagrangean relaxation and Benders decomposition as primary steps for the solution of the LPEN problem. The two decomposition techniques are discussed in more detail in chapter 6.

5.5 An Advanced Basis Method For Solving LPEN Problems

The LPEN problem P_0 can be solved by any well known LP solution algorithm. While this is an obvious approach, our goal is to exploit the EPN structure in the problem thereby finding a computationally superior solution method. Simple LP solutions use a sequence of factored LP basis matrices represented in a sparse form. Advanced basis procedures [63], [83] are known to speed up the SSX algorithm. In our approach, we have taken the concept of advanced basis into consideration. The key idea is that we apply a decomposition method to obtain a good (near optimum and near feasible) solution so that it can be used as a "hot start" for a general SSX solver which then processes the original LP problem to optimality. We first apply Lagrangean relaxation since this procedure is a commonly established procedure to exploit the special structures. As an alternative decomposition procedure, we introduce Benders decomposition of the LPEN problem.

Our approach based on an advanced basis is described here in a summary form. After applying preprocessing and scaling procedures, we extract the embedded network structure in the coefficient matrix of the LP problem by applying one of the network extraction algorithms explained in the first part of the thesis. Then Lagrangean relaxation or Benders decomposition is applied to the LPEN problem. The Lagrangean relaxation procedure creates a pure network flow model by adding the non-network constraints into the objective function with Lagrangean penalties. A series of minimum cost network flow problems are then solved iteratively by assigning trial values to the Lagrangean multipliers at each iteration. The Benders procedure decomposes the LP problem into a master and a subproblem. At each iteration, a cut obtained by solving the subproblem is introduced into the master problem and then solved again iteratively.

From the solution of the last decomposed problem, an advanced basis is created; we call this a network based advanced SSX basis (NSSX). The NSSX basis constructed in this way is introduced as a starting basis to our experimental system FortMP [37] which is a general simplex solver. The choice of the pivotal algorithms (primal, dual or primal-dual) play an important role since one may be faster than the other in the final stage of the solution of the given LPEN problem.

Hsu and Fourer [73] suggested the dual-primal finishing strategy which avoids infeasibility thereby eliminating the need for Phase 1 in the two-phase simplex method. They claimed that this strategy is superior to the other simplex pivotal algorithms in terms of the number of iterations and CPU time. In our computational work, we have investigated three pivotal algorithms: primal, primal-dual and dual as SSX completion strategies. We can now sketch the overall algorithmic framework of our approach which is set out below. The schematic flow of this algorithm in block diagrammatic form is also depicted in Figure 5.5.1.

Solution Algorithm for the LPEN Problem

Step 1 *Preprocessing and scaling*

Apply a preprocessing procedure to reduce the size of the problem and a scaling procedure to increase the number of essential rows and columns that have only $+1, -1$ non-zero elements.

Step 2 *Network extraction*

Detect an EPN structure out of the set of essential rows and columns. Decompose the problem into network and non-network structures as shown in (5.1).

Step 3 *Solve the decomposition problem and create an advanced basis*

Either

Apply Lagrangean relaxation followed by the multiplier adjustment procedure (see section 6.2) and construct a triangular crash basis.

Or

Apply Benders decomposition (see section 6.4) and construct a starting basis by merging bases extracted from the solution of the master problem and the subproblem.

Step 4 *Complete the SSX solution*

Process the given LPEN problem applying the primal, dual or primal-dual SSX algorithm using the advanced basis obtained above.

Step 5 *Terminate the algorithm*

One of the key advantages of the approach described above is that it develops a way to exploit the EPN structure of an LP by using a standard network solver and standard LP solvers. Therefore, the main computational work including the linear algebra and the basis factorisation are taken care of by standard solvers. By contrast, the work described by other researchers in section 5.2 involves specialised SSX algorithms where all of the linear algebra is redeveloped.

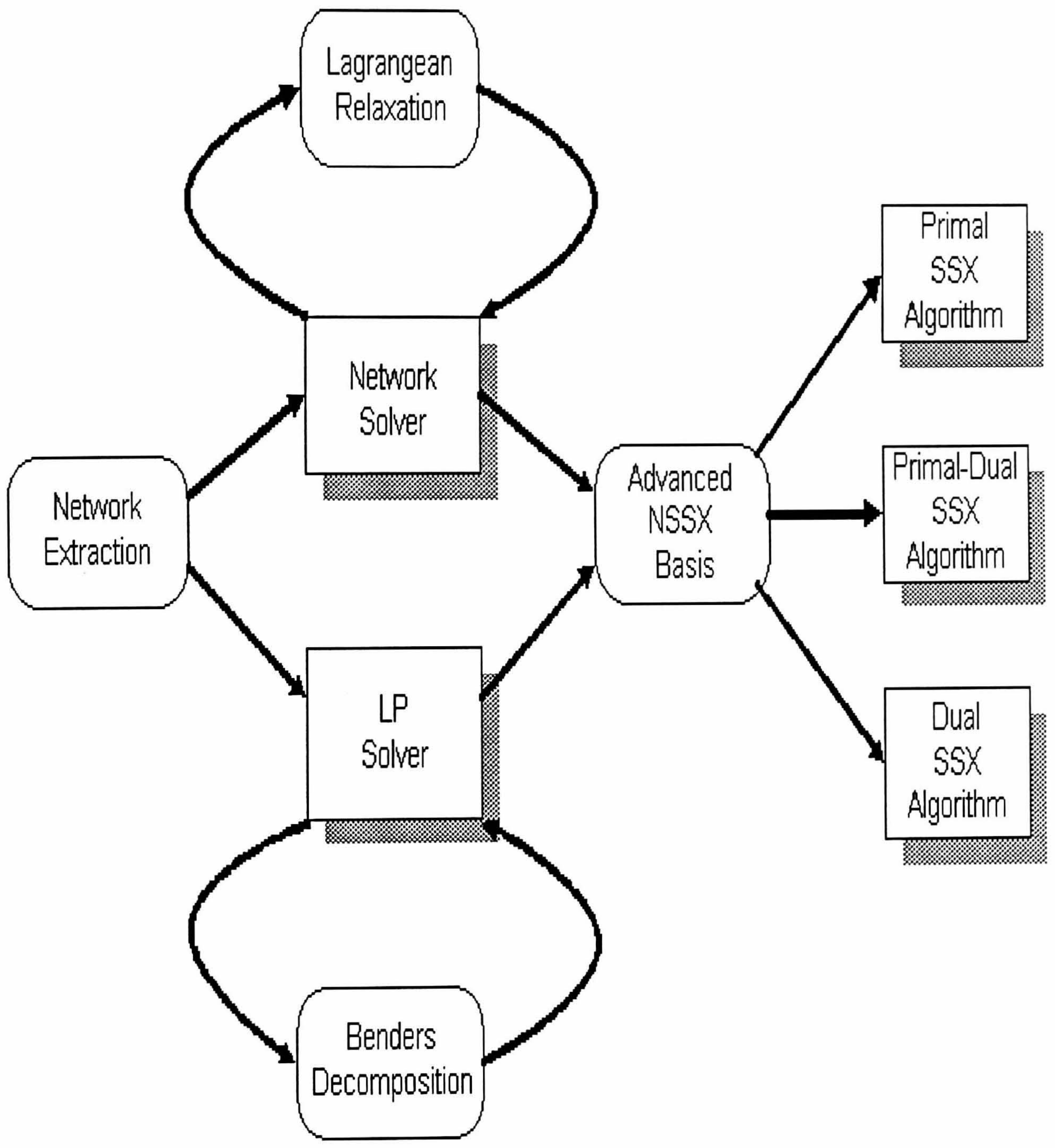

Figure 5.5.1: Schematic diagram of LPEN solution method using advanced NSSX basis.

From a computational point of view, this has some drawbacks since practitioners will almost certainly use commercial optimisation packages rather than developing code themselves. Software developers of optimisation packages such as CPLEX, IBM, OSL and FortMP are not interested in developing a specialised code when they have already developed and maintain a primal simplex, dual simplex and network simplex code.

Another reason why the methods described in section 5.2 have not been widely used is because those methods fail to take advantage of improvements in LP technology. While they may have compared well with LP solvers of late 1970's and early 1980's, see for example methods described in [4], [28], [91], they do not hold up well today since the state-of-the-art in mathematical programming has improved tremendously during the last decade. On the other hand, our approach works with standard solvers so this approach as time defines improved performance as the standard solvers get faster.

Chapter 6

Lagrangean Relaxation and Benders Decomposition

6.1 Introduction

Computing a 'good' feasible solution for a given minimisation or maximisation problem provides a lower and/or an upper bound to the optimum solution of the original problem. Lagrangean relaxation and Benders decomposition are two well established techniques for calculating such solutions. In this chapter, we first consider the theoretical properties of Lagrangean relaxation and Benders decomposition. We then discuss how these methods are applied to solve the LPEN problem given in (5.1).

The rest of this chapter is organised as follows. In section 6.2, we apply Lagrangean relaxation to the LPEN problem and introduce a multiplier adjustment algorithm to solve the Lagrangean dual problem. In section 6.3, we first aggregate the side constraints of the LPEN problem to reduce the size of the original problem and then apply the Lagrangean relaxation procedure. Section 6.4 focuses on an alternative decomposition procedure namely Benders decomposition.

6.2 Lagrangean Relaxation

6.2.1 Basic Methodology

One of the most computationally useful ideas of the 1970's is the observation that many hard problems can be viewed as easy problems complicated by a relatively small set of side constraints. A large scale mathematical programming problem is decomposed into simpler problems where constraints are partitioned into two categories. Constraints in the first category are retained as binding constraints and those in the second category are grouped together as side constraints. The latter constraints are loosely termed as complicating constraints as they hinder the solution of an otherwise easy problem.

Relaxation of these side constraints makes the corresponding sub-problem easier to solve than the original problem. In the Lagrangean relaxation method, the side constraints are attached by multipliers (dualising side constraints), relaxed and then introduced into the objective function. In other words, the complicating constraints are replaced by penalty terms in the objective function. These penalty terms are computed as the amount of violations of the side constraints multiplied by their dual variables.

In the last decade, Lagrangean relaxation has grown from a successful theoretical concept to a tool that has increasingly been used in large scale mathematical programming applications. There are several surveys on Lagrangean relaxation (for example, see [38], [44], [101]) as well as extensive use of Lagrangean relaxation in practical applications of linear, non-linear, dynamic and integer programming (for example, see [24], [39], [100], [52], [53]). In the domain of combinatorial optimisation, the Lagrangean relaxation method was first introduced by Held and Karp [69]. They applied this technique to the travelling salesman problem. Since then, this method has been widely used to solve other classes of constrained optimisation problems [9], [38], [70], [71]. In discrete optimisation, Lagrangean decomposition is preferred to the LP relaxation to provide a lower bound for a minimisation problem and is interfaced with a branch and bound procedure [2].

According to Fisher [40], there are three major questions in designing a Lagrangean based system: 1) which constraints should be relaxed, 2) how to compute good multipliers, 3) how to deduce a good feasible solution to the original problem, given a solution to the relaxed problem. Roughly speaking, the answer to the first question is that the relaxation problem must be significantly easier than the original problem. For the second question, there is a choice to use either a general purpose procedure called the subgradient method or a "smarter" method called multiplier adjustment or various versions of the simplex method implemented using column generation techniques. Similarly, the answer to the third question tends to be problem specific.

6.2.2 A Lagrangean Relaxation of the LPEN Problem

Consider the problem P_0 in (5.1). We group and relax the non-network (side) constraints $Ux' + Vx'' = b''$; weight them using the Lagrangean multipliers λ and introduce in the objective function. The problem is then restated as the Lagrangean relaxation $P_{L(\lambda)}$:

$$\begin{aligned} \text{Minimise}\, z_{L(\lambda)} = \lambda^T b'' + (c'^T - \lambda^T U)x' + (c''^T - \lambda^T V)x'' \\ \text{subject to} \\ Nx' = b' \\ l' \leq x' \leq u' \\ l'' \leq x'' \leq u'' \\ \lambda \in \Re^{m_2} \text{ and unrestricted.} \end{aligned} \tag{6.1}$$

Clearly, the Lagrangean multipliers λ penalise the violation of the corresponding side constraints introduced in the objective function. It is easily seen that the relaxed problem $P_{L(\lambda)}$ is a pure network flow problem in x', the feasibility of x'' can be trivially satisfied, whereas P_0 is a general LP problem. This network flow problem is solved efficiently by special algorithms such as the network simplex algorithm [77], [87].

The Lagrangean function $z_{L(\lambda)}$ is convex, piecewise-linear and continuous; these important structural properties make the Lagrangean relaxation problem easier to solve. However, the Lagrangean function is not everywhere differentiable. It is differentiable whenever the optimal solution of the Lagrangean subproblem is unique. It is, however, subdifferentiable everywhere in the convex hull of the problem. Let $(\pi, \lambda, \sigma_1, \eta_1, \sigma_2, \eta_2)$ be dual variables corresponding to constraints in P_0. The dual of the original LP problem P_0 is then formalised as

D_0 :

$$\begin{gathered} \text{Maximise } b'^T\pi + b''^T\lambda + l'^T\sigma_1 + l''^T\sigma_2 - u'^T\eta_1 - u''^T\eta_2 \\ \text{subject to} \\ N^T\pi + U^T\lambda + \sigma_1 - \eta_1 = c' \\ V^T\lambda + \sigma_2 - \eta_2 = c'' \\ \sigma_1, \sigma_2, \eta_1, \eta_2 \geq 0 \\ \pi, \lambda \text{ unrestricted.} \end{gathered} \tag{6.2}$$

The dual problem of $P_{L(\lambda)}$ is the same as problem D_0 with the different right hand side values. The Lagrangean dual problem P_D with respect to the side constraints is to find the set of Lagrangean multipliers λ^* that maximise the Lagrangean function $z_{L(\lambda)}$. The objective function of P_D is given by $\max\limits_{\lambda}$, $\min\limits_{(x',x'')}$ process and shown as follows;

P_D:

$$z^*_{L(\lambda^*)} = \max_{\lambda}\{\min_{(x',x'')}(c' - \lambda^T S)x' + (c'' - \lambda^T T)x'' + \lambda b''\} \tag{6.3}$$

where the Lagrangean multipliers are computed by solving the LP problem

$$\begin{aligned} z^*_{L(\lambda^*)} &= \text{Maximise } w \\ &\quad \text{subject to} \\ w &\leq f_j + \lambda^T g^j, \;\; j = 1, \cdots, K \end{aligned} \tag{6.4}$$

In (6.4), f_j is the objective value of the original problem and g^j is the subgradient of the jth basic solution

$$f_j = c'^T x'^j + c''^T x''^j, \quad g^j = b'' - Ux'^j - Vx''^j \tag{6.5}$$

and K denotes the number of all basic solutions of the Lagrangean problem.

We may consider the Lagrangean problem $P_{L(\lambda)}$ and its relationship with the original problem P_0 which is described with the following properties.

Property 1:

For any vector λ of Lagrangean multipliers, the optimum value $z^*{}_{L(\lambda)}$ of the Lagrangean function is a lower bound on the optimal objective function value z_0^* of the original primal optimisation problem P_0, that is, $z^*_{L(\lambda)} \leq z_0^*$ for all λ.

Property 2:

There exists a set of Lagrangean multipliers λ^* for which the objective value of the Lagrangean relaxation $z^*_{L(\lambda^*)}$ attains the optimal value of the original problem P_0, that is, $z^*_{L(\lambda^*)} = z_0^*$.

Property 3:

For some choice of the Lagrangean multiplier vector λ, if the solution of the Lagrangean relaxation (x', x'') is feasible in the optimisation problem and satisfies the complementary slackness conditions involving non-negative primal $\{x', x'', s', s''\}$, and non-negative dual $\{\sigma_1, \sigma_2, \eta_1, \eta_2\}$ variables with the property $x'\sigma_1 = x''\sigma_2 = s'\eta_1 = s''\sigma_2 = 0$, then (x', x'') is an optimal solution to P_0.

It is worthwhile to note that when applying Lagrangean relaxation to linear programs, the first property has long been known, but Geoffrion [44] observed that the second property does not hold for integer programs in general.

6.2.3 Determination of the Lagrangean Multipliers

One of the key issues in the use of Lagrangean relaxation is to design a procedure to optimise the Lagrangean dual problem because the Lagrangean dual often requires a specialised algorithm that must be tailored for each application model. In general, the basic approaches to solve the problem P_D in (6.4) can be classified into three categories;

- subgradient optimisation,
- various versions of the simplex method implemented using column generation techniques,

- multiplier adjustment procedures.

In this section, we give general basic issues of these methods and present a multiplier adjustment approach to solve the Lagrangean dual problem of the LPEN problem.

I- Subgradient Optimisation

Subgradient optimisation is a traditional approach used to solve the Lagrangean dual problem. The method is easy to program and has achieved success on many practical problems. Its computational performance and theoretical convergence properties are discussed in Held, Wolfe and Crowder [71] and in several references on non-differentiable optimisation. It is an adaptation of the gradient method in which subgradient is used instead of gradient. The method starts from an initial set of multipliers, λ^0. At iteration k, a sequence of multipliers $\{\lambda^k\}$ is calculated by the rule

$$\lambda^k = \lambda^k + t_k g^k, \tag{6.6}$$

where g^k is a subgradient direction of the Lagrangean dual problem and t_k is the step size which can be calculated commonly in practice as

$$t_k = \frac{\pi\ (\mathrm{Z_{UB}} - \mathrm{Z_{LB}})}{\sum\limits_{i=1}^{m} g_i^2}.$$

The step size depends upon the gap between the current lower bound Z_{LB} and the upper bound Z_{UB} and the user defined parameter, generally $0 < \pi \leq 2$, with $\sum\limits_{i=1}^{m} g_i^2$ being a scaling factor. In subgradient optimisation, bounds do not monotonically improve and the method is terminated upon reaching an arbitrary iteration limit.

II- A Simplex Based Algorithm

Another class of algorithms for solving the Lagrangean dual problem is based on applying a variant of the simplex method to the original problem and generating an appropriate entering variable at each iteration by solving a Lagrangean relaxation problem with the current value of simplex multipliers. Primal simplex with column generation has been used for this class of solution procedures. However, this approach is known to converge very slowly and does not produce monotonically increasing lower bounds [38]. Therefore, researchers have developed column generation implementations of dual forms of the simplex method, especially the dual and primal-dual simplex method. The primal-dual simplex method can also be modified to make it the method of steepest ascent for the Lagrangean dual problem.

Marsten et al. [88] had applied the modification of these simplex procedures which they called "boxstep". The boxstep proceeds as follows. It begins with initialisation of λ^0 and a sequence $\{\lambda^k\}$ is generated. In order to obtain λ^{k+1} from λ^k, the problem given in (6.4) is solved with the additional requirement that $|\lambda_i - \lambda_i^k| \leq \delta$ for some fixed positive δ. Assume that λ' is the optimal solution of this problem. If $|\lambda_i' - \lambda_i^k| \leq \delta$ for all i, then λ' is optimal in the Lagrangean dual problem. Otherwise, set

$$\lambda^{k+1} = \lambda^k + t_k(\lambda'^k - \lambda^k)$$

where t_k is a scalar. The same procedure is carried out until optimality is reached.

III- A Multiplier Adjustment Algorithm

A multiplier adjustment method is a specialised procedure that solves a Lagrangean dual problem by exploiting the structure of a particular model. It is also known as the "Lagrangean dual ascent" as it can be viewed as an ascent procedure. This method is often preferred to subgradient optimisation in solving a Lagrangean dual problem since an ascent procedure guaranties monotone bound improvement. Fisher et al. [39] and Guignard et al. [53] gave an application of this method to an assignment problem and an allocation problem. Guignard et al. [52] and Beasley [9] discussed the theoretical issues of this method.

Developing a multiplier adjustment procedure is considered to be an art; different problems require different multiplier adjustment algorithms unlike subgradient optimisation which is capable of being applied directly to many problems. It is an iterative method and starts with an initialised set of Lagrangean multipliers. The initialisation of multipliers is dependent on the underlying model structure and affects the quality of the final bound. At each iteration, only a small subset of multipliers is examined and one or more multipliers of violated constraints are adjusted. The calculation of the multiplier adjustment amount is also problem specific. At iteration k of the multiplier adjustment method, a set of ascent directions is determined such that the effect on the optimum value of the Lagrangean dual problem by a movement along a direction is evaluated. The common improvement on multipliers is made by the rule

$$\lambda^{k+1} = \lambda^k + t_k g^k, \tag{6.7}$$

where g^k is an ascent direction and t_k is the step size. The step size can be chosen either to maximise $z_{L(\lambda^k+t_k g^k)}$ or to get to the first point at which the directional derivative changes. The ascent direction involves changes to multipliers corresponding to violated (greater and equal) constraints which is made generally with the following rules:

- if ${g_i}^k < 0$, then reduce the multiplier ${\lambda_i}^k$
- if ${g_i}^k = 0$, then do not change the multiplier ${\lambda_i}^k$
- if ${g_i}^k > 0$, then increase the multiplier ${\lambda_i}^k$.

The determination of the set of the directions and order in which directions are scanned are problem specific and affect the final bound. If the set of ascent directions is empty, in other words all constraints are satisfied, or the set has no improving direction, the procedure is terminated even though an optimal dual solution is not found.

6.2.4 Solving Lagrangean Relaxation of the LPEN Problem

Consider the problem $P_{L(\lambda)}$ in (6.1). It actually consists of two subproblems; network and non-network. Thus, solving the kth Lagrangean relaxation problem involves a network part which provides the solution for network variables x' and a trivial non-network subproblem which finds solution for non-network variables x''. The non-network subproblem is solved simply by setting the side variables to bounds in their feasibility ranges according to their reduced costs as

$$\begin{array}{ll} ({x_j}'')^k = ({l_j}'')^k \;\; if \;\; {c_j}'' - (\lambda^k)^T V_j \geq 0, \\ ({x_j}'')^k = ({u_j}'')^k \;\; if \;\; {c_j}'' - (\lambda^k)^T V_j < 0 \end{array} \tag{6.8}$$

where V_j denotes the jth column of the matrix V. The network subproblem can be solved by the primal or dual network simplex algorithm. For our computational work, we use MINET a minimum cost network flow solver developed by Maros [82], [86].

As described in the previous section, there exists three types of techniques to solve the Lagrangean dual problem. The subgradient method, however, fails for the Lagrangean dual of the LPEN problem because $z_{L(\lambda)}$ cannot be guaranteed to be finite for all λ. Another reason is that the subgradient algorithm does not guarantee that there exists the monotone increasing lower bound; it might be worse than the previous lower bound at the process of the algorithm.

As noted by Fisher [38], simplex based methods are generally harder to program and have not performed quite so well computationally as the subgradient method. However, recently Hsu and Fourer [73] described a method for solving the Lagrangean dual problem using the trust region constraints. We consider that this is similar to the Boxstep method. In our procedure, we solve $P_{L(\lambda)}$ only by a multiplier adjustment approach which finds a good (near optimal and near feasible) solution for the original LP problem as it progressively improves the lower bound.

The multiplier adjustment heuristic solves iteratively a sequence of network linear programs with different values of λ. Even though the solution $(x', x'')^k$ is an optimal solution for the Lagrangean relaxation problem at the kth iteration, it is not guaranteed that it is a feasible solution of the original LP problem. To improve the current lower bound, the algorithm finds another direction and updates the multipliers. The procedure is stated below; for more detail the reader is referred to [57].

The Multiplier Adjustment Algorithm

Step 1 *Initialisation*

Let $k = 0$, assign the Lagrangean multipliers an initial value, say $\lambda^0 = 0$.

Step 2 *Solve subproblems*

Solve the network subproblem and find the network flows $(x')^k$. Apply rule (6.8) to bound restrictions on the non-network variables to determine their solution values. If there does not exist a feasible solution for the network subproblem, then terminate the algorithm and conclude that the original LP problem has no feasible solution.

Step 3 *Compute gradients*

Calculate gradients for each side constraint. Check gradients; if all side constraints are satisfied, that is ${g_i}^k = 0$, then stop the algorithm and conclude that a feasible solution to the original problem has been found.

Step 4 *Compute the adjustment*

Compute $\Delta\lambda = \min\{c'^k - (\lambda^k)^T U_j,\ c''^k - (\lambda^k)^T V_j\}$.

Step 5 *Update the Lagrangean multipliers*

Choose a set of the violated constraints. Update the Lagrangean multipliers, ${\lambda_i}^{k+1}$ of the violated constraints as ${\lambda_i}^{k+1} = {\lambda_i}^k + |\Delta\lambda|$.

Step 6 *Update the problem*

Let $k = k + 1$, construct another minimum cost network flow problem with new multipliers and go to step 2.

It is worthwhile to mention here that, for $k = 0$, the problem represents the network components of the embedded network LP problem since the side constraints are all ignored ($\lambda = 0$). Hence, the objective value of the network subproblem and the objective value of the LP problem P_0 are the same. Hsu and Fourer [73] called this the zero-multiplier method.

6.3 Aggregation of Side Constraints

In mathematical programming, aggregation techniques consist of a set of methods for solving optimisation problems by combining data, using an auxiliary model which is reduced in size and complexity relative to the original model. They have been developed to help form the most appropriate reduced models that provide good approximations to the original problem [94]. In order to perform a row or a column aggregation, a set of constraints or variables are replaced with a single row or a single column. If a set of rows is multiplied by different weights and aggregated to a single constraint, this is known as a *weighted aggregation.* If a row is selected such that it dominates a set of rows, the choice of this row is called an *aggregation by dominance.* The same terminology applies to columns.

In this section, we consider another approach which quickly finds a near optimum solution of the LPEN problem by aggregating the side constraints. Instead of solving the problem with the original side constraints, the Lagrangean relaxation problem with respect to the aggregated side constraints is used to find a starting basis. The weight vector consisting of only ones is used to aggregate the non-network constraints. We apply a basic heuristic to aggregate side constraints in the given embedded network problem in (5.1). Each side constraint i is considered to group according to the type of variables. Variables which appear in the given side constraint i are either network variables or non-network variables and can be classified into two subsets as

$$I_i(N) = \{j \mid a_{ij} \neq 0, \ j \text{ is a network variable}\} \text{ and}$$

$$I_i(\bar{N}) = \{j \mid a_{ij} \neq 0, \ j \text{ is a non-network variable}\}.$$

The side constraints are aggregated into three groups using the following criteria:

- the group of rows which have only network columns, that is $I_i(\bar{N}) = \emptyset$. Let this be defined as a subset R_1 of the row indices i,
- the group of rows which have only non-network columns, that is $I_i(N) = \emptyset$. Let this be defined as a subset R_2 of the row indices i,
- the group of rows which have both network and non-network columns, that is $I_i(\bar{N}) \neq \emptyset$ and $I_i(N) \neq \emptyset$. Let this be defined as a subset R_3 of row indices i.

The aggregated embedded LP problem becomes a minimum cost network flow problem with only three side constraints and is set out below.

AP_0 :

Minimize $z_0 = c'^T x' + c''^T x''$

subject to

$$Nx' = b',$$

and the following aggregated constraints

R_1:

$$\sum_{i \in R_1} \Big(\sum_{j \in I_i(N)} s_{ij} \Big) x_j' = \sum_{i \in R_1} b_i''$$

In vector notation, this can be written as $S_1 x' = \beta_1$, where

$$S_1 = [\sum_{i \in R_1} s_{i1}, \cdots, \sum_{i \in R_1} s_{in_1}] \text{ and } \beta_1 = \sum_{i \in R_1} b_i''.$$

R_2:

$$\sum_{i \in R_2} (\sum_{j \in I_i(\bar{N})} t_{ij}) x_j'' = \sum_{i \in R_2} b_i''$$

In vector notation, this can be written as $T_2 x'' = \beta_2$, where

$$T_2 = [\sum_{i \in R_2} t_{i1}, \cdots, \sum_{i \in R_2} t_{in_2}] \text{ and } \beta_2 = \sum_{i \in R_2} b_i''.$$

R_3:

$$\sum_{i \in R_3} [\sum_{j \in I_i(N) \cup I_i(\bar{N})} (s_{ij} x_j' + t_{ij} x_j'')] = \sum_{i \in R_3} b_i''$$

In vector notation, this can be written as $S_3 x' + T_3 x'' = \beta_3$, where

$$S_3 = [\sum_{i \in R_3} s_{i1}, \cdots, \sum_{i \in R_3} s_{in_1}],\ T_3 = [\sum_{i \in R_3} t_{i1}, \cdots, \sum_{i \in R_3} t_{in_2}] \text{ and } \beta_3 = \sum_{i \in R_3} b_i''.$$

$$l' \leq x' \leq u',$$

$$l'' \leq x'' \leq u''.$$

The multiplier adjustment procedure explained in the previous section can now be applied to solve the following Lagrangean relaxation of the aggregated embedded network flow problem in which there are only three multipliers to be adjusted.

$AP_{L(\lambda)}$:

$$\begin{aligned} \text{Minimise } z_{L(\lambda)} = \lambda_1\beta_1 + \lambda_2\beta_2 + \lambda_3\beta_3 + c'^T x' + c''^T x'' - \lambda_1 U_1 x' - \lambda_2 V_2 x'' - \lambda_3(U_3 x' + V_3 x'') \\ \text{subject to} \\ Nx' = b' \\ l' \leq x' \leq u' \\ l'' \leq x'' \leq u'' \end{aligned} \tag{6.9}$$

and vector $\lambda^T = \{\lambda_1, \lambda_2, \lambda_3\}$ is unrestricted.

6.4 Benders Decomposition

This decomposition was introduced by Benders [11] to solve mixed integer programming problems. Since then, it has been applied to many large scale problems in mathematical programming, especially, for solving two stage as well as multistage stochastic programming problems [75]. The embedded network flow problem may be considered as a two stage problem in which the network part is the first stage and the non-network part is the second stage. This has motivated us to use Benders decomposition to create an advanced starting basis for solving the original LPEN problem. In this section, we describe our algorithm based on the Benders decomposition procedure.

6.4.1 Theoretical Framework

We consider the LPEN problem P_0 given in (5.1) and split the original problem into a master P_{master} and a subproblem P_{sub}. The latter is used to generate cuts as in the Benders decomposition method. Initially, the P_{master} problem is a pure network problem stated as

P_{master}:

$$\begin{aligned} \text{Minimise } z_M = c'^T x' \\ \text{subject to} \\ Nx' = b' \\ l' \leq x' \leq u' \end{aligned} \tag{6.10}$$

Let x'^* denote an optimal solution of P_{master}, then the subproblem P_{sub} is defined as
P_{sub}:

$$\begin{gathered}\text{Minimise } z_S = c''^T x'' \\ \text{subject to} \\ Vx'' = b'' - Ux'^* \\ l'' \leq x'' \leq u''\end{gathered} \tag{6.11}$$

The corresponding dual linear program D_{sub} of the subproblem P_{sub} is stated using dual variables $\pi^T = (\pi_1, \pi_2, \pi_3)^T$ as
D_{sub}:

$$\begin{gathered}\text{Maximise } {\pi_1}^T(b'' - Ux'^*) - {\pi_2}^T u'' + {\pi_3}^T l'' \\ \text{subject to} \\ {\pi_1}^T V - {\pi_2}^T + {\pi_3}^T \leq c'' \\ \pi_1 \text{ free} \\ \pi_2, \pi_3 \geq 0\end{gathered} \tag{6.12}$$

The feasibility condition of the problem D_{sub} is that

$$ {\pi_1}^T V - {\pi_2}^T + {\pi_3}^T - c'' \leq 0 \text{ and } \pi_2, \pi_3 \geq 0.$$

The assumption that P_0 is feasible requires the feasibility of P_{sub} for all values of x' satisfying $l' \leq x' \leq u'$ and $Nx' = b'$. In addition, from duality theory, problem D_{sub} is finite if and only if

$$ {\pi_1}^T(b'' - Ux'^*) - {\pi_2}^T u'' + {\pi_3}^T l'' \leq 0. \tag{6.13}$$

This constraint can be appended to the first master problem to ensure that the solution to the new master problem leads to a feasible solution of the original problem; this constraint is called a *feasibility cut.*

By considering an upper bound on the objective function of D_{sub}, the smallest value of this upper bound is denoted by θ and used to formulate the master problem as follows.

$$
\begin{aligned}
&\text{Minimise } z_M = c'^T x' + \theta \\
&\text{subject to} \\
&Nx' = b' \\
&\theta - (\pi_1{}^*)^T(b'' - Ux') + (\pi_2{}^*)^T u'' - (\pi_3{}^*)^T l'' \geq 0 \\
&(\pi_1{}^*)^T(b'' - Ux') - (\pi_2{}^*)^T u'' + (\pi_3{}^*)^T l'' \leq 0 \\
&l' \leq x' \leq u' \\
&\theta : \text{ free}
\end{aligned}
\tag{6.14}
$$

where $\pi^* = \{\pi_1{}^*, \pi_2{}^*, \pi_3{}^*\}$ is the optimal solution of D_{sub}. In (6.14), the constraint

$$\theta - (\pi_1{}^*)^T(b'' - Ux') + (\pi_2{}^*)^T u'' - (\pi_3{}^*)^T l'' \geq 0 \tag{6.15}$$

is called *an optimality cut* which ensures that the subproblem is solved to optimality. At each iteration of the decomposition procedure, either an optimality cut (6.15) or a feasibility cut (6.13) is added to the master problem; if primal infeasibility (dual unboundness) is found for the subproblem, then the feasibility cut is added. If the primal problem is feasible (dual bounded), then the optimality cut is then appended to the master problem. Then the master problem which is a network flow problem with side constraints is solved.

Each optimal solution of the master problem (x'^*, θ^*) is suboptimal and gives a lower bound on the objective value of the LPEN problem, that is $LB = c'^T x'^* + \theta^* \leq z_0^*$. When the master and subproblem are both feasible, the solution (x'^*, x''^*) is a feasible solution for the problem (5.1) and creates an upper bound, that is $UB = c'^T x'^* + c''^T x''^* \geq z_0^*$. At each pass of the decomposition, the lower bound is updated, at the kth pass the lower bound is calculated as

$$LB^k = c'^T (x'^*)^k + \theta^{*k}. \tag{6.16}$$

Initially, the upper bound is set to infinity. If the subproblem at iteration k is solved to optimality, then a new upper bound may be found by the relation

$$UB^k = \min\{UB^{k-1}, c'^T (x'^*)^k + c''^T (x''^*)^k\}. \tag{6.17}$$

When the relative gap satisfies the following property, then the problem is considered to have been solved with sufficient accuracy [75],

$$\frac{UB^k - LB^k}{|LB^k| + 1} \leq TOL \tag{6.18}$$

and the algorithm is terminated. We use $TOL = 10^{-6}$ in our computational experiments.

6.4.2 Benders Decomposition For the LPEN Problem

It is well known that Benders decomposition converges to an optimal solution in a finite number of iterations. Instead of solving the entire embedded network flow problem with Benders decomposition, our aim is to take advantage of a good intermediate solution obtained by the decomposition algorithm and create an advanced starting point. Therefore, we preset the maximum number of iterations MAXP and use it as a termination criteria of Benders decomposition. Our procedure is set out below.

Benders Decomposition For the LPEN Problem

Step 1 *Construct a master and a subproblem*

Decompose the LPEN problem into a master and a subproblem. Initialise the maximum pass number, *MAXP*.

Repeat for $k = 0, \cdots, MAXP$

{ **Step 2** *Solve the master problem*

Solve the master problem by an SSX solver. (When $k = 0$, use a network solver.) If the solution is infeasible, then conclude that the entire LP problem is infeasible and go to step 7.

Step 3 *Construct a new subproblem and solve*

By fixing the solution x'^* of the master problem and revising the right hand side, construct P_{sub} and then solve this subproblem by the Primal SSX algorithm.

```
    Step 4 Obtain the lower bound
        Calculate the lower bound using relation (6.16).
    Step 5 Create a cut
        If an optimal solution is found for the subproblem, then
            calculate the upper bound using relation (6.17),
            check the optimality conditions shown in (6.18).
            If the optimality condition is satisfied, then
                go to step 7.
            Else
                create an optimality cut (6.15).
            Endif
        Elseif the subproblem does not have a feasible solution, then
            create a feasibility cut (6.13).
        Endif
    Step 6 Update the problem
        Add this cut to the master problem.}
Step 7 Terminate the algorithm
```

Chapter 7

Solving an LP with an Embedded Network Structure: Advanced Basis

7.1 Introduction

The simplex method requires a starting point which is a basic solution. The calculation of an initial basis is of great importance as it determines to a large extent the amount of computation that is required to solve the problem to optimality. The traditional way is to use a unit starting basis which consists of all the logical (including artificial) variables. For the solution of large scale LP models, the performance of the revised sparse simplex method or its variants can be considerably enhanced if an advanced basis is introduced instead of using the traditional all-logical initial basis. In general, procedures to create an advanced basis are designed to construct the initial basis computationally in an effective way. Many LP systems provide some forms of crash procedures. The most well established of these procedures constructs a triangular basis using some heuristics; for example, see Bixby [12], Maros and Mitra [83].

There has been relatively little attention paid to this computational aspect of the simplex method compared to other aspects such as reinversion, Phase-I procedures and pricing. In this chapter, we introduce two procedures to create an advanced basis for a general LP solver. The solution of a network flow problem after applying Lagrangean relaxation or iterated solutions of the master and subproblem in Benders decomposition is used to compute a good (near optimal and near feasible) solution for the given LPEN problem. Active variables identified in this way are then used to create an advanced basis.

The rest of this chapter is organised in the following way. In section 7.2, we first briefly describe how to create the all-logical initial basis and review some well known methods of obtaining an advanced starting point reported in the literature and then give the basic description of a lower triangular symbolic crash procedure. In section 7.3 and section 7.4, we describe two procedures for computing an advanced basis by applying Lagrangean relaxation and Benders decomposition, respectively. Computational results are presented in section 7.5 followed by a discussion of the results in section 7.6.

7.2 Unit Starting Basis and Crash Procedures

Consider the LP problem in the standard form given in (1.5). An initial basis for the LP problem is always obtained by augmenting the constraint matrix A by adding the artificial and slack variables so that the augmented matrix contains an identity matrix. This is the traditional method of obtaining a starting basis and leads to well known all-logical (unit) basis. The given LP problem may be reexpressed as

$$\begin{gathered}\text{Minimise } c^T x + {c_a}^T x_a \\ \text{subject to} \\ \begin{bmatrix} A & : & I \end{bmatrix} \begin{bmatrix} x \\ \\ x_a \end{bmatrix} = b \\ x \geq 0, x_a = 0\end{gathered}$$

The vector x_a contains the artificial variables that must be all zero and $c_a = 0$. These variables are known as *logical* variables. The variables x are called *structural (or natural)* variables. A basis B is then immediately created out of the columns of $[A, I]$ which is made up of the logical variables as $B = I$. The objective at this stage is to drive artificial variables to zero (make as many as possible non-basic) in order to obtain a feasible solution to the original problem. This is known as the Phase I procedure which has two outcomes; either a feasible solution is found or it is established that there is no feasible solution to the given constraints. If a feasible solution is obtained after Phase I, then the initial basic feasible solution is found. For the above system, this stage can be formulated as

$$\text{Minimise } \bar{c}^T x + \bar{c}_a^T x_a + x_o$$
$$\text{subject to}$$
$$B^{-1}Ax + B^{-1}Ix_a = B^{-1}b = \beta \geq 0$$
$$x \geq 0, x_a = 0$$

where $\bar{c}$ and $\bar{c}_a$ are the updated objective coefficients and x_o is the current objective value. B^{-1} is the inverse basis and usually represented in a factored form; product and elimination form of the inverse. Thus, computing the factors without going through a series of pivots and corresponding set of updates sequentially can be interpreted as a block-pivot operation.

Since the traditional procedure starts with a unit basis that does not have any structural variables, it has been shown to be computationally unattractive, particularly for large scale problems. However, it has long been recognised that a starting basis which contains some structural variables needs fewer iterations and less time to find an optimal solution compared to the all-logical basis (see [84]). Such a basis is called an *advanced basis* and the procedure by which it is computed is known as a *crash procedure.* In other words, crash procedures are methods of creating a starting basis which contains more structural variables. All crash procedures aim to find block pivots and carry out the corresponding factorisation or reinversion. Alternative heuristic procedures for creating advanced starting bases have been described in the literature. These crash procedures can be classified into two main categories, namely triangular and block triangular.

Triangular Crash

All triangular crash procedures have a common strategy which is to replace as many logicals and artificials in the logical/artificial basis with the structural variables in such a way that the resulting basis matrix has a triangular form with a zero free diagonal. If the matrix found by the triangular crash procedures does not satisfy the full row rank property, then it is usually augmented by logical variables as necessary to form a non-singular triangular matrix that can be used as a starting basis for the simplex algorithm. The triangularity of this basis matrix ensures that a factored inverse representation of the basis with a minimum number of non-zeros can be trivially created. Two types of triangular bases can be extracted out of the LP constraint matrix. The first one is the lower triangular basis in which the non-zeros are located in and below the main diagonal. The other one is the upper triangular basis in which non-zeros are located in and above the main diagonal. If a crash procedure does not take into account the actual numerical values of coefficients but it considers the location of non-zeros, then it is called a *symbolic crash*. If the numerical values are used, then this is a *numerical crash* procedure.

The triangular crash procedure was first introduced by Carstens in [26]. He defined the improvement of the objective value as GAIN and introduced some heuristics for choosing the possible pivots which lead to improvement in GAIN. It is called 'GAIN switch on'. The vector selected for entering the basis is processed serially and the one which has a reduced cost coefficient which will cause a gain in the functional value and in feasibility is taken. He also introduced another strategy called 'GAIN switch off' which ignores the objective function, but considers sparsity of the coefficient matrix alone. Simple three-level strategies are defined using CC_j and RC_i for the number of the non-zeros for column j and row i of the coefficient matrix A. These are set out as follows.

- Consider the nonbasic columns in order of increasing sequence of CC_j and choose column j with the smallest number of non-zero coefficients and the pivot $a_{ij} \neq 0$, in column j and row i for which RC_i is a minimum.
- Consider the rows in order of increasing sequence of RC_i and choose the one with the minimum RC_i and pivot $a_{ij} \neq 0$ for column j with the minimum CC_j.
- Consider non-zeros $a_{ij} \neq 0$ in their increasing order of count $(RC_i - 1) \times (CC_j - 1)$

and select the coefficient which is the smallest.

Recently, Bixby described a procedure for obtaining an initial basis in [12], and called it the CPLEX basis. This procedure does not take into account the traditional sparsity based approach. It can be interpreted as a variation of Carstens' gain switch on approach using the objective function coefficients. On the other hand, he considered a more general problem with bounded variables and described his algorithm in terms of these. Maros and Mitra introduced some crash heuristics which have been used in the solver FortMP system [37]. One of them is a Lower Triangular Symbolic Crash designed for Feasibility (CLTSF). This crash procedure is summarised in the next section since our procedure makes use of a part of its logic. The second one is an anti-degeneracy strategy called CRASH(ADG). This crash procedure is used when the logical basis is degenerate but not completely. These two crash procedures are based on the triangularity feature of the matrix. They also proposed another crash procedure which aims to remove the artificial variables as many as possible by violating the triangularity structure of the CLTSF. This is a numerical crash procedure which is called CRASH(ART). These three crash procedures are explained in more detail in [83], [85]. The experimental results are presented for some Netlib and industrial models and the computational performance of these crash procedures is compared with the CPLEX starting basis.

Block Triangular Crash

The block triangular crash procedure was first introduced by Gould and Reid in [63]. This algorithm is a numerical crash procedure which tries to find a basis close to feasibility. In this procedure, first the coefficient matrix A is permuted into a lower block triangular form. A series of small dense LP problems are then solved based on these diagonal matrices. A basis is then assembled from their solutions. This procedure is computationally more demanding than the triangular crash procedures. However, Gould and Reid suggested that the block triangular crash procedure can lead to computational improvements in the simplex method. They also found that their techniques are better than the 'GAIN switch off' algorithm.

7.2.1 The Crash CLTSF Procedure

Conceptually, CLTSF sets out to replace the variables of the all-logical basis by structural variables. The triangular structure of the basis is obtained using the logical operations only. The simplest way to trace the possible triangular structure is to introduce row and column counts, RC_i and CC_j which are defined as the number of non-zeros in row i and column j of the matrix A. Since elements above the diagonal consist of only zero elements in a lower triangular basis, the first diagonal element can be found easily as pivot row i such that $RC_i = \min\{RC_k\}$. If $RC_i = 1$, then the pivot column is automatically found. Otherwise, the pivot column is the one with a non-zero element in this row having the smallest column count.

The selected element becomes the current pivot and the pivot position is logically permuted (by row and column permutations) to the top left corner. The row i and all remaining columns having non-zero elements in row i, if there are any, are marked unavailable for further consideration. In this way, it is ensured that the remaining columns do not have to be transformed at later stages. The row and column counts are recomputed for the remaining active rows and columns. The same procedure is repeated for finding other pivot positions. If there is not a unique minimum row count and column count for the pivot positions at each stage, then there are multiple choices (ties) that have to be broken. In this procedure, ties for column selection are broken by giving the preference to removal of a logical variable with small feasibility range and inclusion of a structural variable with large feasibility range. The feasibility ranges are ranked in accordance with the type of variables $0, 1, 2, 3$ which are defined as fixed variable, bounded variable, non-negative variable and free variable, respectively.

The definition of row priority (RP) and column priority (CP) are given in Table 7.2.1. In this table, row type and column type are denoted by RT and CT, respectively. Row type refers to the type of the logical variable of a row, and similarly, column type refers to the type of the corresponding structural variable. The interpretation of priorities determines the individual exchanges and the final block pivot. It is therefore easily seen that the most favourable combination is to replace a type 0 logical by a type 3 structural.

The pivot row and column selections are based on row and column priority functions. The Row Priority Function, $RPF(i)$ is defined as

$$RPF(i) = RP(RT(i)) - 10 \times RC(i) \tag{7.1}$$

where $RT(i)$ is the type of the logical variable of row i and $RC(i)$ is the number of non-zeros in row i of the active columns of A. The pivot row selection is made by finding a row r with the maximum value of this function.

ROW PRIORITY			COLUMN PRIORITY		
RT	RP	Comments	CT	CP	Comments
0	3	Equality row (HP)	0	0	Fixed variable (LP)
1	2	Range type row	1	1	Bounded variable
2	1	" $\leq$ " or " $\geq$ " type row	2	2	Non-negative variable
3	0	Free row (LP)	3	3	Free variable (HP)

HP: Highest Priority, **LP**: Lowest Priority.

Table 7.2.1: Row and column priorities.

Similarly, the Column Priority Function $CPF(j)$ is defined as

$$CPF(j) = CP(CT(j)) - 10 \times CC(j) \tag{7.2}$$

where $CT(j)$ is the type of column j and $CC(j)$ is the number of non-zeros in column j of the active rows of A. Having selected row r, this row is traced for non-zero entries and if such an entry is found, the function $CPF(j)$ is evaluated for that column j. Finally, the column is determined by the maximum value of $CPF(j)$. The procedure CLTSF is fully described in [83].

7.3 Constructing An Advanced Basis with Lagrangean Relaxation

In this section, we consider an LPEN problem and describe our network based crash procedure for creating an advanced basis. For a given set of trial values of Lagrangean multipliers, the Lagrangean relaxation problem is itself a minimum cost network flow problem. Therefore, the basis corresponding to an optimum solution to this problem has a triangular form because of the natural structure of network flow problems. This leads us to use a lower triangular crash procedure by considering only variables which are basic in the network optimum solution. As there are generally less basic variables in the optimum solution of network problems than the basis size of the original problem, the logical variables associated with side constraints are introduced to gain the full row rank and to construct a non-singular basis for the original problem.

We construct two network based crash procedures which use the main concept of the CLTSF approach. For the first crash procedure, our choice is restricted to only network variables and network rows. The starting basis is initialised as the one which consists of the basic variables of the network problem and the logical variables of the non-network rows. In other words, all non-network columns and network columns which are not in the optimal basis of the network problem are excluded. We call this crash procedure CNET1.

In the second method we take into account the remaining rows, and apply the CLTSF procedure to the non-network rows that are not processed in the CNET1 procedure. In that case, the initial basis is constructed as in CNET1, and then the CLTSF procedure is applied. Therefore, as many logical variables corresponding to non-network rows as possible are replaced by structural variables. We call this procedure CNET2. Our computational experiments show that, as we expect from theory, the CNET1 and CNET2 procedures in all cases produce a lower triangular basis structure. The main steps of the crash CNET2 procedure are set out below; for more detail the reader is referred to [60].

A Network Based Crash Algorithm: CNET2

Step 1 Initialise the starting basis for the original problem. It contains all basic variables of the optimal solution of the network problem and the logical variables of the non-network rows.

Step 2 Define the set of active rows as all non-network rows and the set of active columns as all nonbasic columns of the optimal network solution. Exclude the free rows and fixed columns.

Step 3 Calculate the row and column counts of non-zero entries for active rows and columns of the coefficient matrix.

Step 4 Make the row selection on the basis of minimum row count. If there is a tie, then break it as in [83]. If the row selection is successful, then select the pivot column based on minimum column count. In case of a tie, break it by [83]. If there is no row to select, then terminate the algorithm with the current basis that may contain some logical variables of the non-network rows.

Step 5 Update the basis by exchanging the basic column corresponding to the pivot row with the selected pivot column.

Step 6 Delete the selected row and column from the active sets and also delete any other active columns intersecting with the selected row.

Step 7 Update the row and column counts and go to step 4 to select the next pivot row.

We present the computational results obtained as the best out of the CNET1 and CNET2 procedures and compare them with the original CLTSF crash procedure in section 7.5.

7.4 Constructing An Advanced Basis with Benders Decomposition

Having applied Benders decomposition with a preset number of passes, we use the combined solution of the master (the embedded network flow problem) and the subproblem to compute a good (near optimal and near feasible) solution for the given LP problem. Let $x^k = ((x')^k, (x'')^k)$ denote the solution vector of the master and the subproblem in the kth pass. This solution may be

- an infeasible,
- a feasible or
- a feasible as well as optimal

solution of the LPEN problem. We create a starting basis for the original problem in the following way. If the variable ${x_i}^k$ for the ith component appears as a basic variable in the solution of the master or the subproblem, then we mark its status as basic. If not, it means that the variable is non-basic, then we analyse the solution values;

Lower bound: If the solution value ${x_i}^k$ for the ith component is at its lower bound, that is, ${x_i}^k = l_i$; normally $l_i = 0$, then we set its status to non-basic at lower bound.

Upper bound: If the solution value ${x_i}^k$ for the ith component is at its upper bound, that is, ${x_i}^k = u_i$, then we set its status to non-basic at upper bound.

The basis factorisation procedure INVERT uses this information to create an initial factorisation of this basis as an SSX starting point for solving the LPEN problem.

7.5 Computational Results

The algorithm described in section 5.5 and the alternative advanced bases are implemented in FORTRAN and FortMP [37] is used as a callable subroutine. FortMP which is an industrial strength mathematical programming system used for both algorithmic research and in collaborative projects with industry has been developed by the mathematical programming research group at Brunel University. The computational experiments have been carried out on a DEC ALPHA 3000/600 computer with 96MB memory. We use CPU time and simplex iterations as alternative performance measures of our procedures. In the network exploitation procedures, the total solution time is calculated by including the time spent in

1. the network extraction including scaling,
2. the Lagrangean multiplier adjustment procedure (solving a series of network flow problems) or Benders decomposition procedure (solving a series of the master and subproblems),
3. an advanced basis construction, and
4. solving the entire LP problem by FortMP.

However, the time to input the original LP problem and preprocessing which are common in all runs is excluded. We present our results for each decomposition method separately and discuss consolidated results.

7.5.1 Results with Lagrangean Relaxation

The results obtained with the Lagrangean relaxation method are set out in Table 7.5.2. We first apply the multiplier adjustment method to the Lagrangean relaxation of the LPEN problem and display results under the heading *LR: Adjusted Multiplier.* We then consider an alternative relaxed problem in which all non-network side constraints are aggregated. We apply the same multiplier adjustment method to solve the corresponding network flow problem with at most three side constraints and the results are displayed under the heading *LR: Row Aggregation.* We also consider the zero-multiplier method (see section 6.2.4) which is a special case of the multiplier adjustment procedure where all multipliers are fixed to zero and display the results in the column *LR: Zero Multiplier.* The asterisk denotes the best time obtained out of different procedures.

For these methods, the advanced bases are constructed by applying two network based crash procedures CNET1, CNET2 discussed in section 7.3. The given problems are then solved using these advanced bases by applying primal, primal-dual and dual SSX as a finishing strategy. The results displayed in Table 7.5.2 are chosen as the best out of all alternative advanced bases and different finishing strategies. The maximum number of passes to solve the Lagrangean relaxation problem is limited to 50 in both cases of applying the multiplier adjustment procedure. We observe that for some LP models,

PROCEDURE NAMES	LR: ZERO MULTIPLIER		LR: ADJUSTED MULTIPLIER		LR: ROW AGGREGATION	
MODEL NAMES	TIME	ITER	TIME	ITER	TIME	ITER
25fv47	*19.68	3164	19.75	3164	19.79	3164
bnl2	55.29	5429	*26.33	3265	51.81	4394
cre-a	*22.92	2810	32.11	3727	23.78	2772
cre-c	16.77	2381	*16.73	2381	17.52	2381
cycle	7.90	1335	7.91	1335	*7.56	1205
czprob	2.89	820	*2.57	759	2.72	820
d2q06c	318.76	20309	*304.59	19138	318.55	20309
d6cube	*20.00	1388	122.99	10383	23.88	1672
energy	*58.50	9112	59.43	9296	58.59	9112
ganges	*3.60	1055	3.89	1110	*3.60	1055
greenbea	*54.37	5084	58.10	5548	55.29	5084
greenbeb	60.06	5915	61.60	5372	*58.66	5915
ken7	6.06	1313	6.19	1079	*5.97	1287
pilot	*331.28	10370	336.89	19138	336.68	10370
scfxm3	3.61	890	*3.55	1086	3.59	1086
sctap2	0.93	255	*0.91	255	1.08	255
sctap3	*1.78	372	1.79	372	2.11	372
scrs8	*0.92	489	1.52	556	*0.92	489
ship12l	2.35	742	*2.14	710	2.36	742
sierra	*2.44	613	2.47	598	2.47	613
stocfor2	9.63	1541	*7.41	896	9.56	1519
woodw	4.36	780	*4.32	780	4.45	780

TIME: CPU Time in seconds, **ITER**: The number of SSX iterations to solve the LP with an advanced basis.

Table 7.5.2: Results with the Lagrangean relaxation method.

the solution obtained using the basis constructed after optimising only the network problem without making any multiplier adjustment (zero multiplier case) decreases the number of iterations as well as the CPU time. However, for other cases, for instance, models `bnl2`, `d2q06c`, and `greenbeb`, `ken7`, LR: Adjusted Multiplier and LR: Row Aggregation methods lead to better performance.

In Table 7.5.2, we also observe that an LP with an advanced starting point constructed from the solution of Lagrangean relaxation of the aggregated embedded network problem is solved to optimality with fewer iterations and less time than the one from the solution of Lagrangean relaxation of the original embedded network problem in most models; for example, see models `d6cube` and `cre-a`. Since the multiplier adjustment procedure is carried out unless the side constraints are satisfied or the maximum iteration number is reached, the total solution time is increased in the Lagrangean relaxation procedure.

It is worthwhile to mention that even though increasing the number of multiplier adjustments might give a better bound, it is still time consuming. In contrast, the aggregated side constraint can be satisfied without reaching the preset limit on the number of passes. In some cases, however, the aggregated side constraint also requires considerable computational time to satisfy it.

7.5.2 Results with Benders Decomposition

In Table 7.5.3, we present the results of applying Benders decomposition to a set of large scale Netlib test models. Since at each pass a master and a subproblem are solved and repeated passes can be time consuming for large models, we have preset the maximum pass number to one. Therefore, only one cut is introduced into the master problem.

In Table 7.5.3, we break up the results for total solution time and total iteration number for the master and subproblems as well as the SSX finishing strategy. In the last column in Table 7.5.3, the total solution time including time taken to solve the master

PROCEDURE NAMES	MASTER PROBLEM		SUB PROBLEM		SSX BENDERS BASIS		TOTAL SOLUTION
MODEL NAMES	TIME	ITER	TIME	ITER	TIME	ITER	TIME
25fv47	0.12	497	3.25	606	18.33	2756	21.89
bnl2	0.60	301	1.04	296	40.95	3766	43.76
cre-a	0.27	201	0.31	40	24.94	3108	26.48
cre-c	0.33	115	0.48	107	19.07	2701	20.67
cycle	0.07	23	0.88	32	9.72	1035	11.08
czprob	0.52	329	0.02	121	2.49	810	3.44
d2q06c	0.89	466	20.50	3684	583.27	33082	606.20
d6cube	0.03	38	109.86	11607	210.07	28208	320.30
energy	7.23	1999	1.60	576	51.59	7267	62.80
ganges	0.88	479	0.04	344	2.07	735	3.36
greenbea	0.32	269	0.60	94	64.41	6239	66.13
greenbeb	0.15	101	0.30	48	56.67	5156	57.91
ken7	3.13	1415	0.01	0	4.15	832	7.83
pilot	0.16	30	21.47	1804	224.37	6322	246.78
scfxm3	0.10	78	0.40	163	3.30	823	3.98
sctap2	0.43	46	0.13	865	0.37	165	1.22
sctap3	0.69	48	0.19	0	0.70	234	2.07
scrs8	0.03	77	0.20	98	0.65	302	0.93
ship12l	0.51	299	0.06	13	2.41	764	3.60
sierra	0.83	604	0.02	8	2.53	960	4.02
stocfor2	1.12	683	0.08	5	14.88	2033	16.79
woodw	0.05	25	4.58	913	8.77	1335	13.63

TIME: CPU Time in seconds, **ITER**: The number of SSX iterations to solve the master, the subproblem and the original LP problem.

Table 7.5.3: Results with Benders decomposition with one cut.

and subproblems, to create an advanced basis and to solve the original LP problem is presented. Considering the results shown in Table 7.5.3, we find that the time spent in solving two master problems does not take the major proportion of the total solution time. However, for model energy, even though the network simplex solver can be used for the first pass of the master problem (the pure network flow problem), solving the master problem with one cut is relatively time consuming.

PROCEDURE NAMES	MASTER PROBLEM		SUB PROBLEM		SSX BENDERS BASIS		TOTAL SOLUTION
MODEL NAMES	TIME	ITER	TIME	ITER	TIME	ITER	TIME
25fv47	0.05	61	1.60	436	17.25	2687	19.09
czprob	0.50	329	0.01	0	2.48	758	3.26
d2q06c	0.51	195	10.00	1391	334.35	20104	346.36
d6cube	0.16	2	109.32	10533	75.99	6020	185.63
ganges	0.05	475	0.02	4	1.89	656	2.72
greenbea	0.13	139	0.30	49	60.86	5733	62.02
pilot	0.02	25	22.68	1084	219.73	6159	243.37
scfxm3	0.07	71	0.31	144	3.20	828	3.76
ship12l	0.48	297	0.04	13	2.00	681	2.89
sierra	0.83	604	0.01	0	2.58	1007	3.97
stocfor2	1.07	682	0.04	3	12.74	1969	14.55

TIME: CPU Time in seconds, **ITER**: The number of SSX iterations to solve the master, the subproblem and the original LP problem.

Table 7.5.4: Results with Benders decomposition with no cut

Time to solve subproblems depends on the size of network structures detected; if the proportion of the network structure is not large, it means that the subproblem is relatively large and cannot be solved fast. The results for models `pilot`, `d6cube` and `woodw` justify this observation. Since the subproblem is the same as the previous one with only different right had side values, the previous basis of the subproblem for a warm start is used. We also observe that an advanced starting basis can always be constructed from the solution of the entire master (pure network flow problem) and subproblem like in the zero multiplier method. Table 7.5.4 displays the results of some example cases where the total solution time reduces by solving the LP with this basis. These results show that for some models such as d2q06c and d6cube the total solution time is decreased by about fifty percent compared to the solution time obtained by Benders decomposition of the LPEN with one cut.

7.5.3 Consolidated Results

We compare our network based crash procedures with the unit basis and CLTSF procedure which has one of the best performances currently reported in the literature, see [83, 85]. We therefore set out the consolidated results in Table 7.5.5 and display the time and iteration number to solve the LP with the advanced basis chosen as the best performance of the Lagrangean relaxation and Benders decomposition, the crash CLTSF and unit basis. The asterisk in this table refers to the best solution time out of all procedures.

In Table 7.5.5, the results displayed in column two under the heading Lagrangean relaxation are obtained as the best out of the two crash procedures CNET1 and CNET2 which can be claimed to be good crash procedures. However, we cannot make any general conclusion as to which of the network based crash procedures fully dominates all others. The performance of the network based crash procedures appear to be problem specific. We observe that the best of the Lagrangean relaxation and Benders decomposition performs marginally better than the crash CLTSF procedure. In general, all the crash procedures perform better than the unit starting basis procedure.

Considering these computational results, we make the following broad observations.

1. Exploiting embedded pure network structures within large scale LP problems improves the total solution time and number of iterations in most of the cases compared with the SSX method with the advanced basis and the unit basis.

PROCEDURE NAMES	LAGRANGEAN RELAXATION		BENDERS DECOMPOSITION		CRASH CLTSF		UNIT BASIS	
MODEL NAMES	TIME	ITER	TIME	ITER	TIME	ITER	TIME	ITER
25fv47	19.68	3164	*19.09	2687	24.26	3514	26.88	4327
bnl2	*26.33	3265	43.76	3766	49.71	4098	61.43	6178
cre-a	*22.92	2810	26.48	3108	36.34	3626	30.47	4154
cre-c	*16.73	2381	20.67	2701	24.38	2892	28.18	4205
cycle	*7.56	1205	11.08	1035	11.01	1293	8.49	1516
czprob	*2.57	759	3.26	758	3.46	1112	4.14	1436
d2q06c	*304.59	19138	346.36	20104	352.40	21644	332.53	21515
d6cube	*20.00	1388	185.63	6020	115.01	9828	125.05	12469
energy	58.50	9112	62.80	7267	*52.36	8288	57.17	9730
ganges	3.60	1055	2.72	656	*2.65	555	4.46	1443
greenbea	*54.37	5084	62.02	5733	61.38	5279	67.42	6805
greenbeb	58.66	5915	*57.91	5156	61.07	5213	65.86	6341
ken7	5.97	1287	7.83	832	*5.63	1506	13.70	2967
pilot	331.28	10370	*243.37	6159	306.44	9172	345.77	10716
scfxm3	*3.55	1086	3.76	828	3.56	882	3.40	1170
sctap2	*0.91	255	1.22	165	1.36	612	2.02	931
sctap3	*1.78	372	2.07	234	2.11	761	4.04	1408
scrs8	*0.92	489	0.93	302	1.45	549	1.80	880
ship12l	*2.14	710	2.89	681	2.28	742	3.61	1070
sierra	*2.44	613	3.97	1007	5.27	1819	3.36	1310
stocfor2	*7.41	896	14.55	1969	10.35	1185	12.36	1939
woodw	*4.32	780	13.63	1335	4.71	903	6.65	1480
RP	16B,18W	–	3B,12W	–	3B, -	–	0B,5W	–
Total Time	956.23	–	1136.00	–	1137.19	–	1208.79	–

RP: Relative performance, **B**: Best out of all methods, **W**: Winner against CLTSF crash procedure. **TIME**: CPU Time in seconds, **ITER**: The number of SSX iterations to solve the LP problem with different advanced bases and the unit basis.

Table 7.5.5: The consolidated results.

2. The SSX solution time and iteration number are decreased by solving the LP problem with the advanced basis obtained by our start up procedure which uses Benders decomposition in a restricted form.

3. The crash CLTSF procedure performs better than the all-logical basis. If we apply the best network based crash procedure chosen out of CNET1 and CNET2, then this result dominates the performance of CLTSF in most cases.

7.6 Discussion

In this chapter, we have shown how the EPN structure within large scale LP problems can be used to create an advanced starting point to solve the original LP problem by the SSX method. The zero multiplier approach has been used to solve a pure network flow problem which is constructed by relaxing all of the side constraints. By considering the non-network side constraints and applying the Lagrangean multipliers to these constraints, we have again constructed a network problem. To solve this network problem with side constraints, a series of minimum cost flow problems which differ only in the objective coefficients have been solved iteratively with a new set of fixed multipliers at each iteration.

As an alternative, the side constraints have been aggregated and the reduced embedded network problem with at most three side constraints has also been treated using the Lagrangean multipliers approach. The near optimal and near feasible solutions obtained by these methods have been used to construct advanced bases and passed to the general SSX solver to process the original LP problem to optimality. As an alternative way of constructing an advanced basis, we have applied Benders decomposition to the LPEN problem. The computational results showed that even for general classes of LP problems this is an effective procedure for creating an advanced basis.

Chapter 8

Summary Conclusions and Future Directions

In this chapter, we discuss the issues raised in the investigation reported in this thesis and present our summary conclusions. In particular, we highlight the novel concepts and the resulting contributions. We also put forward suggestions for further work.

8.1 Summary of Contributions

The investigation of the embedded network structure within LP problems has led us to develop automatic network detection algorithms and a solution procedure which takes advantage of the EPN structure.

- **M-GUB Algorithm**

  We have presented a new GUB based algorithm for detecting an EPN structure within an LPEN problem. We have applied the Markowitz merit count concept in a novel way to improve the GUB detection heuristic, and consequently the maximum number of the network rows within the scope of this heuristic. We have then exploited the relationship between the GUB structures in LP problems and the independent sets in the corresponding graphs. In this procedure, the GUB subsets have been detected by an independent set algorithm. In particular, we have adopted a greedy algorithm for detecting independent sets since the greedy algorithm outperformed the matching algorithm in our computational experiments. The number of network rows and columns detected by different algorithms as well as the computing time taken by the detection algorithm have been used as measures for the comparison. We have taken the row scanning deletion algorithm as a benchmark heuristic. An analysis of the computational results has showed that our approach of extending the two-stage heuristic to the multi-stage heuristic leads to improved performance and our procedures perform favourably when compared with the row scanning deletion algorithm. Taking into consideration the computing time we have observed that the structure detection time is usually less than 5% of the sparse simplex solution time.

- **GSG Algorithm**

We have presented the second EPN structure extraction algorithm which is based on generalised signed graphs. We have shown that this algorithm performs very well compared to other algorithms. Using generalised signed graphs, we have proved that the problem of detecting the maximum EPN structure in an LP problem is NP-hard; even for very special families of matrices. The main contribution of the GSG algorithm is that it determines whether the given LP coefficient matrix is entirely a pure network. This result is in contrast to other known algorithms in the literature.

- **Solution Algorithm**

In order to exploit the EPN structure, we have introduced an advanced basis algorithm which improves the computational time required to solve the LPEN models. The main advantage of this algorithm is that it uses a general network solver and an LP solver. The original coefficient matrix is partitioned into the network and the non-network parts. For this partitioning, we have investigated two alternative decompositions namely, Lagrangean and Benders. In the Lagrangean approach, the optimal solution of a network flow problem and in Benders, the combined solution of the master and the subproblem have been used to compute good (near optimal and near feasible) solutions for the given LP problem.

In both cases, we have terminated the decomposition algorithms after a preset number of passes. The near feasible and near optimal solutions obtained by these methods have been used to construct advanced bases and passed to the general SSX solver to process the original LP problem to optimality. We have presented comparisons with the unit basis and a well established crash procedure. We have found that the computational results of applying these techniques to a selection of Netlib models are promising enough to encourage further research in this area.

8.2 Suggestions for Further Work

Based on our experience of computational algorithms investigated in this thesis, we suggest the following further research.

- The investigation carried out in this study for detecting EPN structures can be naturally extended to extract embedded GNET structures. In this way, the embedded GNET structure can be exploited. Like pure network solvers the methods for solving generalised networks are becoming competitive when compared to simplex based LP solvers [91].

- We have observed that the number of network rows detected depends (sometimes, significantly) on the spanning tree computed in the GSG algorithm. To enhance the results, several spanning trees rather than just one can be built in which case parallel algorithms may be considered. In addition, the independent set S can be enlarged by using local search improvement algorithms.

- In the application of Benders decomposition to the LPEN problem, even though the first master problem can be solved by the network solver, at the other iterations the master problem becomes a network flow problem with cuts. At this stage, the Lagrangean relaxation procedure can be applied to the master problem by dualising the cuts whereby the master problem can be processed by a network solver.

- Instead of using the SSX procedures, the interior point method can be used to exploit the EPN structure in LP problems. Todd [104] suggested computational schemes which take advantage of specially structured constraints, especially various upper bounding constraints, in a variant of Karmarkar's projective algorithm for LPs. These constraints are used to generate improved bounds on the optimal value of the problem and also to compute the necessary projections more efficiently.

References

[1] Ahuja, R.K., Magnanti, T.L. and Orlin, J.B., *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, (1993).

[2] Ali, A.I. and Thiagarajan, H., "A Network Relaxation Based Enumeration Algorithm for Set Partitioning", *European Journal of Operational Research*, Vol. 38, (1989), p. 76–85.

[3] Baker, B.M. and Maye, P.J., "A Heuristic for Finding Embedded Network Structure in Mathematical Programmes", *European Journal of Operational Research*, Vol. 67, (1993), p. 52–63.

[4] Barr, R.S., Farhangian, K. and Kennington, J.L, "Networks with Side Constraints: An LU Factorization Update", *The Annals of the Society of Logistics Engineers*, Vol. 1, No. 1, (1986), p. 66–85.

[5] Barr, R.S., Glover, F. and Klingman, D., "Enhancements of Spanning Tree Labelling Procedures for Network Optimization", *Infor*, Vol. 17, (1979), p. 16–34.

[6] Bartholdi, J.J., "A Good Submatrix is Hard to Find", *Operations Research Letters*, Vol. 1, (1982), p. 190–193.

[7] Baston, V.J.D., Rahmouni, M.K. and Williams, H.P., "The Practical Conversion of Linear Programmes to Network Flow Models", *European Journal of Operational Research*, Vol. 50, (1991), p. 325–334.

[8] Bazaraa, M.S., Jarvis, J.J. and Sherali, H.D., *Linear Programming and Network Flows*, Second Edition, John Wiley and Sons, (1990).

[9] Beasley, J.E., "Lagrangean Relaxation", in *Modern Heuristic Techniques for Combinatorial Problems*, Ed. Reeves, C.R., Blackwell Scientific Publications, Oxford. (1993).

[10] Belling-Seib, K., Mevert, P. and Muller, C., "Network Flow Problems with One Side Constraint: A Comparison of Three Solution Methods", *Computers and Operational Research*, Vol. 15, No. 4, (1988), p. 381–394.

[11] Benders, J.F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems", *Numerische Mathematik*, 4, (1962), p. 238–252.

[12] Bixby, R.E., "Implementing the Simplex Method: The Initial Basis", *ORSA Journal on Computing*, Vol. 4, No. 3, (1992), p. 267–284

[13] Bixby, R.E. and Fourer, R., "Finding Embedded Network Rows in Linear Programs I. Extraction Heuristics", *Management Science*, Vol. 34, No. 3, (1988), p. 342–376.

[14] Bixby, R.E. and Fourer, R., "Finding Embedded Network Rows in Linear Programs II. Augmentation Heuristics", unpublished, (1995).

[15] Bixby, R.E., "Hidden Structure in Linear Programs", in *Computer-Assisted Analysis and Model Simplification*, Ed. Greenberg, H., Maybee, J., Academic Press, New York, (1981), p. 327–360.

[16] Bixby, R.E. and Cunningham, W.H., "Converting Linear Programs to Network Problems", *Mathematics of Operations Research*, Vol. 5, No. 3, (1980), p. 321–356.

[17] Brearley, A.L., Mitra, G. and Williams, H.P., "Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm", *Mathematical Programming*, Vol. 8, (1975), p. 54–83.

[18] Brown, G.G. and Olson, M.P., "Dynamic Factorization in Large-Scale Optimization", *Mathematical Programming*, Vol. 64, No. 1, (1994), p. 17–51.

[19] Brown, G.G. and Olson, M.P., "Dynamic Factorisation in Large Scale Optimisation", *Technical Report, NPSOR-93-008*, (1993).

[20] Brown, G.G. and Wright, W.G., "Automatic Identification of Embedded Network Rows in Large-Scale Optimization Models", *Mathematical Programming*, Vol. 29, (1984), p. 41–56.

[21] Brown, G.G., McBride, R.D. and Wood, R.K., "Extracting Embedded Generalized Networks from Linear Programming Problems", *Mathematical Programming*, Vol. 32, (1985), p. 11–31.

[22] Brown, G.G. and Wright, "Automatic Identification of Embedded Structure in Large-Scale Optimization Models", in *Computer-Assisted Analysis and Model Simplification*, Ed. Greenberg, H., Maybee, J., Academic Press, New York, (1981), p. 369–388.

[23] Brown, G.G. and Thomen, D.S., "Automatic Identification of Generalized Upper Bounds in Large-Scale Optimization Models", *Management Science*, Vol. 26, No. 11, (1980), p. 1166–1184.

[24] Bryson, N., "Parametric Programming and Lagrangian Relaxation: The Case of the Network Problem with a Single Side Constraint", *Computers and Operations Research*, Vol. 18, No. 2, (1991), p. 129–140.

[25] Bryson, N., "A Parametric Programming and Methodology to Solve the Dual for Network Problems with Multiple Side Constraints", *Computers and Operations Research*, Vol. 28, No. 5, (1993), p. 541–552.

[26] Carstens, D.M., "Crashing Techniques", in Orchard-Hays, W., *Advanced Linear Programming Computing Techniques*, McGraw-Hill, (1968), p. 131–141.

[27] Chartrand, G., *Graphs as Mathematical Models*, Prindle, Weber and Schmidt, Boston, (1977).

[28] Chen, S. and Saigal, R., "A Primal Algorithm for Solving a Capacitated Network Flow Problem with Additional Linear Constraints", *Networks*, Vol. 7, (1977), p. 59–79.

[29] Cormen, T.H., Leiserson, C.E. and Rivest, R.L., *Introduction to Algorithms*, MIT Press, Cambridge, (1990).

[30] Daniel, R.C., "A Generalisation of Variable Upper Bounding and Generalised Upper Bounding", *European Journal of Operational Research*, Vol. 2, (1978), p. 202–206.

[31] Dantzig, G.B. and Van Slyke, R.M., "Generalized Upper Bounding Techniques", *Journal of Computer and System Sciences*, Vol. 1, (1967), p. 213–226.

[32] Dantzig, G.B., *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, (1963).

[33] Dantzig, G.B. and Orchard-Hays, W., "The Product Form of the Inverse in the Simplex Method", *Mathematical Tables and Aids to Computation*, Vol. 8, (1954), p. 64–67.

[34] Dantzig, G.B. and Wolfe, P., "Decomposition Principle for Linear Programs", *Operations Research*, Vol. 8, (1960), p. 101–111.

[35] Darby-Dowman, K. and Mitra, G., "An Investigation of Algorithms Used in Restructuring of Linear Programming Basis Matrices Prior to Inversion", in *Studies on Graphs and Discrete Programming*, Ed. Hansen, P., North Holland, (1981).

[36] Duff, S., Erisman, A.M. and Reid, J.K., *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford-London, (1986).

[37] Ellison, E.F.D., Hajian, M., Levkovitz, R., Maros, I. and Mitra, G., "A Fortran Based Mathematical Programming System: FortMP", (1995), Brunel University and NAG Ltd.

[38] Fisher, M.L., "The Lagrangean Relaxation Method for Solving Integer Programming Problems", *Management Science*, Vol. 27, No. 1, (1981), p. 1–18.

[39] Fisher, M., Jaikumar, R. and Wassenhove, L.N.V, "A Multiplier Adjustment Method for the Generalized Assignment Problem", *Management Science*, Vol. 32, No. 9, (1986), p. 1095–1103.

[40] Fisher, M.L., "An Applications Oriented Guide to Lagrangian Relaxation", *Interfaces*, Vol. 15, No. 2, (1985), p. 10–21.

[41] Ford, L.K. and Fulkerson, D.K., *Flows in Networks*, Princeton University Press, New Jersey, (1962).

[42] Fragniere, E., Gondzio, J., Sarkissian R. and Vial, J., "Structure Exploiting Tool in Algrebraic Modeling Languages", *Logilab Technical Report*, 1997.2, (1997).

[43] Gay, D.M., "Electronic Mail Distribution of Linear Programming Test Problems", *Mathematical Programming Society, Coal, Newsletter*, Vol. 13, (1985), p. 10–12.

[44] Geoffrion, A.M., "Lagrangean Relaxation for Integer Programming", *Mathematical Programming Study*, Vol. 2, (1974), p. 82–114.

[45] Geoffrion, A.M., "Elements of Large-Scale Mathematical Programming Part I: Concepts", *Management Science*, Vol. 16, No. 11, (1970), p. 652–675.

[46] Geoffrion, A.M., "Elements of Large-Scale Mathematical Programming Part II: Synthesis of Algorithms and Bibliography", *Management Science*, Vol. 16, No. 11, (1970), p. 676–691.

[47] Glover, F. and Klingman, D., "The Simplex SON Algorithm for LP/Embedded Network Problems", *Mathematical Programming Study*, Vol. 15, (1981), p. 148–176.

[48] Glover, F. and Klingman, D., "Basis Exchange Characterizations for the Simplex SON Algorithm For LP/Embedded Networks", *Mathematical Programming Study*, Vol. 24, (1985), p. 141–157.

[49] Glover, F., Karney, D., Klingman, D. and Russell, R., "Solving Singly Constrained Transshipment Problems", *Transportation Science*, Vol. 12, No. 4, (1978), p. 277–297.

[50] Graves, G.W. and McBride, R.D., "The Factorization Approach to Large-Scale Linear Programming", *Mathematical Programming*, Vol. 10, (1976), p. 91–110.

[51] Greenberg, H.J., "A Functional Description of Analyze: A Computer Assisted Analysis System for Linear Programming Models", *ACM Trans. Math. Software*, Vol. 9, No. 1, (1983).

[52] Guignard, M. and Rosenwein, M.B., "An Application Oriented Guide for Designing Lagrangean Dual Ascent Algorithms", *European Journal of Operational Research*, Vol. 43, (1989), p. 197–205.

[53] Guignard, M., "A Lagrangean Dual Ascent Algorithm for Simple Plant Location Problems" *European Journal of Operational Research*, Vol. 35, (1988), p. 193–200.

[54] Gunawardane, G., Hoff, S. and Schrage, L., "Identification of Special Structure Constraints in Linear Programs", *Mathematical Programming*, Vol. 21, (1981), p. 90–97.

[55] Gülpınar, N., Maros, I. and Mitra, G., "Detecting Embedded Pure Network Structures in Linear Programs", *Technical Report, TR/20/96*, Brunel University, (1996).

[56] Gülpınar, N., Gutin, G. and Mitra, G., "Detecting Embedded Pure Network Structure Using Independent Set Algorithms", *Technical Report, TR/12/97*, Brunel University, (1997).

[57] Gülpınar, N., Mitra, G. and Maros, I., "Computational Solution of Large Scale Programs Exploiting Embedded Network Structures", *Technical Report, TR/04/98*, Brunel University, (1998).

[58] Gülpınar, N., Maros, I. and Mitra, G., "Detecting Embedded Pure Network Structures in LP Problems", *TOP Operational Research in Practice Journal*, Vol. 6, No. 1, (1998), p. 67–95.

[59] Gülpınar, N., Gutin, G., Mitra, G. and Maros, I., "Detecting Embedded Networks in LP Using GUB Structures and Independent Set Algorithms", Accepted for publication in *Computational Optimization and Applications.*

[60] Gülpınar, N., Mitra, G. and Maros, I., "Creating Advanced Bases for Large-Scale Linear Programs Exploiting Embedded Network Structure" Submitted to *Computational Optimization and Applications.*

[61] Gutin, G., Gülpınar, N., Mitra, G. and Zverovich, A., "Extracting Pure Network Submatrices in Linear Programs Using Generalised Signed Graphs", Submitted to *Mathematical Programming.*

[62] Gutin, G., Gülpınar, N., Mitra, G. and Zverovich, A., "Extracting Pure Network Submatrices in Linear Programs Using Generalised Signed Graphs", *Technical Report, TR/14/98*, Brunel University, (1998).

[63] Gould, N.I.M. and Reid, J.K., "New Crash Procedures for Large Systems of Linear Constraints", *Mathematical Programming*, Vol. 45, (1989), p. 475–501.

[64] Hansen, P., "Labelling Algorithms for Balance in Signed Graphs." in *Problémes Combinatoires et Theorie des Graphes* (Colloq. Internat., Orsay, 1976), p. 215–217, Colloques Internat. du CNRS, 260, Paris, (1978).

[65] Harary, F., Norman, R.Z. and Cartwright, D., *Structural Models: An Introduction to the Theory of Directed Graphs*, John Wiley and Sons, (1965).

[66] Harary, F., "On the Notion of Balance of a Signed Graph", *Michigan Mathematical Journal*, Vol. 2, (1954), p. 143–146.

[67] Harary, F. and Kabell, J.A., "A Simple Algorithm to Detect Balance in Signed Graphs", *Mathematical Social Science*, Vol. 1, (1980), p. 131–136.

[68] Hartman, J.K. and Lasdon, L.S., "A Generalized Upper Bounding Algorithm for Multicommodity Network Flow Problems", *Networks*, Vol. 1, (1972), p. 333–354.

[69] Held, M. and Karp, R.M., "The Travelling Salesman Problem and Minimum Spanning Trees", *Operations Research*, Vol. 18, (1970), p. 1138–1162.

[70] Held, M. and Karp, R.M., "The Travelling Salesman Problem and Minimum Spanning Trees, Part II". *Mathematical Programming*, Vol. 1, (1971), p. 6–25.

[71] Held, M., Wolfe, P. and Crowder, P., "Validation of Subgradient Optimisation", *Mathematical Programming*, Vol. 6, (1974), p. 62–88.

[72] Hellerman, E. and Rarick, D., "Reinversion with the Preassigned Pivot Procedure", *Mathematical Programming*, Vol. 1, (1971), p. 195–216.

[73] Hsu, A.C. and Fourer, R., "Exploiting Network Structure for Solving Large Scale Linear Programming Models", Working Paper, January (1996).

[74] Hultz, J. and Klingman, D., "Solving Singularly Constrained Generalized Network Problems", *Applied Mathematics and Optimization*, Vol. 4, No. 2, (1978), p. 103–119.

[75] Infanger, G., "Planning Under Uncertainty: Solving Large Scale Stochastic Linear Programs", Boyd and Fraser, Danvers, M.A., (1994).

[76] Khanna, S., Motwani, R., Sudan, M. and Vazirani, U., "On Syntactic Versus Computational Views of Approximability", *Proceedings Symposium on Foundations of Computer Science*, (1994), p. 819–830.

[77] Kennington, J.L. and Helgason, R., *Algorithms for Network Programming*, John Wiley and Sons, New York, (1980).

[78] Klingman, D. and Russell, R., "Solving Constrained Transportation Problems", *Operations Research*, Vol. 23, No. 1, (1975), p. 91–114.

[79] Lasdon, L. S., *Optimisation Theory for Large Systems*, Collier-Macmillan Limited, London, (1970).

[80] Lucas, C., Messina, E. and Mitra, G., "Risk and Return Analysis of a Multi-period Strategic Planning Problem", in *Stochastic Modelling in Innovative Manufacturing Lecture Notes in Economics and Mathematical Systems 445*, Ed. Christer, A. and Osaki S., (1995).

[81] Markowitz, H.M., "The Elimination Form of the Inverse and its Application to Linear Programming", *Management Science*, Vol. 3, (1957), p. 255–267.

[82] Maros., I., "A Practical Anti-Degeneracy Row Selection Technique in Network Linear Programming", *Annals of Operations Research*, Vol. 47, (1993), p. 431–442.

[83] Maros, I. and Mitra, G., "Strategies for Creating Advanced Bases for Large-Scale Linear Programming Problems", *Informs Journal on Computing*, Vol. 10, No. 2, (1998), p. 248–260.

[84] Maros, I. and Mitra, G., "Simplex Algorithms", in *Advances in Linear and Integer Programming*, Ed. Beasley, J., Oxford University Press, (1996), p. 1–46.

[85] Maros, I. and Mitra, G., "Finding Better Starting Bases for Simplex Method", in *Operations Research Proceedings 1995*, Ed. Kleinschmidt, P., Springer Verlag, (1996), p. 7–12.

[86] Maros, I., "Performance Evaluation of the MINET Minimum Cost Netflow Solver", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 2, (1993), p. 199–217.

[87] Maros, I., "A Structure Exploiting Pricing Procedure in the Primal Network Simplex Algorithm", *Rutcor Research Report*, 18–91, (1991).

[88] Marsten, R.E., Hogan, W.W and Blankenship J.W., "The Boxstep Method for Large-Scale Optimization", *Operations Research*, Vol. 23, No. 3, (1975), p. 389–405.

[89] Mashc, H.V., "The Cyclic Method of Solving the Transshipment Problem with an Additional Linear Constraints", *Networks*, Vol. 10, (1980), p. 17–31.

[90] Mitra, G. and Tamiz, M., "Alternative Methods for Representing the Inverse of Linear Programming Basis Matrices", in *Recent Developments in Mathematical Programming*, Ed. Kumar, S., (1991), p. 273–301.

[91] McBride, R.D., "Solving Embedded Generalised Network Problems", *European Journal of Operational Research*, Vol. 21, (1985), p. 82–92.

[92] Paschos, V.Th., "A $\Delta/2$-Approximation Algorithm for the Maximum Independent Set Problem", *Information Processing Letters*, Vol. 44, (1992), p. 11–13.

[93] Reeves, C.L., *Modern Heuristic Techniques For Combinatorial Problems*, McGraw-Hill Book Company, (1995).

[94] Rogers, D.F., Plante, R.D., Wong, R.T. and Evans, J.R., "Aggregation and Disaggregation Techniques and Methodology in Optimisation", *Operations Research*, Vol. 39, No. 4, (1991), p. 553-582.

[95] Saab, Y., "Iterative Improvement of Vertex Covers", *Information Proceedings Letters*, Vol. 55, (1995), p. 5–98.

[96] Sharda, R., *Linear and Discrete Optimisation and Modelling Software*, Unicom Seminars Ltd, Lionheart Publishing, (1993).

[97] Schrage, L., "Some Comments on Hidden Structure in Linear Programs", in *Computer-Assisted Analysis and Model Simplification*, Ed. Greenberg, H., Maybee, J., Academic Press, New York, (1981), p. 389–395.

[98] Schrage, L., "Implicit Representation of Generalized Variable Upper Bounds in Linear Programming", *Mathematical Programming*, Vol. 14, (1978), p. 11–20.

[99] Schrage, L., "Implicit Representation of Variable Upper Bounds in Linear Programming", *Mathematical Programming*, Vol. 4, (1975), p. 118–132.

[100] Shapiro, J.F., *Mathematical Programming: Structures and Algorithms*, John Wiley and Sons, (1979).

[101] Shapiro, J.F., "A Survey of Lagrangean Techniques for Discrete Optimization", *Annals of Discrete Mathematics*, Vol. 5, (1979), p. 113–138.

[102] Shetty, B., "A Heuristic Algorithm for a Network Problem with Variable Upper Bounds", *Networks*, Vol. 20, (1990), p. 373–389.

[103] Todd, M.J., "Large-Scale Linear Programming: Geometry, Working Bases and Factorizations", *Mathematical Programming*, Vol. 26, (1983), p. 1–20.

[104] Todd, M.J., "Exploiting Special Structure in Karmarkar's Linear Programming Algorithm", *Mathematical Programming*, Vol. 41, (1988), p. 97–113.

[105] Truemper, K., "Unimodular Matrices of Flow Problems with Additional Constraints", *Networks*, Vol. 7, (1977), p. 343–358.

[106] Venkataraman, M.A., Dinkel, J.J. and Mote, J., "A Surrogate and Lagrangian Approach to Constrained Network Problems", *Annals of Operations Research*, Vol. 20, (1989), p. 283–302.

[107] Zaslavsky, T., "Signed Graphs", *Discrete Applied Mathematics*, Vol. 4, (1982), p. 47–74.

[108] Zaslavsky, T., "How Colorful the Signed Graph?", *Discrete Mathematics*, Vol. 52, (1984), p. 279–284.