School of Mathematics and Statistics

Restricted Spanning Trees and Graph Partitioning

Bee Khim Lam

This thesis is presented as part of the
requirements for the award of
the Degree of Doctor of Philosophy
of
Curtin University of Technology

November 1999

Summary

A network is a system that involves movement or flow of some commodities such as goods and services. In fact any structure that is in the form of a system of components some of which interact can be considered as a network. In network design the problem is often to construct economical and reliable networks which satisfy certain requirements and which are optimal according to some criterion such as cost, output or performance. Graph theory is useful when the requirements of the network can be expressed in terms of graph parameters, usually as bounds. Some of the graph parameters that have been considered include: degree; distance; diameter; and connectivity. Problems with these parameter restrictions are usually from a class of NP-complete problems with instances that require exponential computer time to solve by available algorithms.

The major focus of this thesis is to develop fast and efficient heuristics for some of these NP-complete problems. The two main topics analysed are Restricted Spanning Trees and Graph Partitioning. The aim of the Restricted Spanning Trees section is to construct the most efficient spanning tree (connected network) subject to various degree constraints. These degree constraints imposed are usually in the form of an upper bound. The upper bound represents the maximum number of connections allowed on a particular vertex. The Graph Partitioning section considers the problem of clustering vertices of the graph into sets such that the overall cost of the edges in the different sets is minimised.

Chapter 1 provides the notation and terminology used throughout the thesis and a review and summary of the thesis.

A literature review of related work that has been carried out to date is presented in Chapter 2. Some of the more promising results are discussed. The first part of the chapter surveys work related to the Restricted Spanning Tree

problem. Analysis of both exact and heuristic methods is given. The second part of Chapter 2 provides a survey of the Graph Partitioning problem. We discuss the many different approaches that have been proposed to solve this problem. The quality of computational results achieved is discussed.

Chapter 3 considers the Degree Constraint Minimum Weight Spanning Tree problem. This problem arises in networks where a given terminal is only allowed connections to a maximum number of specified terminals. We consider a number of cases including: same degree constraint on each vertex; different degree constraint on some vertices; and when the degree constraint is only on one or two vertices. A number of heuristics are developed and implemented and compared against an exact Branch and Cut algorithm. Our computational results demonstrated the value of our better performing heuristics.

Chapter 4 considers the complexity of the $(1,k)$-tree problem. This problem is defined as given a graph G with maximum degree k find a spanning tree T with all vertices having degree 1 or k. Analysis is done on graphs with maximum degree 3, 4 and 5. Results establishing that the $(1,3)$-tree and $(1,4)$-tree problems are NP-complete are presented. Further consideration is also given to the complexity of spanning trees with degree from the set $\{1,3,5\}$. Analysis is also carried out on the number of degree one vertices in the $(1,k)$-tree. Presentation of heuristic procedures to solve this NP-complete problem concludes the chapter.

Chapter 5 is devoted to the Graph Partitioning problem. A number of heuristics are presented and extensive computational work carried out. Computational findings support the usefulness of the heuristic methods both in terms of quality and time.

We conclude this thesis by detailing some future work that can be carried out.

Certification

I certify that this thesis is my own work and that all references are duly acknowledged. This thesis has not been submitted previously, in whole or in part, in respect of any academic award at Curtin University of Technology or elsewhere.

Bee Khim Lam

November 1999

Acknowledgements

With all my heart I thank my Ph.D. supervisor, Professor Lou Caccetta, for all the help and guidance he has given to me in creating this thesis.

Special thanks also to Dr. Jamie Simpson for the occasional suggestions and ideas. My heartfelt thanks also to Dr. Peg-Foo Siew, the Postgraduate Coordinator for his continuous support during my time at the school.

My endless gratitude goes out to Curtin University of Technology for awarding me the COSS scholarship for two years of my Ph.D. studies. Thanks also to the School of Mathematics and Statistics for all the funding I received during this project without which this project would not have been completed.

The love and support of my other half, Luan Lam, made me stuck to this project until the end. His great humour, patience and love kept the faith when mine faltered. He magnanimously stood by me during the whole project with such great understanding, care and wisdom. I only pray that he can read the love and gratitude in my heart between every line.

A special word of thanks and appreciation to my Mum and Dad for the love and patience throughout the years.

In closing, I like to thank all the people who have in one way or another contributed to this thesis. It is to these special people that I offer my sincerest thanks.

Contents

Chapter 1

Introduction

A network is a system which involves movement or flow of some commodities such as information, mail, goods, products and passengers. The commodity usually originates from a source and moves towards a destination using the connections available in the network. The connections that link the sources and destinations are mostly routes through which the commodity flows. Therefore a network is considered as any structure in the form of a system of lines and a system of components having a common purpose which is to move some commodities. For example, a communication network is a collection of centres (telephone exchanges, computer terminals, etc.) some of which communicate directly via some satellite links to exchange information; a printed circuit board is a collection of terminals connected by conductor paths; and a transportation network is a collection of depots and outlets which are linked up by roads or railways to enable goods to be transported from one depot to another.

Networks arise in many areas of modern society. These networks are large and complex often involving thousands of connections. There are enormous costs involved in designing, constructing and maintaining such networks. Therefore, the objective is to come up with efficient networks that are economical, both in terms of cost and efficiency, to operate and maintain. Network analysis based on graph theoretic concepts has proven useful (Caccetta 1989, Caccetta

and Vijayan 1987) in studying many network problems including: the design of minimal cost networks; the design of integrated circuits; the design of data communication networks capable of supporting large scale on-line applications; the determination of optimal routes in a network; and many more (Ahuja et al. 1995).

A graph consists of a set of points or vertices some pairs of which are joined by lines, or edges. This simple structure can be used to model complex network systems. Usually in the graph model the vertices represent components and the edges represent the interconnections or relationship between the components. This gives us a structural model of the network being studied. Many factors influence the design of the network such as: message delay time; throughput; message traffic density; reliability; and equipment capacity. These factors can be incorporated into the graph theoretic model by assigning weights to the vertices and edges of the graph representing the network; the result is a weighted graph.

In many network applications, the problem that arises is to construct a network which satisfies certain requirements and which is optimal according to some criterion such as cost, output or performance. Graph theory is useful when the requirements of the network can be expressed in terms of graph parameters, usually as bounds. A fundamental problem is when the objective is to construct a minimum cost network. The many graph parameters that have been considered in network design include: degree, distance, diameter and connectivity. For example, in a telecommunication network a degree restriction on the vertex (centre) could represent the number of links (line interfaces) allowed at the centre; in a transportation network a distance restriction on the goods (flow commodities) could represent the maximum distance allowed for delivery; and in a radio network a diameter restriction on the station (centre)

could represent the maximum number of links allowed for communication between the terminals.

In the application of graph theory to many real-life problems that arise in network analysis, consideration is normally given to one or more of the graph parameters. A number of graph parameters have been studied including: diameter (Achuthan et al. 1994, Caccetta 1989), distance (Caccetta 1989), degree (Gabow 1978, Gabow et al. 1986, Garey and Johnson 1979, Gavish 1982, Glover and Klingman 1974, Malyshko 1985, Narula and Ho 1980, Savelsbergh and Volgenant 1985, Volgenant 1989), and connectivity (Caccetta 1989).

In this thesis we consider problems which can be formulated in terms of edge weighted graphs (representing costs). Our aim is to find a minimum cost network satisfying one or more of the specified graph parameter restrictions. We are particularly interested in heuristic methods for obtaining near optimal solutions of trees satisfying prescribed degree restrictions. Focus is on how well these methods perform and on ways of improving them. We shall first introduce the basic graph theory notation and terminology used throughout this thesis in Section 1.1. The review and summary of the thesis is presented in Section 1.2.

1.1 Notation and Terminology

In the graph theory literature there exists a wide variation of notation and terminology. Hence, in this section we present the basic notation and terminology used throughout this thesis. For most part, we follow the graph theoretic notation and terminology in Bondy and Murty (1976).

A graph G is an ordered triple $(V(G), E(G), \phi_G)$ consisting of a non-empty set $V(G)$ of **vertices**, a set $E(G)$ of **edges** disjoint from $V(G)$ and an incidence function ϕ_G that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G. If both $V(G)$ and $E(G)$ are finite sets then graph G is said to be finite. The number of vertices in graph G is called the **order** of G and is denoted by $\nu(G)$; very often we set $\nu(G) = n$. The number of edges of G is called the **size** of G and is denoted by $\varepsilon(G)$. It is common to write $\nu(G) = |\mathrm{V(G)}|$ and $\varepsilon(G) = |\mathrm{E(G)}|$.

If u and v are vertices of the graph G identified by an edge e then edge e is **incident** to u and v. We write edge $e = (u, v)$ where u and v are the **end-points** of edge e. Further, vertices u and v are said to be **adjacent**. If every edge $e = (u, v)$ has a positive weight c_{uv} associated with it then the graph is weighted. An edge with identical ends is called a **loop** and an edge with distinct ends is called a **link**. Two or more edges joining the same pair of vertices are called **multiple** edges. A graph that is without loops and multiple edges is a **simple** graph. The **trivial** graph is a simple graph having one vertex. A graph is **empty** if it consists of isolated vertices (no edges).

A directed edge is an **arc**. The arc set of G is denoted by $A(G)$. A **directed graph** or **digraph** D is a graph with all its edges as arcs. It is formally defined as an ordered triple $(V(D), A(D), \phi_D)$ consisting of a non-empty set of $V(D)$ of vertices, a set of $A(D)$, disjoint from $V(D)$, of arcs and an incidence function ϕ_D that associates with each arc of D an ordered pair of (not necessarily distinct) vertices of D. If arc a has vertices u and v as its end-points and $\phi_D(a) = (u, v)$ then vertex u is the **tail** of arc a and vertex v is the **head** of arc a.

The **neighbour set** $N_G(S)$ of a vertex set S in graph G is the set of vertices

adjacent to vertices in S. When $S = \{v\}$ we write $N_G(v)$ instead of $N_G(\{v\})$.

The **degree**, $d_G(u)$, in graph G of a vertex u is the number of edges incident to u. For any graph G,

$$\sum_{u \in V(G)} d_G(u) = 2\varepsilon(G).$$

Therefore the number of vertices of odd degree in G is even. The maximum and minimum degrees of a graph G are denoted by $\Delta(G)$ and $\delta(G)$ respectively. An **r-regular** graph is one where $\Delta(G) = \delta(G) = r$. A **complete** graph is a simple graph where every vertex is adjacent to every other vertex. The complete graph on n vertices is denoted by K_n. K_n is $(n-1)$-regular with $\varepsilon(K_n) = \frac{1}{2}n(n-1)$. K_1 is the trivial graph.

Two graphs G and H are considered **isomorphic** if there exists a one-to-one mapping of their vertex sets which preserves adjacency. A graph $H = (V(H), E(H), \phi_H)$ is a **subgraph** of $G = (V(G), E(G), \phi_G)$ if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and ϕ_H is a restriction of ϕ_G to $E(H)$. If $H \neq G$ then H is a **proper subgraph** of G. If $V(H) = V(G)$ then H is said to be a **spanning subgraph** of G. The subgraph of G **induced** by V', denoted by $G[V']$, where V' is a non-empty subset of $V(G)$, is a graph with vertex set V' and $E(G[V']) = \{(u,v) \in E(G) : u, v \in V'\}$. The subgraph $G[E']$ of G induced by $E' \subseteq E(G)$ is defined similarly. The subgraph $G - V'$ is the graph obtained from G by deleting all the vertices of V' together with their incident edges.

A **matching** in G is a one-regular subgraph of G. A one-regular spanning subgraph is called a **perfect matching**. A **vertex cover** of graph G is a subset of vertices $V' \subseteq V$ such that for each edge of $E(G)$ at least one of its end-point belongs to V'.

The **complement** $\bar{G}$ of graph G is the simple graph with vertex set $V(G)$ and two vertices being adjacent in $\bar{G}$ if and only if they are not adjacent in G. When H is a subgraph of G, the **complement** $\bar{H}(G)$ of H in G is the subgraph $G - E(H)$.

The **line graph** of graph G, denoted by $L(G)$, is the graph whose vertex set is the set of edges of G, with two distinct vertices in $L(G)$ adjacent if and only if the edges of G they represent share a common end-point. If G is a digraph graph then it may contain a pair of arcs, say e_1 and e_2 sharing the same two end-points, say v_1 and v_2. The associated line graph is then called the **multi line graph** where a double link is imposed between the two end-points v_1 and v_2.

A graph G is said to be **embedded** in a surface S when it is drawn on S so that no two edges intersect. A graph is said to be **planar** if it can be drawn in the plane so that its edges intersect only at their ends.

A **walk** in G is a finite non-empty sequence $W = v_0e_1v_1e_2 \dots e_zv_z$ of vertices and edges such that for $1 \leq i \leq z$, the ends of edge e_i are v_{i-1} and v_i. W is said to be a walk from v_0 (the **origin**) to v_z (the **terminus**), or a (v_0, v_z)-walk. The **length** of W is the number of edges in W. For our purposes, and since our graphs are simple, we denote the walk $W = v_0v_1v_2 \dots v_z$. A walk whose origin and terminus are the same is called a **closed walk**. A **path** is a walk where all the vertices $v_0v_1v_2 \dots v_z$ are distinct. A **cycle** C is a closed walk $v_0v_1v_2 \dots v_zv_0$, where $n \geq 3$ and $v_1, v_2, \dots, v_z$ are distinct vertices.

A **Hamilton cycle** of graph G is a cycle containing every vertex of G. Any

graph that contains a Hamilton cycle is said to be **Hamiltonian**. A **Hamilton path** of graph G is a path that contains every vertex of G.

Two vertices u and v of G are **connected** if there is a (u, v)-path in G. Graph G is said to be connected if there is a path between every pair of vertices; otherwise it is **disconnected**. The **connectivity** of graph G is the minimum number of vertices that must be removed to disconnect G or result in the trivial graph consisting of one vertex. A maximal connected subgraph is called a **component**. A **tree** T is a graph that is **acyclic** (without cycles). The **distance** of two vertices in a tree T is the number of edges in the (unique) path joining the two vertices. The **diameter** of a tree T is the maximum distance in T. A **forest** is a graph with each component a tree.

A **spanning tree** of a graph G is a spanning subgraph of G that is a tree. It is well known that if T is a tree then

$$\varepsilon(T) = \nu(T) - 1.$$

Moreover, every connected graph G contains a spanning tree. Therefore, every connected graph G has

$$\varepsilon(G) \geq \nu(G) - 1.$$

Hence, in an edge-weighted graph with positive weights a minimum weight spanning connected subgraph is a spanning tree of the minimum weight. This is often referred to as the **Minimum Weight Spanning Tree (MWST)**. A **spanning forest** is an acyclic graph that is not necessarily connected.

Every graph G corresponds to a $\nu \times \nu$ matrix called the **adjacency matrix**. If the vertices of G are denoted by $v_1, v_2, \ldots, v_\nu$ then the adjacency matrix of G is the matrix $A(G) = [a_{ij}]$ where a_{ij} is the number of edges joining v_i and v_j.

Typically, a graph optimisation problem may be expressed as:

$$Minimise \qquad cx \tag{1.1}$$

subject to

$$Ax \leq b \tag{1.2}$$

$$x \geq 0$$

where x are unknown variables and A, b and c are the given matrices.

Equation (1.1) is referred to as the **objective function** whilst equations (1.2) are the **constraints**. When both the objective function and constraints are linear, the problem is known as a **Linear Program**. If some of the variables are specified as integer then it is referred to as a **Mixed Integer Linear Program (MILP)**. For the case when all the variables must be integer the problem is a pure **Integer Linear Program (ILP)**.

MILP and ILP problems belong to the class of problems known as **NP-complete**. Basically, there are three classes of problems. The first class, the class **P**, consists of problems that can be solved in polynomial time. This means there exist algorithms that solve all instances of these problems using a maximum number of steps that increases polynomially with some measure of the problem size. The second class, the class **NP**, consists of all the problems in P as well as other problems that can be solved by a non-deterministic algorithm in polynomial time. The third class, the class **NP-complete** problems, is a subset of NP having the property that all problems in NP can be reduced in polynomial time to one of them. This means that a problem P_1 can be reduced in polynomial time to problem P_2 if any instance of P_1 can be transformed in polynomial time into an instance of P_2, such that the solution of P_1 can be obtained in polynomial time from the solution of the instance of

P_2. A problem is considered **NP-hard** if every problem in NP is polynomially reducible to it.

A **heuristic** is an algorithm that is not guaranteed to find optimal solutions. For NP-complete problems where good exact algorithms are unlikely to exist, heuristic procedures are frequently produced. Good heuristic procedures can generate near-optimal solutions efficiently (time-wise). The quality of the heuristics is often determined by how far their results are from the optimal. This can be evaluated by $\frac{H-Opt}{Opt}$ where H is the results produced by the heuristic and Opt is the optimal solution. In cases where the optimal is unknown, it is often replaced by the lower bound LB to produce $\frac{H-LB}{LB}$.

We say that a function $f(n)$ is $O(g(n))$ whenever there exists a constant c such that $|f(n)| \leq c|g(n)|$ for all values of $n \geq 0$. A **polynomial time** algorithm is defined as an algorithm whose time complexity function is $O(p(n))$ for some polynomial function p where n denotes the input length.

The performance of an algorithm is usually determined by extensive testing on randomly generated data. For a fixed order n, the test graphs are generated by specifying the **probability** p of an edge occurring. The **density** of the graph is defined as $p \times \frac{n(n-1)}{2}$.

Besides the mentioned basic graph theory terminology, we shall now define some further terminology which we shall use in this thesis. For vertex u, $r(u)$ denotes the **degree restriction** on u. It represents the maximum number of incident edges allowed beside vertex u in a particular graph.

Let $V(G) = \{1, 2, \ldots, n\}$ and denote the degree of vertex i by $d_G(i)$. Where

G is understood we write d_i for $d_G(i)$. Let $n_i \geq 1$ be the number of vertices of degree d_i and suppose without loss of generality that $\delta = d_0 \leq d_1 \leq d_2 \leq \ldots \leq d_u = \Delta$. We call the sequence $d = (d_0, d_1, d_2, \ldots, d_u)$ the **degree spectrum** of G; note that for regular graphs $d_0 = d_u$.

A $(1,k)$-**tree** is a tree in which every vertex has degree 1 or k. A ($d_0 = 1, d_1, d_2, \ldots, d_u = k$)-tree is a tree with degree spectrum ($d_0 = 1$, d_1, d_2, ..., d_{u-1}, $d_u = k$).

In our algorithms we make use of the following terminology. The term **current tree** in an algorithm refers to the tree being considered at that instant. An edge e is said to be **available** if it has not been chosen in the current tree and its inclusion in the current tree does not form a cycle.

A **degree constrained spanning tree (DCST)** is a tree where every vertex satisfies the specified degree restrictions. In a weighted graph G, the **degree constrained minimum weight spanning tree**, is a DCST tree of minimum weight. We call such a tree an **optimal tree**.

Let G be a weighted graph and T a spanning tree of G. Consider an edge e of T. The subgraph $T - e$ consists of two components. We can reconnect these components by adding an appropriate edge of G not in T. The least cost edge f that can be chosen for this is called the **mate** of e. Thus, we get the tree $T' = T - e + f$ with $c(T') = c(T) - c_e + c_f$, where $c(T)$ is the cost of all the edges in tree T. Therefore, the gain of this exchange is $c(T) - c(T')$. If this gain is positive, then the exchange helps move the objective function nearer to the optimum. However, the gain could be positive or negative. The process of swapping edges is called an **edge exchange analysis**. When choosing an

edge to be included in the DCST if its edge weight is less than a certain **critical value** (CV) it is included in the tree. For the case when the objective function is to maximise some function, edges are chosen if their edge weights are greater than CV. Hence, the critical value (CV) is a value that acts either as a lower or upper restriction to the edge weights.

If $V = V_1 \cup V_2 \cup \ldots \cup V_k$ is a disjoint union of k sets of distinct vertices, we call each of the k partitions a **set**. The number of sets is denoted by b. The simple case when $b = 2$ is known as **bisection**. The sets V_i and V_j are said to be **disjoint** if there are no edges connecting vertices in V_i and V_j. The **cut set** (V_i, V_j), where $(i \neq j)$, of graph G are all the edges that have an end-point in V_i and the other in V_j. For our purposes, they are also called the **external** edges. The **internal** edges are the edges connecting vertices in the same set. We define the **difference** of vertex u, $Df(u)$, where $u \subseteq V_i$, as the sum of the external edge weights (edges in the cut of (V_i, V_j)) less the sum of the internal edge (edges in (V_i, V_i) weights.

If A is an $n \times n$ matrix, then a nonzero vector x in the plane R^n is called an eigenvector of A if Ax is a scalar multiple of x; that is $Ax = \lambda x$ for some scalar of λ. The scalar λ is called an **eigenvalue** of A and x is said to be an **eigenvector** corresponding to λ. When more than one eigenvector is utilised we get **multiple eigenvectors.** If A is a large sparse symmetric positive definite matrix in the form $Ax = b$, then the **Cholesky factor** L of A, is $A = LL^T$ where L is the lower triangular matrix of A and L^T is the transpose of matrix L.

1.2 Review and Summary of Thesis

The major focus of this thesis is to examine some of the different graph parameter restrictions on networks. Our main interest is in networks whose requirements can be formulated in terms of the degree restriction on vertices. The main objective is to determine optimal (minimum cut) networks. For the most part, these graph optimisation problems are NP-complete and exact algorithms are computationally expensive and time consuming. We focus on producing fast and efficient heuristics that produce near-optimal solutions.

The two main topics that we will explore are Restricted Spanning Trees and Graph Partitioning. The section on Restricted Spanning Trees deals with degree restrictions on the vertices of the tree. We examine a number of different cases including when all the vertices in the graph have a degree restriction bounded from above; when the degree restriction is only on a subset of V; and when a specified degree spectrum is to be present in the tree. Algorithms to efficiently obtain an effective solution are presented and discussed. The section on Graph Partitioning deals with various methods of partitioning the vertices of the graph G in such a way that the overall cost of the edges with end-points in different sets is minimum. The partitioning is usually restricted by constraints on the number of specified sets and the number of vertices permitted in each set. Algorithms for doing this are presented and discussed.

We present a literature survey consisting of work that has been carried out to date in Chapter 2. The first section, Section 2.1, provides some background of the types of graph optimisation problems that arise in network design. Different types of network problems are mentioned including the Minimum Cost problems, Capacitated Spanning Tree problems and Facility Location problems. Literature related to the applications of network design in the fields of

Operations Research, Computer Science and Engineering is also presented.

Section 2.2 deals with the topic of NP-completeness. A number of different NP-complete problems are outlined and the reasons for opting to heuristics are presented. The heuristic methods include: constructive methods; local transformation methods; decomposition methods; and feature extraction methods.

In Section 2.3 we investigate various types of Restricted Spanning Tree problems including: the Minimum Weight Spanning Tree problem; the Degree Constrained Minimum Weight Spanning Tree problem; the Capacitated Minimum Spanning Tree problem; the Balanced Spanning Tree problem; the Diameter Restricted Spanning Tree problem; and the Multi-Constrained Spanning Tree problem. We discuss methods that have been proposed for solving these problems. These methods consist of Branch and Bound techniques, Branch and Cut techniques, Lagrangean Relaxation techniques and various heuristic methods that utilised different edge exchange analyses. Relevant computational work is discussed.

Section 2.4 provides a survey of the Graph Partitioning problem where the objective is to cluster vertices in such a way that the sum of the weights of the edges that have end-points in different sets is minimum. This problem has various applications not only in graph theory but in Computer Science (circuit board design) where the aim is to keep the cost of connecting different components as low as possible. We review literature that deals with this problem and discuss the different methods proposed to solve it. These methods consist of parallel algorithms, edge exchange procedures, matchings and simulated annealing. Relevant computational results are discussed. Some equivalent formulations are also discussed. For example, instead of minimising the sum of the weights of the edges between sets, maximising the sum of the

weights of the edges that are in the same sets will also produce the same result.

In Chapter 3, we consider, the **Degree Constrained Minimum Weight Spanning Tree (DCMWST)** problem. The problem can be simply stated as:

> *Given an edge weighted graph* $G = (V, E)$ *with* $V = \{1, 2, ..., n\}$ *and positive integers* $r_1, r_2, ..., r_n$, *find a spanning tree* T *of* G *of minimum weight such that the degree,* $d_T(i)$, *in* T *of vertex* i *is at most* r_i, $1 \leq i \leq n$.

The reason for exploring this problem is its wide application in telecommunication networks. In these networks, the vertices in the graph correspond to communication centres (telephone exchanges, computer terminals, etc.), the edges in the graph correspond to communication links (telephone lines, conductor paths, etc.) and the degree restriction on a vertex represents the number of line interfaces available at a terminal or the traffic capacity of the terminal.

This problem has been analysed by many authors and both heuristic and exact methods have been developed. Gavish (1982), Narula and Ho (1980), Savelsbergh and Volgenant (1985), Volgenant (1989) and Malyshko (1985) presented different exact methods that utilised various Branch and Bound procedures and Lagrangean relaxations. These methods were tested on graphs ranging in size from 20 to 200 vertices. Recently, Caccetta and Hill (1997) developed an exact Branch and Cut method that was tested on graphs with 100 to 1000 vertices. The exact methods of Narula and Ho (1980), Savelsbergh and Volgenant (1985) and Caccetta and Hill (1997) utilised heuristics in their implementation.

We present a number of Mixed Integer Linear Programming formulations for

the DCMWST problem (Chapter 3). Much of our work is for the case when $d_i = r$, for every vertex i in G. Our work can also be found in Caccetta et. al (1998). Due to the NP-completeness of this problem and the extensive amount of work and time required to solve it we concentrate on heuristics. We start by outlining the famous algorithm of Kruskal (1956) and the algorithm of Prim (1957) for obtaining a MWST (as noted in Matousek and Nesetril 1996, the idea of Prim's algorithm goes back to Jarnik and Boruvka). We present seven heuristics to solve the DCMWST problem out of which four are new. We first carry out a **preprocessing** step which is common to all heuristics. This step efficiently includes all the edges that must be in every optimal tree.

As mentioned, of the seven heuristics we consider, four are new. The reason for presenting the other three heuristics (Heuristics 3.1.1, 3.1.2 and 3.1.6) is to provide a means of evaluating the quality of the new heuristics. These heuristics consist of modifications of Prim's (Heuristic 3.1.1) and Kruskal's (Heuristic 3.1.2) algorithms as well as a heuristic by Savelsbergh and Volgenant (1985) which is claimed to be the best available heuristic (Heuristic 3.1.6). As for our four new heuristics we utilise various methods of selecting edges to be included in the near-optimal trees. They include: sorting edges in increasing weight; checking the degree beside the relevant vertex before including the edge in the tree; building up spanning forests until a tree is obtained; checking if the edge weight satisfies a certain critical value; and sequentially building up a tree from a root vertex.

We implemented all heuristics in the C programming language on a SUN SPARC 2 workstation operating at 28.5 MIPS. We simulated 2500 random graphs. The values of the density p used were: 0.05, 0.25, 0.50, 0.75 and 1.00. We ranged n from 50 to 500 with stepsize of 50. We tested all the seven heuristics on different classes of 50 problems and recorded the statistic $\frac{H-LB}{LB}$,

where H is the average (of the 50 problems) of the DCST obtained from applying the Heuristic H and LB is the average lower bound obtained from the average weight (of the 50 problems) of the MWST. Besides that, we recorded the average CPU time (secs) required to solve the 50 problems. Our computational findings are presented in Section 3.2 together with a comprehensive comparison of the heuristics. From the results obtained our best heuristic always outperform the Savelsbergh and Volgenant (1985) heuristic by as much as 4%. In comparison to the Branch and Cut method by Caccetta and Hill (1997) our heuristic solutions were always within 4% from the optimal with the average speed (in terms of CPU time (secs)) at least 15 times smaller than that of the exact method. The computational results demonstrate the value of our better performing heuristics especially on large graphs of more than 100 vertices. Until now exact methods had only managed to efficiently solve the problem for graphs with at most 50 vertices.

We devote a short section (Section 3.3) to generalisations of the DCMWST problem. This section simply shows that the algorithm can be modified in such a way that the degree restriction imposed on all the vertices in the graph need not be the same. We implemented a random number generator that randomly assigns a different degree restriction (not necessarily distinct) to each vertex. The results of the generalisation are only presented for the best four heuristics. Our results show that the heuristics produced good results which are no more than 0.40% from the optimal with average computational time no more than a minute.

Section 3.4 looks at a special class of DCMWST where the degree restriction is only on one vertex of graph G. We refer to this problem as the **Degree Constraint One Vertex Problem**. This problem is useful in telecommunication networks when only one terminal is the centre and is to communicate with no

more than a certain number of terminals. The problem is not NP-complete and can be solved in polynomial time. Gabow (1978) presented an algorithm that runs in *O(E loglog V+V log V)*. The motivation behind analysing this problem is due to the fact that Gabow's algorithm (1978) when extended to solve problems with degree restrictions on more than one vertex of the graph does not perform well. We present two heuristics for the One Vertex problem which can easily be extended to tackle problems with degree restriction on more than one vertex of the graph.

Our first heuristic uses the idea of first satisfying the degree on the vertex with the degree restriction and then building up the tree using Kruskal's algorithm (1956). Our second heuristic first obtains a MWST and depending on the degree of the constrained vertex adds or deletes edges to it until its degree in the tree is satisfied. If an edge is added then we move along the created cycle to delete the most expensive edge that is not incident to the constrained vertex. On the other hand, if an edge is deleted we connect the created spanning forest using the cheapest edge not incident to the constrained vertex. Without loss of generality we always consider the first vertex of graph G as the vertex with the degree restriction. We implemented the two heuristics together with Gabow's algorithm (1978) in the C programming language on a SUN SPARC 2 workstation operating at 28.5 MIPS. We present a comparison of the heuristics with their respective CPU time (secs). For simplicity we only present analysis for graphs with edge density 0.25 and 0.75. Our obtained computational results showed that our heuristics are within 0.09% from the optimal with average speed (in terms of CPU time (secs)) within two to forty-six times smaller than Gabow's algorithm (1978).

Section 3.5, extends the Degree Constraint One Vertex problem to the case when the degree restriction is on two of the vertices of graph G. We call

this problem the **Degree Constraint Two Vertex Problem**. This problem arises in any network in which only two of its terminals can be the centres. These centres need not necessarily be adjacent. We first present a Mixed Integer Linear Programming formulation for this problem. As with the DCMWST problem we analyse heuristics for solving this problem due to the extensive amount of work and time required to solve it optimally. We give explanations on how the heuristics for the One Vertex Problem can be extended to solve the Two Vertex Problem. Without loss of generality we consider the first two vertices in graph G as the vertices with the degree restrictions r_1 and r_2. Note that this problem is equivalent to the case when the first two vertices have the imposed degree restrictions and all the other vertices have $(n-1)$ as their degree restriction.

We present three main simple ideas, namely:

- First satisfy the degree restrictions r_1 and r_2 on the two constrained vertices then build up the tree sequentially using Kruskal's algorithm (1956).
- Satisfy the degree restriction on either one of the constrained vertices and then apply Kruskal's algorithm (1956) to obtain a DCST. Depending on the degree of the other constrained vertex in this tree either add or delete edges incident to the vertex until its degree restriction is satisfied.
- Satisfy the degree requirement on either one of the constrained vertices then apply Gabow's algorithm (1978) with the aim of satisfying the degree on the other constrained vertex.

With these three ideas we come up with eight heuristics to solve the Two Vertex Problem. We implemented these heuristics in the C Programming Language like with all the previous heuristics. Comprehensive analysis of their

computational results with the average speed (in terms of CPU time (secs)) are recorded. The results presented are the statistic $\frac{H-LB}{LB}$ together with the average CPU time (secs) of the 50 problems. Again, we only present test runs for graphs with edge density 0.25 and 0.75. Our computational results show that our new heuristics not only produced results that are of higher quality than the extension of Gabow's One Vertex algorithm (by at least 2%) but are at least five times faster than the extension of Gabow's algorithm.

The next chapter of the thesis, Chapter 4, deals with a different type of degree restriction on trees. This chapter examines the $(1,k)$-**tree** problem which is defined as: given a graph G with maximum degree k, find a spanning tree consisting of vertices of degree 1 or k. Douglas (1992) provided the motivation for this problem with his $(1,3)$-tree result:

> **Theorem 1.2.1** *Given a planar graph $G=(V,E)$ with maximum degree 3, it is NP-complete to decide if there exists a spanning tree T for G such that $deg(x,T)=1$ or 3 for all $x \in V$.*

We strengthen this result by proving that the non-planar case of this problem still holds (Section 4.2 and Caccetta and Lam 1998). We achieve this by converting the problem to its equivalent Hamiltonian path problem. Our proof uses the idea of Garey and Johnson's proof (1979) for the Hamiltonian circuit problem. Similarly, we prove the complexity of the $(1,4)$-tree problem in graphs with maximum degree 4 (Section 4.2). Further, we prove the complexity of the $(1,3,5)$-tree problem in graphs with all internal vertices having degrees 3 and 5 to be NP-complete.

Due to the fact that the number of degree one vertices is always of interest in any tree, we provide some simple counting to determine this value in the general $(1,k)$-tree. We also determine the number of degree k vertices in the

$(1,k)$-tree. We establish upper and lower bounds on the number of degree one vertices in a spanning $(d_0 = 1, d_1, \ldots, d_m = k)$-trees. These bounds are:

$$\frac{(d_1 - 2)\nu + 2 + (d_2 - d_1)(m-1) + \sum_{i=3}^{m}(d_i - d_2)}{(d_1 - 1)} \leq n_0$$

$$\leq \frac{(d_m - 2)\nu + 2 - \sum_{i=1}^{m-1}(d_m - d_i)}{(d_m - 1)}$$

where n_i is the number of vertices of degree d_i.

Due to the complexity of the $(1,k)$-tree problem, we approach the computational aspect of this problem by heuristics. The aim is to come up with fast procedures that can find a $(1,k)$-tree given any graph G, if such a tree exists. We present two heuristics in Section 4.3. We first carry out a preprocessing step to check the graphs for certain attributes like connectedness and maximum degree k. Our heuristics work on building up the $(1,k)$-tree sequentially.

We implemented our two heuristics in the C programming language similarly to the implementation of our other previous heuristics. Our first heuristic starts with any vertex of degree k in graph G and includes all its incident edges and neighbouring vertices in the tree. For the included neighbouring vertices find one of degree k and include its other $(k-1)$ edges in the tree if their inclusion does not form a cycle. The heuristic builds up the $(1,k)$-tree in this continuous manner. Our second heuristic partitions the graphs into two sets, one with vertices of degree k and the other with vertices of degree $\leq (k-1)$. It systematically includes and excludes certain edges in the tree until a $(1,k)$-tree is found. Due to the fact that a large portion of random graphs do not have a $(1,k)$-tree, we implemented a random generator to produce graphs having such trees. This generator randomly adds edges to an initial $(1,k)$-tree until the prescribed density p is achieved. It is on these graphs that our heuristics

are tested. Our computational results (Section 4.2) show that our heuristics are fast and can find a $(1, k)$-tree at least 70% of the time (our best heuristic has 90% success rate).

Chapter 5 of the thesis is devoted to the Graph Partitioning problem. This problem is that of dividing the vertices in graph G into clusters of specified sizes such that the sum of the weights of the edges with end-points in different clusters are kept to a minimum. This NP-complete problem (Garey and Johnson 1979) has wide application in many scientific and engineering problems including: parallel computing (Barnard and Simon 1993, Diniz et al. 1995, Gilbert and Zmijewski 1987, Hendrickson and Leland 1995b, Savage and Wloka 1991, Simon 1991, Vanderstraeten et al. 1995, Walshaw et al. 1995); placement of circuit elements (Feo and Khellaf 1990); and pattern recognition (Blake 1994). For example, in the case of parallel computing, efficient use of distributed parallel computers requires that the computational load be balanced across processors in a way that minimises interprocessor communication. As for the placement of circuits, since there are only a maximum number of components which may be placed on any one circuit card, and the connection cost between cards are much higher than the connections between boards, the objective is to minimise the number of interconnections between cards and maximise the number of interconnections between boards. Hence, in all such applications the aim is to minimise the connection between terminals that are in different systems.

In Chapter 5 we first present a Mixed Integer Linear Formulation of the Graph Partitioning problem. Some equivalent formulations are also mentioned. For example, instead of minimising the sum of the weights of the edges between sets, maximising the sum of the weights of the edges that are in the same set will also produce the same results. This is so since the total cost of all the

weights of the edges in any graph is always a constant.

The classic Kernighan and Lin (1970) heuristic for bisecting graphs is presented in Section 5.1. This heuristic partitions the vertices in the graph into two sets such that the weights of the edges in the cut set are minimal. We outline the main idea used by Kernighan and Lin (1970) for swapping vertices between sets. The heuristic of Feo and Khellaf (1990) based on the idea of matchings to partition vertices into sets with four vertices is also analysed.

We use the Kernighan and Lin idea to develop a new method for calculating gains to swap the vertices between sets. The vertices that produce the most gain when swapped are interchanged. This method is also used as a refinement step for all our heuristics. The main idea of our heuristics is to build up the sets sequentially according to incident edge weights of the vertices that are to be included in a certain set. The various methods used in building up the sets include: sorting and choosing the edges in decreasing weights, checking if the edge weights satisfy a certain prescribed critical value and using matchings. With these we come up with nine fast and simple heuristics.

Further, analysis is done to extend the heuristic of Kernighan and Lin to solve problems involving four and eight sets. The heuristic of Feo and Khellaf (1990) is also extended to solve problems with eight vertices in each set. Refinement of these heuristics is also carried out.

We implemented and tested all the heuristics in the C programming language on a SUN SPARC 2 workstation operating at 28.5 MIPS. A comparative analysis between the heuristics is carried out and presented. Our computational work indicates that the two best performing heuristics are those that select

edges according to a prescribed critical value. One such critical value is the average edge weights of all the edges in the graph. Only edges that have weights more than this value are included in the sets. Further, our results showed that although Kernighan and Lin's heuristic (1970) is very efficient for bisecting graphs its quality deteriorates when applied to four and eight partitions. This is also the case with the heuristic of Feo and Khellaf (1990) when further applied to solve problems with eight vertices in each set. In all cases our heuristics required little CPU times (secs).

We conclude this thesis with a discussion on some open problems in the final Chapter 6.

Chapter 2

Literature Review

In this chapter we present a survey of some Restricted Spanning Tree problems and the Graph Partitioning problem. Section 2.1 provides some background of network design together with applications in various fields including Operations Research, Chemistry, Mining and Computer Science. Section 2.2 provides some basic background on NP-completeness and heuristics.

Restricted Spanning Tree problems arise in many network applications. Some fundamental problems are listed below. In our description we adopt the notation of $V = \{1, 2, \ldots, n\}$ and $c_{ij} =$ *the cost of edge* (i, j).

- The classic **Minimum Weight Spanning Tree problem** (MWST) which is:

  Given an edge weighted graph $G = (V, E)$, *find a spanning tree* $T = (V, E')$, $E' \subseteq E$ *such that* $\sum_{(i,j)\subseteq E'} c_{ij}$ *is minimum.*

- The **Degree Constrained Minimum Weight Spanning Tree problem** (DCMWST). This problem is defined as:

  Given an edge weighted graph $G = (V, E)$ *with* $V = \{1, 2, \ldots, n\}$ *and positive integers* $r_1, r_2, \ldots, r_n$ *find a spanning tree* T *of* G

of minimum weight such that the degree, $d_T(i)$, in T of vertex i is at most r_i, $1 \leq i \leq n$.

- The **Capacitated Minimum Spanning Tree problem** (CMST) which is:

  Given an edge weighted graph $G = (V, E)$ with a centre (source vertex) specified, find a minimal spanning tree T for transferring commodity from the centre to every other vertex such that no more than a certain number of r terminals can be directly connected to the centre, each arc capacity is satisfied and the vertex demands are met.

- The **$(1, k)$-tree problem** which is:

  Given a graph G find a spanning tree T with all vertices having degree 1 or k.

- The **Balanced Spanning Tree problem** (BST) which is:

  Given a graph $G = (V, E)$, find a spanning tree $T = (V, E')$, $E' \subseteq E$ such that the number of vertices in any subtree created when disconnected from a chosen vertex (the root vertex) does not differ by more than one.

- The **Minimum Weight Spanning Tree with Bounded Diameter problem** (MWSTBD) which is:

  Given an edge weighted graph $G = (V, E)$, find a minimum weight spanning tree T having diameter at most D.

- The **Minimal-Cost Spanning Tree problem subject to Resource constraint and Flow requirement** (MCSTRF). In this problem there are two weights on each arc, one representing the arc capacity and the other the cost of the arc. The problem is:

Given a graph $G = (V, E)$ with one source vertex of infinite supply and every other vertex a sink with known constant demand together with the cost of constructing the arcs, the amount of each scarce resource consumed during the construction of the respective arc, and the flow capacity of the arc, find a minimum cost spanning tree T with the consumption satisfying the scarce resource constraint and the flow from the source to the sinks satisfying the demand and capacity constraints.

In Section 2.3 we discuss the progress made on the above problems including the new results on the DCMWST and the $(1, k)$-tree problems presented in this thesis.

The last section of this chapter, Section 2.4, deals with the analysis of the Graph Partitioning problem where the aim is to cluster vertices of graph G in such a way that the sum of the edges with end-points in different sets is minimised. We analyse work that has been done to solve this problem and comparison is made between the different methods.

2.1 Background of Network Design

Network design is of utmost importance in many real-life applications because it provides accurate representations of the problems at hand. It is a core domain for many Operations Research problems as well as in many areas of Computer Science, Applied Mathematics, Engineering and Management. Golden et al. (1981) stated that the motivation for efficient network algorithms is:

- Efficient solution of large scaled networks has resulted in improvements in productivity and service, reduced distribution, communication and transmission costs and hence benefited society overall.

- Network design models are easier to visualise, more informative and intuitively appealing thus more likely to be implemented and used.

- Network problems can be formulated as mathematical programs and linear algebra used to characterise the underlying structure of the optimisation problem.

Many authors (see Caccetta and Vijayan 1987, Caccetta 1989 and Ahuja et al. 1995) have devoted time in providing a comprehensive analysis of the many applications of network optimisation. Caccetta and Vijayan (1987) discuss many applications of graph theory in areas of Chemistry, Communication Networks and Operations Research. They showed how Smolenski (1964) made use of graph theory to obtain some chemical structures which are isomorphic to certain graph structures; how the hydrocarbon structure C_nH_{2n+2} (Cayley 1874) is in fact a representation of a tree and the problem of finding the number of isomers is equivalent to the problem of counting the number of unrooted trees with n vertices (Cayley 1874 deleted, without loss of generality, the vertices of degree 1 representing the hydrogen atoms). Caccetta and Vijayan (1987) listed some ways of classifying other chemical structures using graph theory mainly in the areas of chemical documentation and classification. Furthermore, graph optimisation was used to highlight the importance of network design in Communication Networks, for example the Minimum Cost Network problem. The authors mentioned many different optimisation problems like the Capacitated Minimum Spanning Tree problem and the Minimum Order Graph problem. They provided some insight of the extent each relevant problem had been solved. Applications of graph theory in routing problems (such as the Travelling Salesman problem) were discussed. A subsection was devoted to the Open Pit Mine problem with the classic Lerchs-Grossman algorithm presented. The authors proposed some implementation strategies for the Pit Mine problem like the 'bounding' technique based on dynamic programming and graph manipulation techniques. The article concluded with applications

of graph theory in other disciplines like Geography (stream systems, location problems), Anthropology (classification of interested areas) and Physics (lattice and lines between crystal structures).

Ahuja et al. (1995) presented 42 applications drawn from the various fields of Operations Research, Computer Science, Physical Sciences, Medicine, Engineering and Applied Mathematics. They provided models that depict a large range of applications and cover many basic types of network optimisation. In addition to the 42 applications they included references for an additional 140 applications. Some of their applications are: DNA sequence alignment; optimal loading of a hopping plane; determining chemical bonds; urban traffic flows; multi-vehicle tanker scheduling; multi-item production planning; and levelling mountain terrain. These applications were used to illustrate the various problems like: the Shortest Path problem; the Maximum Flow problem; the Minimum Cost Flow problem; the Assignment problem; the Maximum Matchings problem; the Minimum Spanning Tree problem; the Convex Cost Flow problem; the Generalised Flow problem; the Multicommodity Flow problem; the Travelling Salesman problem; and Network Design problems.

Golden and Magnanti (1977) provided a bibliography of nearly 1000 entries that are related to network optimisation articles and books. They divided the bibliography into three main sections which are network analysis, network synthesis and unifying topics. Under the category of network analysis they listed entries from Network Reliability, Traffic Equilibrium, the Shortest Path Problem, the Single and Multi-Commodity Flow problem, Matching and the Chinese Postman problem, the Travelling Salesman problem and the Vehicle Routing problem. The category of network synthesis provided entries for the Minimum Spanning Tree problem, the Facility Location problem and the Design of Optimal Network problem. Unifying topics listed articles on Imple-

mentation Issues, Complexity Theory and Matriods, and Graph Theory. Note that these categories are not mutually exclusive.

Caccetta (1989) detailed a list of graph optimisation problems which arise in network design. The purpose of the article was to introduce relevant network problems, highlighting the graph theoretic aspects. For each identified problem the author indicated what is known and the future work that can be carried out to solve the respective problems. In the Diameter and Connectivity section, the author analysed problems regarding Edge Minimal Graphs and Diameter-Critical Graphs. The author discussed the applications of graph theory to problems arising in the design of large communication networks as well as in very large scale integration (VLSI) circuit design. The importance of graph theory in assisting to solve some of these problems was illustrated. For example, in the case of probabilistic graphs in communication systems, where the weights of the graph represent the probability of the component functioning correctly, the aim is to determine the most reliable network.

Minoux (1989) provided a survey in the area of network synthesis and optimum network design. The article was aimed to assist researchers to identify the fundamental structure of a problem and relate it to already published work. The three main models covered in the article were the tree-like networks, the Minimum Cost Multicommodity Flow model and the Non-simultaneous Single-commodity or Multicommodity Flow model. For each model the most important variants of the problem were mentioned together with the various ways of solving the problems that had been carried out by other researchers. For instance, in the Minimum Cost Multicommodity Flow model the variants discussed are: the Minimum Concave Cost Multicommodity Flow problem; the Minimum Linear Cost (Fixed Cost) Multicommodity Flow problem; the Minimum Linear Cost Multicommodity Flow problem with a budget constraint;

the Minimum Cost Multiterminal (Single Commodity) Network Flow problem; the Optimum Rented Lines Network problem; and the relation between the Minimum Cost Multiterminal Network Flow problem and the Steiner Tree problem (trees whose set of vertices is a subset of V). Some of the Mixed Integer Linear Programming formulations of those problems were presented together with the most efficient technique (up to then) outlined.

2.2 NP-completeness and Heuristics

Due to the practical importance of network design in many real-life applications, researchers are always looking for new methodologies that will provide better feasible solutions, require less execution time and storage space, are flexible to model changes and simple to implement. However, the problem that many researchers face is the complexity of the problem at hand. Many network optimisation problems are NP-complete and cannot be solved optimally in polynomial time.

Garey and Johnson (1979) provided a list of many NP-complete network problems. Their book presented the various techniques that are normally utilised in proving that a problem is NP-complete. They stated the six basic NP-complete problems which are: 3-Satisfiablility, 3-Dimensional Matching, Vertex Cover, Clique, Hamiltonian Circuit, and Partitioning. These problems are normally utilised to prove that other graph theory problems are NP-complete. A comprehensive list of other such problems is included as a guide for future reference.

Garey et al. (1976) showed that a number of NP-complete problems remain NP-complete even when their domains are substantially restricted. Their paper was motivated by the fact that many real-life applications do not occur with generality but in restricted conditions due to the additional constraints

imposed. They showed that the Simple Max Cut problem which is the Max Cut with edge weights restricted to 1 is NP-complete. Furthermore, they showed that even if the domains of the Node Cover and Directed Hamiltonian Path problems are restricted to planar graphs, the problems are still NP-complete. This is also true with some other graph problems even when their domains are restricted to graphs with low vertex degree. For the Graph 3-Colourability, Node Cover and Undirected Hamiltonian Circuit problems, they determined the lowest possible upper bounds on vertex degree for the problems to remain NP-complete.

Johnson has an ongoing guide of NP-complete problems that appears in the Journal of Algorithms every quarter. The column provides continuing coverage of new developments in the theory of NP-completeness. We are particularly interested in the fourteenth edition (1985) which provides further insight into the Minimum Spanning Tree problem. It looked at various types of spanning trees like the Minimum Max-Flow Spanning Tree, Degree Constrained Subgraph, Minimum Cycle Basis Spanning Tree (spanning trees that yields a "cycle basis" for G of total length B or less), Bounded Diameter Decomposition (graphs with a subset $E' \subseteq E$ where both $G' = (V, E')$ and $G'' = (V, E - E')$ have a diameter of D or less), Maximum Weight Connected Subgraph (a connected subgraph of G whose total vertex weight is at least B), and a few others. The constraints that render these problems NP-complete are outlined. For example, the Bounded Diameter Decomposition problem remains NP-complete even if the diameter $D = 3$ and G is bipartite. The paper also analysed the suitability of a network for a given task, its reliability and the way the network works. The paper concluded with some problems of network design used primarily for organising and representing data rather than transmitting it. For instance in the 'PERT network' of an acyclic directed graph, the arcs represent tasks to be performed and the length of an arc represents the expected time to

perform the corresponding task. Once constructed the network is analysed to determine the bottlenecks in the overall project (arcs along the longest path from the start vertex to the finish vertex).

Papadimitriou and Yannakakis (1982) provided some complexity of other Restricted Spanning Tree problems. Given a distance matrix they were interested in finding a shortest spanning tree that is isomorphic to a given prototype. They presented some isomorphism properties (isomorphic to a path, isomorphic to a full binary tree) that caused the Minimum Spanning Tree problem to be NP-complete. They examined a class of families of trees called the d-bouquets (trees with d disjoint flowers as subtrees) which included the 3-stars and the full binary trees. They showed that the Minimum Spanning Tree problem is NP-complete if the problem is defined by a family of d-bouquets with d increasing as a positive power of the number of vertices in the graph. They also looked at the *dissociation number* of the tree, that is, the removal of the minimum number of vertices that would reduce the tree to isolated vertices and edges.

Camerini et al. (1980) wrote an article regarding the complexity of the spanning tree problems. Their main concern was finding an optimum restricted weighted tree, since they reasoned that network design is always concerned with trees and that spanning trees are the fundamental structure of graphs. They limited their attention to undirected graphs and considered only the complexity of optimum single weighted spanning tree problems. They stated a list of tree weight functions like the Shortest Path and the Maximum Flow, and gave a few ways of solving these proposed functions. Their aim was to identify the borderline between easy and hard problems.

Golden et al. (1981) provided a list of some network-related NP-complete

problems and stated various methodologies that can be utilised to solve these problems. They believe that the objective in future research is to develop methodologies for solving new abstract network problems as well as the well-studied network problems. Their aim is to improve current methodologies with respect to: quality of solution (feasibility), running time and storage requirements, difficulty of implementation, flexibility to model changes, robustness and simplicity. The methodologies proposed included analysis of heuristics, characterisation of polyhedra for NP-complete problems, transformation between NP-complete problems, better formulation of network optimisation problems, finding network structure in linear programs and a few others. They concluded with a list of some research applications of network problems which included: routing and scheduling of vehicles and crew, project scheduling using PERT/CPM, pipeline layout and design, facility location, cash flow management, marketing and distribution.

Due to the fact that most Restricted Spanning Tree problems are NP-complete and good efficient algorithms may not exist for solving these problems, researchers have opted for heuristics. In particular, heuristics that produce results that are 'close' to the optimal in polynomial time. Lin (1975) provided some insight to the nature of problems that are solved by heuristics. He also gave some guidelines for constructing efficient heuristics that utilised some exhaustive searches, some transformation methods and some reduction procedures to solve a variety of combinatorial optimisation problems.

Weiner (1975) listed various methods that can be used for developing heuristics like: constructive methods; local transformation methods; decomposition methods; feature extraction methods; inductive methods; and approximation methods. He commented that these methods are particularly successful for instances that included the modification of the objective function, addition of

constraints and elimination of constraints. Further, these heuristic techniques can be usefully combined. For example, after an approximation method is applied, constructive or inductive methods can be used to obtain 'initial' solutions that can then be improved by the local transformation methods. Weiner (1975) noted that the way to evaluate heuristics is to experimentally compare several alternate heuristics and that results are needed to establish error bounds for heuristic techniques.

Silver et al. (1980) defined a heuristic as a procedure that solves a well-defined mathematical problem by an intuitive approach in which the structure of the problem can be interpreted and exploited intelligently to obtain a reasonable solution. They analysed the different types of heuristic methods (feature extraction method, model manipulation method, local improvement method, and constructive method) as well as the measurement of the quality of these heuristics. Other issues that are discussed are the interactive role humans play during the execution of a heuristic method and the factors that may be important in the choice of testing of a heuristic method. These factors include: amount of computational effort permitted; number of decision variables; size of the problems; discrete versus continuous; and deterministic versus continuous.

Although heuristic methods do generate a tremendous interest in computational research, this does not mean that exact algorithms are no longer of interest. Researchers are always looking for ways to develop fast exact methods to solve NP-complete problems since they produce optimal solutions. An active area of current research is that of the Branch and Bound and Branch and Cut methods for solving MILP. Work is always being carried out with the aim of producing fast exact algorithms.

2.3 Restricted Spanning Trees

Among the various network related problems we focus on the Restricted Spanning Tree problems stated in the introduction of this Chapter. The objective of these problems is to find a minimum cost tree that satisfies a given set of constraints. We begin our discussion with the classic MWST problem and then progress through each of the problems listed in the introduction.

- **The MWST Problem**

The MWST problem has attracted much attention since publication of Kruskal's greedy algorithm in 1956. His method repeatedly selects the cheapest edge to build up the components. Another classic algorithm was discovered independently by Prim (1957) and Dijkstra (1957). This algorithm repeatedly selects the cheapest edge beside the component to build up the tree. Both algorithms are efficient and easy to implement. Over the years many researchers have come up with various efficient algorithms that use sophisticated sorting and storage methods in both the sequential and parallel settings.

Cheriton and Tarjan (1976) presented several algorithms for finding the minimum spanning trees in graphs that have *O(m log log n)* worst-case running times with n and m respectively denoting the number of vertices and edges in the graph. They studied the relationship between the minimum spanning tree problem and certain data manipulation needed to implement the algorithms that may lead to a general lower bound on the complexity of the problem. These methods include combining sets of vertices, utilising priority queues and some cleaning up steps like deleting superfluous edges. They also presented an algorithm with worst-case time of $O(m)$ for dense graph and another of $O(n)$ for planar graphs. Their algorithm uses the implementation of priority queues

and merges the trees repeatedly until one tree is left.

Eppstein (1994) developed an efficient algorithm for maintaining a minimum spanning tree subject to a sequence of edge weight modifications. A reduction and contraction method that contracts edges and merges vertices was used. This way the graph is made smaller and thus a more efficient procedure results. The algorithm is efficient for off-line computation of minimum spanning tree. This means that if the input of the graph is a long sequence of updates, the corresponding sequence of minimum spanning tree changes can only occur when the entire output sequence is known. The algorithm can be applied to planar and rectilinear point sets. The time taken for the planar graph is $O(k\ log^2\ n)$ and for the rectilinear metric is $O(k\ log\ n\ loglog\ n)$ where k is the number of updates performed and n is the number of vertices in the graph.

Johnson and Metaxas (1995) looked at parallel algorithms for computing the minimum spanning trees of graphs. They presented an algorithm that runs in $O(log^{\frac{3}{2}}\ n)$ where n is the number of vertices in the graph. The major innovation of their algorithm is that vital information about components of the graph can be extracted without ever shrinking the components unlike other parallel algorithms. The algorithm creates a linked list representing a preorder traversal of the components in the tree and uses this linked list to obtain information about the components. Thus there is no need for any shrinking. This algorithm showed an improvement in the run time, by a factor of $\sqrt{log|n|}$, over previous results of the same tested problem.

Hwang (1979) presented an algorithm for finding a minimum spanning tree for graphs with rectilinear distances that runs in $O(n\ log\ n)$, n being the number of vertices in the graph. The algorithm used the idea of a **Voronoi diagram**. Given a set of coplanar points P, for each point P_x in the set P, a boundary

can be drawn that encloses all the intermediate points lying closer to P_x than to other points in the set P. Such a boundary is called a **Voronoi polygon** and the set of all Voronoi polygons for a given point set is called a **Voronoi diagram.** Voronoi diagrams depend critically on the distance function and since the author was considering rectilinear distance, the objective was to obtain a rectilinear minimal spanning tree. Hwang (1979) gave a recursive method for constructing the Voronoi diagram for n given points that identifies the intersection of some line segments.

Ravi et al. (1996) approached the spanning trees in a slightly different manner. Their interest was to find a minimum spanning tree that consists of only k specified vertices of the graph. This is called the k **Minimum Spanning Tree problem.** This problem is NP-complete even for Euclidean graphs. They presented an algorithm that starts with each vertex as a singleton-connected component and then merges and clusters these components until the required tree is found. The algorithm has a performance ratio of $O(2k^{\frac{1}{2}})$ for the general edge-weighted graph and $O(k^{\frac{1}{4}})$ for Euclidean problems. They also presented the NP-completeness proof of this k Minimum Spanning Tree problem and proceeded to investigate k-trees with minimum diameter. They presented a simple technique that provides a polynomial-time solution for finding k-trees of minimum diameter.

Gabow (1977) presented two algorithms for generating spanning trees of a connected graph in increasing weight. His first algorithm generates the k smallest spanning trees of graph G where the value of k can be specified before or during the generation of the trees. This algorithm runs in $O(k\ m\ \alpha(m,n) + m\ log\ m)$ where n and m represent the number of vertices and edges in the graph respectively and α is Tarjan's inverse of Ackermann's function. Ackermann's function is a function of two parameters whose value grows very fast. It is

defined as $A(1,j) = 2^j$, $A(i,1) = A(i-1,2)$, and $A(i,j) = A(i-1, A(i,j-1))$. Tarjan's inverse of Ackermann's function finds the inverse of this function. The algorithm of Gabow (1977) uses an edge exchange procedure to obtain other trees and uses the minimum weight spanning tree as a reference. Gabow (1977) proceeded to modify the algorithm to generate all spanning trees of the graph in increasing weight. He tested his algorithms on random connected graphs with number of vertices, n, ranging from 10 to 60. The edge weights are integral, chosen in the interval [50,10000]. His results and analysis indicated that the algorithm has almost a linear performance.

Deo and Kumar (1997) presented a compilation of 29 constrained spanning tree problems like the DCMWST problem, the CMWST problem, the MWSTBD problem and the Spanning Tree Optimisation in VLSI Design problem. They presented two generic algorithms for solving large graphs on massively-parallel SIMD (Single Instruction, Multiple Data) machines. Their first method starts by finding a Minimum Weight Spanning Tree and then sequentially increasing the weights of selected edges so that the next Minimum Spanning Tree has fewer violations of the constraint. This process is repeated until a spanning tree without constraint violations is obtained. As for their second method they constructed a spanning tree satisfying the imposed constraints by employing problem-specific heuristics every step of the tree construction. Deo and Kumar (1997) tested their methods on large graphs with as many as 8192 vertices and approximately 18 million edges. Their analysis of the DCMWST problem supports the efficiency of their methods with results that are always within 0.2% from the lower bound (the weight of the Minimum Weight Spanning Tree). Their computational times are also promising with the second method taking less than a minute of CPU time to compute a spanning tree on graphs with approximately 16 million edges.

• **The DCMWST Problem**

We next look at the case when the minimum spanning tree is to satisfy a prescribed degree constraint. This problem is the Degree Constrained Minimum Weight Spanning Tree (DCMWST) problem. The problem was proven to be NP-complete by Garey and Johnson (1979) for the case when the degree restriction is on two or more vertices of the graphs. Note that for the case when $r_i = r = 2$, $1 \leq i \leq n$, the problem is equivalent to the Hamiltonian Path problem.

For the simple case when the degree restriction is only on one vertex of the graph, Gabow (1978) developed an 'edge exchange' method to find the spanning tree. The algorithm runs in *O(m log log n+n log n)* (n and m are the number of vertices and edges in the graph respectively) and uses the idea of merging queues once the best 'mate' of an edge is found and an edge exchange has been carried out. The author claimed that his algorithm is suitable for solving the problem when the constraint on the vertex is bounded from below, above and even when it is an equality bound. Gabow et al. (1986) used F-heaps to obtain a fast algorithm for solving the Minimum Weight Spanning Tree problem. Their algorithm can be extended to allow degree restriction at one vertex. The algorithm uses the insertion and deletion methods and can be applied to both the directed and undirected graphs. For directed graphs the algorithm runs in *O(m log n)* time where n and m are the number of vertices and edges in the graph respectively.

Glover and Klingman (1974) used a 'quasi-greedy' algorithm to obtain a degree constrained spanning tree at one vertex. They provided labelling procedures that were analogous to the basis exchange steps that were used in specialised linear programming procedures for solving the Minimum Cost Flow Network

problem. This procedure enables the primal and dual methods to be applied by means of 'modified pivot steps'. In the dual case the procedure gives optimal spanning tree of the next higher or lower order hence producing an algorithm that is very efficient.

We have developed two simple heuristics to solve the one vertex constrained problem (Section 3.4). Our first heuristic consists of first satisfying the degree on the vertex with the degree restriction and then using Kruskal's algorithm (1956) to build up the remaining tree. Our second heuristic starts with a MWST and then adds or deletes edges incident to the vertex with the degree restriction to obtain the required tree. We tested our heuristics against Gabow's algorithm (1978) and our computational results showed that our heuristics were within 0.09% from the optimal with CPU time (secs) at least two to forty-six times faster than Gabow's algorithm (1978). We further extended the algorithm to the case when the degree restrictions are on two of the vertices in the graph (Section 3.5). This case is applicable to telecommunication networks when at any one time only two of its terminals can be centres. We develop eight heuristics based on various edge exchange procedures utilising Kruskal's algorithm (1956) and Gabow's algorithm (1978). We carried out a comprehensive analysis on these heuristics and found that our best heuristic is the one that starts with a MWST and subsequently adds or deletes edges incident to the vertices with the degree restrictions. Our computational results always produced solutions that are closer to the optimal than that of the extension of Gabow's algorithm (1978) with substantially less CPU time (secs).

The problem when all the vertices in the graph have the same degree restriction, that is $r_i = r$ for every i, has been considered by a number of authors and both heuristic and exact methods have been proposed. Gavish (1982) developed a Lagrangean relaxation method for the problem with a subgradient

procedure employed to obtain the optimal Lagrange multipliers. The method was tested on 630 Euclidean problems ranging in size from 20 to 200 vertices with r taking values from 2 to 20. His results showed that the subgradient procedure generated very tight bounds and even for the more difficult problems the bounds were only 0.74% from the optimal.

Narula and Ho (1980) developed three procedures (2 heuristics and 1 exact) to solve the DCMWST problem. One heuristic is basically a modified Prim's algorithm whilst the other starts with a minimum weight spanning tree and moves towards feasibility through a series of edge exchanges. These heuristics along with a penalty method were used in a Branch and Bound procedure to develop an exact algorithm. Their procedures were tested on 120 randomly generated problems of size 30, 50 and 100 with the Euclidean distances as the cost of the edges. They restricted their testing to degrees 2, 3 and 4 since they used the fact that for Euclidean distances the maximum degree on any vertex in the degree constrained spanning tree is no more than 6. Their computational results showed that their heuristics do not vary by more than 2% from the optimal on the 100 vertex graphs. Even though their exact algorithm is efficient for graphs of order 30 and 50, the Branch and Bound procedure required more core storage space (approximately twice) compared to the two heuristics.

Savelsbergh and Volgenant (1985) described a Branch and Bound procedure adapted from the Travelling Salesman problem. The procedure is based on edge exchange and utilises three heuristics (the primal method of Narula and Ho (1980), a heuristic of Christofides (1976) and a new heuristic that is based on edge exchange). They too tested their methods on Euclidean problems with $r = 3$ and 4. Computational results based on 200 problems were produced and their comparative analysis demonstrated superiority over the methods of Narula and Ho (1980). This superiority is due to: an improved branching

scheme, a better initial upper bound, the use of fast heuristics; and the elimination of edges by using an edge exchange analysis.

Volgenant (1989) also presented a Lagrangean approach to the DCMWST problem. He showed that Lagrange multipliers compute better lower bounds that are much closer to the optimum. Computational results were presented for the Euclidean problems by Narula and Ho (1980) as well as for randomly generated problems with up to 150 vertices. Their results demonstrate the usefulness of the Lagrangean approach. Their Lagrange multipliers resulted in lower bounds that are much closer to the optimum and together with their edge elimination procedure reduced the branching search considerably.

Malyshko (1985) discussed an exact method that reduces the DCMWST to a $p = \lceil \frac{n}{k} \rceil$- median problem by using the Simplex method and a Branch and Bound procedure. He proposed a reduction method of the problem to the p-median problem and an approximation algorithm of local optimisation. The methods were tested on problems with $(n, r) = (28, 3)$ and $(100, 5)$. On average the computational results obtained suggest that the local optimisation algorithm is far superior than the reduction method in terms of accuracy but is more time consuming.

Mao et al. (1997) approached the DCMWST problem from the point of parallel algorithm. They implemented two parallel approximation algorithms known as TC (tree construction) and IR (iterative refinement). The TC algorithm, based on a one-time tree construction approach, employs a minimum spanning tree algorithm to obtain the required DCST by checking the degree constraint at every step of the tree construction. The IR algorithm, based on an iterative refinement approach, first constructs a MWST and then iteratively employs an edge exchange procedure until the required DCST is obtained. At each it-

eration the edges that cause violations of the degree constraints are penalised by assigning larger weights to them hence discouraged from appearing in subsequent iterations. Their computational results, based on matrix size of 500 to 3500, showed that although the IR algorithm is faster, the TC algorithm consistently produced DCSTs of smaller weights. Moreover, when the degree restriction, r, is 2 and 3 the TC algorithm appears to compute feasible DCST more often than the IR algorithm.

Zhou and Gen (1997) developed generic algorithms for the DCMWST problem. They adopted the Prüfer number (Prüfer 1918) to encode a spanning tree (Zhou and Gen 1998). Prüfer number made use of the graphical enumeration of Cayley's theorem that there are $n^{(n-2)}$ distinct labelled trees on a complete graph with n vertices. Prüfer (1918) provided a constructive proof of Cayley's theorem by establishing a one-to-one correspondence between such trees and the set of all strings of $(n-2)$ digits. Hence, given any permutation of the Prüfer number of $(n-2)$ digits, we are always assured that it can be decoded into a feasible tree T. The algorithm of Zhou and Gen uses various degree modification methods. This generic approach was tested on five randomly generated graphs of order 10 to 50. The edge weights are integer values randomly generated and uniformly distributed over [10,100]. The computational results obtained showed that the generic approach is within less than 8% from the lower bounds. Hence, this verifies the effectiveness of generic algorithms and Prüfer number as a tree encoding procedure for solving the DCMWST problem.

Krishnamoorthy et al. (1998) presented three heuristics based on generic algorithms for the DCMWST problem. They too made use of the Prüfer number representation of a tree. In their first heuristic GA-F (generic algorithm based on feasible solutions) they always work with spanning trees having degree-

feasible solutions. In their second heuristic GA-P (generic algorithm based on penalty-alteration method) they relaxed the degree constraint and violations are added to the objective function via a penalty multiplier. Their third heuristic **PSS** (problem space search) combines a simple constructive heuristic with a genetic algorithm. Its main idea is: instead of searching through the space of feasible solutions directly, a base heuristic is used as a mapping function between a space of perturbations and the solution space. Hence, it is sufficient to just search the perturbation space to find a good solution to the original problem.

Krishnamoorthy et al. (1998) tested their heuristics against a simulated annealing method, their Branch and Bound algorithm and Volgenant's Lagrangean approach (1989). For the two dimensional euclidean problems they tested graphs of order 30, 50, 70 and 100. For the higher dimensional euclidean problems graphs of order 30, 50 and 70 were tested and for the artificially difficult problems constructed using non-euclidean graphs, graphs of order 15, 20, 25 and 30 were tested. For each order they tested 10 problems. Their computational findings showed that the PSS method performed best over all the problems while the simulated annealing performs averagely. Their new generic heuristics however did not perform that well when compared to Volgenant's approach (1989) both in terms of CPU time and quality of the gap from optimality.

For this DCMWST problem we develop four new heuristics (in Section 3.1 and in Caccetta et. al 1998) which we compare with three heuristics from literature (modified Kruskal's 1956, modified Prim's 1957, Savelsbergh and Volgenant's 1985). Our heuristics utilise different edge exchange procedures depending on edge weights and certain critical values. We implemented our heuristics in the C programming language on a SUN SPARC 2 workstation

operating at 28.5 MIPS. We simulated 2500 random graphs. The values of the density p used were: 0.05, 0.25, 0.50, 0.75 and 1.00. We ranged the number of vertices in the graphs, n, from 50 to 500 with stepsize of 50. We tested all the seven heuristics with r ranging from 3 to 10 and presented the computational findings in Section 3.2. From the results obtained our best heuristic always outperformed the heuristic of Savelsbergh and Volgenant (1985) by as much as 4% (the heuristic of Savelsbergh and Volgenant is considered the best heuristic known). We further compared our heuristics with an exact Branch and Cut method by Caccetta and Hill (1997).

Caccetta and Hill (1997) developed a Branch and Cut method utilising a version of the Lagrangean relaxation method of Volgenant (1989) in a preprocessing procedure. The major features of their method are: a depth first search procedure was used, two search procedures for finding violating constraints were utilised, the lower bound for the relaxed LP was revised whenever a "quality" violating cut was found within 6 successive attempts and a branching variable identification procedure based on the work of Applegate et al. (1995) was described. They tested their method on graphs with number of vertices, n, ranging from 100 to 1000 in increments of 100. When tested against the exact Branch and Cut method of Caccetta and Hill (1997) our heuristics were always within 4% from the optimal with the average CPU time at least 15 times faster than that of the exact method. The best heuristic produced results within 0.06% from the optimum and the fastest heuristic was 82 times faster then the exact method.

- **The CMWST Problem**

The next problem of the Restricted Spanning Tree problems that we survey is the Capacitated Minimum Spanning Tree (CMST) problem. Esau and

Williams (1966) pioneered work on this problem by presenting a heuristic for constructing these multipoint networks. Their heuristic used the idea of swapping edges connected to the centre until the centre is finally linked to the required number of vertices. The edges incident to the centre are swapped in such a way that the edge's end-point that is not incident to the centre is now connected to its nearest neighbour. Proceeding in this manner the authors were able to construct communication networks where the distance between the terminals and the centre can be established. The authors, however, did not provide any computational testing of their heuristic.

Papadimitriou (1978) established the NP-completeness of the CMWST problem. He also proved that the Euclidean case of the CMWST problem is NP-complete. He presented the exact 'Sharma-El Bardai' algorithm that produced solutions with relative error almost certainly arbitrarily close to zero for Euclidean problems. This algorithm divides the vertices of the graphs (represented by polar coordinates) into sectors and uses the algorithm by Shamos and Hoey (1975) to construct the minimum spanning trees. In the final stage the algorithm adds edges joining the sink and the terminal in each sector that is closest to the tree. This algorithm produces a feasible solution in $O(n \log n)$ time where n is the number of vertices in the graph.

Chandy and Russell (1972) were the first to carry out the computational work for the CMST problem. They presented a Mixed Integer Programming formulation of the problem. They coded their heuristic based on Vogel's approximation that uses a 'trade-off' value which is the difference between the cost of the cheapest link connecting that terminal to any other terminal and the next terminal to the cheapest link. The authors tested their heuristic against two other heuristics. The first heuristic was the heuristic of Esau and Williams (1966) and the second was the heuristic of Martin (1967). Computational re-

sults on graphs of order 10 to 50 were presented. The heuristic was tested on graphs with the assumption that all lines of the network have the same capacity. They concluded that Esau and Williams' heuristic (1966) gave the best near-optimal solution with Martin's (1967) requiring less time.

Gavish (1982) pointed out that the CMWST problem is a simple extension of the Minimal Spanning Tree problem. The extra restriction is that each vertex now has a capacity and demand restriction and the problem can be reduced to a flow problem. He proposed a Benders Decomposition procedure that partitions the constraints and variables of the integer linear programming formulation into binary and continuous variables. He does this in order to obtain lower and upper bounds of the optimal solution. The procedure was tested on graphs with number of vertices, n, ranging from 6 to 12. Their results were disappointing with over 900 cuts generated for $n = 12$ without reaching optimality. The only useful result that came out of their findings is that Benders Decomposition can be used to identify valid cuts for a linear programming characterisation of the problem.

Kershenbaum (1974) presented an algorithm that is based on a modification of Kruskal's algorithm for solving the capacitated problem. The algorithm chooses edges according to the trade-off function (value found by subtracting a component weight from the edge weight). The complexity of the procedure is $O(m^2 + n^2 \times C_w)$ where n is the number of vertices, m is the number of edges and C_w is the complexity associated with calculating the component weights. The author showed that with the use of heaps the complexity of the algorithm can be reduced to $O(n \log n)$. He claimed that his algorithm is sufficiently fast even for very large problems however, he did not provide any computational results.

Kershenbaum and Boorstyn (1983) studied methods of evaluating the effectiveness of heuristics and Branch and Bound methods for the CMWST problem. They referred to the CMST problem as the Centralised Teleprocessing Network problem. They outlined the two approaches available for the Branch and Bound procedure, namely considering the subproblem with the least lower bound and using depth-first search. The authors also analysed an exact technique for node partitioning. This exact technique restricts the subtree a vertex is allowed into. In addition, an edge restriction technique that forces an edge to be either 'prohibited' or 'required' was outlined. Computational results based on Euclidean problems with up to 20 vertices were presented. They concluded that the bound yielded by the partitioning technique provided better bounds than that of the Minimum Spanning Tree-based technique (Kruskal's algorithm 1957 or Prim's algorithm 1957). Further, their analysis suggested that the performance of known heuristics deteriorated as the constraint tightness increased but improved with increased n.

Chandy and Lo (1973) presented an algorithm that uses a Branch and Bound procedure from the Travelling Salesman problem to solve the Capacitated Minimal Spanning Tree problem. Their branching procedure starts by finding a MWST to obtain the lower bound on the cost of the Capacitated Minimal Spanning Tree. The algorithm then branches from the root vertex to cluster the vertices and edges that are allowed in the tree. Edges that are prohibited in the tree (based on their previously determined results) are assigned a cost of infinity. Their algorithm was tested on random graphs of order 10, 20, 30 and 40. Comparison was carried out with Martin's heuristic (1967). The results showed that the exact method requires more time and is more sensitive to problem structure (number of vertices, capacity, edge weights). Although in 77% of the cases Martin's heuristic (1967) was not optimal the results were very close to the optimal. From their comparison, pragmatism dictates the use

of heuristics in preference to an optimal algorithm.

Gavish (1983) presented a new Linear Integer Programming formulation which leads to a Dantzig-Wolfe (1961) decomposition and a new Lagrangean relaxation procedure for the Capacitated Minimal Spanning Tree problem. The author analysed problems with asymmetric costs and called the problem the Capacitated Minimal Directed Tree problem. The aim of the Lagrangean relaxation is to obtain tighter lower bounds on the optimal solution. The Lagrangean relaxation procedure was tested on a set of 160 problems of order 20 to 70. Comparison was made to a Minimal Directed Tree Relaxation problem and a Degree Constrained Minimal Directed Tree problem. Their analysis showed that the Lagrangean relaxation generated tighter bounds in a shorter amount of computing time than other relaxations.

Malik and Yu (1993) presented a Branch and Bound procedure for the CMWST problem. They added a few tightening constraints in the Lagrangean relaxation to obtain tight bounds that can solve large-scale problems to optimality. Further, they presented a multilevel approach to reduce and manage large data. They gave two heuristics, a Prim-like heuristic and a Lagrangean heuristic, to calculate the upper bounds. A full description of their Branch and Bound procedure was presented detailing how optimality was achieved. They also described how the Lagrangean reduced costs throughout the branching process. They implemented their algorithm in the Pascal programming language and tested their method on Euclidean problems with up to 50 vertices. The authors made comparison of their results with that obtained by applying Chandy and Lo's Branch and Bound algorithm (1973). Their results showed that their algorithm provided a comparative advantage over previous research on this problem in that it provided tighter bounds.

Sharaiha et al. (1997) utilised a tabu search for the CMST problem. A tabu search is a local search heuristic that moves at each iteration from a solution to its best admissible neighbour, even if it causes the objective function to deteriorate. The heuristic presented starts with an initial feasible solution which is constructed as: the closest vertex u incident to the centre is selected as the root of the first main subtree. A shortest spanning tree is then constructed starting from u using Prim's algorithm (1957) until it is no longer possible to extend it without violating the capacity constraint. This procedure is repeated with the next closest vertex (not yet included) incident to the centre until all the vertices have been included in a main subtree. The heuristic uses a neighbouring structure based on cut and paste operations. The operations allow for easy reoptimisation of the spanning tree and for the current number of subtrees to vary dynamically in order to minimise the solution cost. The heuristic was coded in Fortran and tested on different sets of instances (unit demand, non-unit demand) with the number of vertices, n, ranging from 40 to 200. The heuristic was compared to Esau and Williams' heuristic (1966). The tabu search required between 50 to 500 CPU seconds, in contrast to Esau and Williams' heuristic (1966) with less than 1 CPU second. However, it outperformed Esau and Williams' heuristic (1966) 36 out of 45 times with an average gap of 2.2% from the lower bound. The authors claimed that this improvement in solution quality is well worth the extra computational time.

- **The $(1,k)$-tree and Leaf-counting Problems**

We next consider the $(1,k)$-tree problem. Recall that Douglas (1992) was interested in the complexity of the $(1,3)$-tree in planar graphs. He proved that:

> *Given a planar graph $G = (V,E)$ with maximum degree 3, it is NP-complete to decide if there exists a spanning tree T for G such*

that $deg(x,T) = 1$ *or* 3 *for all* $x \in V$.

He achieved this by modifying the proof of the planar Hamiltonian Circuit problem (Garey et al. 1976). The author extended his results to spanning trees with degrees from a fixed proper subset S of positive integers where $|S| \geq 2$. He established bounds for the number of degree one vertices in the $(1,3)$-tree. Further, he showed that the complexity of a connected dominating set in planar graphs with maximum degree 3 and cardinality $(\frac{n}{2} - 1)$ is also NP-complete. Douglas' work (1992) provided the motivation for our Chapter 4 where we examined some cases of the problem for non-planar graphs. Our results can also be found in Caccetta and Lam (1998).

We strengthen Douglas' results (Section 4.1) by proving the complexity of the $(1,3)$-tree problem for non-planar graphs. This complexity is established by converting the problem to its equivalent Hamiltonian Path problem. Similarly, we prove the NP-completeness for the $(1,4)$-tree problem in graphs with maximum degree 4. Moreover, we prove that finding a spanning tree T with a certain degree spectrum d where $d = (\delta = d_0, d_1, \ldots, d_m = k)$ is also NP-complete. We prove the following:

Given a graph $G = (V,E)$ *with degree spectrum* $d = (3,5)$, *it is NP-complete to decide if there exists a spanning* $(1,3,5)$*-tree* T *for* G.

Due to the fact that the number of degree one vertices is always of interest in any tree, we provide some simple counting to determine this value in the general $(1,k)$-tree. We also determine the number of degree k vertices in the $(1,k)$-tree. Further, we develop a theorem for the upper and lower bounds of the number of degree one vertices in the $(d_0 = 1, d_1, \ldots, d_m = k)$-trees. These bounds are:

$$\frac{(d_1-2)\nu+2+(d_2-d_1)(m-1)+\sum_{i=3}^{m}(d_i-d_2)}{(d_1-1)} \leq n_0$$
$$\leq \frac{(d_m-2)\nu+2-\sum_{i=1}^{m-1}(d_m-d_i)}{(d_m-1)}$$

where n_i is the number of vertices of degree d_i. We show with an example that these bounds are achievable.

We also analyse the computational aspects of the $(1,k)$-tree problem. We develop two fast and simple heuristics to find such a tree given any graph G. Our heuristics build up the $(1,k)$-tree sequentially (Section 4.2). Since a large portion of random graphs do not have a $(1,k)$-tree, we implement a random generator that randomly adds edges to an initial $(1,k)$-tree until the prescribed density p is achieved. It is on these graphs that our heuristics are tested. Our computational results (Section 4.2) show that our two heuristics managed to find a $(1,k)$-tree at least 70% of the time (our best heuristic has 90% success rate).

When analysing spanning trees with degree restrictions, one common interest that arises is to establish the bounds of the number of vertices of degree one. Galbiati et al. (1994) provided a short note to show that counting the maximum number of vertices of degree one (leaves) in spanning trees is an NP-complete problem. They mentioned different problems like the Minimum Bounded Dominating Set problem and the Maximum Leaves Spanning Tree problem. Griggs and Wu (1992) looked at spanning trees with minimum degree 4 or 5. They proved that for graphs with n vertices and minimum degree $\delta = 4$, the number of leaves in the tree is bounded from below by $(\frac{2}{5}n+\frac{8}{5})$ and when $\delta = 5$ the lower bound is $(\frac{1}{2}n+2)$. They provided examples to show that these bounds are indeed sharp.

Kleitman and West (1991) showed that every connected n-vertex graph of minimum degree at least 3 has spanning trees with at least $(\frac{n}{4}+2)$ leaves. Further, when the minimum degree is at least 4, the spanning tree has at least $(\frac{2n+8}{5})$ leaves. They generalised the bound to the case when the minimum degree in the n-vertex graph is at least k. They established the upper bound of $(n-3\lfloor\frac{n}{k+1}\rfloor+2)$ for all k. They also proved that when k is sufficiently large there is an algorithm for constructing spanning trees with at least $(1-\frac{b\ ln\ k}{k})n$ leaves in any graph of minimum degree k where b is any constant greater than 2.5

Fernandes and Gouveia (1998) were interested in finding, given a graph $G=(V,E)$ with cost c_{ij} for each edge $(i,j)\subseteq E$ and a natural number k $(1\leq k\leq n)$, a minimal spanning tree with k leaves. For their convenience the authors converted the graphs to directed graphs with two arcs of opposite direction replacing an edge. They presented two formulations and a local search heuristic for the problem. Their first formulation is from a reformulation (Gouveia 1995) of a single-commodity flow based formulation for the Directed Minimal Spanning Tree problem. The second formulation adds variables and constraints to the formulation of the Directed Minimal Spanning Tree problem in order to obtain valid formulation of the problem. Their paper gives detail as to how the relaxation schemes (Lagrangean relaxations) were employed for both formulations.

In order to obtain upper bounds on the optimal solutions Fernandes and Gouveia (1998) employed a 2-phase transformation local search heuristic. In the first phase a spanning tree with at least k leaves is obtained. If such solution satisfies the leaf constraint the heuristic terminates. Otherwise, the second phase that consists of a local exchange procedure is carried out. In each iteration of the second phase the current tree solution is transformed into another

tree solution with smaller number of leaves until eventually a tree with k leaves is obtained.

Fernandes and Gouveia (1998) implemented their methods in the FORTRAN programming language on a 486/66Mhz PC. They carried out computational testing on a class of symmetric graphs with up to 40 vertices and with k up to 35. The cost matrix was taken as the integer part of the euclidean distance between the coordinates of $(n+1)$ points in a square grid of dimension 100 by 100. The authors compared the best lower bounds (cost of the corresponding minimal spanning tree) with the best upper bounds produced by their methods. For the most part the bounds given by the Lagrangean schemes were not better than the minimal spanning tree bounds. It is only for small k that the Lagrangean schemes produced improvements on the minimal spanning tree bounds. However, when the subgradient optimisation procedure were carried out to 100 or 500 iterations their results were always within 1% of the optimum. The authors concluded with suggestions of finding tighter constraints for the problem.

- **Some Other Tree Problems**

Ali and Huang (1991) considered the Balanced Spanning Tree problem (BST). The aim of this problem is to find a spanning tree such that the number of vertices in any subtree created when disconnected from a chosen vertex (the root vertex) does not differ by more than one. The authors made use of clique and augmented-clique inequalities to represent the balance constraints of the BST. Basis-change dual-ascent techniques were developed to maximise the Lagrangean duals obtained with the balanced constraints dualised. The relaxations were used to obtain lower bounds on the optimal primal value and to generate upper bounds using a dual-information-based heuristic. Their algo-

rithm was tested on graphs of order 20, 50 and 100. The computational results obtained indicated that their optimal and suboptimal solutions provided very tight bounds and on average, in 180 to 300 iterations for 100-vertex problems.

Fürer and Raghavachari (1992) looked at ways of obtaining a spanning tree for a graph $G = (V,E)$ with n vertices whose maximal degree in the tree is the smallest among all spanning trees of G. The authors reduced the problem to that of computing a sequence of maximal matchings on some auxiliary graphs. They developed an iterative polynomial time approximation algorithm to tackle this NP-complete problem. They extended their algorithm to solve Steiner problems (problems where their set of vertices is a subset of V) and directed graphs. The main idea of their algorithm is to add an edge to the tree and move along the created cycle to delete an edge that is beside a vertex whose degree is greater than the required. They applied this idea recursively until the desired spanning tree is obtained. They utilised the local optimality property that is: no edge should be able to reduce the maximal degree of vertices on the basic cycle induced by that edge. Their article concluded with some open problems on the relationship among problems which can be approximated to within one from their optimal.

The Minimum Weight Spanning Tree with Bounded Diameter problem (MWSTBD) was analysed by Achuthan et al. (1994). The authors analysed this NP-complete problem for diameter four and above. They presented a Mixed Integer Linear Programming formulation for the problem. The authors developed and implemented a number of Branch and Bound algorithms based on various branching, bounding and search strategies. They tested their algorithms on graphs with number of vertices, n, ranging from 10 to 50 and D from 4 to 8. From their computational results they concluded that their new branching rule is superior when the diameter restriction is tight and the

superiority of their algorithm increases as the number of vertices in the graph increases.

Abdalla et al. (1997) approached the MWSTBD using parallel algorithms. They employed an iterative refinement approach as well as a greedy procedure. In their iterative approach they start by obtaining a MWST and for the edges on the path that violates the diameter restriction they assign a penalty to these edges. The authors developed a penalty function which encodes how many edges to penalise, which edges to penalise, and what the penalty amount must be. Obviously, edges in the center of a long path are ideal candidates to penalise since it will break the path into short subpaths and hence reduce the diameter. In their second approach, the greedy procedure, the authors employed a modified Prim's algorithm. At each iteration an edge is added to the current tree if its inclusion does not violate the diameter restriction. If it does then it is put aside for further iterations. The algorithm terminates when a diameter-constrained minimum spanning tree is obtained or the constructed tree contains less than $(n-1)$ edges. Abdalla et al. (1997) tested their algorithms on complete graphs with matrix size ranging from 10 to 3000. Their computational results suggest that although the iterative approach is fast it is only effective if the number of vertices, n, of the graphs increases with increased diameter bound. This is so since the approach of penalising edges close to the center of the trees does not produce reduction in the diameter of the spanning tree during later iterations when the spanning trees contain a large number of paths of almost equal length. On the contrary, the modified Prim's algorithm seems very effective even for small diameter bound irrespectively of n. Its only drawback is that it becomes prohibitive to obtain the best solution for large n, since spanning trees have to be constructed starting with each of the n vertices as the root vertex.

The problem of constructing a Minimum-Cost Spanning Tree problem satisfying a resource constraint and a flow requirement (MCSTRF) was studied by Shogan (1983). The importance of this problem rests with its wide application in many real-life problems. For example, reconstructing a network after a natural disaster such as earthquake, the distribution of energy networks (electrical or natural gas networks), communication networks (telephone, telegraph), transportation networks (highway and railroad network), and water networks (networks for the distribution of potable water or the removal of sewage). Shogan (1983) exploited a Branch and Bound algorithm based on Lagrangean relaxation to solve this problem. The algorithm utilises a feasibility check in the fathoming procedure and iteratively updates the flow on the arcs until the capacity on the corresponding arcs is satisfied. This algorithm was coded in the FORTRAN computer language. Shogan (1983) tested his methods on graphs of 50 vertices with 5 resource constraints as well as on graphs of 20 vertices with 3 resource constraints of unit demands. His results showed that the algorithm has the potential of solving large problems that will produce results within 5% from the optimal. The author discussed the modifications necessary to enable the algorithm to solve the more general problem like when the capacity constraint on every arc need not be satisfied. He also extended the problem to the case of multiple arcs and multiple sources.

Camerini et al. (1983) provided the complexity for finding **multi-constrained spanning trees**. They focused on undirected and unweighted graphs. They started by presenting the one-constrained spanning tree problem with cost functions of distance, diameter, height and a few other parameters. They commented on how some of these problems are equivalent to other NP-complete problems. For example, under certain conditions the sum of the distance problem is equivalent to the Hamiltonian Path problem. The two-constrained spanning tree problem that consisted of combinations of their various one-

constrained functions was also discussed. The authors concluded with the Multi-Constrained Spanning Tree problem. Any problem is considered multi-constrained if it has three or more cost functions. An example of such an instance is provided.

2.4 Graph Partitioning

The Graph Partitioning problem is concerned with partitioning the vertices of an edge weighted graph into b disjoint sets such that the sum of the edges that have end-points in different sets is minimised. The problem can be stated as:

> *Given an edge weighted graph $G = (V, E)$, partition the vertices into b equal sets such that the cost of the edges with end-points in different sets is minimum.*

This problem has wide applications in VLSI networks, circuit design, group technology classifications, processor allocation, pattern recognition and parallel computing. A typical example of the problem was presented by Feo and Khellaf (1990) regarding the routing states of VLSI design. The objective is to minimise the total wire length as well as the area required by the chip. By modelling each chip as a vertex and the electrical connections between them as edges, the aim is to cluster highly connected chips to facilitate the routing of the electrical connection and reduce the layout area.

This Graph Partitioning problem was proven to be NP-complete by Garey and Johnson (1979) for fixed $b \geq 3$. For $b = 2$ the problem can be solved in polynomial time using matchings since it is equivalent to the weighted matching problem for bipartite graphs. Due to its practical importance in network analysis, researchers opted for heuristics. Kernighan and Lin (1970) pioneered

the work by providing a classic algorithm for partitioning vertices of a graph into two sets. Their algorithm recursively calculates for each edge of the cut the gains that would result from interchanging ends of the edge. The edge that produces the maximum gain is selected and then the partition is updated. This algorithm is well illustrated in Chapter 5 of this thesis. Kernighan and Lin (1970) mentioned ways of speeding up the algorithm like first sorting the edges with end-points in different sets. Their algorithm requires at most $O(n^3)$ time (Gilbert and Zmijewski 1987) with n representing the number of vertices in the graph. The authors stressed that their algorithm can be repeatedly applied to solve m-way partitioning problems as well as sets of unequal sizes. However, from our analysis we found that when we further extend Kernighan and Lin's heuristic (1970) to sets of four or eight its quality deteriorates significantly.

Feo and Khellaf (1990) addressed the lack of practical heuristics for partitioning vertices into b sets when b is large. They presented various heuristics to partition vertices of a graph into b sets with 4 vertices in each set. Their heuristics utilised matchings obtained using different initial steps like: choosing a vertex at random and selecting its incident edge with the maximum weight to be in the matching, finding a maximum weight perfect matching, or starting with a random matching. Once a matching is obtained its edges are contracted. Then the heuristics are reapplied. This way each edge corresponds to 4 vertices. The authors tested their heuristics on graphs with the number of vertices, n, varying from 12 to 20 with edge weights drawn uniformly from the interval (1,20). They compared their heuristics with a Branch and Bound procedure that employed a Lagrangean relaxation technique imbedded within an implicit enumeration scheme (Khellaf 1987). They claimed that almost all their heuristics produced solutions that are approximately 2% from the optimal.

Johnson et al. (1989) approached the Graph Partitioning problem by simulated annealing. They presented details of their implementation, describing a parameterised, generic annealing algorithm that called problem-specific subroutines, and hence, can be used in a variety of problem domains. The algorithm starts by reading in an instance and constructing an initial solution. Then it chooses a random neighbour S' of the current solution S and returns the difference of cost $c(S') - c(S)$. It replaces S by S' in the local memory and updates the data structures as appropriate. If S' is a feasible solution and $c(S')$ is better than c^* (a locally stored variable which was initially set to some trivial upper bound on the optimal feasible solution cost) it sets S^* (a locally stored 'champion' solution) to S'. Finally, it modifies the current solution S to obtain a feasible solution S'' ($S'' = S$ if S is already feasible). If $c(S'') \leq c^*$, the algorithm outputs S''; otherwise it outputs S^*.

Johnson et al. (1989) compared their computational results with those of Kernighan and Lin (1970) and those obtained from a local optimisation method. This local optimisation method is based on the same neighbourhood structure used in their annealing algorithm. Their results showed that the simulated annealing always outperformed the local optimisation method for sparse graphs and Kernighan and Lin's algorithm (1970) for larger graphs.

Johnson et al. (1989) also studied various optimisation parameters that affect the simulated annealing like the temperature and size factors. They investigated ways of improving their algorithms by modifying the generic structure of the algorithms. These included procedures like choosing moves according to random permutation, using better-than-random starting solution and using appropriate exponentiation. The article concluded with some observations as to how best to adapt the simulated annealing to other type of problems like the Graph Colouring problem, the Number Partitioning problem and the

Travelling Salesman problem.

Falkner et al. (1994) approached the Graph Partitioning problem in a different manner. They made use of eigenvalues to obtain upper and lower bounds for $b = 2$. They provided detail of their method which is a non-linear optimisation method that utilised the Bundle Trust (BT) method (Rendl and Wolkowicz 1990). They presented computational results on a variety of randomly generated graphs having various characteristics like sparse graphs, denser graphs, geometric graphs, large graphs and graphs obtained from real-world applications. Their findings showed that the eigenvalue approach for finding partitions to graphs consistently produced lower and upper bounds having a relative gap of 10%. Further, by taking advantage of the possible sparsity of the adjacency matrix they computed the upper bound efficiently, and this leads to good partitions which are a few percent from the optimum. The method also performed well on real-world problems that consisted of several connected components and on graphs having some underlying geometric structure. However, extending the method to $b = 3$ did not give as good a performance as for $b = 2$.

The Graph Partitioning problem has also attracted much interest in the area of parallel computing. This is so since parallel processing requires efficient use of distributed memory in such a way that the computational load is balanced across processors and the interprocessor communication is minimised. Hence, parallel computing resembles the Graph Partitioning problem. Hendrickson and Leland (1995a) provided a multilevel algorithm for partitioning graphs where the graphs are approximated by a series of increasingly smaller graphs. The algorithm generates a sequence of smaller graphs that approximate the original graph, partitions the smallest graph using a spectral method and then propagates the resulting partition back to the original graph while periodically performing a local refinement. The local refinement method is a variant

of Kernighan and Lin's heuristic (1970) which is good for finding local optimal solutions. They tested their algorithm on several large graphs ($n = 4720$ and $m = 13722$, $n = 1434374$ and $m = 409593$) corresponding to computational grids for large scientific computing problems where n and m represent the number of vertices and edges of the graph respectively. These graphs were divided through six levels of bisection producing 64 sets. The authors observed that their algorithm demonstrated excellent performance with relatively good run time. The only shortcoming is that the construction of a sequence of graphs requires the use of a lot of memory.

Hendrickson and Leland (1995b) presented an improved spectral graph partitioning algorithm for mapping parallel computations. The algorithm uses multiple eigenvectors to divide the problem into four or eight groups where the first eigenvector defines a surface that bisects the graph, the second represents an intersecting surface that bisects the first two pieces, and so on. By doing so the authors got around the look ahead problem associated with the bisection. Further, the use of multiple eigenvectors gave rise to substantial savings in the net cost of eigenvector calculations. The algorithm was shown to perform well for hypercube topology in the communication networks and can also be applied to other machine topologies like the d-dimensional meshes. The authors tested their algorithm on a variety of meshes and compared their results with other methods like Kernighan and Lin's heuristic (1970) and the recursive spectral bisection method by Simon (1991). Their findings showed that their improved spectral partitioning algorithm generated better partitions than its competitors, which are themselves considered quite good.

Gilbert and Zmijewski (1987) developed a parallel algorithm based on Kernighan and Lin's heuristic (1970) to solve the Graph Partitioning problem for a message-passing multiprocessor. They used machines that consisted of sev-

eral identical processors, each containing some local memory. These machines communicate by passing messages along some links. Their aim was to develop algorithms that do as much computation as possible locally with little communication among the processors. The authors used their parallel algorithms for finding small separators to solve systems of linear equations by computing the Cholesky factor of the lower triangular matrix of the matrix representing the graph. They showed that their parallel Kernighan-Lin algorithm takes $O(m \log n \log b)$ time (n and m are the number of vertices and edges in the graph respectively and b is the number of partitions), ignoring the time for message passing. The authors compared three algorithms which were: the narrow and wide algorithms based on their parallel Kernighan-Lin algorithm, and a simple sequential strategy. For their tested problems, the narrow algorithm required 32% to 66% fewer messages passing than the sequential strategy, while the wide algorithm required 14% to 34% fewer messages passing than the narrow. The authors commented that in general, parallel algorithm performs better if it first decomposes the problem at hand into parts with high locality that require low communication overhead.

Savage and Wloka (1991) investigated parallelism in graph partitioning. The authors presented a new parallel heuristic which they named *Mob* that used a local search procedure to bisect the vertices of a graph. Vertices in Mob are swapped according to some increase in gains. These are vertices in each set whose exchange results in the largest individual change in the bisection width (number of edges linking the two sets). Testings were carried out on Mob, Kernighan and Lin's heuristic (1970) and simulated annealing (Johnson et al. 1989) on some large random graphs of 1 million vertices and 2 million edges with degree $d_G(i)$ ranging from 3 to 16. Their findings showed that Mob produced results that are consistently good in a short time and the quality improved with increased running time. For graphs with more than two million

edges Mob generated partitions that were within 2% of the best ever found in 9 minutes. Further, the running time grows logarithmically if the number of processors is proportional to the size of the graph.

Kučera (1995) studied the Graph Colouring problem and the Cut into Equal Parts problem. Both these problems are NP-complete. The Graph Colouring problem was looked at in view of testing for $b=O(\sqrt{\frac{n}{logn}})$, whether two vertices of a b-colourable graph can be b-coloured by the same colour in time $O(b\ log\ n)$ per pair of vertices with $O(b^4\ log^3 n)$-time preprocessing in such a way that for almost all b-colourable graphs the solution is correct for all pairs of vertices. The author managed to obtain a sublinear (with respect to the number of edges) expected time algorithm for b-colouring of b-colourable graphs (assuming the uniform input distribution). Similarly, for the Cut into Equal Parts problem the author tested for $c \leq (\frac{1}{8} - \varepsilon)n^2$, $\varepsilon > 0$ is a constant, in graph G having a cut vertex set in two equal sets with at most c cross-edges. He wanted to know if there exist two vertices from the same class of some c-cut in time $O(log\ n)$ per vertex with $O(log^3 n)$-time preprocessing such that for almost all graphs having a c-cut the answer is true for all pairs of vertices. The author commented that his approach can be used to solve other Graph Partitioning problems like the Independent Subset problem.

Arbib (1987/88) studied the Uniform Partition and the Simple Max Partition problems of the Graph Partitioning problem. The Uniform Partition problem is concerned with partitioning the vertices of G into two sets ($b = 2$) such the number of edges (on an unweighted graph) in the cut set that is to be removed is minimised whilst making sure that the two subsets of vertices have the same cardinality. The Simple Max Partition problem is similar to the Uniform Partition problem with the exception that one set always has at least the same number of vertices as the other set. It is the Uniform Partition prob-

lem with the cardinality constraint relaxed. Arbib (1987/88) outlined that the two problems are NP-complete. He derived some properties of the feasibility and optimality regions of the two problems. He presented the notion and properties of the cut-symmetric graphs and proposed a polynomial heuristic procedure utilising the class of line graphs to solve both the problems. The article concluded with results claiming that the two problems can also be solved in polynomial time on multi line graphs.

For the Graph Partitioning problem we are particularly interested in methods of partitioning the vertices into b user-specified sets. We describe and implement, in Section 5.1, sixteen different heuristics and compare them to Kernighan and Lin's heuristic (1970) and Feo and Khellaf's heuristic (1990). We extended the Kernighan and Lin's heuristic to partition vertices into four and eight sets and Feo and Khellaf's heuristic to partition vertices into sets with eight vertices in each set. Our heuristics build up the sets sequentially using methods like: sorting and choosing the edges in increasing weights, checking if the edge weights satisfy a certain prescribed critical value and using matchings. We implemented all the heuristics in the C programming language on a SUN SPRAC 2 workstation operating at 28.5 MIPS. We tested all the heuristics and recorded their average edge weights of the edges with end-points in the same sets. We also recorded the average speed (in terms of CPU time (secs)) of all the heuristics. Our computational work indicates that the two best performing heuristics are those that select edges according to a prescribed critical value. Further, our results showed that although Kernighan and Lin's heuristic (1970) is very efficient for bisecting graphs its quality deteriorates when applied to four and eight sets. This is also the case with Feo and Khellaf's heuristic (1990) when further applied to solve problems with eight vertices in each set. In all cases our heuristics required little CPU times (secs).

Chapter 3

Degree Constrained Spanning Tree

In this chapter we consider the problem of finding, in a given weighted graph G, a minimum weight spanning connected subgraph T satisfying the requirement that the degrees of the vertices are bounded above. Here the edge weights represent costs and thus are non-negative. Our T must be a tree. Recall that this problem is NP-complete except for the case when all but one of the vertices have $(n-1)$ as their bounded degree.

In Section 3.1 we consider the case when each vertex of the graph has the same degree restriction. We present seven simple heuristics to solve the problem. Computational results are presented in Section 3.2. A comparative analysis is carried out based upon 2500 random test problems ranging from 50 to 500 vertices. The heuristics are also compared to an exact method based on Branch and Cut. Generalisations of the problem when the degree constraint takes on values in the range $[2, n-1]$ is also tested (Section 3.3). Section 3.4 considers the case when the degree restriction is only on one of the vertices in the graph. Heuristics are also developed and a comparative analysis with Gabow's (1978) polynomial time algorithm is carried out. The Degree Constraint One Vertex problem is then extended to the case when the restriction is on two of the

vertices in the graph. Eight simple approaches are developed from the heuristics of the One Vertex problem. Their computational results are presented in Section 3.5. A Mixed Integer Programming Formulation of the problem is also presented.

3.1 Maximum Degree Problem

Here we are concerned with the following problem which we refer to as the **Degree Constrained Minimum Weight Spanning Tree (DCMWST)-problem:**

> *Given an edge weighted graph* $G = (V, E)$ *with* $V = \{1, 2, ..., n\}$ *and positive integers* $r_1, r_2, ..., r_n$, *find a minimum weight spanning tree* T *of* G *such that the degree,* $d_T(i)$, *in* T *of vertex* i *is at most* r_i, $1 \leq i \leq n$.

The DCMWST-problem is easily formulated as a Mixed Integer Linear Programming (MILP) problem. One formulation is:

$$Minimise \sum_i \sum_j c_{ij} x_{ij} \tag{3.1}$$

subject to

$$1 \leq \sum_{j=1, i \neq j}^{n} x_{ij} \leq r_i, \qquad i = 1, \ldots, n \tag{3.2}$$

$$\sum_{i,j} x_{ij} = n - 1 \tag{3.3}$$

$$\sum_{i,j \in V'} x_{ij} \leq |V'| - 1, \qquad \forall \quad \emptyset \neq V' \subseteq V \tag{3.4}$$

$$x_{ij} = 0 \, or \, 1, \qquad 1 \leq i \neq j \leq n. \tag{3.5}$$

The variable x_{ij} has a value 1 if edge (i, j) is included in the tree and 0 otherwise. Constraints (3.2) specify the degree restrictions on the vertices. Constraint (3.3) ensures that $(n-1)$ edges are selected. Constraints (3.4) eliminate cycles guaranteeing that the selected edges form a spanning tree.

In this section we present seven heuristics, four of which are new, for the DCMWST-problem with $r_1 = r_2 = ... = r_n = r$. These heuristics and their results can also be found in Caccetta et al. (1997). The first two heuristics make use of Prim's and Kruskal's algorithms with the additional step of testing for degree violations along the way. **Prim's algorithm** (1957) repeatedly selects the cheapest edge inside the component to build up the tree. **Kruskal's algorithm** (1956), on the other hand, repeatedly selects the cheapest edge to build up components and eventually (for feasible problems) these components are connected and a tree is obtained.

We first describe a procedure that is common to all our heuristics. Observe that given any graph G we can easily specify those edges that must be in every spanning tree of G. More precisely, let T be a spanning tree of G. Then every edge of G that is incident to a vertex of degree one must be in T. Let E_0 be the set of all such edges and consider the subgraph $G - E_0$. Clearly every edge of $G - E_0$ that is incident to a vertex of degree one in $G - E_0$ must be in T. Now proceeding in this way we can identify all the edges E^* of G that must be in every spanning tree. It may be possible that the forest formed by the subgraph $G[E^*]$ induced by E^* violates the degree restrictions in which case G does not have a DCST and the problem is infeasible. In all our heuristics

we first identify E^* and test $G[E^*]$ for feasibility; we refer to this step as **preprocessing**. If $G[E^*]$ is feasible we include the edges of E^* in our tree and basically apply the heuristic to the edges of $G - E^*$.

We begin with a modification of Prim's algorithm which was presented by Gavish (1982) and also by Narula and Ho (1980). The modification is simply that the edges are included in the tree only if they satisfy the additional degree restriction. In our procedure we have added the preprocessing step. We implemented this heuristic with the aim of comparing our other heuristics.

Heuristic 3.1.1:

Step 1. Apply the preprocessing procedure and initialise the graph if infeasibility is not detected.

Step 2. Sort the edges by increasing weight.

Step 3. Initialise by selecting the cheapest edge.

Step 4. Repeat

Select the cheapest available incident edge whose inclusion will not violate the degree constraint

until

a DCST is found or the edge set is exhausted.

Note that if $E^* \neq \emptyset$ we can chose our initial edge (Step 3) as the cheapest edge incident to an edge in E^*.

Our second heuristic is a similar modification of Kruskal's algorithm. More specifically, we have:

Heuristic 3.1.2:

Step 1. Apply the preprocessing procedure and initialise the graph if infeasibility is not detected.

Step 2. Sort the edges by increasing weight.

Step 3. Repeat

Select the cheapest available edge. If both its vertices have degree at most (r-1) then

include this edge in the tree

else

discard this edge.

until

a DCST is found or all the edges have been exhausted.

The above two heuristics build up a DCST. Our next heuristic starts with a MWST and then deletes appropriate edges incident to vertices that violate the degree restriction. The result is a spanning forest F with every vertex satisfying the degree requirement. Now F is a spanning subgraph of $G - E'$, where E' denotes the set of deleted edges. Heuristic 3.1.2 is then applied to $G - E'$ in an effort to extend F to a DCST of G. More precisely:

Heuristic 3.1.3:

Step 1. Apply the preprocessing procedure and initialise the graph if infeasibility is not detected.

Step 2. Sort the edges by increasing weight.

Step 3. Find a MWST using Kruskal's algorithm.

Step 4. Repeat

Delete edges incident to the vertex with the largest degree

violation starting with the most expensive edge until the degree constraint is satisfied

until

no more vertices violate the degree constraint. The resulting subgraph is a spanning forest F.

Step 5 Extend F using Heuristic 3.1.2.

Our next two heuristics select the edges forming the spanning tree according to an edge weight criterion. The criterion is that an edge is selected if its weight is less than a prescribed critical value, CV. We use a critical value of $\alpha(\frac{LB}{n-1})$ where LB is the lower bound (the weight of the MWST) and α is a specified parameter (we use CV $= 0.60(\frac{LB}{n-1})$ in our computations). The following heuristic incorporates this idea in Heuristic 3.1.1.

Heuristic 3.1.4:

Step 1. Apply the preprocessing procedure. If infeasibility is not detected initialise the graph and set CV $= \alpha(\frac{LB}{n-1})$.

Step 2. Sort the edges by increasing weight.

Step 3. Initialise by selecting the cheapest edge.

Step 4. Repeat

Select the cheapest available incident edge whose weight is less than the critical value and whose inclusion will not violate the degree constraint.

If no such edge is found select the next cheapest available edge and repeat Step 4 to build up another component.

until

a DCST is found or the edge set is exhausted. If a DCST is not found go to Step 5.

Step 5. If the components are not connected, connect them up by applying Heuristic 3.1.2 on the remaining unselected edges.

Our next heuristic uses the idea of a depth first search. We first add the edges E^* that must be in the tree. From the remaining edges we choose the cheapest available edge and construct a component containing this edge and satisfying the degree restriction by examining vertices in a depth first search manner. As with Heuristic 3.1.4, our criterion for selecting edges in the component is based on the edge weight.

Heuristic 3.1.5:

Step 1. Apply the preprocessing procedure. If infeasibility is not detected initialise the graph and set CV $= \alpha(\frac{LB}{n-1})$.

Step 2. Sort the remaining edges with $e_1 \leq e_2 \leq ... \leq e_m$.

Step 3. For $i = 1$ to m do

if e_i is unavailable then

discard this edge

else

let $e_i = (u, v)$. If $d(u) \geq d(v)$ then

push v on top of the stack followed by u

else

push u on top of the stack followed by v.

While stack $\neq \emptyset$ do

pop the element from top of the stack. Try to include at most $(r-1)$ of its cheapest available edges in the tree only if the weight of the edge is less than the critical value (this inclusion could merge with F).

For the newly selected vertex/vertices push the vertex of

lower degree on top of the stack followed by the vertex of higher degree.

When the stack is empty and a DCST is found, stop. If not, go to Step 4.

Step 4. If the components are not connected, connect them up by applying Heuristic 3.1.2 on the remaining unselected edges.

Remark 1: In the computational experimentation for Heuristics 3.1.4 and 3.1.5 we used critical value $\alpha(\frac{LB}{n-1})$ with α ranging from 0.01 to 2.00. We found $\alpha = 0.60$ to be the most satisfactory.

Remark 2: If in Step 3 of Heuristic 3.1.5 the stack is built with the lower degree vertex on top, the same results are obtained.

The heuristic of Savelsbergh and Volgenant (1985) is claimed to be the best available. It is based on the idea by Prim (1957) and Dijkstra (1957) that states: *any component can be connected to a nearest neighbour of the component.* More precisely the heuristic is:

Heuristic 3.1.6:

Step 1. Initialise the graph.

Step 2. Construct a MWST using the Prim's and Dijkstra's idea.

Step 3. Delete the edges that connect two vertices that violate the degree constraint.

Step 4. For any vertex i whose degree is greater than its degree constraint r delete its incident edges until its degree is r.

Step 5. Connect up the components as cheaply as possible using Heuristic

3.1.2.

The next heuristic is a modification of Savelsbergh and Volgenant's AH heuristic (1985). The heuristic first generates an upper bound (UB) using Savelsbergh and Volgenant's AH heuristic (1985) and then performs an edge exchange analysis until every vertex satisfies the degree constraint. The heuristic is presented as follows:

Heuristic 3.1.7:

Step 1. Choose a vertex i whose degree d_i is greater or the same as degree constraint r_i and label this the root vertex.

Step 2. Determine the level structure of the graph with the root vertex i in level 0.

Step 3. Choose a vertex q with $d_q > r_q$ that is the furthest from the root vertex. If no such vertex is found go to Step 7.

Step 4. For each edge e incident to q calculate the minimum cost of deleting e and reconnecting the graph T by the addition of an edge $e' = (j, k)$ such that vertices j and k satisfy the degree restriction in the resulting tree $T' = T - e + e'$. This cost is determined by using the procedure of Volgenant and Jonker (1983) where e' is calculated as the edges in the cutset (after deleting edge e) that gives rise to the minimum cost when included in T to produce T'. Observe that some of the information maybe useable from previous iterations however, caution must be taken when this information identifies an edge e' with the degree of one of its end-points j or k equal to the prescribed upper bound. In this case instead of recalculating the information we ignore this edge exchange by assigning a large cost.

Step 5. Let the cheapest edge exchange result in a tree $T^* = T' - (q, l) + e'$. If the weight of T^* is greater than or equal to the upper bound then stop; otherwise perform the edge exchange.

Step 6. If the level of vertex l is lower than that of q set $i = q$; otherwise retain the same root i. Return to Step 2.

Step 7. If the current feasible DCST has a weight less than the upper bound UB, set UB as this weight. Exit.

3.2 Computational Results

The seven heuristics presented in Section 3.1 were implemented in the C programming language on a SUN SPARC 2 workstation operating at 28.5 MIPS. We tested the heuristics on 2500 simulated problems generated as follows:

- The number of vertices ranged from 50 to 500 with stepsize of 50.
- For each order, we generated 50 connected random graphs with edge probability p; we used p values of 0.05, 0.25, 0.50, 0.75 and 1.00. For a given p the edge (i, j) is selected if the random number chosen from the unit is less than p. Note that the expected number of edges in the graph is $\binom{n}{2} p$. Disconnected graphs are rejected.
- The edge weights are real numbers uniformly distributed in the range $[1, 100]$ rounded off to two decimal places.
- The common upper bound r on the vertex degree takes on the values $3, 4, \ldots, 10$.

Our results are presented in Tables 3.1 to 3.5; we only give results for n =100, 200, 300, 400 and 500. Our tables record the statistic $\frac{H-LB}{LB}$ where H is the average weight of the DCST obtained from applying the Heuristic H and LB

is the average lower bound obtained from the average weight of the MWST. In addition, we record the average CPU time (secs) of the 50 problems.

p	r	$\frac{H1-LB}{LB}$	$\frac{H2-LB}{LB}$	$\frac{H3-LB}{LB}$	$\frac{H4-LB}{LB}$	$\frac{H5-LB}{LB}$	$\frac{H6-LB}{LB}$	$\frac{H7-LB}{LB}$	Ave CPU Time (secs)						
									H1	H2	H3	H4	H5	H6	H7
0.05	3	0.101209	0.095619	0.088098	0.087630	0.092226	0.119979	0.079674	0.0334	0.0160	0.0444	0.0214	0.0302	0.0312	0.1618
	4	0.020909	0.019532	0.018084	0.018398	0.019678	0.030403	0.012201	0.0314	0.0168	0.0268	0.0210	0.0300	0.0228	0.1106
	5	0.004401	0.004401	0.003558	0.003558	0.004324	0.007686	0.002425	0.0300	0.0158	0.0200	0.0212	0.0306	0.0190	0.0612
	6	0.001552	0.001552	0.001184	0.001184	0.001630	0.002526	0.000342	0.0304	0.0168	0.0176	0.0212	0.0322	0.0176	0.0462
	7	0.000088	0.000088	0.000088	0.000088	0.000236	0.000315	0.000005	0.0302	0.0164	0.0166	0.0198	0.0296	0.0174	0.0378
	8	0.000000	0.000000	0.000000	0.000000	0.000140	0.000000	0.000000	0.0310	0.0156	0.0170	0.0210	0.0306	0.0168	0.0358
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0312	0.0170	0.0176	0.0208	0.0306	0.0170	0.0352
0.25	3	0.110785	0.099351	0.093230	0.099018	0.098630	0.101828	0.064394	0.1048	0.0852	0.1434	0.0918	0.1464	0.1228	0.3740
	4	0.021215	0.019519	0.019501	0.019519	0.019598	0.022193	0.008991	0.1056	0.0884	0.1052	0.0952	0.1472	0.1016	0.3180
	5	0.004943	0.004943	0.004857	0.004943	0.005007	0.005331	0.001500	0.1054	0.0878	0.0910	0.0914	0.1476	0.0934	0.2702
	6	0.000961	0.000961	0.000961	0.000961	0.001024	0.001728	0.000229	0.1048	0.0874	0.0884	0.0902	0.1480	0.0912	0.2490
	7	0.000038	0.000038	0.000038	0.000038	0.000101	0.000279	0.000038	0.1048	0.0880	0.0898	0.0930	0.1460	0.0914	0.2422
	8	0.000000	0.000000	0.000000	0.000000	0.000063	0.000000	0.000000	0.1040	0.0868	0.0880	0.0910	0.1464	0.0904	0.2408
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.1044	0.0882	0.0860	0.0930	0.1446	0.0900	0.2390
0.50	3	0.093677	0.083250	0.080496	0.083217	0.083854	0.083380	0.053323	0.2048	0.1870	0.2986	0.1974	0.3558	0.2628	0.6826
	4	0.019000	0.017188	0.016450	0.017188	0.017197	0.018528	0.007702	0.2050	0.1882	0.2282	0.1992	0.3602	0.2234	0.6210
	5	0.003031	0.003015	0.003015	0.003015	0.002913	0.004326	0.001241	0.2050	0.1870	0.2022	0.1974	0.3594	0.2038	0.5774
	6	0.000481	0.000481	0.000481	0.000481	0.000513	0.000748	0.000174	0.2020	0.1872	0.1934	0.1984	0.3574	0.2002	0.5520
	7	0.000181	0.000181	0.000181	0.000181	0.000213	0.000166	0.000034	0.2044	0.1894	0.1912	0.1976	0.3584	0.1998	0.5466
	8	0.000000	0.000000	0.000000	0.000000	0.000032	0.000000	0.000000	0.2060	0.1880	0.1914	0.2010	0.3572	0.1986	0.5528
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.2038	0.1882	0.1922	0.1986	0.3584	0.2000	0.5530
0.75	3	0.082743	0.074886	0.072130	0.074828	0.074828	0.074372	0.043974	0.3146	0.2944	0.4574	0.3070	0.5768	0.4114	0.9932
	4	0.016157	0.015212	0.014876	0.015212	0.015212	0.015457	0.006237	0.3126	0.2952	0.3524	0.3088	0.5796	0.3482	0.9412
	5	0.002226	0.002226	0.002092	0.002226	0.002226	0.002697	0.001076	0.3126	0.2952	0.3112	0.3088	0.5812	0.3208	0.8867
	6	0.000488	0.000488	0.000488	0.000488	0.000488	0.000757	0.000220	0.3126	0.2944	0.3036	0.3088	0.5848	0.3162	0.8502
	7	0.000264	0.000264	0.000264	0.000264	0.000264	0.000242	0.000061	0.3114	0.2958	0.3008	0.3108	0.5844	0.3148	0.8354
	8	0.000218	0.000218	0.000218	0.000218	0.000218	0.000046	0.000018	0.3134	0.2952	0.3022	0.3094	0.5858	0.3142	0.8402
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.3116	0.2950	0.3018	0.3116	0.5798	0.3136	0.8374
1.00	3	0.067249	0.060372	0.059118	0.060372	0.060372	0.065023	0.038591	0.4258	0.4092	0.6128	0.4238	0.7998	0.5702	1.2768
	4	0.012013	0.012009	0.011437	0.012009	0.012009	0.013012	0.005677	0.4262	0.4080	0.4788	0.4234	0.8046	0.4844	1.2168
	5	0.002587	0.002498	0.002498	0.002498	0.002498	0.002863	0.000994	0.4276	0.4092	0.4348	0.4258	0.7880	0.4534	1.1708
	6	0.000600	0.000600	0.000600	0.000600	0.000600	0.000336	0.000137	0.4250	0.4082	0.4208	0.4170	0.7936	0.4462	1.1460
	7	0.000103	0.000103	0.000103	0.000103	0.000103	0.000161	0.000016	0.4240	0.4068	0.4328	0.4224	0.7954	0.4462	1.1412
	8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.4232	0.4086	0.4184	0.4244	0.8018	0.4438	1.1342

Table 3.1: Results for n=100

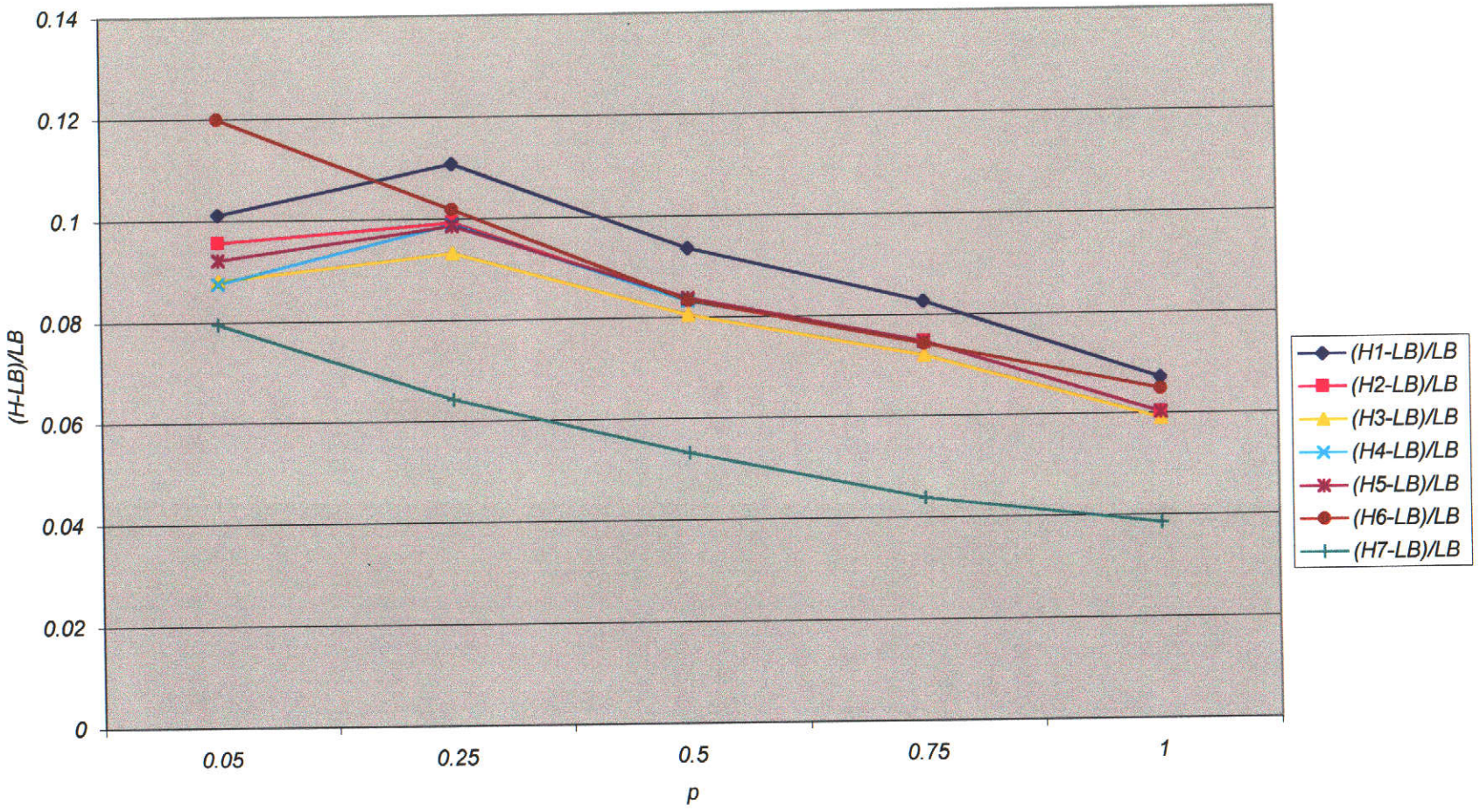


Table 3.1(b): Results for n = 100 with increasing p for r = 3

p	r	$\frac{H1-LB}{LB}$	$\frac{H2-LB}{LB}$	$\frac{H3-LB}{LB}$	$\frac{H4-LB}{LB}$	$\frac{H5-LB}{LB}$	$\frac{H6-LB}{LB}$	$\frac{H7-LB}{LB}$	Ave CPU Time (secs)						
									H1	H2	H3	H4	H5	H6	H7
0.05	3	0.130910	0.119432	0.112127	0.120142	0.120136	0.114957	0.076274	0.1596	0.0780	0.2868	0.0876	0.1494	0.1808	0.8742
	4	0.031925	0.031345	0.029754	0.031290	0.031292	0.029662	0.013183	0.1514	0.0788	0.1698	0.0886	0.1480	0.1264	0.6824
	5	0.008618	0.008544	0.008401	0.008544	0.008545	0.007536	0.002398	0.1490	0.0774	0.1094	0.0888	0.1502	0.0976	0.5166
	6	0.001231	0.001231	0.001231	0.001231	0.001233	0.001354	0.000415	0.1478	0.0764	0.0836	0.0890	0.1484	0.0820	0.4040
	7	0.000403	0.000403	0.000403	0.000403	0.000405	0.000358	0.000093	0.1472	0.0760	0.0794	0.0886	0.1508	0.0802	0.3704
	8	0.000014	0.000014	0.000014	0.000014	0.000016	0.000011	0.000011	0.1482	0.0766	0.0768	0.0888	0.1482	0.0790	0.3588
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.1488	0.0770	0.0762	0.0888	0.1496	0.0784	0.3582
0.25	3	0.098824	0.089048	0.084566	0.089048	0.089406	0.090388	0.057563	0.5164	0.4308	0.9110	0.4394	1.0796	0.7452	2.8606
	4	0.020073	0.019292	0.018374	0.019292	0.019296	0.021089	0.008262	0.5042	0.4314	0.6364	0.4356	1.0768	0.5610	2.6470
	5	0.003819	0.003748	0.003748	0.003748	0.003752	0.004472	0.001362	0.5032	0.4298	0.4894	0.4406	1.0908	0.4836	2.4768
	6	0.000878	0.000878	0.000878	0.000878	0.000883	0.001300	0.000316	0.5034	0.4302	0.4570	0.4400	1.0844	0.4608	2.3628
	7	0.000213	0.000213	0.000213	0.000213	0.000217	0.000498	0.000086	0.5040	0.4300	0.4396	0.4438	1.0788	0.4522	2.3070
	8	0.000097	0.000097	0.000097	0.000097	0.000101	0.000114	0.000043	0.5022	0.4288	0.4406	0.4454	1.0834	0.4508	2.2867
	9	0.000041	0.000041	0.000000	0.000041	0.000045	0.000088	0.000017	0.5040	0.4296	0.4415	0.4430	1.0850	0.4496	2.2899
	10	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.5056	0.4308	0.4399	0.4424	1.0938	0.4526	2.2874
0.50	3	0.078853	0.071924	0.066402	0.071924	0.071924	0.069358	0.044372	1.0032	0.9154	1.7572	0.9282	2.3122	1.4716	5.2212
	4	0.016102	0.015136	0.015024	0.015136	0.015136	0.017050	0.006853	0.9856	0.9254	1.2536	0.9284	2.3218	1.1576	5.0090
	5	0.003383	0.003385	0.003262	0.003385	0.003385	0.003926	0.001263	0.9880	0.9114	1.0094	0.9354	2.3178	1.0146	4.7993
	6	0.000887	0.000887	0.000887	0.000887	0.000887	0.000841	0.000348	0.9844	0.9160	0.9536	0.9158	2.3174	0.9804	4.6823
	7	0.000430	0.000430	0.000430	0.000430	0.000430	0.000411	0.000111	0.9882	0.9130	0.9368	0.9018	2.3194	0.9642	4.6335
	8	0.000136	0.000136	0.000136	0.000136	0.000136	0.000131	0.000034	0.9832	0.9170	0.9328	0.9076	2.3170	0.9618	4.6285
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.9876	0.9120	0.9256	0.9076	2.3162	0.9640	4.6144
0.75	3	0.064936	0.059345	0.055680	0.059345	0.059345	0.058264	0.036119	1.5194	1.4320	2.5610	1.4106	3.6274	2.2268	7.6392
	4	0.013281	0.012431	0.012009	0.012431	0.012431	0.013823	0.005720	1.5030	1.4334	1.8540	1.4108	3.6432	1.7596	7.3791
	5	0.002821	0.002821	0.002821	0.002821	0.002821	0.003303	0.000810	1.5018	1.4294	1.5488	1.4122	3.6562	1.5710	7.1722
	6	0.000376	0.000376	0.000376	0.000376	0.000376	0.000350	0.000109	1.5048	1.4268	1.4630	1.4154	3.6534	1.5102	7.0340
	7	0.000106	0.000106	0.000106	0.000106	0.000106	0.000030	0.000019	1.4986	1.4334	1.4550	1.4114	3.6570	1.4996	7.0039
	8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.5014	1.4338	1.4502	1.4108	3.6522	1.5030	6.9830
1.00	3	0.054966	0.049077	0.046193	0.049077	0.049077	0.045807	0.030236	2.0506	1.9514	3.4424	2.0036	2.3530	3.0100	9.8742
	4	0.011006	0.010443	0.009949	0.010443	0.010443	0.010730	0.004749	2.0364	1.9600	2.5130	2.0208	2.3798	2.3922	9.6356
	5	0.002312	0.002208	0.002207	0.002208	0.002208	0.002453	0.000840	2.0350	1.9558	2.1342	2.0314	2.3900	2.1496	9.4686
	6	0.000321	0.000321	0.000321	0.000321	0.000321	0.000441	0.000165	2.0312	1.9576	2.0198	2.0532	2.3890	2.0764	9.2832
	7	0.000124	0.000124	0.000124	0.000124	0.000124	0.000134	0.000040	2.0396	1.9596	1.9934	2.0370	2.3902	2.0630	9.2496
	8	0.000085	0.000085	0.000085	0.000085	0.000085	0.000033	0.000018	2.0314	1.9566	1.9866	2.0346	2.3888	2.0582	9.2524
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.0356	1.9600	1.9866	2.0310	2.3912	2.0578	9.3237

Table 3.2: Results for n=200

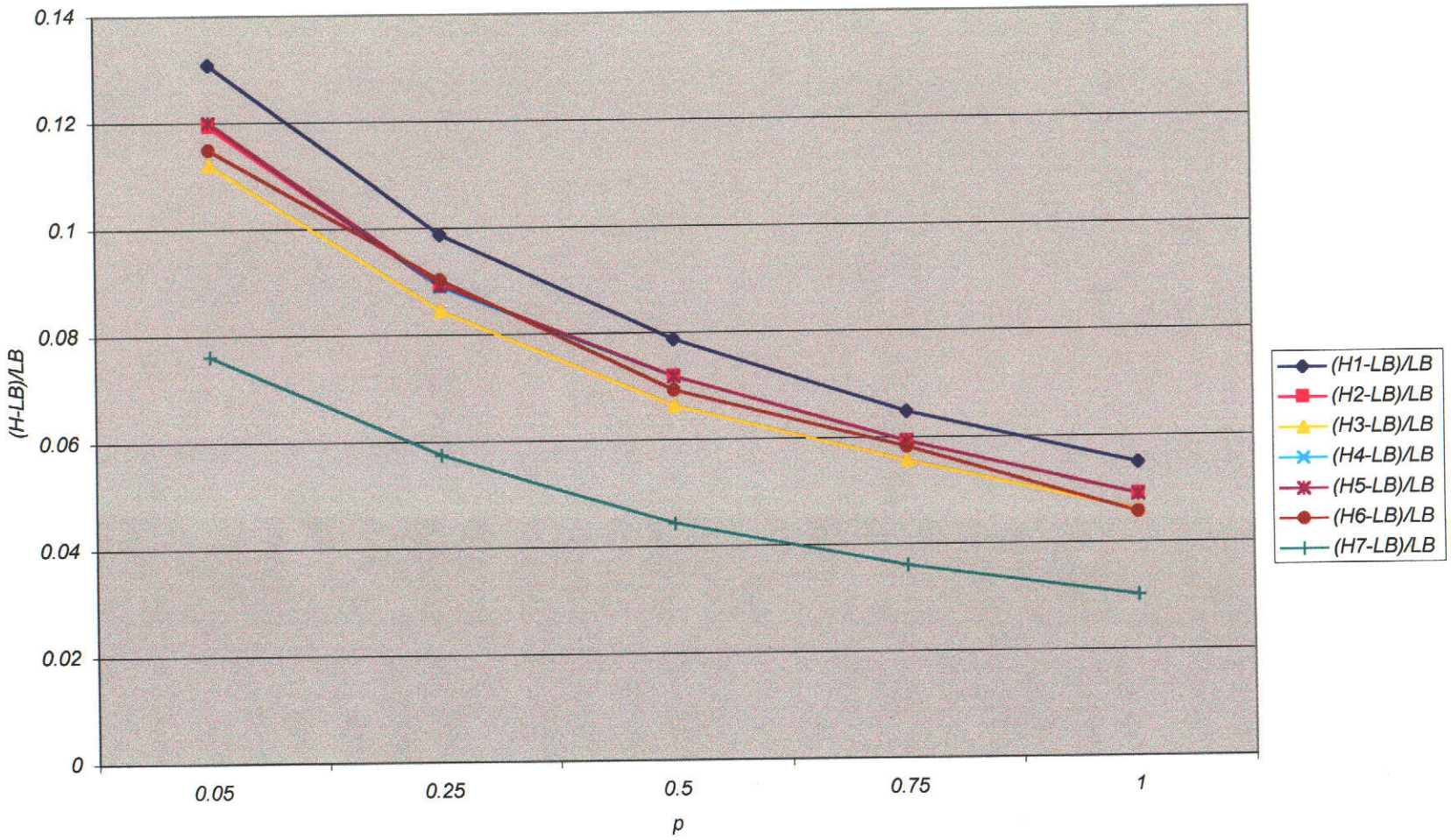


Table 3.2(b): Results for n = 200 with increasing p for r = 3

p	r	$\frac{H1-LB}{LB}$	$\frac{H2-LB}{LB}$	$\frac{H3-LB}{LB}$	$\frac{H4-LB}{LB}$	$\frac{H5-LB}{LB}$	$\frac{H6-LB}{LB}$	$\frac{H7-LB}{LB}$	Ave CPU Time (secs)						
									H1	H2	H3	H4	H5	H6	H7
0.05	3	0.122157	0.109573	0.102201	0.109939	0.109939	0.112913	0.070427	0.3870	0.1928	0.8138	0.2216	0.5246	0.5016	2.7420
	4	0.025085	0.023298	0.022636	0.023298	0.023246	0.027956	0.010630	0.3644	0.1910	0.4932	0.2210	0.5520	0.3466	2.2714
	5	0.005272	0.004949	0.004949	0.004949	0.004927	0.006854	0.001971	0.3590	0.1910	0.2892	0.2210	0.5520	0.2444	1.8550
	6	0.001610	0.001581	0.001581	0.001581	0.001599	0.001661	0.000424	0.3584	0.1938	0.2196	0.2220	0.5474	0.2104	1.6410
	7	0.000161	0.000161	0.000161	0.000161	0.000179	0.000127	0.000050	0.3598	0.1930	0.1986	0.2220	0.5494	0.2038	1.4429
	8	0.000036	0.000036	0.000036	0.000036	0.000054	0.000029	0.000016	0.3604	0.1936	0.1958	0.2220	0.5426	0.2026	1.4469
	9	0.000002	0.000002	0.000002	0.000002	0.000019	0.000027	0.000002	0.3596	0.1944	0.1952	0.2210	0.5416	0.2020	1.4368
	10	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.3602	0.1940	0.1946	0.2218	0.5312	0.2020	1.4404
0.25	3	0.092560	0.083521	0.077981	0.083541	0.083541	0.081910	0.049917	1.2360	1.0398	2.6322	1.0650	3.2756	2.0212	9.2476
	4	0.019575	0.018397	0.017820	0.018397	0.018397	0.021603	0.008530	1.2158	1.0390	1.7244	1.0614	3.2924	1.4919	8.7334
	5	0.004242	0.004134	0.004077	0.004134	0.004134	0.005235	0.001766	1.2068	1.0400	1.2493	1.0612	3.2968	1.2118	8.4619
	6	0.000953	0.000953	0.000953	0.000953	0.000953	0.001089	0.000363	1.2052	1.0403	1.1050	1.0636	3.3026	1.1206	8.1846
	7	0.000203	0.000203	0.000203	0.000203	0.000204	0.000329	0.000073	1.2075	1.0394	1.0659	1.0604	3.2994	1.1032	8.0076
	8	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	0.000006	1.2053	1.0402	1.0581	1.0604	3.2996	1.0990	7.8963
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.2066	1.0386	1.0565	1.0604	3.2996	1.0973	7.9732
0.50	3	0.065415	0.058908	0.055651	0.058908	0.058908	0.057939	0.037098	2.4159	2.2400	4.8838	2.2298	7.1202	3.9760	17.5084
	4	0.013933	0.013450	0.012888	0.013450	0.013450	0.014668	0.005791	2.4014	2.2402	3.3572	2.2254	7.1784	3.0082	16.7374
	5	0.002906	0.002904	0.002913	0.002904	0.002904	0.003545	0.000969	2.3896	2.2454	2.5548	2.2318	7.1916	2.5500	16.3236
	6	0.000373	0.000373	0.000373	0.000373	0.000373	0.000544	0.000112	2.3963	2.2424	2.3280	2.2250	7.1890	2.3778	15.9821
	7	0.000015	0.000015	0.000015	0.000015	0.000015	0.000013	0.000006	2.3952	2.2424	2.2786	2.2220	7.1914	2.3578	15.8575
	8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.3934	2.2418	2.2698	2.2228	7.1882	2.3490	15.8441
0.75	3	0.049534	0.049034	0.042229	0.044587	0.044587	0.044302	0.027541	3.6826	3.4782	7.3918	3.5834	3.7638	5.9754	25.0768
	4	0.009515	0.002005	0.008965	0.009034	0.009034	0.011369	0.004446	3.6508	3.4834	5.0196	3.6136	3.8032	4.5554	24.6270
	5	0.002026	0.000449	0.002009	0.002005	0.002005	0.002846	0.000833	3.6560	3.4832	3.9448	3.6258	3.8106	3.8808	24.5339
	6	0.000449	0.000074	0.000449	0.000449	0.000449	0.000617	0.000163	3.6490	3.4810	3.6270	3.6290	3.8134	3.6936	24.4153
	7	0.000074	0.000019	0.000074	0.000074	0.000074	0.000130	0.000033	3.6476	3.4794	3.5518	3.6280	3.7426	3.6508	24.3059
	8	0.000019	0.000000	0.000019	0.000019	0.000019	0.000031	0.000007	3.6528	3.4798	3.5314	3.6606	3.7382	3.6396	24.2979
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	3.6506	4.9249	3.5312	3.7242	3.7376	3.6394	24.3018
1.00	3	0.042074	0.036905	0.035191	0.036905	0.036905	0.036986	0.022789	4.9482	4.7605	9.4863	4.9338	5.0238	7.9870	33.1042
	4	0.008683	0.007903	0.007552	0.007903	0.007903	0.009091	0.003792	4.9335	4.7673	6.8040	4.9688	5.0714	6.2195	32.5958
	5	0.001556	0.001488	0.001480	0.001488	0.001488	0.001953	0.000648	4.9200	4.7675	5.3733	5.0020	5.0920	5.3395	32.2558
	6	0.000288	0.000288	0.000272	0.000288	0.000288	0.000466	0.000110	4.9215	4.7555	5.0018	4.9960	5.1004	5.0908	32.0683
	7	0.000009	0.000009	0.000009	0.000009	0.000009	0.000038	0.000004	4.9265	4.7530	4.8570	4.9548	5.0950	5.0083	31.8568
	8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.9325	4.7648	4.8378	5.0842	5.0938	4.9988	31.7952

Table 3.3: Results for n=300

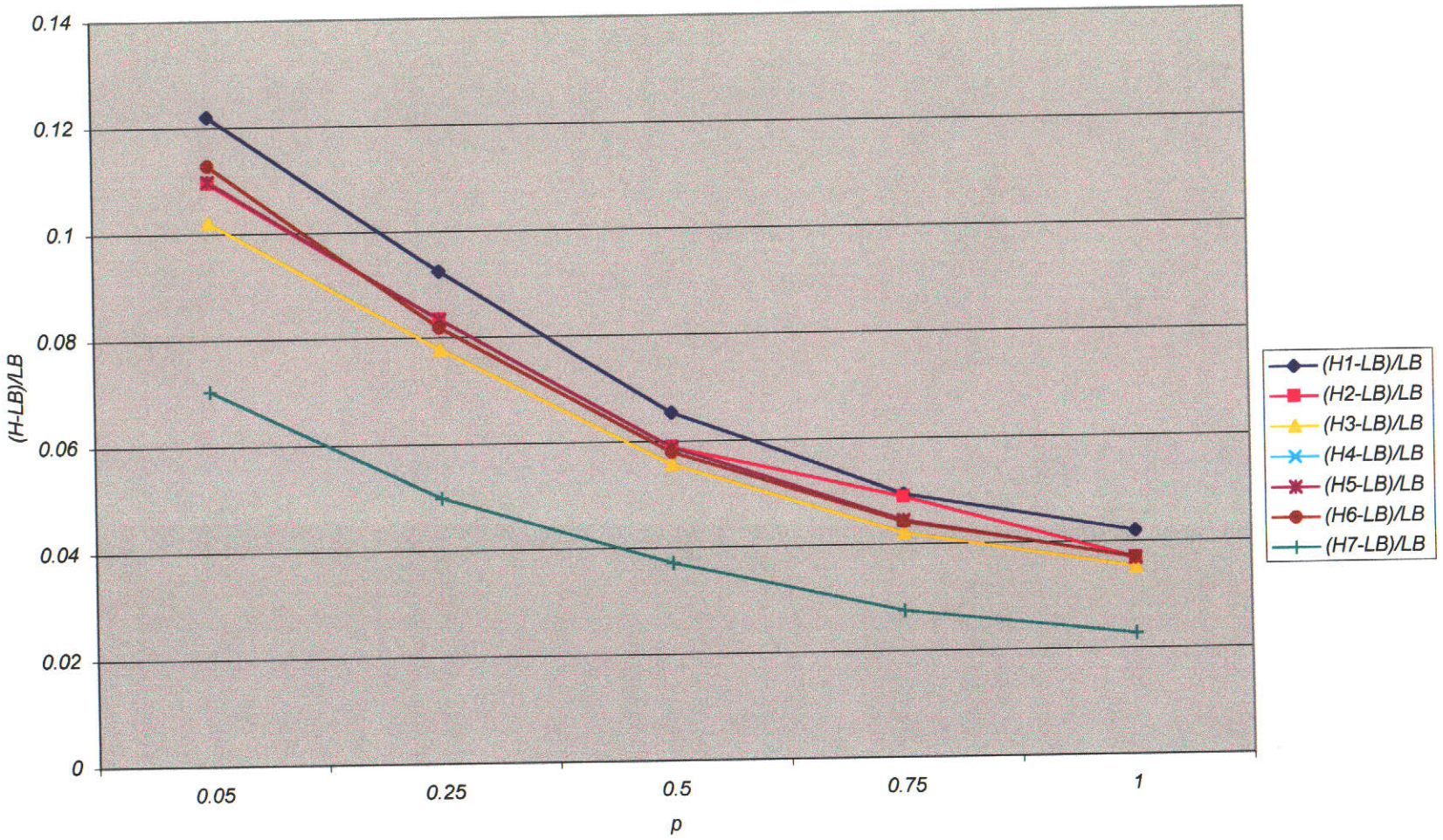


Table 3.3(b): Results for n = 300 with increasing p for r = 3

p	r	$\frac{H1-LB}{LB}$	$\frac{H2-LB}{LB}$	$\frac{H3-LB}{LB}$	$\frac{H4-LB}{LB}$	$\frac{H5-LB}{LB}$	$\frac{H6-LB}{LB}$	$\frac{H7-LB}{LB}$	Ave CPU Time (secs)						
									H1	H2	H3	H4	H5	H6	H7
0.05	3	0.129244	0.115520	0.108380	0.115677	0.115692	0.114035	0.073679	0.7444	0.3726	1.7994	0.4190	1.2502	1.0750	6.0018
	4	0.028846	0.027566	0.027384	0.027533	0.027583	0.029777	0.012706	0.7000	0.3706	1.1032	0.4228	1.2806	0.7190	5.1996
	5	0.007860	0.007730	0.007535	0.007763	0.007779	0.008570	0.002593	0.6904	0.3696	0.5848	0.4262	1.2550	0.4892	4.6396
	6	0.002158	0.002158	0.002158	0.002158	0.002174	0.002149	0.000572	0.6888	0.3694	0.4290	0.4178	1.2576	0.4170	4.0795
	7	0.000344	0.000344	0.000344	0.000344	0.000359	0.000475	0.000095	0.6870	0.3712	0.3846	0.4182	1.2640	0.3914	3.7626
	8	0.000000	0.000000	0.000000	0.000000	0.000015	0.000000	0.000000	0.6872	0.3708	0.3708	0.4442	1.2532	0.3834	3.6198
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.6858	0.3704	0.3716	0.4318	1.2490	0.3844	3.6192
0.25	3	0.078549	0.071193	0.066490	0.071260	0.071211	0.070130	0.042973	2.3682	1.9922	5.3692	2.0472	7.4678	4.1418	21.2310
	4	0.016493	0.015647	0.015242	0.015647	0.015648	0.017406	0.007150	2.3186	1.9930	3.5788	2.0474	7.4806	3.0094	20.4884
	5	0.003618	0.003531	0.003531	0.003531	0.003531	0.004253	0.001241	2.3252	1.9964	2.5016	1.9872	7.5778	2.3734	19.9210
	6	0.000821	0.000792	0.000792	0.000792	0.000792	0.000942	0.000181	2.3088	1.9930	2.1262	1.9924	7.5144	2.1418	19.3393
	7	0.000233	0.000233	0.000233	0.000233	0.000234	0.000140	0.000012	2.3092	1.9930	2.0436	1.9942	7.5312	2.1018	19.2440
	8	0.000041	0.000041	0.000041	0.000041	0.000042	0.000001	0.000000	2.3090	1.9922	2.0288	2.0056	7.4612	2.0898	19.3044
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.3158	1.9878	2.0168	1.9918	7.4788	2.0886	19.1177
0.50	3	0.062327	0.055913	0.052755	0.055913	0.055913	0.053754	0.030159	4.0106	3.6242	8.8168	4.3066	4.4276	7.0972	40.2834
	4	0.012995	0.012602	0.012557	0.012602	0.012602	0.013382	0.004801	3.9634	3.6232	5.9234	4.3476	4.4714	5.1380	39.6293
	5	0.002747	0.002659	0.002659	0.002659	0.002659	0.002995	0.000809	3.9460	3.6250	4.2504	4.3674	4.4832	4.1918	38.7110
	6	0.000453	0.000452	0.000452	0.000452	0.000452	0.000614	0.000122	3.9524	3.6246	3.7824	4.3670	4.4904	3.8710	37.9736
	7	0.000014	0.000014	0.000014	0.000014	0.000014	0.000052	0.000013	3.9456	3.6280	3.6770	4.3688	4.4952	3.8122	37.7641
	8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	3.9440	3.6178	3.6700	4.3688	4.4840	3.8076	38.2264
0.75	3	0.040256	0.037076	0.035695	0.037076	0.037076	0.035861	0.023622	6.9173	6.5285	15.5413	6.6785	6.8788	12.2297	60.0221
	4	0.008506	0.007767	0.007603	0.007767	0.007767	0.008973	0.003402	6.8710	6.5393	10.2348	6.7730	6.9557	8.9816	58.8573
	5	0.001781	0.001781	0.001726	0.001781	0.001781	0.001957	0.000631	6.8459	6.5331	7.7855	6.8013	7.0858	7.4397	58.4651
	6	0.000266	0.000266	0.000266	0.000266	0.000266	0.000331	0.000086	6.8635	6.5307	6.9057	6.9134	6.9894	6.9566	57.2784
	7	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	6.8535	6.5438	6.6482	6.7934	6.9765	6.8292	56.2318
1.00	3	0.035593	0.033333	0.031805	0.033333	0.033333	0.030335	0.019462	9.3049	8.9101	20.1288	9.1246	9.3785	15.8613	78.0078
	4	0.004313	0.006927	0.006564	0.006927	0.006927	0.007071	0.003052	9.2415	8.9171	13.5720	9.2097	9.4367	11.6677	77.1859
	5	0.001380	0.001372	0.001367	0.001372	0.001372	0.001610	0.000488	9.2359	8.8995	10.3722	9.3548	9.4869	10.0284	77.5356
	6	0.000276	0.000276	0.000276	0.000276	0.000276	0.000367	0.000097	9.2480	8.9125	9.3640	9.3678	9.5028	9.4906	77.5978
	7	0.000074	0.000074	0.000074	0.000074	0.000074	0.000079	0.000012	9.2474	8.9062	9.0961	9.3448	9.5061	9.3224	75.6462
	8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	9.2438	8.9197	9.0244	9.2931	9.4993	9.3044	76.2781

Table 3.4: Results for n=400

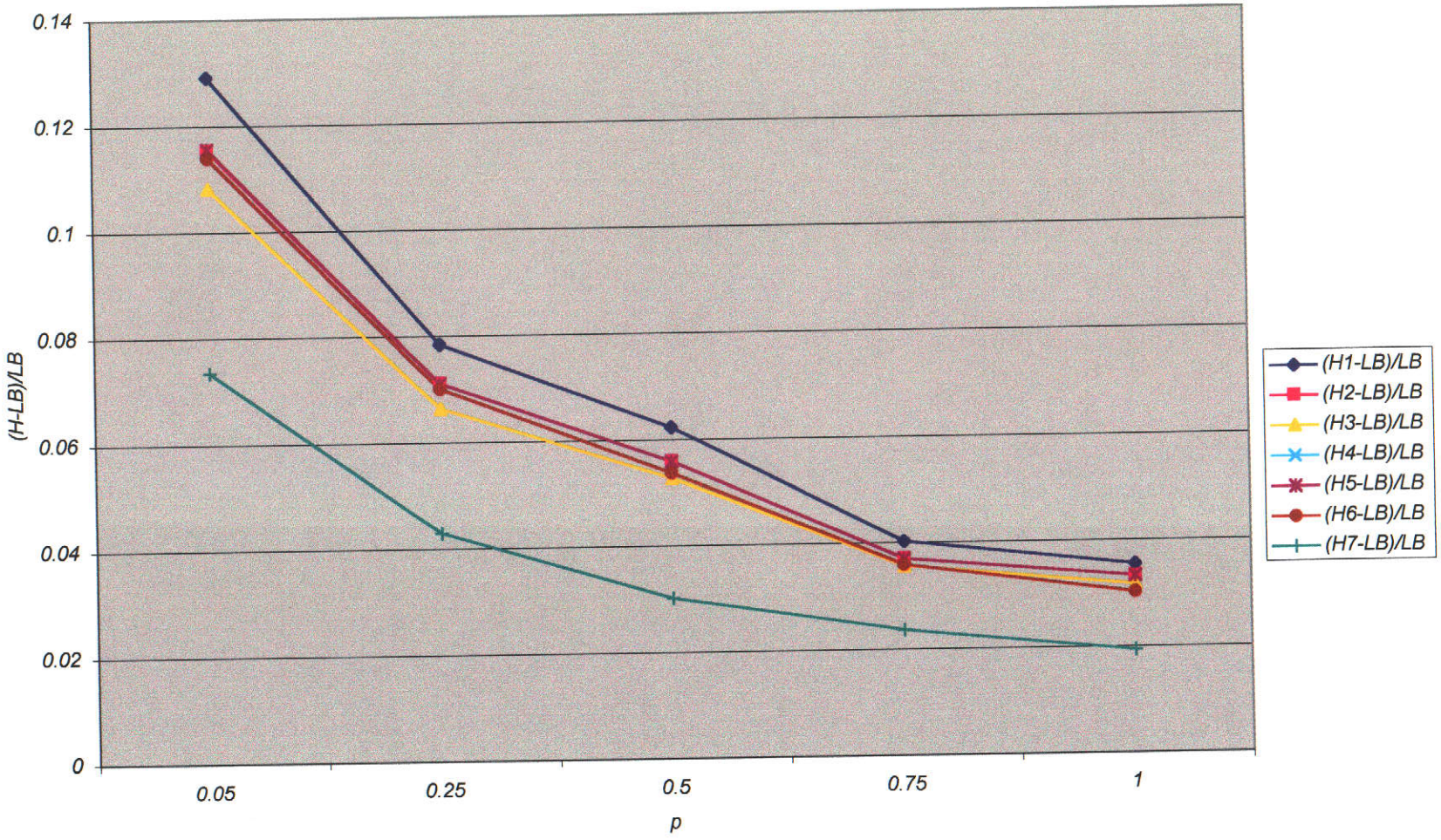


Table 3.4(b): Results for n = 400 with increasing p for r = 3

p	r	$\frac{H1-LB}{LB}$	$\frac{H2-LB}{LB}$	$\frac{H3-LB}{LB}$	$\frac{H4-LB}{LB}$	$\frac{H5-LB}{LB}$	$\frac{H6-LB}{LB}$	$\frac{H7-LB}{LB}$	Ave CPU Time (secs)						
									H1	H2	H3	H4	H5	H6	H7
0.05	3	0.123358	0.110521	0.103960	0.111004	0.111004	0.106996	0.068377	1.2062	0.6256	3.4748	0.6796	2.4012	1.9728	10.8540
	4	0.025801	0.024476	0.023787	0.024476	0.024476	0.026412	0.011336	1.1434	0.6222	2.1380	0.6824	2.4204	1.3078	9.8010
	5	0.006308	0.005985	0.005926	0.005985	0.005985	0.007549	0.002192	1.1272	0.6214	1.0936	0.6814	2.4140	0.8694	8.6268
	6	0.001371	0.001368	0.001368	0.001368	0.001368	0.001254	0.000345	1.1284	0.6236	0.7146	0.6780	2.4226	0.6800	7.7197
	7	0.000412	0.000370	0.000370	0.000370	0.000370	0.000245	0.000071	1.1322	0.6234	0.6526	0.6802	2.4156	0.6452	7.3770
	8	0.000151	0.000151	0.000151	0.000151	0.000151	0.000047	0.000011	1.1326	0.6240	0.6342	0.6824	2.4226	0.6410	7.2858
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.1303	0.6208	0.6315	0.6824	2.4196	0.6394	7.2664
0.25	3	0.072750	0.065097	0.061422	0.065097	0.065094	0.063593	0.041107	3.8423	3.2459	10.1962	3.2310	14.0806	7.3442	41.6014
	4	0.016333	0.015608	0.015410	0.015609	0.015610	0.017472	0.007070	3.8342	3.2484	6.8302	3.2200	14.1796	5.2138	40.6296
	5	0.003657	0.003601	0.003468	0.003601	0.003605	0.004293	0.001344	3.8184	3.2469	4.3974	3.2316	14.1964	4.0654	39.6881
	6	0.000787	0.000788	0.000788	0.000788	0.000789	0.000887	0.000233	3.8181	3.2560	3.5572	3.2242	14.2040	3.5576	38.4932
	7	0.000126	0.000126	0.000126	0.000126	0.000127	0.000149	0.000022	3.7616	3.2568	3.3434	3.2254	14.1962	3.4460	38.1441
	8	0.000009	0.000009	0.000009	0.000009	0.000010	0.000012	0.000002	3.7538	3.2574	3.3094	3.2274	14.2010	3.4164	37.4803
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	3.7529	3.2568	3.3064	3.2198	14.2016	3.4060	37.0953
0.50	3	0.048683	0.044607	0.043105	0.044607	0.044607	0.042006	0.026957	7.5388	6.9114	18.1476	7.0540	7.3076	14.1631	78.2863
	4	0.010487	0.009886	0.009629	0.009886	0.009886	0.010150	0.004072	7.4572	6.9128	11.8798	7.1083	7.4139	10.2061	76.9266
	5	0.002250	0.002123	0.002119	0.002123	0.002123	0.002714	0.000740	7.4439	6.9045	8.4724	7.1301	7.5090	8.1332	75.5932
	6	0.000674	0.000659	0.000659	0.000659	0.000659	0.000756	0.000117	7.4385	6.9014	7.3223	7.1723	7.5322	7.4438	74.8825
	7	0.000082	0.000082	0.000082	0.000082	0.000082	0.000119	0.000020	7.4437	6.9056	7.0654	7.1948	7.5158	7.2590	74.3539
	8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	7.4432	6.9012	6.9956	7.2055	7.5240	7.2026	75.7859
0.75	3	0.035167	0.031659	0.029338	0.031659	0.031659	0.030472	0.019071	11.2905	10.6680	26.0120	10.9340	11.1940	20.9580	116.5345
	4	0.007306	0.006853	0.006581	0.006853	0.006853	0.007695	0.003039	11.2240	10.6555	17.2975	11.1750	11.3020	15.2495	114.9648
	5	0.001723	0.001667	0.001667	0.001667	0.001923	0.001930	0.000571	11.2290	10.6570	12.6485	11.1505	11.3360	12.3525	114.1779
	6	0.000562	0.000532	0.000532	0.000532	0.000532	0.000663	0.000104	11.2055	10.6555	11.1810	11.1670	11.3540	11.4485	115.2408
	7	0.000112	0.000112	0.000112	0.000112	0.000112	0.000073	0.000014	11.1840	10.6690	10.8685	11.0515	11.3580	11.1620	113.6789
	8	0.000042	0.000042	0.000042	0.000042	0.000042	0.000027	0.000000	11.2055	10.6720	10.8595	11.0800	11.3575	11.1300	112.1871
	9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	11.1885	10.6600	10.8085	11.0280	11.3540	11.1075	112.7195
1.00	3	0.031926	0.028263	0.026082	0.028263	0.028263	0.025771	0.016453	15.1840	14.5300	36.5020	14.8120	15.1960	27.9240	154.7078
	4	0.007077	0.006464	0.006058	0.006464	0.006464	0.006302	0.002548	15.1040	14.5420	23.2740	15.0940	15.3320	20.3000	151.6363
	5	0.001863	0.001870	0.001450	0.001870	0.001870	0.001430	0.000529	15.0880	14.5520	17.4240	14.9760	15.3760	16.4500	150.5242
	6	0.000392	0.000392	0.000392	0.000392	0.000392	0.000215	0.000037	15.0640	14.5240	15.1680	15.0080	15.4060	15.3800	151.5906
	7	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	15.0480	14.5500	14.7000	15.0040	15.4200	15.1820	151.3906

Table 3.5: Results for n=500

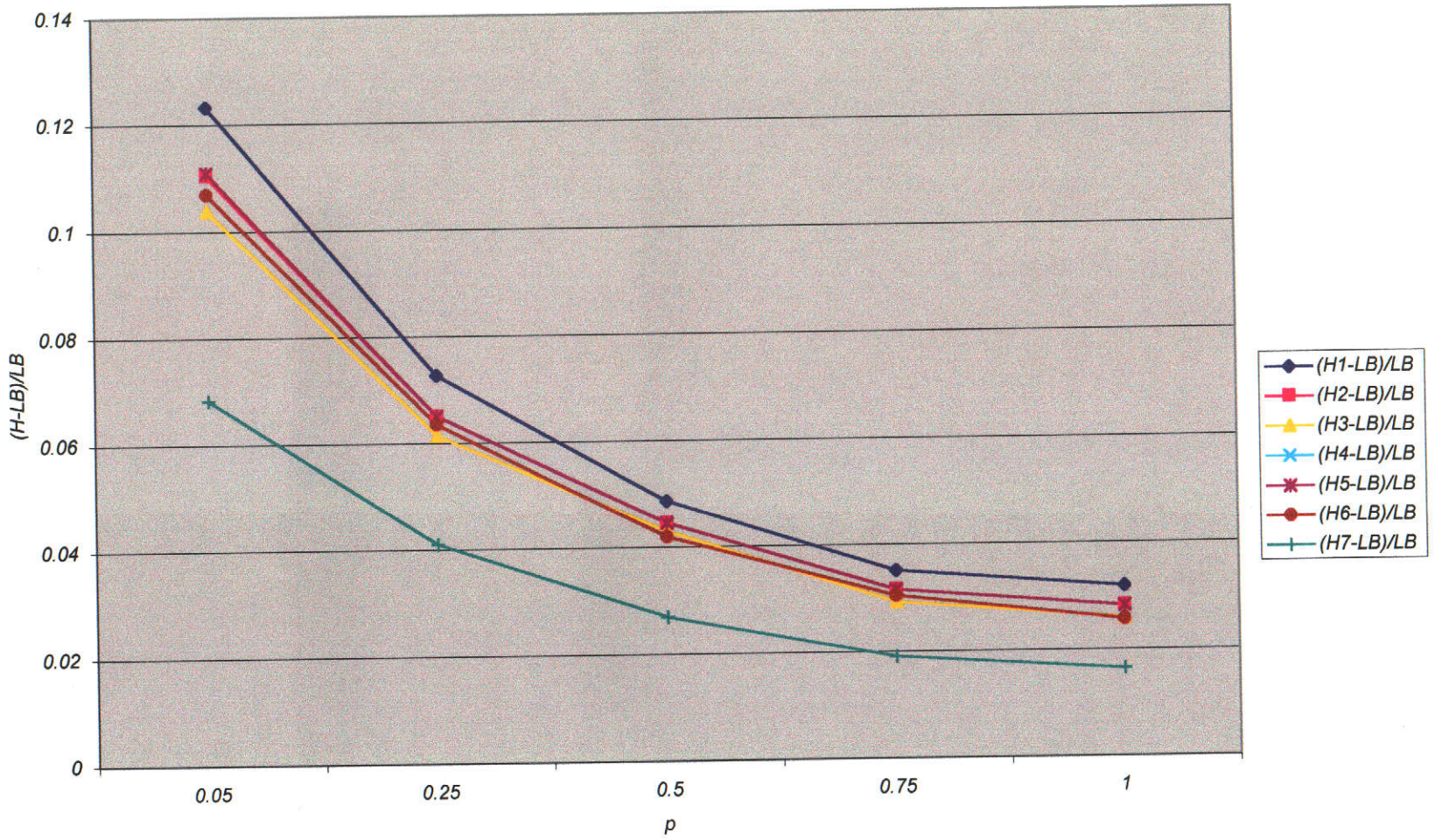


Table 3.5(b): Results for n = 500 with increasing p for r = 3

From the tables we observe that as the value of r and p increased all the heuristics generated DCSTs whose weights are closer to the LB value. Among the heuristics, Heuristic 3.1.7 produced results that are the closest to the LB value followed by Heuristic 3.1.3. This is so for all values of r. For small values of r and p Heuristic 3.1.4 and 3.1.5 gave results that are slightly worse than those of Heuristic 3.1.2. As r and p increased the two heuristics coincide with Heuristic 3.1.2. Heuristic 3.1.2 almost always produced results that are better than Heuristic 3.1.6 except for some occasional cases when $r = 3$. Of the 2500 test problems attempted Heuristic 3.1.1 to 3.1.7 generated a DCST in 98.88%, 98.96%, 97.96%, 97.72%, 97.68%, 99.12% and 98.63% of the cases respectively.

The fastest heuristic is Heuristic 3.1.2. This is followed by Heuristic 3.1.4 and Heuristic 3.1.1. The time for Heuristic 3.1.5 greatly depends on the edge weight in the graphs. We observe that as the density of the graph increases, for $n = 300, 400$ and 500, the time for density 0.50 is faster than the time for 0.25. This is so since the increase in density increases the number of edges in the graph with weights greater than the critical value. Hence, Heuristic 3.1.5 does not have to carry out the edge weight checking procedure so many times. Therefore its time improves with increasing p. As for Heuristic 3.1.3, it is fast for large values of r but slow for $r = 3$ and 4. As the value of r increases Heuristic 3.1.3 is faster than Heuristic 3.1.6. The time of Heuristic 3.1.7 improved with increasing r.

To provide a measure of the quality of the heuristics we tested them against the Branch and Cut method of Caccetta and Hill (1997). We present results for the case of $n = 500$ and $r = 3$. We have recorded the values of $\frac{H-O}{O}$ from our seven heuristics where O is the optimal weight produced by the exact method.

p	Optimal (O)	$\frac{H1-O}{O}$	$\frac{H2-O}{O}$	$\frac{H3-O}{O}$	$\frac{H4-O}{O}$	$\frac{H5-O}{O}$	$\frac{H6-O}{O}$	$\frac{H7-O}{O}$
0.05	3028.55536	0.064012	0.051854	0.045643	0.052311	0.052311	0.048545	0.011956
0.25	1011.02842	0.038267	0.030859	0.027301	0.030859	0.030855	0.029408	0.007467
0.50	770.27100	0.026604	0.022595	0.021135	0.022595	0.022595	0.020024	0.005290
0.75	669.89100	0.019198	0.015745	0.013459	0.015745	0.015745	0.014578	0.003354
1.00	628.30060	0.017732	0.014119	0.011966	0.014119	0.014119	0.011663	0.002474

Table 3.6 : Results of Heuristics compared with Exact values for n=500 and r=3

p	Optimal (O)	H1	H2	H3	H4	H5	H6	H7
0.05	378.4220	1.2062	0.6256	3.4748	0.6796	2.4012	1.9728	10.8540
0.25	689.7880	3.8423	3.2459	10.1962	3.2310	14.0806	7.3442	20.8007
0.50	869.0280	7.5388	6.9114	18.1476	7.0540	7.3076	14.1631	39.1432
0.75	952.7300	11.2905	10.6680	26.0120	10.9340	11.1940	20.9580	58.2673
1.00	1233.1600	15.1840	14.5300	36.5020	14.8120	15.1960	27.9240	77.3539

Table 3.7 : Average CPU Time (secs) for Exact and Heuristics for n=500 and r=3

From the tables we observe that on average our heuristics are 0.033163, 0.027034, 0.023901, 0.027126, 0.027125, 0.024844 and 0.006108 from the optimal respectively. Even though the Branch and Cut method produced exact solutions its computational time is at least 15 times higher than that of the heuristics.

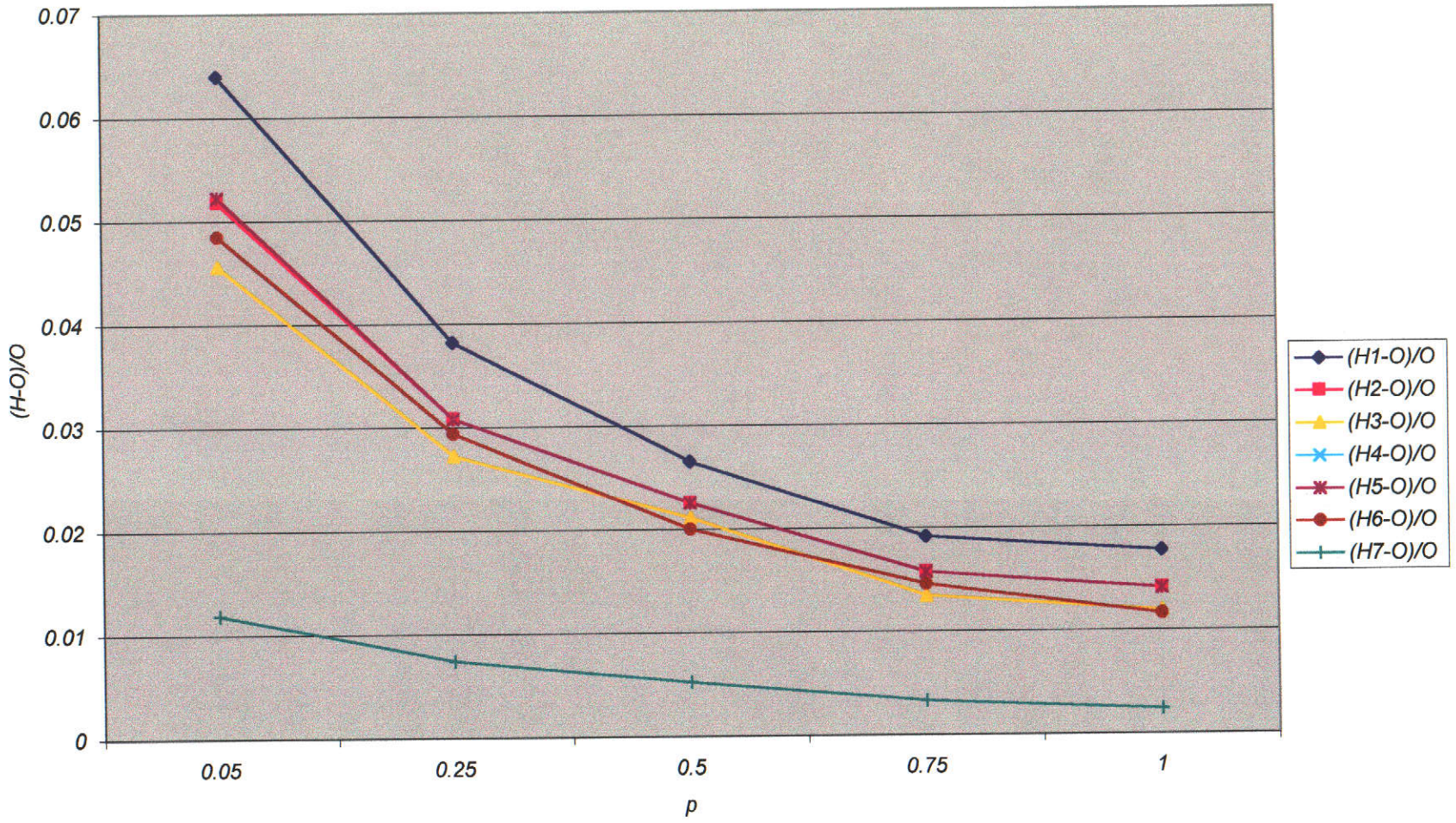


Table 3.6 (b): Results for heuristics compared with exact values for= 500 and r = 3

3.3 Generalisations

The seven heuristics presented in Section 3.1 can easily be extended to solve the more general problem when every vertex in the graph has its own degree constraint, that is each r_i takes on a value in the range $[2, n-1]$ and the r_i's may not necessarily be distinct. We tested four generalisations: GH2 is a generalisation of Heuristic 3.1.2, GH3 is a generalisation of Heuristic 3.1.3, GH6 a generalisation of Heuristic 3.1.6 and GH7 a generalisation of Heuristic 3.1.7. Table 3.8 gives the results of these four heuristics on the same set of graphs used in Section 3.2 with $2 \leq r_i \leq (n-1)$. Note that here the degree restriction on each vertex is chosen randomly from the set $[2, n-1]$. GH7 performed the best followed by GH3, GH2 and GH6. Among all the graphs tested only two cases resulted in a non-feasible solution.

n	p	$\frac{GH2-O}{O}$	$\frac{GH3-O}{O}$	$\frac{GH6-O}{O}$	$\frac{GH7-O}{O}$	Ave CPU Time (secs)			
						GH2	GH3	GH6	GH7
100	0.05	0.003878	0.003766	0.005158	0.001156	0.0160	0.0198	0.0190	0.0296
	0.25	0.002829	0.002731	0.003599	0.000262	0.0860	0.0864	0.0906	0.1434
	0.50	0.003069	0.002712	0.004420	0.000331	0.1824	0.1898	0.1956	0.3492
	0.75	0.001856	0.001817	0.002834	0.000375	0.2860	0.2966	0.3086	0.5536
	1.00	0.001969	0.001765	0.002947	0.000332	0.4076	0.4156	0.4330	0.7516
200	0.05	0.001412	0.001412	0.002893	0.000165	0.0770	0.0838	0.0844	0.2224
	0.25	0.000369	0.000375	0.000927	0.000045	0.4160	0.4220	0.4392	1.5394
	0.50	0.000956	0.000754	0.001075	0.000118	0.8904	0.9030	0.9382	3.1334
	0.75	0.001151	0.001109	0.001763	0.000318	1.3896	1.4284	1.4774	4.6512
	1.00	0.000786	0.000756	0.000834	0.000023	1.9034	1.9510	2.0202	6.1422
300	0.05	0.001411	0.001370	0.001582	0.000285	0.1898	0.2068	0.2090	0.9422
	0.25	0.000802	0.000787	0.000908	0.000080	1.0132	1.0604	1.0802	2.6396
	0.50	0.000395	0.000392	0.000627	0.000085	2.1790	2.2484	2.2962	5.2841
	0.75	0.000423	0.000298	0.000653	0.000057	3.3896	3.4530	3.5504	7.9847
	1.00	0.000470	0.000439	0.000499	0.000152	4.6508	4.8000	4.9205	10.6016
400	0.05	0.000937	0.000848	0.001223	0.000139	0.3608	0.3826	0.3912	2.5032
	0.25	0.000432	0.000417	0.000675	0.000056	1.9414	1.9864	2.0536	3.0754
	0.50	0.000579	0.000562	0.000677	0.000114	3.5448	3.6550	3.7396	6.3437
	0.75	0.000341	0.000337	0.000442	0.000082	6.3815	6.6208	6.7954	12.6447
	1.00	0.000214	0.000214	0.000261	0.000042	8.6914	8.8538	9.1023	18.9667
500	0.05	0.000734	0.000707	0.000985	0.000091	0.6008	0.6356	0.6424	4.9640
	0.25	0.000379	0.000376	0.000606	0.000072	3.1762	3.2860	3.3792	12.5381
	0.50	0.000191	0.000191	0.000179	0.000089	6.7267	6.8202	7.0152	25.0893
	0.75	0.000213	0.000140	0.000275	0.000051	10.4145	10.6370	11.0450	38.0540
	1.00	0.000397	0.000397	0.000644	0.000089	14.2260	15.2780	15.2920	50.2891

Table 3.8: Results of General Heuristics

3.4 Degree Constraint One Vertex Problem

As noted in the introduction to this chapter, it is easy to determine a minimum weight spanning tree with only one vertex having a degree restriction. The problem can be solved in polynomial time. Gabow (1978) presented an algorithm to solve this problem in *O(E log log V+V log V)*. In this section we present two very simple heuristics for this problem. The motivation behind this problem is due to the fact that Gabow's algorithm (1978) does not perform well when extended to problems with degree restrictions on more than one vertex in the graph. Our heuristics, on the other hand, produce much better computational results when extended to problems with degree restrictions on more than one vertex in the graph.

We let u be the vertex of our graph that has a specified degree. Suppose $r(u) = r$. Our first heuristic is:

Heuristic 3.4.1:

Select the r cheapest edges incident to u so as to satisfy the degree requirements on u and then extend this to a spanning tree of G by applying Kruskal's algorithm on the subgraph G' obtained by deleting all the edges of G incident to u which have not been selected.

Another approach is to start with a MWST and then exchange edges until the degree requirement on vertex u is met. More precisely:

Heuristic 3.4.2:

Step 1. Initialise the graph.

Step 2. Sort the edges by increasing weight.

Step 3. Find a MWST using Kruskal's algorithm.

Step 4. Examine vertex u. If the degree of vertex u is higher than its degree constraint then

Repeat

Delete the most expensive edge adjacent to vertex u.

Connect up the subgraph using the cheapest available edge not adjacent to vertex u

until

vertex u satisfies the degree constraint.

else

Repeat

Include the cheapest available edge adjacent to vertex u.

Move along the created cycle and delete the most expensive edge (not adjacent to vertex u) in the cycle

until

vertex u satisfies its degree constraint.

We have also programmed Gabow's algorithm (1978) to compare with our heuristics. Gabow's algorithm uses two queues and the idea of merging queues. We call it the Exact Algorithm.

Exact Algorithm (Gabow's):

Step 1. Initialise the graph and make X an empty priority queue.

Step 2. Find S, the minimum spanning forest of $G - u$.

Step 3. Let U be the set of edges of G incident to vertex u.

Step 4. Remove all edges from G except those in $S \cup U$.

Step 5. Find T, the smallest spanning tree containing U.

Step 6. For each edge $e \in U$

let $F(e)$ be a priority queue containing all the edges f in S joining components of $T - e$.

If $F(e) \neq \emptyset$

let f be the smallest exchange edge in $F(e)$.

Add (e, f) to X.

Step 7. While $d(u) > r$

Remove the edge (e, f) that give the smallest exchange from X and remove f from $F(e)$.

Let e' be the edge in $U - e$ where f joins the components of $T - e'$.

Remove the edge (e', f') from X and remove f from $F(e)$.

Merge $F(e)$ and $F(e')$ into $F(e')$.

If $F(e') \neq \emptyset$

let f' be the smallest exchange edge in $F(e')$.

Add (e', f') to X.

Hence $T \leftarrow T - e + f$.

Our two heuristics and Gabow's algorithm (1978) were implemented in the C programming language on a SUN SPARC 2 workstation operating at 28.5 MIPS. We tested the heuristics and Gabow's algorithm (1978) on the same sets of graphs as in Section 3.2. For simplicity we only did test runs on graphs with edge density 0.25 and 0.75. For the two heuristics we recorded the statistic $\frac{H-O}{O}$ where H is the average weight of the DCST obtained from applying the heuristic H and O is the optimum generated by Gabow's algorithm. We also recorded the average CPU time (secs) of the 50 problems. The results are

presented in Tables 3.9 and 3.10.

Our results show that both heuristics generated near optimal solution; on average within 0.04% and 0.03% of the optimum. Further, Heuristic 3.4.2 is always better than Heuristic 3.4.1. However, as the value of r increases Heuristic 3.4.1 sometimes coincides with Heuristic 3.4.2. In terms of speed Heuristic 3.4.1 is faster than Heuristic 3.4.2. Gabow's algorithm (1978) is slower than both the heuristics.

n	r	$\frac{H1-O}{O}$	$\frac{H2-O}{O}$	Exact (O)	Ave CPU Time (secs)		
					H1	H2	Exact
200	2	0.00025	0.00016	676.3755	0.2338	0.1230	1.2152
	3	0.00030	0.00022	678.5009	0.2324	0.1684	1.1884
	4	0.00024	0.00013	682.7687	0.2326	0.2708	1.2272
	5	0.00028	0.00028	688.9723	0.2328	0.4102	1.2770
	6	0.00037	0.00037	697.1086	0.2316	0.5376	1.2278
	7	0.00084	0.00084	707.1879	0.2336	0.6670	1.1786
	8	0.00067	0.00067	719.4085	0.2316	0.8172	1.2590
	9	0.00072	0.00072	733.1776	0.2316	0.9682	1.2716
	10	0.00079	0.00079	749.1169	0.2326	1.1216	1.3172
300	2	0.00014	0.00013	776.1157	0.5602	0.3360	3.9014
	3	0.00024	0.00010	777.6969	0.5590	0.4016	3.8200
	4	0.00023	0.00008	780.6414	0.5570	0.6290	3.7332
	5	0.00046	0.00021	785.0627	0.5592	0.9612	3.7366
	6	0.00041	0.00026	790.4191	0.5592	1.3584	3.7312
	7	0.00065	0.00051	797.4280	0.5594	1.6960	3.7316
	8	0.00045	0.00030	805.6696	0.5574	2.1176	3.7662
	9	0.00034	0.00026	815.2656	0.5604	3.0078	3.7806
	10	0.00057	0.00031	825.8989	0.5656	3.0054	3.7852
400	2	0.00025	0.00022	877.7145	1.0518	0.9898	13.7846
	3	0.00038	0.00006	878.7607	1.0532	1.3902	13.8786
	4	0.00014	0.00012	880.9194	1.0514	2.1746	13.8812
	5	0.00010	0.00010	884.2709	1.0584	3.2340	14.0094
	6	0.00002	0.00002	888.6181	1.0572	4.4288	13.9802
	7	0.00005	0.00005	893.8267	1.0578	5.6560	13.9768
	8	0.00005	0.00005	900.1816	1.0562	6.7728	13.9604
	9	0.00012	0.00012	907.6463	1.0554	7.9522	13.9304
	10	0.00009	0.00009	916.0938	1.0554	9.3900	13.9308
500	2	0.00017	0.00008	984.3172	1.7140	1.0280	26.9676
	3	0.00009	0.00006	985.3934	1.7220	1.9776	26.9476
	4	0.00005	0.00001	987.5574	1.7180	3.4220	26.9352
	5	0.00004	0.00004	990.6403	1.7172	4.7372	26.9360
	6	0.00017	0.00017	994.1372	1.7160	6.2636	26.9361
	7	0.00001	0.00001	998.5681	1.7140	8.1488	26.9452
	8	0.00014	0.00014	1003.9423	1.7160	9.8320	26.9420
	9	0.00017	0.00017	1010.1309	1.7224	11.7056	26.9280
	10	0.00014	0.00014	1016.9263	1.7168	13.9144	26.9048

Table 3.9: Results for p=0.25

n	r	$\frac{H1-O}{O}$	$\frac{H2-O}{O}$	Exact (O)	Ave CPU Time (secs)		
					H1	H2	Exact
200	2	0.00049	0.00042	360.7542	0.0723	0.5316	14.8656
	3	0.00035	0.00001	361.4945	0.7513	0.8263	14.9506
	4	0.00040	0.00029	363.1152	0.7516	1.2423	14.8843
	5	0.00023	0.00023	365.3816	0.7506	1.7226	14.8883
	6	0.00020	0.00020	368.5220	0.7553	2.2876	14.8803
	7	0.00036	0.00036	372.2230	0.7563	2.8046	14.8470
	8	0.00044	0.00044	376.5482	0.7503	3.3236	14.9306
	9	0.00040	0.00040	381.2992	0.7563	3.9020	14.8596
	10	0.00062	0.00062	386.5394	0.7563	4.6703	14.8816
300	2	0.00018	0.00018	458.7394	1.8360	1.9365	50.1345
	3	0.00004	0.00004	459.2954	1.8375	2.0665	50.0650
	4	0.00018	0.00015	460.3174	1.8365	3.2200	50.1105
	5	0.00036	0.00006	461.7632	1.8250	4.5725	50.1395
	6	0.00060	0.00008	463.7685	1.8285	5.8495	50.1005
	7	0.00068	0.00013	465.9983	1.8345	7.3845	50.2880
	8	0.00062	0.00017	468.6849	1.8295	8.6550	50.2685
	9	0.00083	0.00009	471.8107	1.8275	10.1825	50.1625
	10	0.00090	0.00010	475.3719	1.8285	11.8910	50.2045
400	2	0.00008	0.00006	560.9706	3.4160	3.0185	119.1425
	3	0.00031	0.00004	561.2841	3.4190	3.3770	119.1385
	4	0.00026	0.00005	561.9660	3.4250	4.9735	119.2215
	5	0.00052	0.00009	563.0442	3.4125	7.7765	119.3060
	6	0.00062	0.00007	564.4143	3.4310	10.0730	119.1135
	7	0.00070	0.00070	566.1294	3.4215	12.5575	119.1695
	8	0.00079	0.00079	568.0683	3.4280	15.3755	119.4385
	9	0.00089	0.00089	570.2973	3.4195	18.8080	119.1425
	10	0.00088	0.00088	572.8286	3.4265	21.8720	119.2320
500	2	0.00020	0.00008	659.8667	5.5995	3.4980	231.5855
	3	0.00028	0.00002	660.3127	5.5925	5.5430	230.3580
	4	0.00035	0.00035	661.0754	5.6015	8.7385	230.0540
	5	0.00040	0.00040	662.2439	5.6035	14.0065	229.9845
	6	0.00055	0.00055	663.7672	5.6010	18.7020	230.0105
	7	0.00055	0.00055	665.4764	5.5915	23.2420	230.1830
	8	0.00051	0.00051	667.4248	5.5925	27.0580	230.0650
	9	0.00060	0.00060	669.6557	5.5995	32.6215	230.0065
	10	0.00069	0.00069	672.1468	5.5970	38.9580	230.0050

Table 3.10: Results for p=0.75

3.5 Heuristics for the Degree Constraint Two Vertex Problem

In this section we consider the case when the degree constraint is on two of the vertices in the graph. The two degree constraints need not be the same. Throughout we let u and v be the two vertices of our graphs that have the specified degree. Suppose $r(u) = r_1 \geq r(v) = r_2$.

The Degree Constraint Two Vertex Problem can be formulated as a 0-1 MILP as follows:

$$Minimise \sum_i \sum_j c_{ij} x_{ij}$$

subject to

$$\sum_{i,j} x_{ij} = n - 1$$

$$\sum_{i,j \in V'} x_{ij} \leq |V'| - 1, \qquad \forall \quad \emptyset \neq V' \subset V$$

$$\sum_j x_{1j} = r_1, \qquad \sum_j x_{2j} = r_2$$

$$x_{ij} = 0 \, or \, 1, \qquad 1 \leq i \neq j \leq n.$$

In this section we modify the heuristics from Section 3.4 to develop eight simple approaches for this problem.

One simple approach is to extend the idea of Heuristic 3.4.1 to two vertices. First select edges incident to u and v so as to satisfy the degree requirements on u and v and then extend this to a spanning tree of G by applying Kruskal's algorithm on the subgraph G' obtained by deleting all the edges of G incident to u and v which have not been selected. Three simple procedures for satisfying the degree requirements on u and v are:

1. Select the r_1 cheapest edges incident to u followed by the r_2 cheapest available edges incident to v.

2. Select the r_2 cheapest edges incident to v followed by the r_1 cheapest available edges incident to u.

3. Select the cheapest available edge incident to u or v until the degree requirements on the two vertices are met.

Another approach is to extend Heuristic 3.4.2, the idea of first finding a MWST and then exchanging edges until the degree requirements are met. For this approach there are also three different procedures that can be carried out.

4. Select the vertex u or v whose number of incident edges in the MWST is furthest from its degree constraint. Without loss of generality let it be u. Depending on whether the number of incident edges for u is more or less than its degree constraint, add or delete edges to u according to Heuristic 3.4.2. Then do the same for v.

5. Satisfy the degree requirement on u first, then apply Kruskal's algorithm until a DCST is found. Check the number of incident edges beside v. Depending on whether this number is more or less than the degree constraint, add or delete edges to v according to Heuristic 3.4.2.

6. Satisfy the degree requirement on v first, then apply Kruskal's algorithm until a DCST is found. Check the number of incident edges beside u.

Depending on whether this number is more or less than its degree constraint, add or delete edges to u according to Heuristic 3.4.2.

Similarly, we can extend Gabow's one vertex algorithm (1978) to produce two different procedures.

7. Satisfy the degree requirement on u first. Then apply Gabow's algorithm (1978) with the aim of satisfying the degree on v.

8. Satisfy the degree requirement on v first. Then apply Gabow's algorithm (1978) with the aim of satisfying the degree on u.

Note that if vertices u and v are adjacent, then satisfying the degree requirement on vertex u (vertex v) followed by applying the algorithm of Gabow (1978) on vertex v (vertex u) could affect the degree on vertex u (vertex v). This happens because Gabow's algorithm (1978) starts with a DCST that includes all the incident edges beside vertex v (vertex u) and subsequently swaps edges until the degree requirement on vertex v (vertex u) is satisfied. To get around this problem we go back and check the degree on vertex u (vertex v) in the obtained DCST after applying Gabow's algorithm (1978). If the degree of vertex u (vertex v) is not the required degree that is, if it has been affected by Gabow's algorithm (1978), we find its cheapest available incident edge (not incident to vertex v (vertex u)) and include it in the DCST. We then move along the created cycle and delete the most expensive edge which is not incident to both u and v.

The above eight simple procedures give rise to eight heuristics represented by Hi, where Hi uses rule i. Our heuristics were implemented in the C programming language the same way as all our previous heuristics. For simplicity we tested graphs with density 0.25 and 0.75 and recorded the statistic $\frac{H-LB}{LB}$ (LB is the average lower bound obtained from the average MWST) and the average

CPU time (secs) of the 50 problems. The two degree constraints are r_1 and r_2. The results of the test runs are recorded in Tables 3.11 to 3.15.

p	r_1	r_2	$\frac{H1-LB}{LB}$	$\frac{H2-LB}{LB}$	$\frac{H3-LB}{LB}$	$\frac{H4-LB}{LB}$	$\frac{H5-LB}{LB}$	$\frac{H6-LB}{LB}$	$\frac{H7-LB}{LB}$	$\frac{H8-LB}{LB}$	Ave CPU Time (secs)							
											H1	H2	H3	H4	H5	H6	H7	H8
0.25	3	3	0.0303	0.0303	0.0304	0.0283	0.0298	0.0298	0.0235	0.0235	0.05	0.05	0.05	0.07	0.08	0.08	0.26	0.26
	3	4	0.0440	0.0440	0.0440	0.0393	0.0452	0.0434	0.0570	0.0593	0.05	0.05	0.05	0.09	0.10	0.08	0.27	0.28
	3	5	0.0870	0.0687	0.0687	0.0573	0.0687	0.0667	0.0794	0.0907	0.05	0.05	0.05	0.11	0.12	0.08	0.27	0.28
	3	6	0.0952	0.0949	0.0949	0.0834	0.0988	0.0955	0.1071	0.1301	0.05	0.05	0.05	0.14	0.15	0.08	0.27	0.28
	3	7	0.1312	0.1304	0.1304	0.1156	0.1358	0.1317	0.1425	0.1782	0.05	0.05	0.05	0.17	0.18	0.08	0.27	0.28
	3	8	0.1755	0.1747	0.1747	0.1578	0.1809	0.1761	0.1852	0.2362	0.05	0.05	0.05	0.20	0.21	0.08	0.27	0.28
	3	9	0.2242	0.2237	0.2237	0.2010	0.2309	0.2248	0.2335	0.3042	0.05	0.05	0.05	0.23	0.24	0.08	0.27	0.28
	3	10	0.2830	0.2825	0.2825	0.2567	0.2910	0.2836	0.2928	0.3787	0.05	0.05	0.05	0.25	0.26	0.08	0.27	0.28
	7	7	0.2427	0.2427	0.2412	0.1918	0.2497	0.2497	0.2743	0.2743	0.05	0.05	0.05	0.25	0.17	0.17	0.26	0.26
	7	8	0.2872	0.2861	0.2857	0.2232	0.3002	0.2951	0.3309	0.3458	0.05	0.05	0.05	0.27	0.19	0.18	0.27	0.27
	7	9	0.3347	0.3348	0.3331	0.2592	0.3522	0.3447	0.3796	0.4132	0.05	0.05	0.05	0.30	0.22	0.18	0.27	0.27
	7	10	0.3936	0.3927	0.3910	0.3070	0.4131	0.4048	0.4383	0.4877	0.05	0.05	0.05	0.33	0.24	0.18	0.27	0.27
	8	8	0.3313	0.3313	0.3286	0.2595	0.3436	0.3436	0.3791	0.3791	0.05	0.05	0.05	0.30	0.19	0.19	0.26	0.26
	8	9	0.3788	0.3797	0.3761	0.2876	0.3994	0.3938	0.4382	0.4582	0.05	0.05	0.05	0.32	0.22	0.19	0.27	0.27
	8	10	0.4377	0.4376	0.4340	0.3348	0.4606	0.4540	0.4970	0.5321	0.05	0.05	0.05	0.35	0.24	0.19	0.27	0.27
	9	9	0.4276	0.4276	0.4250	0.3326	0.4502	0.4502	0.4986	0.4986	0.05	0.05	0.05	0.35	0.21	0.21	0.26	0.26
	9	10	0.4866	0.4898	0.4831	0.3749	0.5178	0.5110	0.5654	0.5851	0.05	0.05	0.05	0.37	0.24	0.21	0.27	0.27
	10	10	0.5425	0.5425	0.5387	0.4247	0.5749	0.5749	0.6329	0.6329	0.05	0.06	0.05	0.38	0.29	0.22	0.28	0.26
0.75	3	3	0.0241	0.0241	0.0244	0.0221	0.0247	0.0247	0.1189	0.1189	0.17	0.17	0.16	0.22	0.26	0.26	1.90	1.89
	3	4	0.0361	0.0366	0.0364	0.0323	0.0364	0.0367	0.1854	0.2259	0.17	0.17	0.16	0.28	0.33	0.26	2.00	2.00
	3	5	0.0531	0.0538	0.0531	0.0471	0.0537	0.0537	0.1993	0.3010	0.17	0.17	0.16	0.35	0.40	0.26	1.98	2.00
	3	6	0.0757	0.0762	0.0757	0.0682	0.0762	0.0763	0.2179	0.3983	0.17	0.17	0.16	0.44	0.47	0.26	1.98	1.99
	3	7	0.1025	0.1027	0.1022	0.0907	0.1036	0.1031	0.2436	0.5287	0.17	0.17	0.16	0.51	0.55	0.26	1.98	2.00
	3	8	0.1351	0.1354	0.1349	0.1198	0.1368	0.1358	0.2743	0.6758	0.17	0.17	0.16	0.59	0.63	0.26	1.98	2.00
	3	9	0.1731	0.1724	0.1719	0.1530	0.1754	0.1740	0.3105	0.8179	0.17	0.17	0.16	0.69	0.72	0.27	1.99	1.99
	3	10	0.2158	0.2148	0.2143	0.1894	0.2192	0.2169	0.3505	0.9992	0.17	0.17	0.16	0.78	0.81	0.27	1.98	2.00
	7	7	0.1863	0.1863	0.1845	0.1461	0.1950	0.1950	0.5583	0.5583	0.17	0.17	0.16	0.77	0.54	0.55	1.89	1.90
	7	8	0.2177	0.2189	0.2158	0.1725	0.2294	0.2275	0.6443	0.7551	0.17	0.17	0.16	0.84	0.62	0.55	2.00	2.01
	7	9	0.2556	0.2559	0.2527	0.1992	0.2709	0.2664	0.6810	0.8988	0.17	0.17	0.16	0.95	0.70	0.54	2.00	2.01
	7	10	0.2960	0.2969	0.2928	0.2358	0.3164	0.3095	0.7216	1.0801	0.17	0.17	0.16	1.03	0.79	0.55	2.00	2.02
	8	8	0.2496	0.2496	0.2469	0.1946	0.2652	0.3047	0.7324	0.7324	0.17	0.17	0.16	0.93	0.64	0.63	1.89	1.89
	8	9	0.2866	0.2877	0.2828	0.2283	0.3068	0.3047	0.8193	0.9314	0.17	0.17	0.16	1.01	0.67	0.63	2.01	2.01
	8	10	0.3271	0.3287	0.3229	0.2588	0.3533	0.3480	0.8608	1.1129	0.17	0.17	0.16	1.10	0.75	0.63	2.00	2.02
	9	9	0.3262	0.3262	0.3224	0.2545	0.3483	0.3483	0.9126	0.9126	0.17	0.17	0.16	1.11	0.72	0.72	1.89	1.90
	9	10	0.3674	0.3686	0.3625	0.2924	0.3947	0.3922	1.0223	1.1489	0.17	0.17	0.16	1.19	0.74	0.71	2.01	2.02
	10	10	0.4104	0.4104	0.4047	0.3183	0.4402	0.4402	1.1622	1.1622	0.17	0.17	0.16	1.30	0.81	0.81	1.90	1.89

Table 3.11: Results for n=100

p	r_1	r_2	$\frac{H1-LB}{LB}$	$\frac{H2-LB}{LB}$	$\frac{H3-LB}{LB}$	$\frac{H4-LB}{LB}$	$\frac{H5-LB}{LB}$	$\frac{H6-LB}{LB}$	$\frac{H7-LB}{LB}$	$\frac{H8-LB}{LB}$	Ave CPU Time (secs)							
											H1	H2	H3	H4	H5	H6	H7	H8
0.25	3	3	0.0108	0.0108	0.0108	0.0104	0.0107	0.0107	0.0136	0.0136	0.30	0.29	0.28	0.51	0.53	0.52	1.22	1.22
	3	4	0.0168	0.0168	0.0168	0.0159	0.0167	0.0166	0.0289	0.0270	0.29	0.29	0.28	0.67	0.67	0.52	1.26	1.26
	3	5	0.0258	0.0258	0.0258	0.0242	0.0258	0.0257	0.0377	0.0385	0.29	0.29	0.28	0.84	0.86	0.52	1.26	1.26
	3	6	0.0379	0.0379	0.0379	0.0355	0.0378	0.0377	0.0493	0.0540	0.29	0.29	0.28	1.04	1.05	0.52	1.25	1.25
	3	7	0.0526	0.0526	0.0526	0.0469	0.0524	0.0524	0.0638	0.0727	0.29	0.29	0.28	1.23	1.25	0.53	1.25	1.25
	3	8	0.0698	0.0698	0.0698	0.0635	0.0698	0.0697	0.0809	0.0972	0.30	0.29	0.28	1.49	1.50	0.53	1.25	1.26
	3	9	0.0906	0.0906	0.0906	0.0821	0.0906	0.0905	0.1015	0.1242	0.30	0.29	0.28	1.72	1.73	0.53	1.25	1.28
	3	10	0.1145	0.1145	0.1145	0.1030	0.1145	0.1143	0.1249	0.1548	0.29	0.29	0.28	1.94	1.96	0.53	1.24	1.26
	7	7	0.0964	0.0964	0.0964	0.0781	0.0967	0.0967	0.1083	0.1083	0.29	0.29	0.28	1.95	1.30	1.27	1.22	1.22
	7	8	0.1137	0.1137	0.1137	0.0908	0.1138	0.1140	0.1415	0.1394	0.29	0.29	0.28	2.20	1.44	1.24	1.26	1.25
	7	9	0.1346	0.1346	0.1346	0.1075	0.1346	0.1349	0.1620	0.1667	0.29	0.29	0.28	2.43	1.62	1.23	1.25	1.27
	7	10	0.1585	0.1586	0.1585	0.1271	0.1584	0.1588	0.1858	0.1974	0.29	0.29	0.28	2.64	1.85	1.22	1.25	1.25
	8	8	0.1315	0.1315	0.1315	0.1061	0.1321	0.1321	0.1501	0.1501	0.29	0.29	0.28	2.41	1.48	1.47	1.22	1.22
	8	9	0.1525	0.1525	0.1525	0.1213	0.1529	0.1531	0.1860	0.1850	0.29	0.29	0.28	2.64	1.61	1.45	1.26	1.27
	8	10	0.1764	0.1765	0.1764	0.1400	0.1768	0.1771	0.2095	0.2156	0.29	0.29	0.28	2.87	1.82	1.43	1.25	1.26
	9	9	0.1732	0.1732	0.1732	0.1370	0.1739	0.1739	0.1969	0.1969	0.29	0.29	0.28	2.84	1.69	1.68	1.22	1.23
	9	10	0.1972	0.1972	0.1972	0.1549	0.1976	0.1979	0.2367	0.2360	0.29	0.29	0.28	3.08	1.77	1.65	1.25	1.26
	10	10	0.2206	0.2206	0.2206	0.1760	0.2218	0.2218	0.2512	0.2512	0.29	0.29	0.28	3.30	1.88	1.85	1.22	1.22
0.75	3	3	0.0080	0.0080	0.0080	0.0069	0.0076	0.0076	0.1281	0.1268	0.79	0.79	0.76	1.44	1.51	1.51	15.07	15.08
	3	4	0.0129	0.0129	0.0129	0.0118	0.0129	0.0126	0.1140	0.1675	0.79	0.79	0.76	1.71	1.62	1.51	15.99	15.69
	3	5	0.0192	0.0192	0.0192	0.0172	0.0192	0.0188	0.1197	0.1961	0.79	0.79	0.76	2.11	2.03	1.48	15.94	15.80
	3	6	0.0266	0.0266	0.0266	0.0245	0.0267	0.0262	0.1267	0.2297	0.79	0.79	0.76	2.66	2.59	1.45	15.92	15.89
	3	7	0.0366	0.0366	0.0366	0.0340	0.0367	0.0363	0.1352	0.2731	0.79	0.79	0.77	3.18	3.13	1.45	15.98	15.84
	3	8	0.0475	0.0475	0.0475	0.0424	0.0478	0.0471	0.1459	0.3213	0.79	0.79	0.76	3.72	3.60	1.46	15.90	15.87
	3	9	0.0613	0.0613	0.0613	0.0552	0.0615	0.0610	0.1580	0.3751	0.79	0.79	0.76	4.23	4.10	1.46	15.93	15.82
	3	10	0.0771	0.0771	0.0771	0.0696	0.0774	0.0712	0.1734	0.4390	0.79	0.79	0.76	4.87	4.67	2.39	15.93	15.73
	7	7	0.0656	0.0656	0.0660	0.0534	0.0656	0.0656	0.2666	0.2658	0.79	0.79	0.77	5.17	3.34	3.33	15.09	15.12
	7	8	0.0765	0.0769	0.0769	0.0637	0.0771	0.0765	0.2523	0.3434	0.79	0.78	0.76	5.70	3.64	3.34	15.96	15.83
	7	9	0.0905	0.0909	0.0909	0.0746	0.0911	0.0905	0.2645	0.3973	0.79	0.79	0.76	6.13	4.18	3.28	15.98	15.72
	7	10	0.1064	0.1067	0.1067	0.0874	0.1071	0.1064	0.2801	0.4602	0.79	0.79	0.76	6.80	4.77	3.23	15.94	15.71
	8	8	0.0884	0.0884	0.0888	0.0730	0.0884	0.0884	0.3226	0.3225	0.79	0.78	0.77	6.02	3.80	3.81	15.09	15.12
	8	9	0.1024	0.1028	0.1028	0.0847	0.1030	0.1024	0.3088	0.4071	0.79	0.79	0.76	6.66	4.18	3.74	15.90	15.73
	8	10	0.1184	0.1188	0.1188	0.0987	0.1191	0.1184	0.3239	0.4701	0.79	0.79	0.76	7.17	4.74	3.71	15.88	15.67
	9	9	0.1157	0.1157	0.1157	0.0959	0.1157	0.1157	0.3868	0.3880	0.79	0.79	0.76	7.17	4.28	4.29	15.10	15.08
	9	10	0.1317	0.1317	0.1317	0.1085	0.1317	0.1317	0.3741	0.4816	0.79	0.79	0.77	7.91	4.71	4.21	15.98	15.69
	10	10	0.1457	0.1457	0.1457	0.1197	0.1463	0.1463	0.4608	0.4637	0.79	0.79	0.76	8.55	4.86	4.85	15.09	15.10

Table 3.12: Results for n=200

p	r_1	r_2	$\frac{H1-LB}{LB}$	$\frac{H2-LB}{LB}$	$\frac{H3-LB}{LB}$	$\frac{H4-LB}{LB}$	$\frac{H5-LB}{LB}$	$\frac{H6-LB}{LB}$	$\frac{H7-LB}{LB}$	$\frac{H8-LB}{LB}$	Ave CPU Time (secs)							
											H1	H2	H3	H4	H5	H6	H7	H8
0.25	3	3	0.0078	0.0078	0.0078	0.0072	0.0077	0.0077	0.0126	0.0126	0.59	0.59	0.57	1.16	1.11	1.11	6.21	6.22
	3	4	0.0118	0.0118	0.0118	0.0111	0.0117	0.0116	0.0211	0.0235	0.59	0.59	0.57	1.43	1.43	1.12	6.27	6.35
	3	5	0.0182	0.0182	0.0182	0.0167	0.0181	0.0180	0.0273	0.0314	0.59	0.59	0.57	1.91	1.89	1.10	6.27	6.34
	3	6	0.0258	0.0258	0.0258	0.0234	0.0258	0.0257	0.0348	0.0415	0.59	0.59	0.57	2.38	2.38	1.11	6.28	6.36
	3	7	0.0351	0.0351	0.0351	0.0316	0.0351	0.0350	0.0442	0.0537	0.59	0.59	0.57	2.93	2.90	1.13	6.29	6.38
	3	8	0.0461	0.0461	0.0461	0.0419	0.0461	0.0460	0.0552	0.0683	0.59	0.59	0.57	3.48	3.47	1.09	6.28	6.41
	3	9	0.0590	0.0591	0.0591	0.0531	0.0591	0.0588	0.0674	0.0856	0.59	0.59	0.57	4.01	4.02	1.07	6.27	6.41
	3	10	0.0732	0.0733	0.0733	0.0670	0.0733	0.0730	0.0815	0.1058	0.58	0.59	0.57	4.68	4.65	1.08	6.27	6.40
	7	7	0.0613	0.0613	0.0613	0.0516	0.0611	0.0611	0.0742	0.0742	0.59	0.59	0.57	4.97	3.04	3.04	6.23	6.22
	7	8	0.0723	0.0723	0.0723	0.0592	0.0723	0.0722	0.0914	0.0935	0.59	0.59	0.57	5.42	3.43	2.94	6.32	6.39
	7	9	0.0852	0.0854	0.0854	0.0677	0.0854	0.0851	0.1035	0.1109	0.58	0.59	0.57	5.89	3.93	2.86	6.32	6.39
	7	10	0.0994	0.0993	0.0993	0.0796	0.0997	0.0994	0.1177	0.1311	0.58	0.59	0.57	6.49	4.50	2.93	6.31	6.40
	8	8	0.0828	0.0828	0.0828	0.0678	0.0826	0.0826	0.0991	0.0991	0.59	0.59	0.57	6.05	3.52	3.55	6.23	6.22
	8	9	0.0953	0.0954	0.0953	0.0765	0.0956	0.0954	0.1173	0.1214	0.59	0.59	0.57	6.45	3.87	3.46	6.32	6.38
	8	10	0.1095	0.1093	0.1092	0.0868	0.1100	0.1097	0.1315	0.1416	0.59	0.59	0.57	7.04	4.45	3.50	6.30	6.38
	9	9	0.1078	0.1078	0.1078	0.0876	0.1081	0.1081	0.1288	0.1288	0.59	0.59	0.57	7.03	3.98	3.97	6.23	6.22
	9	10	0.1220	0.1218	0.1218	0.0975	0.1225	0.1224	0.1482	0.1540	0.59	0.59	0.57	7.57	4.39	3.99	6.25	6.39
	10	10	0.1356	0.1356	0.1354	0.1101	0.1365	0.1365	0.1628	0.1628	0.59	0.59	0.57	8.25	4.52	4.52	6.22	6.23
0.75	3	3	0.0043	0.0043	0.0043	0.0040	0.0043	0.0043	0.0707	0.0707	1.92	1.91	1.86	3.68	3.81	3.80	50.52	50.54
	3	4	0.0064	0.0064	0.0064	0.0058	0.0065	0.0064	0.0782	0.0986	1.91	1.92	1.86	4.36	4.41	3.73	52.09	52.41
	3	5	0.0096	0.0097	0.0097	0.0080	0.0097	0.0096	0.0811	0.1147	1.93	1.92	1.86	5.38	5.52	3.66	52.01	52.86
	3	6	0.0140	0.0141	0.0141	0.0120	0.0141	0.0140	0.0840	0.1317	1.92	1.92	1.86	6.16	6.39	3.70	51.94	52.66
	3	7	0.0194	0.0195	0.0195	0.0166	0.0198	0.0194	0.0890	0.1537	1.92	1.92	1.85	7.51	7.63	3.69	51.96	52.75
	3	8	0.0256	0.0258	0.0258	0.0221	0.0261	0.0256	0.0954	0.1793	1.92	1.93	1.85	9.22	9.56	3.65	52.19	52.71
	3	9	0.0331	0.0333	0.0333	0.0296	0.0336	0.0331	0.0954	0.2099	1.92	1.92	1.86	10.97	11.31	3.59	52.19	52.53
	3	10	0.0414	0.0413	0.0413	0.0370	0.0424	0.0414	0.0954	0.2457	1.92	1.92	1.86	12.30	12.67	3.67	52.19	52.53
	7	7	0.0334	0.0334	0.0334	0.0273	0.0335	0.0335	0.0954	0.1480	1.91	1.92	1.85	13.17	8.69	8.72	52.19	50.51
	7	8	0.0393	0.0393	0.0393	0.0312	0.0399	0.0394	0.0954	0.1907	1.92	1.92	1.86	14.83	9.30	8.89	52.19	52.54
	7	9	0.0470	0.0470	0.0470	0.0378	0.0477	0.0471	0.0954	0.2213	1.92	1.92	1.86	16.61	11.02	8.79	52.19	52.37
	7	10	0.0550	0.0550	0.0550	0.0435	0.0565	0.0554	0.0954	0.2571	1.92	1.92	1.86	17.66	12.45	8.93	52.19	52.92
	8	8	0.0453	0.0453	0.0453	0.0356	0.0454	0.0454	0.0954	0.1789	1.92	1.92	1.86	16.18	10.26	10.26	52.19	50.73
	8	9	0.0530	0.0530	0.0530	0.0418	0.0538	0.0530	0.0954	0.2276	1.92	1.92	1.85	18.00	10.88	10.02	52.19	52.84
	8	10	0.0606	0.0610	0.0608	0.0473	0.0626	0.0614	0.0954	0.2634	1.91	1.93	1.85	19.39	12.32	10.47	52.19	52.73
	9	9	0.0595	0.0595	0.0595	0.0476	0.0597	0.0597	0.0954	0.2159	1.91	1.92	1.86	19.73	11.65	11.67	52.19	50.82
	9	10	0.0672	0.0675	0.0673	0.0537	0.0692	0.0681	0.0954	0.2685	1.92	1.92	1.86	20.97	12.43	12.16	52.19	52.63
	10	10	0.0749	0.0749	0.0746	0.0601	0.0759	0.0759	0.0954	0.2590	1.92	1.91	1.86	22.72	13.76	13.75	52.19	51.05

Table 3.13: Results for n=300

p	r_1	r_2	$\frac{H1-LB}{LB}$	$\frac{H2-LB}{LB}$	$\frac{H3-LB}{LB}$	$\frac{H4-LB}{LB}$	$\frac{H5-LB}{LB}$	$\frac{H6-LB}{LB}$	$\frac{H7-LB}{LB}$	$\frac{H8-LB}{LB}$	Ave CPU Time (secs)							
											H1	H2	H3	H4	H5	H6	H7	H8
0.25	3	3	0.0049	0.0049	0.0049	0.0043	0.0046	0.0046	0.0080	0.0080	1.10	1.11	1.07	2.72	2.41	2.42	14.08	14.12
	3	4	0.0074	0.0074	0.0074	0.0066	0.0072	0.0071	0.0110	0.0132	1.10	1.10	1.07	3.22	2.92	2.37	14.45	14.16
	3	5	0.0112	0.0111	0.0112	0.0102	0.0111	0.0109	0.0148	0.0186	1.11	1.10	1.07	4.20	3.93	2.44	14.44	14.18
	3	6	0.0160	0.0160	0.0160	0.0146	0.0160	0.0158	0.0196	0.0255	1.11	1.11	1.07	5.42	5.09	2.45	14.45	14.20
	3	7	0.0221	0.0221	0.0222	0.0204	0.0222	0.0218	0.0253	0.0339	1.11	1.11	1.07	6.70	6.21	2.52	14.44	14.23
	3	8	0.0291	0.0292	0.0292	0.0267	0.0292	0.0288	0.0323	0.0436	1.12	1.10	1.07	8.08	7.67	2.46	14.44	14.22
	3	9	0.0373	0.0374	0.0374	0.0345	0.0375	0.0370	0.0404	0.0549	1.11	1.10	1.07	9.22	8.80	2.57	14.37	14.23
	3	10	0.0466	0.0467	0.0467	0.0429	0.0468	0.0463	0.0493	0.0679	1.11	1.10	1.07	10.57	10.14	2.59	14.37	14.22
	7	7	0.0390	0.0390	0.0392	0.0338	0.0390	0.0390	0.0496	0.0496	1.12	1.11	1.07	10.85	6.65	6.65	14.10	14.14
	7	8	0.0460	0.0462	0.0462	0.0396	0.0462	0.0461	0.0565	0.0607	1.12	1.10	1.07	12.15	7.61	6.46	14.49	14.18
	7	9	0.0543	0.0545	0.0545	0.0465	0.0545	0.0543	0.0647	0.0720	1.11	1.10	1.07	13.24	8.71	6.59	14.48	14.15
	7	10	0.0636	0.0638	0.0638	0.0533	0.0639	0.0636	0.0736	0.0850	1.11	1.10	1.07	14.68	10.08	6.53	14.43	14.15
	8	8	0.0533	0.0533	0.0534	0.0458	0.0533	0.0533	0.0665	0.0665	1.11	1.10	1.07	13.34	7.60	7.60	14.12	14.06
	8	9	0.0615	0.0616	0.0616	0.0528	0.0617	0.0615	0.0748	0.0793	1.12	1.10	1.07	14.42	8.69	7.62	14.47	14.13
	8	10	0.0708	0.0710	0.0710	0.0604	0.0711	0.0708	0.0838	0.0923	1.12	1.10	1.07	15.71	10.03	7.57	14.47	14.13
	9	9	0.0701	0.0701	0.0701	0.0588	0.0701	0.0701	0.0865	0.0865	1.12	1.10	1.07	15.63	8.78	8.78	14.16	14.06
	9	10	0.0795	0.0795	0.0795	0.0666	0.0795	0.0795	0.0954	0.1007	1.12	1.10	1.07	16.85	9.95	8.68	14.45	14.12
	10	10	0.0891	0.0891	0.0891	0.0743	0.0891	0.0891	0.1090	0.1090	1.12	1.79	1.07	18.28	9.97	10.03	14.10	14.05
0.75	3	3	0.0027	0.0027	0.0027	0.0024	0.0026	0.0026	0.0393	0.0393	3.59	3.60	3.48	6.74	6.56	6.55	120.39	120.26
	3	4	0.0041	0.0041	0.0041	0.0039	0.0041	0.0041	0.0877	0.0597	3.59	3.60	3.49	9.07	9.37	6.76	123.17	123.90
	3	5	0.0062	0.0062	0.0062	0.0059	0.0062	0.0062	0.0894	0.0674	3.59	3.60	3.49	11.51	11.63	6.81	122.98	123.83
	3	6	0.0090	0.0090	0.0090	0.0086	0.0090	0.0089	0.0914	0.0774	3.60	3.60	3.49	14.50	14.55	6.86	122.95	124.01
	3	7	0.0121	0.0121	0.0121	0.0115	0.0121	0.0120	0.0943	0.0886	3.60	3.60	3.49	18.25	18.23	6.89	122.97	123.85
	3	8	0.0159	0.0159	0.0159	0.0151	0.0159	0.0159	0.0977	0.1021	3.60	3.60	3.49	21.25	21.26	6.91	123.02	123.73
	3	9	0.0203	0.0203	0.0203	0.0191	0.0203	0.0202	0.1017	0.1183	3.59	3.60	3.49	25.28	25.18	7.07	123.37	123.86
	3	10	0.0254	0.0254	0.0254	0.0238	0.0254	0.0253	0.1063	0.1369	3.59	3.59	3.49	28.69	28.32	7.24	122.85	123.66
	7	7	0.0204	0.0204	0.0204	0.0176	0.0204	0.0204	0.0820	0.0820	3.59	3.60	3.49	28.49	15.46	15.46	120.30	120.24
	7	8	0.0242	0.0242	0.0242	0.0205	0.0242	0.0242	0.1354	0.1094	3.60	3.61	3.49	31.59	21.87	15.59	124.57	123.48
	7	9	0.0286	0.0286	0.0286	0.0239	0.0286	0.0286	0.1397	0.1257	3.59	3.60	3.49	36.06	25.38	16.09	124.94	123.68
	7	10	0.0337	0.0337	0.0337	0.0281	0.0337	0.0337	0.1443	0.1443	3.59	3.59	3.48	39.15	28.54	16.54	124.68	123.61
	8	8	0.0276	0.0276	0.0276	0.0236	0.0276	0.0276	0.0993	0.0993	3.60	3.60	3.48	34.50	19.12	19.06	120.28	120.72
	8	9	0.0320	0.0320	0.0320	0.0265	0.0320	0.0320	0.1535	0.1289	3.60	3.60	3.49	38.62	25.41	19.70	123.49	123.70
	8	10	0.0371	0.0371	0.0371	0.0306	0.0371	0.0371	0.1581	0.1475	3.60	3.59	3.49	42.85	28.36	20.29	123.53	123.61
	9	9	0.0360	0.0360	0.0360	0.0296	0.0360	0.0360	0.1187	0.1187	3.60	3.59	3.49	42.33	23.14	23.13	120.24	120.24
	9	10	0.0411	0.0411	0.0411	0.0338	0.0411	0.0411	0.1741	0.1512	3.62	3.60	3.49	45.57	29.06	23.84	123.57	123.52
	10	10	0.0456	0.0456	0.0456	0.0369	0.0456	0.0456	0.1415	0.1415	3.60	3.59	3.49	48.75	26.91	26.94	120.26	120.62

Table 3.14: Results for n=400

p	r_1	r_2	$\frac{H1-LB}{LB}$	$\frac{H2-LB}{LB}$	$\frac{H3-LB}{LB}$	$\frac{H4-LB}{LB}$	$\frac{H5-LB}{LB}$	$\frac{H6-LB}{LB}$	$\frac{H7-LB}{LB}$	$\frac{H8-LB}{LB}$	Ave CPU Time (secs)							
											H1	H2	H3	H4	H5	H6	H7	H8
0.25	3	3	0.0034	0.0034	0.0034	0.0031	0.0033	0.0033	0.0057	0.0057	1.80	1.79	1.74	3.93	3.75	3.76	27.33	27.30
	3	4	0.0053	0.0053	0.0053	0.0048	0.0052	0.0052	0.0085	0.0107	1.79	1.79	1.74	5.59	5.34	3.59	27.83	27.80
	3	5	0.0079	0.0079	0.0079	0.0068	0.0078	0.0078	0.0112	0.0144	1.79	1.79	1.74	6.98	6.69	3.52	27.95	27.91
	3	6	0.0113	0.0113	0.0113	0.0100	0.0112	0.0112	0.0146	0.0198	1.79	1.79	1.74	8.54	8.48	3.47	27.90	28.01
	3	7	0.0155	0.0155	0.0155	0.0142	0.0155	0.0154	0.0188	0.0261	1.79	1.79	1.74	10.36	10.24	3.57	27.95	28.12
	3	8	0.0204	0.0204	0.0204	0.0184	0.0204	0.0203	0.0237	0.0336	1.79	1.79	1.74	12.81	12.71	3.57	27.92	27.92
	3	9	0.0260	0.0260	0.0260	0.0239	0.0260	0.0259	0.0294	0.0422	1.80	1.80	1.74	14.78	14.73	3.52	27.97	27.70
	3	10	0.0330	0.0330	0.0330	0.0305	0.0330	0.0329	0.0363	0.0524	1.79	1.79	1.73	16.74	16.75	3.48	27.93	27.73
	7	7	0.0290	0.0290	0.0290	0.0240	0.0290	0.0290	0.0368	0.0368	1.80	1.79	1.74	17.14	9.87	9.87	27.29	27.24
	7	8	0.0339	0.0339	0.0339	0.0285	0.0338	0.0339	0.0403	0.0469	1.79	1.80	1.74	19.43	12.69	9.98	27.91	27.88
	7	9	0.0395	0.0395	0.0395	0.0347	0.0395	0.0395	0.0461	0.0560	1.79	1.81	1.74	21.26	14.73	9.88	27.84	27.89
	7	10	0.0464	0.0464	0.0464	0.0408	0.0464	0.0464	0.0530	0.0661	1.80	1.80	1.74	23.09	16.74	9.79	27.91	27.88
	8	8	0.0395	0.0395	0.0395	0.0334	0.0395	0.0395	0.0498	0.0498	1.79	1.79	1.74	21.47	11.46	11.50	27.31	27.25
	8	9	0.0451	0.0451	0.0451	0.0387	0.0451	0.0451	0.0528	0.0614	1.80	1.79	1.74	22.94	14.74	11.38	27.86	27.90
	8	10	0.0520	0.0520	0.0520	0.0453	0.0520	0.0520	0.0597	0.0716	1.79	1.80	1.74	24.95	16.70	11.37	27.85	27.86
	9	9	0.0514	0.0514	0.0514	0.0442	0.0514	0.0514	0.0651	0.0651	1.80	1.80	1.74	24.88	13.18	13.21	27.31	27.27
	9	10	0.0583	0.0583	0.0583	0.0503	0.0583	0.0583	0.0677	0.0783	1.79	1.79	1.74	26.63	16.24	13.12	27.93	28.13
	10	10	0.0652	0.0652	0.0652	0.0553	0.0652	0.0652	0.0822	0.0822	1.80	1.80	1.74	29.23	15.19	15.16	27.28	27.25
0.75	3	3	0.0016	0.0016	0.0016	0.0015	0.0016	0.0016	0.0461	0.0461	5.85	5.85	5.68	11.93	11.24	11.29	232.47	232.57
	3	4	0.0026	0.0026	0.0026	0.0024	0.0026	0.0026	0.0531	0.0603	5.86	5.85	5.69	15.96	15.40	11.53	239.86	243.54
	3	5	0.0038	0.0038	0.0038	0.0035	0.0038	0.0037	0.0542	0.0662	5.85	5.85	5.69	22.04	21.21	11.85	239.18	243.33
	3	6	0.0056	0.0056	0.0056	0.0051	0.0056	0.0055	0.0559	0.0737	5.85	5.85	5.68	27.21	27.05	11.70	239.42	244.11
	3	7	0.0077	0.0077	0.0077	0.0072	0.0077	0.0077	0.0579	0.0827	5.85	5.85	5.68	32.65	32.65	11.64	239.13	244.60
	3	8	0.0101	0.0101	0.0101	0.0094	0.0101	0.0100	0.0602	0.0929	5.85	5.85	5.71	39.07	38.51	11.58	239.11	241.46
	3	9	0.0127	0.0127	0.0127	0.0121	0.0127	0.0127	0.0632	0.1051	5.85	5.85	5.68	45.09	44.73	11.66	239.83	241.32
	3	10	0.0159	0.0159	0.0159	0.0151	0.0159	0.0159	0.0661	0.1184	5.85	5.85	5.69	52.46	51.79	11.91	241.19	241.27
	7	7	0.0153	0.0153	0.0153	0.0130	0.0153	0.0153	0.0801	0.0801	5.85	5.85	5.68	51.47	30.22	30.30	233.14	232.82
	7	8	0.0177	0.0177	0.0177	0.0151	0.0177	0.0177	0.0925	0.1001	5.85	5.86	5.69	59.17	41.29	30.00	240.84	241.17
	7	9	0.0204	0.0204	0.0204	0.0175	0.0204	0.0204	0.0955	0.1125	5.85	5.86	5.69	66.16	47.80	30.44	240.58	240.51
	7	10	0.0235	0.0235	0.0235	0.0203	0.0235	0.0235	0.0985	0.1258	5.85	5.85	5.69	72.81	54.76	31.62	240.16	239.91
	8	8	0.0207	0.0207	0.0207	0.0170	0.0207	0.0207	0.0933	0.0933	5.85	5.85	5.68	61.80	33.98	33.98	232.74	232.95
	8	9	0.0234	0.0234	0.0234	0.0195	0.0234	0.0234	0.1074	0.1153	5.85	5.85	5.68	69.08	47.54	34.50	240.57	240.48
	8	10	0.0265	0.0265	0.0265	0.0229	0.0265	0.0265	0.1104	0.1286	5.85	5.85	5.69	76.06	54.53	35.76	240.76	240.99
	9	9	0.0267	0.0267	0.0267	0.0220	0.0267	0.0267	0.1088	0.1088	5.85	5.85	5.69	75.22	39.73	39.60	232.64	236.05
	9	10	0.0298	0.0298	0.0298	0.0251	0.0298	0.0298	0.1237	0.1319	5.85	5.85	5.68	81.65	55.12	41.23	241.03	243.58
	10	10	0.0335	0.0335	0.0335	0.0280	0.0335	0.0335	0.1258	0.1258	5.85	5.86	5.70	88.89	47.45	47.33	232.59	233.24

Table 3.15: Results for n=500

We tested the heuristics with values of (r_1, r_2): (2,2), (2,3), ... , (2,10), (3,3), ... , (3,10), (4,4), ... and (10,10). Here we only present our results for the values: (3,3), ..., (3,10) and (7,7), ... and (10,10).

From the results we observe that on average Heuristic 4 produced results that are closest to the LB followed by: Heuristic 6, Heuristic 3, Heuristic 1, Heuristic 2, Heuristic 5, Heuristic 7 and Heuristic 8.

Among all the heuristics Heuristic 3 is the fastest with Heuristic 1 and Heuristic 2 next in line. This is followed by Heuristics 6, 5 and 4. The extension of Gabow's (1978) One Vertex algorithm to Two Vertex heuristics are the slowest compared to other heuristics. However, Heuristic 7 did perform slightly faster than Heuristic 8.

Chapter 4

The $(1,k)$-tree Problem

This chapter looks at another type of Restricted Spanning Tree, the $(1,k)$-tree. The problem is defined as: given a graph G with maximum degree k find a spanning tree consisting of vertices of degree 1 or k. We are interested in establishing the complexity of this $(1,k)$ problem with $3 \leq k \leq 5$. Recall from Chapter 1 that Douglas (1992) provided the motivation for this problem with his $(1,3)$-tree result:

> *Given a planar graph $G = (V,E)$ with maximum degree 3, it is NP-complete to decide if there exists a spanning tree T for G such that $deg(x,T) = 1$ or 3 for all $x \in V$.*

We use the idea of Douglas' proof (1992) to provide an alternate proof to this NP-complete problem for non-planar graphs with maximum degree 3 (Section 4.1). We also establish the complexity of the $(1,4)$-tree in graphs with maximum degree 4. We provide simple countings to determine the number of vertices of degree 1 and k in the general $(1,k)$-tree.

We also study spanning trees whose degrees are from a certain degree spectrum. Recall that we call $d = (d_1, d_2, \ldots, d_m)$ the degree spectrum of G and a $(d_0, d_1, \ldots, d_m)$-tree is a tree with degree spectrum $(d_0, d_1, d_2, \ldots, d_m)$. We

establish the NP-completeness of the $(1,3,5)$-tree problem in graphs with $d = (3,5)$. Further, we establish upper and lower bounds for the number of vertices of degree one in spanning trees with degree spectrum ($d_0 = 1 < d_1 < \ldots < d_m = k$).

The chapter concludes with heuristics for finding $(1,k)$-trees. These heuristics are tested on connected random graphs that possess $(1,k)$-trees and computational results based on our testings are presented.

4.1 The Complexity

We begin our work with the following important lemma.

Lemma 4.1.1 *Given a graph $G = (V,E)$ with exactly two vertices A and B of degree 1 and all the others of degree k, it is NP-complete to decide if there is a Hamilton path between A and B.*

PROOF: A Hamilton path between A and B in G exists if and only if $G + AB$ contains a Hamilton cycle (must use edge AB). So we must show that the Hamilton cycle problem in $G+AB$ is NP-complete. The proof of this follows from the proof of Garey and Johnson (1979) which shows that the Hamilton cycle is NP-complete. We have included this proof here for completeness. The idea is to transform the Vertex Cover problem (given a graph G the aim is to find a subset of vertices $V' \subseteq V$ such that for each edge of $E(G)$ at least one of its end-point belongs to V') to the Hamilton cycle problem. Given a graph $G = (V,E)$ and a positive integer $Q \leq |V|$, we construct a graph $G' = (V',E')$ such that there is a Hamilton cycle in G' if and only if there is a Vertex Cover of size Q or less in G.

We use Q 'selector' vertices $a_1, a_2, \ldots, a_Q$ in G'. Further, G' contains a 'cover testing' component that ensures that every edge of E is incident to at least one of the selected

Q vertices.

Each edge $e = (u, v) \in E$ is replaced by the structure shown in Figure 4.1 that contains 12 vertices and 14 edges. The vertices are:

$$V'_e = \{(u, e, i), (v, e, i) : 1 \leq i \leq 6\}$$

and the edges are:

$$\begin{aligned} E'_e \;=\; & \{\{(u, e, i), (u, e, i+1)\}, \{(v, e, i), (v, e, i+1)\} : 1 \leq i \leq 5\} \\ & \cup\{(u, e, 3), (v, e, 1)\}, \{(v, e, 3), (u, e, 1)\} \\ & \cup\{(u, e, 6), (v, e, 4)\}, \{(v, e, 6), (u, e, 4)\}. \end{aligned}$$

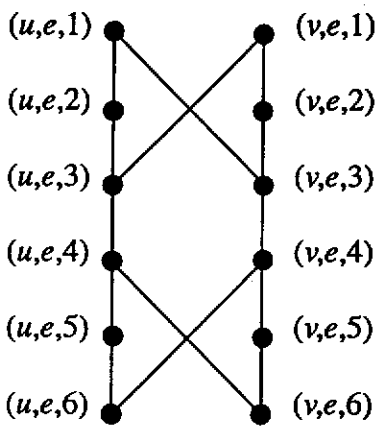


Figure 4.1

We refer to each such structure as a component.

Only vertices $(u, e, 1), (v, e, 1), (u, e, 6)$ and $(v, e, 6)$ of this cover-testing component will be involved in any additional edges of G'. The additional edges are to join the pairs of cover-testing components or to join the cover-testing components to the selector vertices. This implies that if the Hamilton cycle of G' enters the component at $(u, e, 1)$ (respectively $(v, e, 1)$), it will have to leave at $(u, e, 6)$ (respectively $(v, e, 6)$) and visit all the 12 vertices in the component or just the 6 vertices (u, e, i), $1 \leq i \leq 6$

(respectively (v, e, i), $1 \leq i \leq 6$). Figure 4.2 illustrates this point.

For each vertex $v \in V(G)$, let the edges incident to a vertex v be ordered (arbitrarily) as $e_{v[1]}, e_{v[2]}, \ldots, e_{v[d_G(v)]}$ where $d_G(v)$ denotes the degree of v in G. Then the cover-testing components are joined together by the edges:

$$E'_v = \{\{(v, e_{v[i]}, 6), (v, e_{v[i+1]}, 1)\} : 1 \leq i < d_G(v)\}.$$

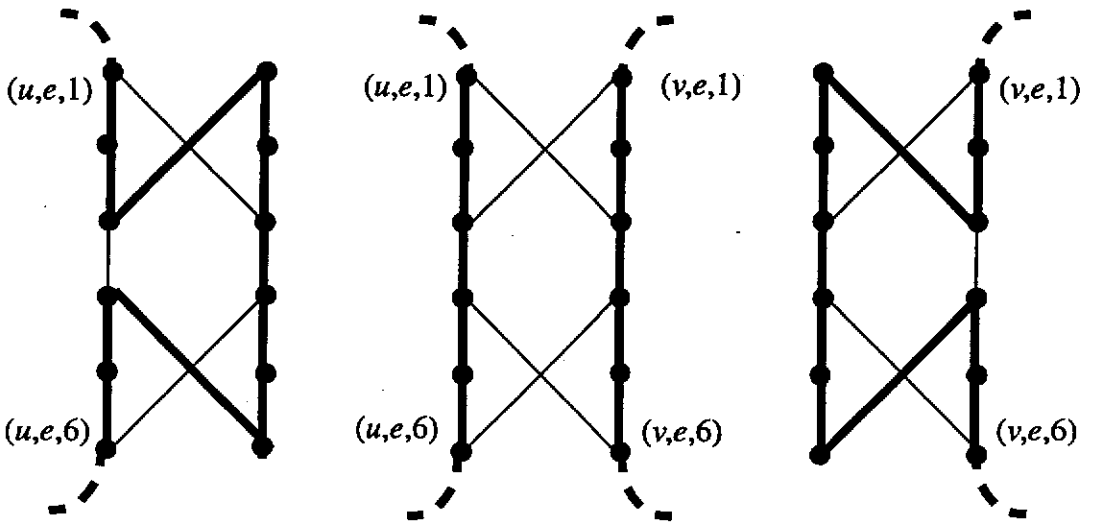


Figure 4.2

To join the cover-testing components to the selector vertices we add the edges:

$$E'' = \{\{(a_i, (v, e_{v[1]}, 1)\}, \{(a_i, (v, e_{v[d_G(v)]}, 6)\} : 1 \leq i \leq Q, v \in V\}.$$

Hence, $G' = (V', E')$ with

$$V' = \{a_i : 1 \leq i \leq Q\} \cup (\bigcup_{e \in E} V'_e)$$

and

$$E' = (\bigcup_{e \in E} E'_e) \cup (\bigcup_{v \in V} E'_v) \cup E''.$$

Note that G' can be constructed from G and Q in polynomial time.

We prove that there is a Hamilton cycle in G' if and only if there is a Vertex Cover of size Q or less in G.

Let $\{v_1, v_2, \ldots, v_n, v_1\}$, where $n = |V'|$, be a Hamilton cycle in G' and consider any portion of the cycle that begins and ends at the selector vertices and encounters no such vertices internally. This portion of the circuit corresponds to those edges of E which are incident to a vertex $v \in V$. Hence, the Q vertices $\{a_1, \ldots, a_Q\}$ divide the Hamilton cycle into Q paths which corresponds to a distinct vertex $v \in V$.

We know that Hamilton cycle must traverse all the vertices in each of the 'cover-testing' component and since the 'cover-testing' component corresponds to an edge $e \in E$ which can only be traversed on a path that is joined to an end-point of e, E must have an end-point among those Q vertices. Therefore, the set of Q vertices forms the Vertex Cover for G.

Conversely, let $V^* \subseteq V$ be a Vertex Cover for G with $|V^*| \leq Q$. A Hamilton cycle for G' can be obtained by choosing all the edges in

(i) E'_{v_i} for $1 \leq i \leq Q$,

(ii) $\{a_i, (v_i, e_{v_i[1]}, 1)\}$, $1 \leq i \leq Q$,

(iii) $\{a_{i+1}, (v_i, e_{v_i[d_G(v_i)]}, 6)\}$, $1 \leq i < Q$, and

(iv) $\{a_1, (v_Q, e_{v_Q[d_G(v_Q)]}, 6)\}$.

This completes the proof. □

We now illustrate the proof of Lemma 4.1.1 with an example. Let graph $G = (V, E)$ be the graph in Figure 4.3 without edge e_{14}.

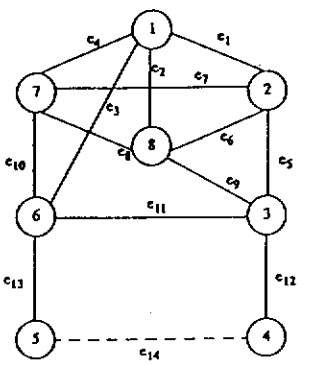
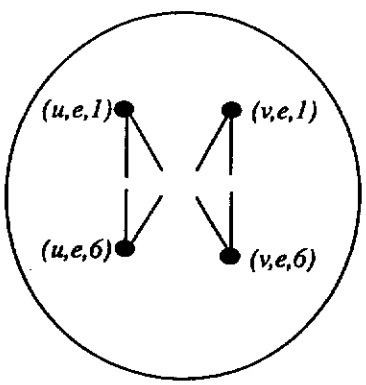


Figure 4.3

Obviously, if there is a Hamilton path between v_4 and v_5 in G then there is a Hamilton cycle in $G + e_{14}$. Note that for this graph the value of $k = 5$.

We obtain our graph $G' = (V', E')$ by replacing each edge $e \in E$ by the structure in Figure 4.1. This structure is represented by the following diagram



in Figures 4.4 and 4.5 where only vertices $(u, e, 1), (u, e, 6), (v, e, 1)$ and $(v, e, 6)$ are shown together with some edges in the component. Each of the component is joined to another component by

$$E'_v = \{\{(v, e_{v[i]}, 6), (v, e_{v[i+1]}, 1)\} : 1 \leq i < d_G(v)\}$$

which are shown in Figures 4.4 and 4.5 by the edges between components e_i and e_j where $(i \neq j)$ and $1 \leq i \leq j \leq 14$.

Further, they are connected to the $a_1, \ldots, a_5$ selector vertices by

$$E'' = \{\{(a_i, (v, e_{v[1]}, 1)\}, \{(a_i, (v, e_{v[d_G(v)]}, 6)\} : 1 \leq i \leq 5, v \in V\}.$$

Hence, the transformation produces G' (Figure 4.4) from G in polynomial time. Therefore, in Figure 4.5, we observe that a Hamilton cycle exists if and only if a Vertex Cover exists.

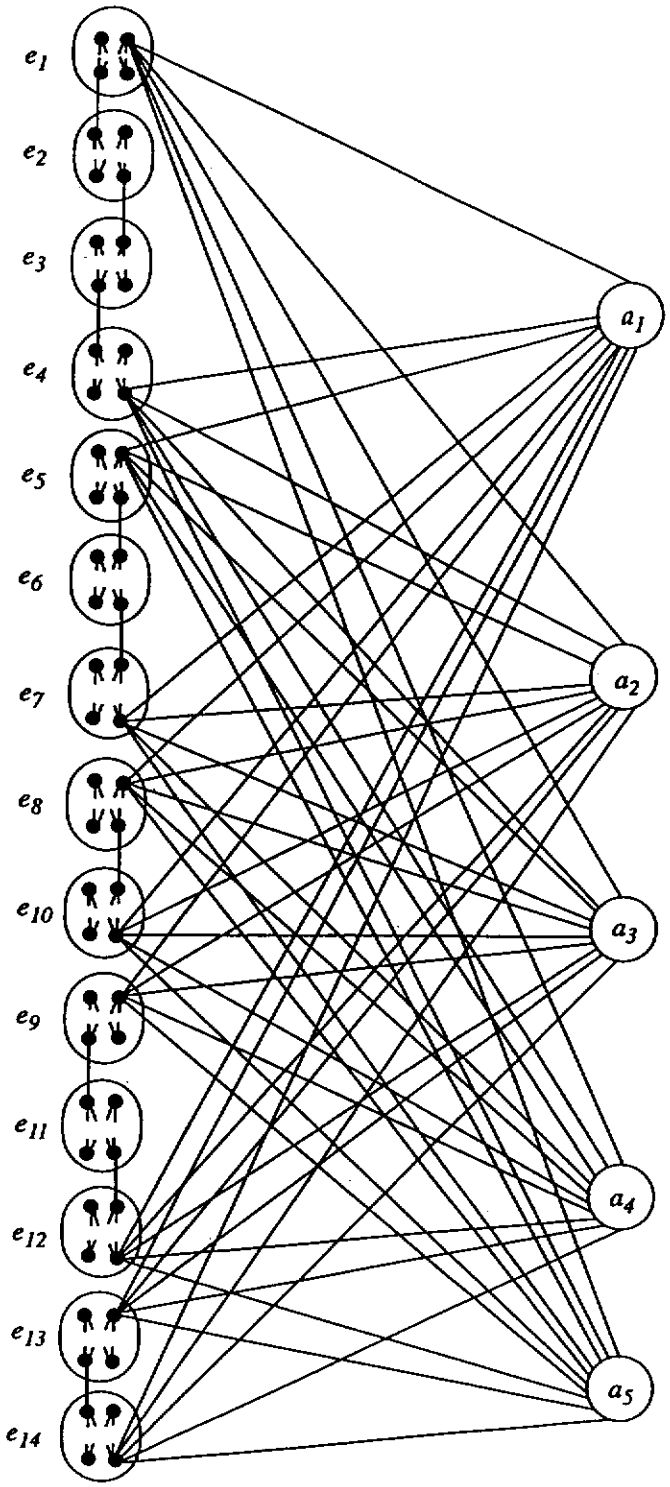


Figure 4.4

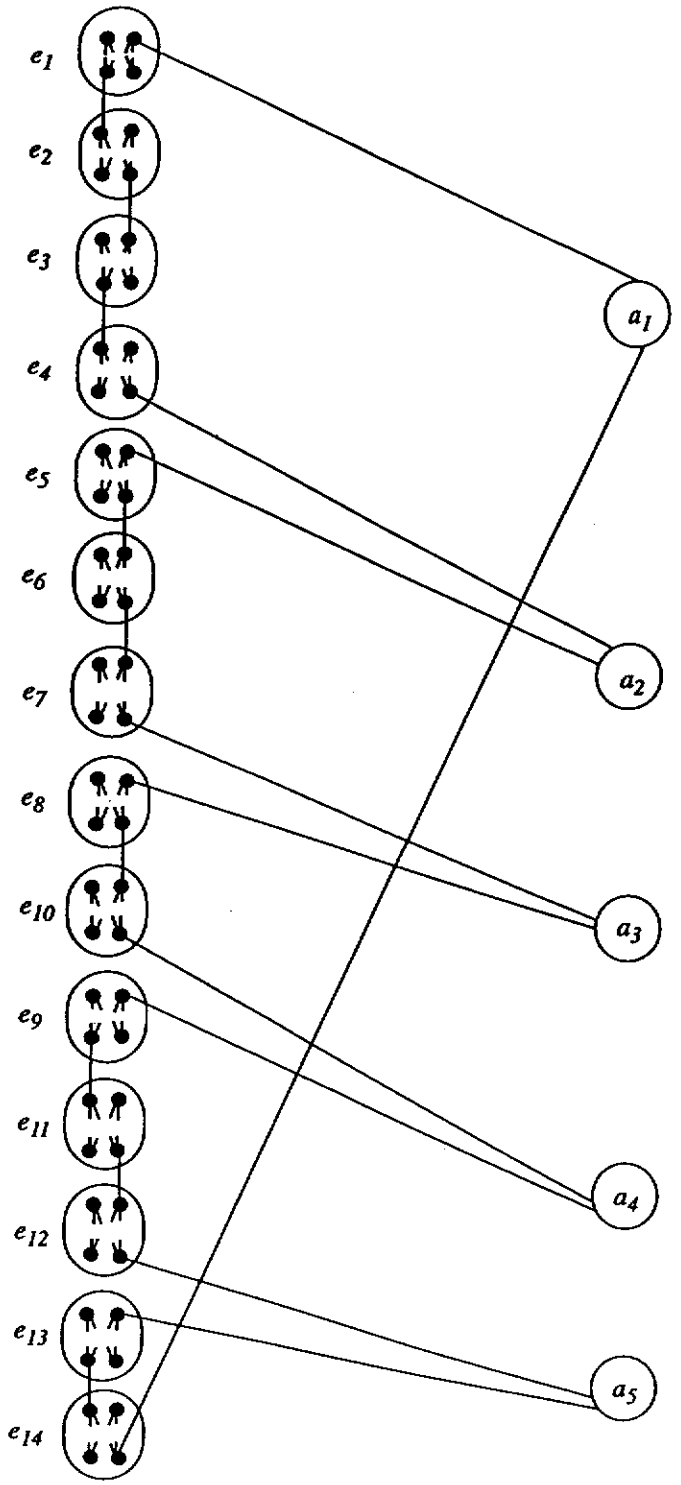


Figure 4.5: Hamilton cycle for graph G in Figure 4.3

From the previous lemma we provide an alternative proof for the complexity of the $(1,3)$-tree using the following theorem.

Theorem 4.1.2 *Given a graph $G = (V, E)$ with maximum degree 3, it is NP-complete to decide if there exists a spanning $(1,3)$-tree T for G.*

PROOF: Let $P^{1,3}$ denote the problem where we are given a graph G with all vertices of degree ≤ 3, and ask if G has a spanning $(1,3)$-tree. Let $G = (V, E)$ be a connected graph with x and y the two vertices of degree 1 and all other vertices of degree 3. The graph B displayed in Figure 4.6 is the basic building block in our constructions. We locally replace each vertex of degree 3 in G with graph B. Let $G' = (V', E')$ be the resulting graph. Observe that G' can be constructed from G in polynomial time.

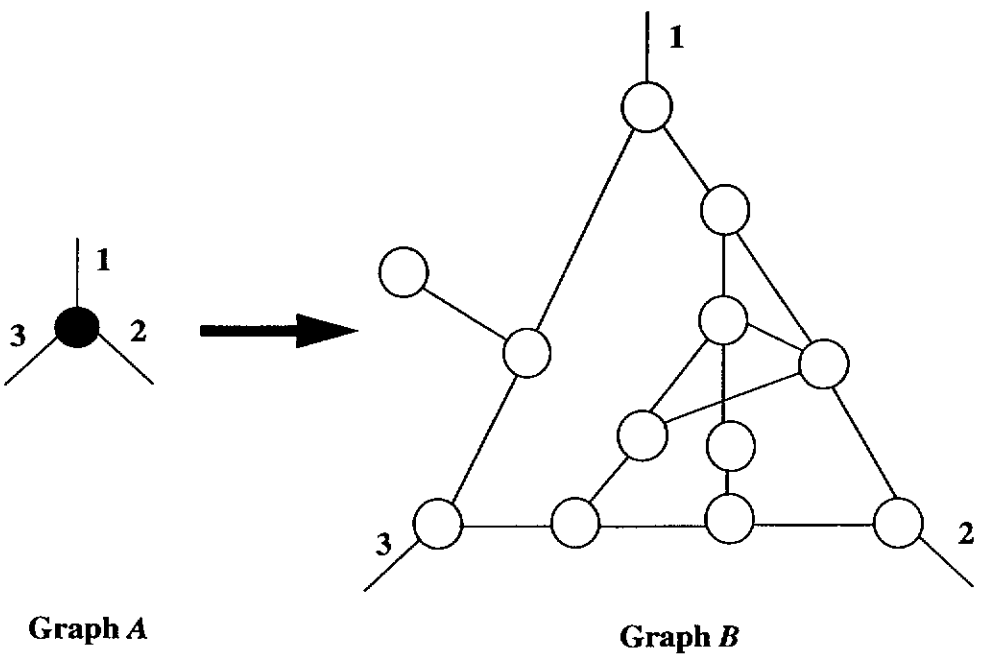


Figure 4.6: Replacing a degree 3 vertex with graph B.

We will establish that there exists a Hamilton path P between x and y in G if and only if there is a spanning $(1,3)$-tree T in G'. Let P be a Hamilton path between x and y in G.

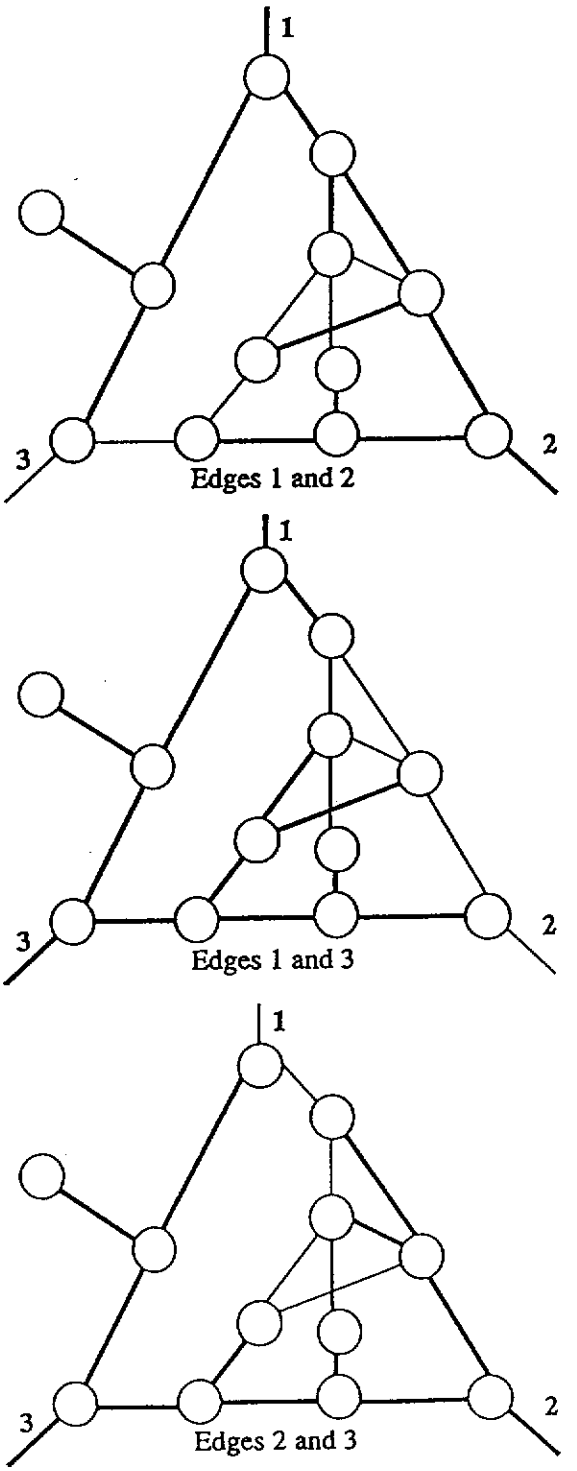
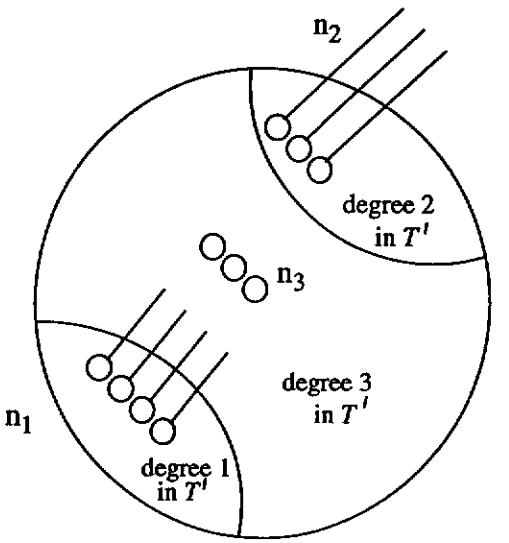


Figure 4.7: Analysis for Block B.

Then P will have to enter and leave any degree 3 vertex in G using two of its incident edges. There are three possible ways P can do that by using: edges 1 and 2, edges 1 and 3, and edges 2 and 3, Whatever the choice, we can identify a $(1,3)$-tree in G'. To see this consider a block B in G'. Listed on the previous page (page 107) are all possible $(1,3)$-trees (bold edges) that can be obtained from B for each of the choices. In all cases a spanning $(1,3)$-tree can be found in G'.

Conversely, suppose that T is a $(1,3)$-tree in G'. Consider a block B in G' and let $T' = B \cap T$. Let n_i, $i = 1, 2, 3$, be the number of vertices of degree i in T'; the situation is as depicted below.



We will show that $n_2 = 2$. Clearly $n_2 \leq 3$. Suppose T' has ω components. Then

$$n_1 + n_2 + n_3 = 12 \tag{4.1}$$

and

$$n_1 + 2n_2 + 3n_3 = 24 - 2\omega. \tag{4.2}$$

Subtracting equation (4.1) from (4.2) we get:

$$n_2 + 2n_3 = 12 - 2\omega. \tag{4.3}$$

We know that $\omega \leq n_2 \leq n_3$ and $n_3 \geq 2$ (from the B block). Hence,

$$\begin{aligned} \omega + 4 &\leq 12 - 2\omega \\ 3\omega &\leq 8 \\ \omega &\leq 2. \end{aligned}$$

From (4.3) we get

$$n_3 = \frac{1}{2}(12 - n_2 - 2\omega).$$

This together with (4.1) yields

$$\begin{aligned} n_1 &= 12 - n_2 - \frac{1}{2}(12 - n_2 - 2\omega) \\ &= \frac{1}{2}(12 - n_2 + 2\omega). \end{aligned}$$

Now since n_1 and n_3 are integers n_2 must be even. Clearly the only possibility is for $n_2 = 2$.

This means that in both cases the spanning $(1,3)$-tree T in G' would always use two of the original edges incident to any vertex of degree 3 in G. Therefore, when we collapse each block B to a single vertex, T becomes a connected subgraph P of G in which every degree 3 vertex in G has exactly two edges in P and x and y have degree one in P. Thus P must be a Hamilton path in G.

This completes the proof of Theorem 4.1.2. □

Theorem 4.1.3 *Given a graph $G = (V, E)$ with maximum degree 4, it is NP-complete to decide if there exists a spanning $(1,4)$-tree T for G.*

PROOF: Let $P^{1,4}$ denote the problem where we are given a graph G with all vertices of degree ≤ 4, and ask if G has a spanning $(1,4)$-tree. Let $G = (V, E)$ be a connected graph with x and y the two vertices of degree 1 and all other vertices

of degree 4. The graph C displayed in Figure 4.8 is the basic building block in our constructions. We locally replace each vertex of degree 4 in G with graph C. Let $G' = (V', E')$ be the resulting graph. Observe that G' can be constructed from G in polynomial time.

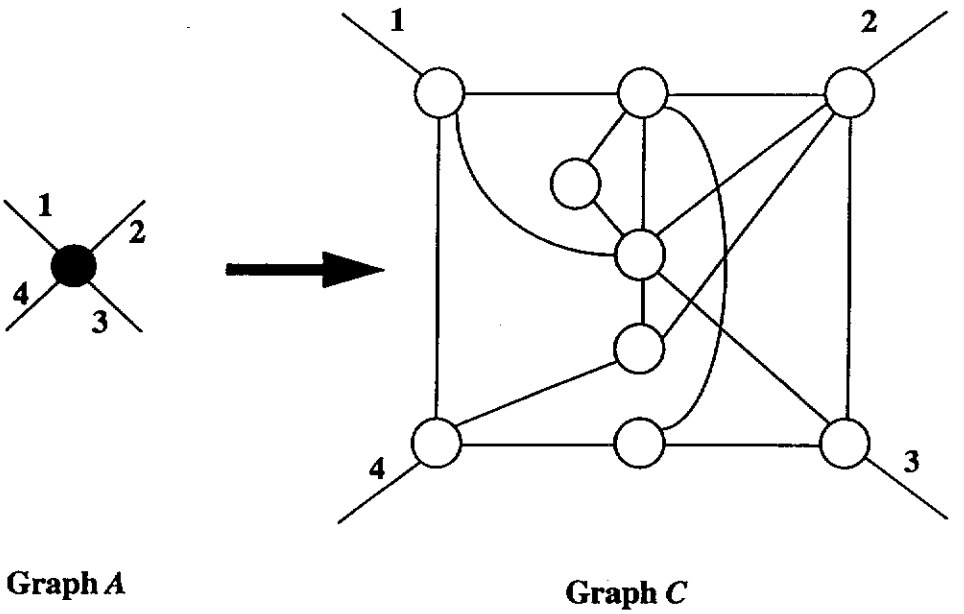


Figure 4.8: Replacing a degree 4 vertex with graph C.

We will establish that there exists a Hamilton path P between x and y in G if and only if there is a spanning $(1,4)$-tree T in G'.

Let P be a Hamilton path between x and y in G. Then P will enter and leave any degree 4 vertex in G using two of its incident edges. There are six possible ways P can do that by using: edges 1 and 2, edges 1 and 3, edges 1 and 4, edges 2 and 3, edges 2 and 4, and edges 3 and 4. Whatever the choice, we can identify a $(1,4)$-tree in G'. To see this consider a block C in G'. Listed on the next page are possible $(1,4)$-trees (bold edges) that can be obtained from C for each of the choices. In all cases a spanning $(1,4)$-tree can be found in G'.

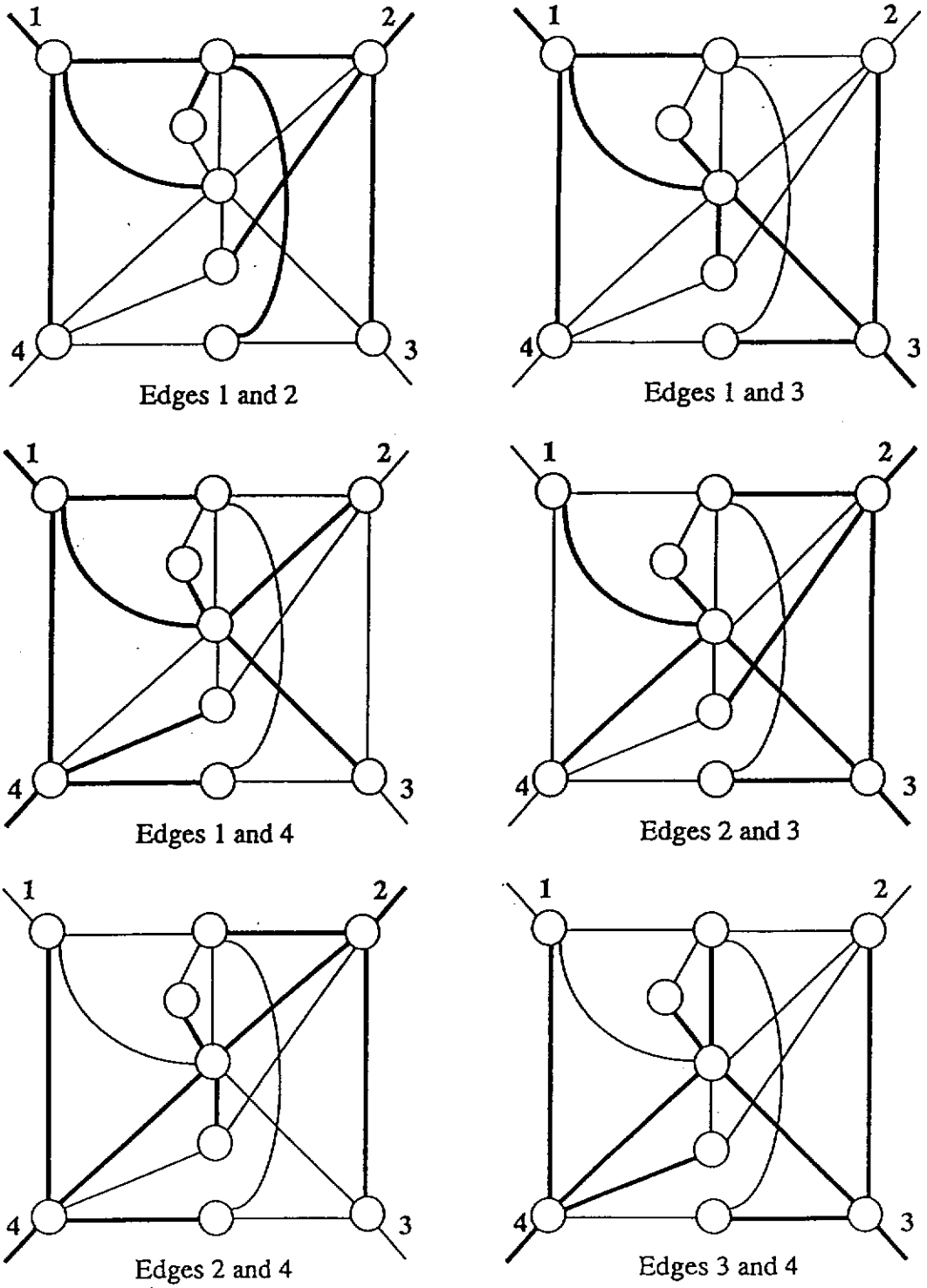
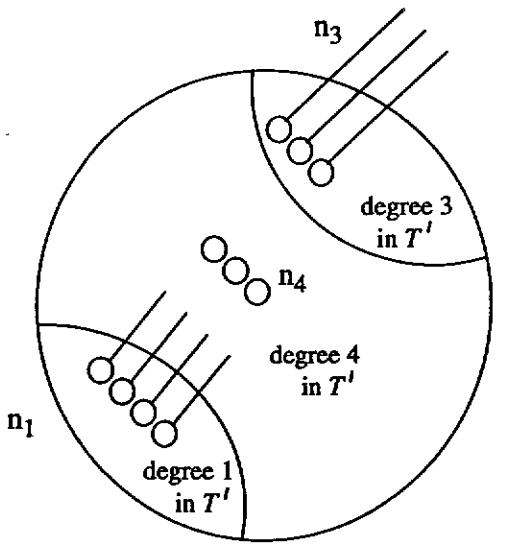


Figure 4.9: Analysis for block C.

Conversely, suppose that T is a $(1,4)$-tree in G'. Consider a block C in G' and let $T' = C \cap T$. Let n_i, $i = 1, 3, 4$, be the number of vertices of degree i in T'; the situation is depicted below.



We will show that $n_3 = 2$. Suppose T' has ω components. Then

$$n_1 + n_3 + n_4 = 9 \tag{4.4}$$

and

$$n_1 + 3n_3 + 4n_4 = 18 - 2\omega. \tag{4.5}$$

Equations (4.4) and (4.5) give

$$2n_3 + 3n_4 = 9 - 2\omega. \tag{4.6}$$

Since T is a tree we must have $\omega \leq n_3 \leq 4$. Further, it is clear from block C that $n_4 \geq 1$. Consequently $4\omega \leq 6$ and hence $\omega \leq 1$. This proves that T' is a tree. Now with $\omega = 1$ equation (4.6) yields

$$n_4 = \frac{7 - 2n_3}{3}$$

into (4.4) we get

$$n_1 = \frac{20 - n_3}{3}.$$

Since n_1 and n_4 are integers the only possibility is for $n_3 = 2$.

This means that the spanning $(1,4)$-tree T in G' would always use two of the original edges incident to any vertex of degree 4 in G. Therefore, when we collapse each block C to a single vertex, T becomes a connected subgraph P of G in which every degree 4 vertex in G has exactly two edges in P and x and y have degree one in P. Thus P must be a Hamilton path in G.

This completes the proof of Theorem 4.1.3. □

We now look at spanning trees whose degrees are from a degree spectrum d where $d = (\delta = d_0, d_1, \ldots, d_m = k)$. Recall that $\delta = d_0 < d_1 < d_2 < \ldots < d_m = k$.

Lemma 4.1.4 *Given a graph $G = (V, E)$ with degree spectrum $(d_1, \ldots, d_m = k)$ and exactly two vertices A and B of degree 1, it is NP-complete to decide if there is a Hamilton path between A and B.*

PROOF: This proof follows from the proof of Lemma 4.1.1 and the reasoning in Garey and Johnson (1979) in their proof that the Hamilton path problem is NP-complete.

We modify the graph of G' obtained at the end of the proof of Lemma 4.1.1 by adding three new vertices a_0, a_{Q+1}, and a_{Q+2} and two edges (a_0, a_1) and (a_{Q+1}, a_{Q+2}). Further, we replace the edges of the form $(a_1, (v, e_{v[d_G(v)]}, 6))$ by $(a_{Q+1}, (v, e_{v[d_G(v)]}, 6))$.

□

Theorem 4.1.5 Given a graph $G = (V, E)$ with degree spectrum $d = (3, 5)$, it is NP-complete to decide if there exists a spanning $(1, 3, 5)$-tree T for G.

PROOF: Let $P^{1,3,5}$ denote the problem where we are given a graph G with degree spectrum $d = (3, 5)$, and ask if G has a spanning $(1, 3, 5)$-tree. Let $G = (V, E)$

be a connected graph with x and y the two vertices of degree 1 and all other vertices of degree 3 or 5. The graphs B and D displayed in Figures 4.6 and 4.10 are the basic building blocks in our constructions. We locally replace each vertex of G having degree 3 and 5 by the graph B and D, respectively. Call the resulting graph $G' = (V', E')$. Observe that G' can be constructed from G in polynomial time.

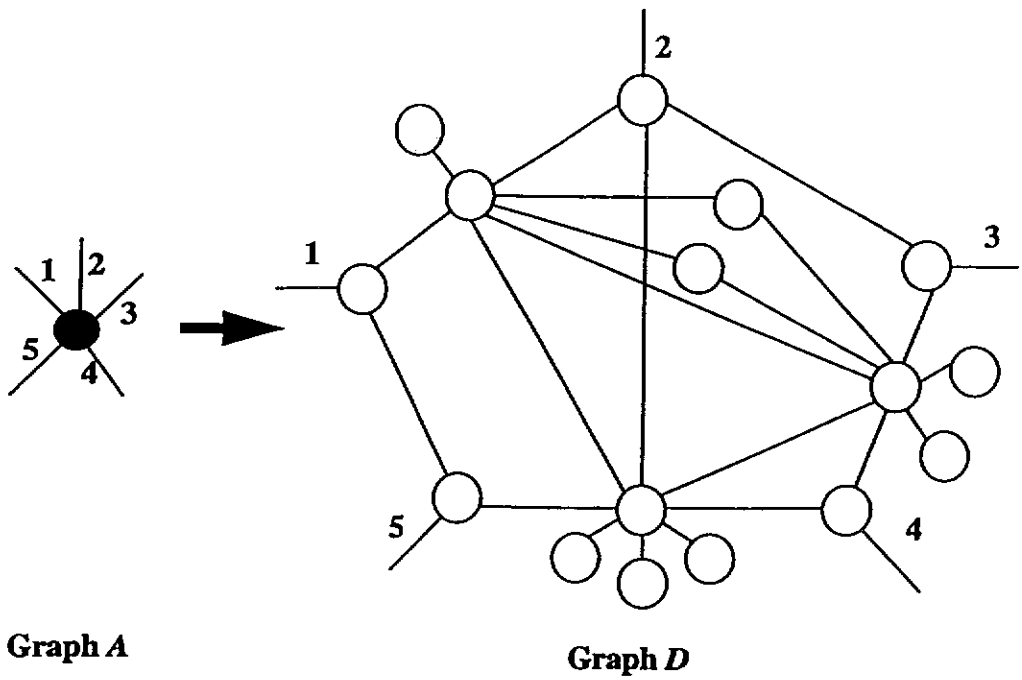
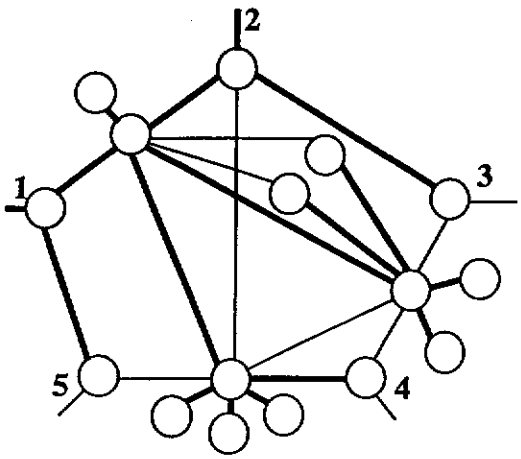


Figure 4.10: Replacing a degree 5 vertex with graph D.

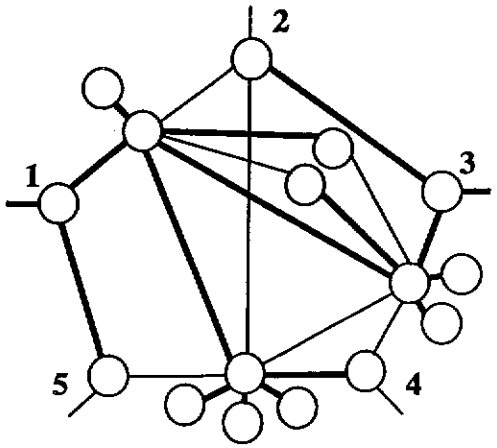
We establish that there exists a Hamilton path P between x and y in G if and only if there is a spanning $(1,3,5)$-tree T in G'.

Let P be a Hamilton path between x and y in G. Then P will enter and leave any degree 3 or 5 vertex in G using two of its incident edges. For vertices of degree 3 in G the proof of Theorem 4.1.2 shows the existence of the $(1,3)$-tree in the associated components of G'.

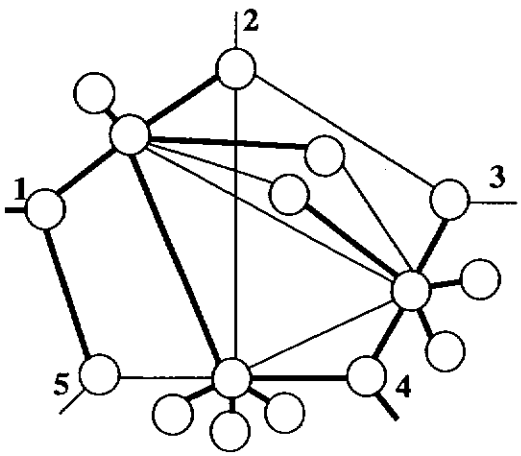
For vertices of degree 5 in G there are ten possible ways P can enter and leave such a vertex by using: edges 1 and 2, edges 1 and 3, edges 1 and 4, edges 1 and 5, edges 2 and 3, edges 2 and 4, edges 2 and 5, edges 3 and 4, edges 3 and 5, and edges 4 and 5. Whatever the choice, we can easily identify a $(1,3,5)$-tree in G'. To see this consider a block D in G'. Listed in Figure 4.11 are possible $(1,3,5)$-trees (bold edges) that can be obtained for D for each of the choices. In all cases a spanning $(1,3,5)$-tree can be found in G'.



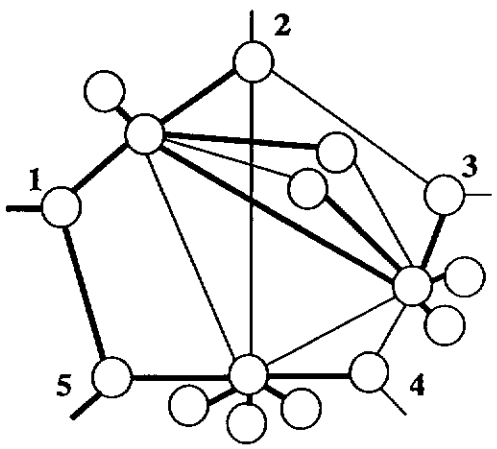
Edges 1 and 2



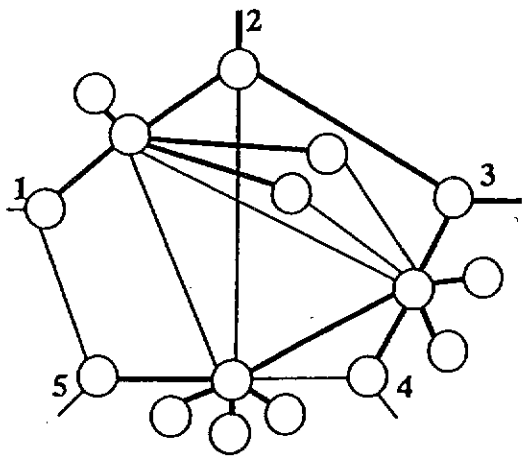
Edges 1 and 3



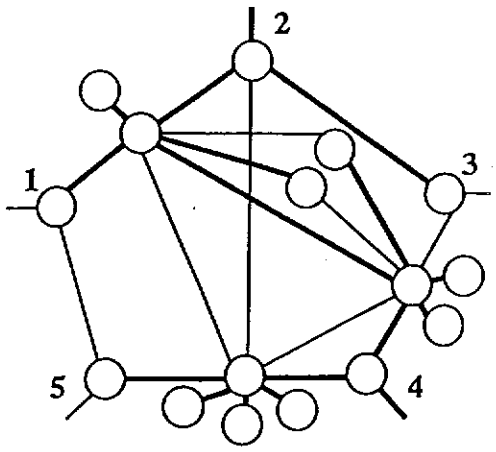
Edges 1 and 4



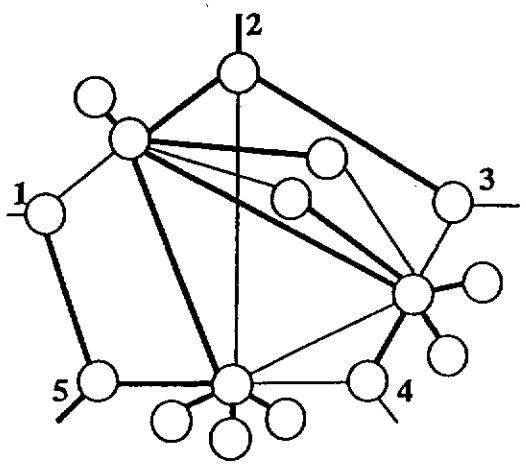
Edges 1 and 5



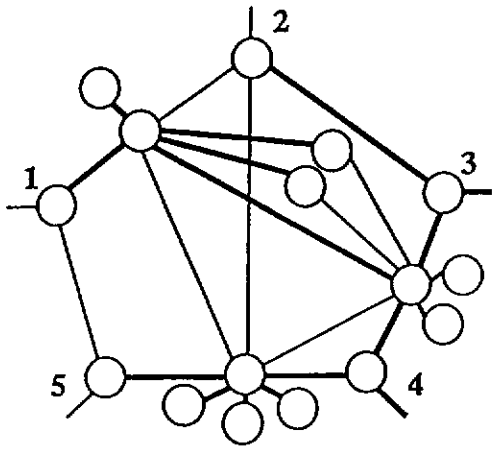
Edges 2 and 3



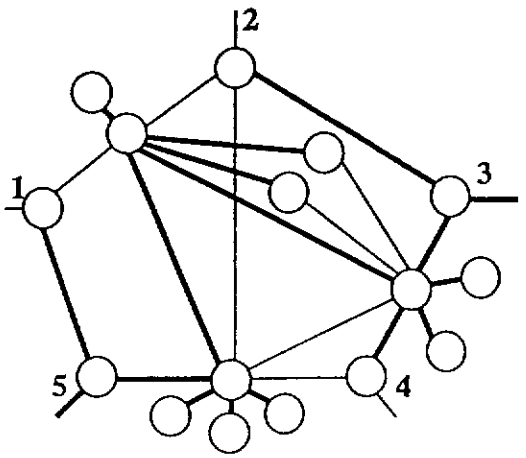
Edges 2 and 4



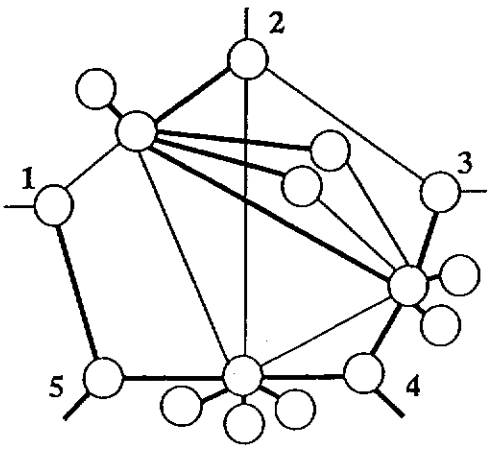
Edges 2 and 5



Edges 3 and 4



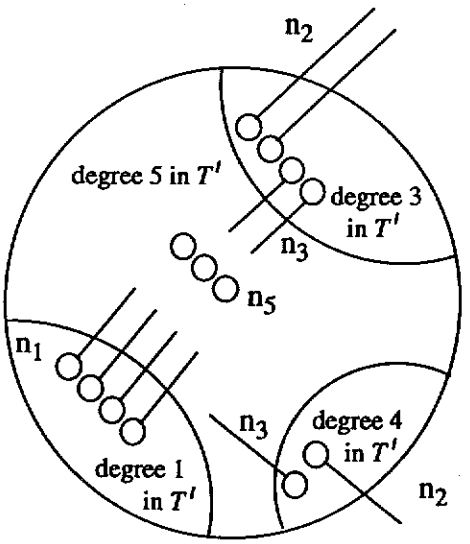
Edges 3 and 5



Edges 4 and 5

Figure 4.11: Analysis for block D.

Conversely, suppose that T is a $(1,3,5)$-tree in G'. For the B blocks, following from the proof of Theorem 4.1.2, we know that T must use only 2 of the original edges beside any vertex of degree 3 in G. For the D blocks let $T' = D \cap T$. Let n_i, $i = 1$, 2, 3, 5, be the number of vertices of degree i in T'; the situation is depicted below.



We first show that $n_2 = 2$ or 4. Suppose T' has ω components. Then

$$n_1 + n_2 + n_3 + n_5 = 16 \tag{4.7}$$

and

$$n_1 + 2n_2 + 3n_3 + 5n_5 = 32 - 2\omega. \tag{4.8}$$

Equation (4.4) - (4.5)

$$n_2 + 2n_3 + 4n_5 = 16 - 2\omega. \tag{4.9}$$

Since T is a tree we must have $\omega \leq n_2 \leq 5$. Further, it is clear from block D that $n_5 \geq 1$ and $n_3 + n_5 \geq 3$. Consequently,

$$\begin{aligned} \omega + 2(n_3 + n_5) + 3 &\leq 16 - 2\omega \\ 3\omega &\leq 7 \\ \omega &\leq 2. \end{aligned}$$

From (4.9) we get

$$n_5 = \frac{1}{4}(16 - 2\omega - n_2 - 2n_3)$$

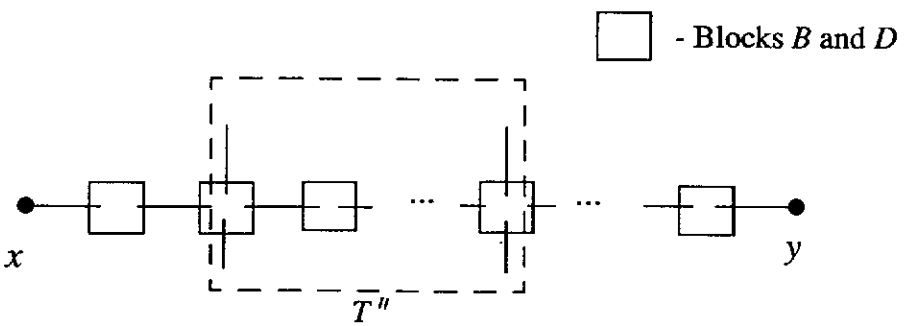
into (4.6) we get

$$\begin{aligned} n_1 &= 32 - 2\omega - 2n_2 - 3n_3 - \frac{5}{4}(16 - 2\omega - n_2 - 2n_3) \\ &= 12 + \frac{1}{2}\omega - \frac{3}{4}n_2 + \frac{1}{2}n_3 \\ &= \frac{1}{4}(48 + 2\omega - 3n_2 - 2n_3). \end{aligned}$$

Since n_1 and n_5 are integers, it is clear that n_2 must be even. Hence, $n_2 = 2$ or 4 as required.

If $n_2 = 2$ for each block, then collapsing each block to a single vertex our T becomes a connected subgraph P of G in which each degree 3 or 5 vertex has exactly two edges in P and x and y have degree 1 in P. Thus P must be a Hamilton path G, as required.

The only case that we need to consider further is when for some D blocks of G' $n_2 = 4$. If $n_2 = 4$ then tree T will look like:



Tree T

For this case consider starting at x and moving along T deleting blocks with $n_2 = 2$ until the first D-block is reached with $n_2 = 4$. We repeat this process starting at y. Call the resulting graph T''. Now we know that T'' must be a tree with each block having at least two edges from G. However, any graph with this degree property must have cycles, hence a contradiction. Therefore, no block of T can have $n_2 = 4$.

This means that the spanning $(1,3,5)$-tree T in G' would always use two of the original edges incident to any vertex of degree 3 or 5 in G. Therefore, when we collapse each block B and block D to a single vertex, T becomes a connected subgraph P of G in which every degree 3 or 5 vertex in G has exactly two edges in P and x and y have degree one in P. Thus P must be a Hamilton path in G.

This completes the proof of Theorem 4.1.5. □

In problems relating to trees there is always interest in establishing the number of vertices of a specified degree. We now provide such counting for the $(1,k)$-tree.

Lemma 4.1.6 *If graph G has a spanning $(1,k)$-tree then $\nu(G) = (k-1)n_k + 2$ vertices, where n_k is the number of vertices of degree k.*

PROOF: We know that

$$2(\nu - 1) = n_k k + (\nu - n_k)(1)$$

Hence,

$$\begin{aligned} \nu &= n_k k - n_k + 2 \\ &= (k-1)n_k + 2, \end{aligned}$$

as required. □

As a corollary we have:

Corollary 4.1.7 *If graph G has a spanning $(1,k)$-tree then G has exactly $L = (k-2)n_k + 2$ degree one vertices where n_k is the number of vertices of degree k.*

PROOF: We know in the $(1,k)$-tree there are only vertices of degree 1 and k. Further, there are:

$$\begin{aligned} L &= \nu - n_k \\ &= (k-1)n_k + 2 - n_k \\ &= (k-2)n_k + 2 \end{aligned}$$

vertices of degree 1, as required. □

The following result provides bounds for the number of vertices of degree 1 in trees with a specified degree spectrum.

Theorem 4.1.8 *Let G be a graph with maximum degree k containing a $(d_0 = 1, d_1, \ldots, d_{m-1}, d_m = k)$-tree T with $d_0 < d_1 < \ldots < d_m$ and $m \geq 3$. Then the number of vertices of degree 1, n_0, in T satisfies*

$$\begin{aligned} \frac{(d_1-2)\nu + 2 + (d_2 - d_1)(m-1) + \sum_{i=3}^{m}(d_i - d_2)}{(d_1 - 1)} &\leq n_0 \\ &\leq \frac{(d_m - 2)\nu + 2 - \sum_{i=1}^{m-1}(d_m - d_i)}{(d_m - 1)}. \end{aligned}$$

PROOF: Let n_i denote the number of vertices of degree d_i in T. We have

$$\nu = n_0 + n_1 + \ldots + n_m$$

and

$$2\nu - 2 = n_0 + n_1 d_1 + \ldots + n_m d_m.$$

Consequently,

$$n_0 = n_1(d_1 - 2) + n_2(d_2 - 2) + \ldots + n_m(k - 2) + 2.$$

We can write it as:

$$\begin{aligned} n_0 &= (d_1 - 2)[n_1 + n_2 + \ldots + n_m] + n_2(d_2 - d_1) + \\ &\quad n_3(d_3 - d_1) + \ldots + n_m(d_m - d_1) + 2 \\ &= (d_1 - 2)(\nu - n_0) + \sum_{i=2}^{m} n_i(d_i - d_1) + 2. \end{aligned}$$

Thus

$$n_0(d_1 - 1) = (d_1 - 2)\nu + \sum_{i=2}^{m} n_i(d_i - d_1) + 2. \tag{4.10}$$

Let

$$F = \sum_{i=2}^{m} n_i(d_i - d_1).$$

Observe that for $2 \leq j \leq m - 1$ we can write

$$\sum_{i=j}^{m} n_i(d_i - d_{j-1}) = (d_j - d_{j-1})(\nu - \sum_{i=0}^{j-1} n_i) + \sum_{i=j+1}^{m} n_i(d_i - d_j).$$

Consequently,

$$\begin{aligned} F &= \sum_{i=2}^{m} n_i(d_i - d_1) \\ &= (d_2 - d_1)(\nu - n_0 - n_1) + \sum_{i=3}^{m} n_i(d_i - d_2) \end{aligned}$$

$$
\begin{aligned}
&= (d_2 - d_1)(\nu - n_0 - n_1) + (d_3 - d_2)(\nu - n_0 - n_1 - n_2) + \ldots + \\
&\quad (d_{m-1} - d_{m-2})(\nu - n_0 - n_1 - \ldots - n_{m-2}) + (d_m - d_{m-1})n_m \\
&= (d_m - d_1)(\nu - n_0 - n_1) - \sum_{i=2}^{m-1} n_i(d_m - d_i).
\end{aligned}
\tag{4.11}
$$

Now (4.9) and (4.10) together yield

$$
\begin{aligned}
n_0(d_1 - 1) - 2 &= (d_1 - 2)\nu + (d_m - d_1)(\nu - n_0 - n_1) - \\
&\quad \sum_{i=2}^{m-1} n_i(d_m - d_i).
\end{aligned}
$$

That is,

$$
\begin{aligned}
n_0(d_1 - 1 + d_m - d_1) - 2 &= (d_1 - 2 + d_m - d_1)\nu - n_1(d_m - d_1) - \\
&\quad \sum_{i=2}^{m-1} n_i(d_m - d_i),
\end{aligned}
$$

or

$$
n_0(d_m - 1) - 2 = (d_m - 2)\nu - \sum_{i=1}^{m-1} n_i(d_m - d_i).
$$

Hence,

$$
n_0 = \frac{(d_m - 2)\nu + 2 - \sum_{i=1}^{m-1} n_i(d_m - d_i)}{(d_m - 1)}.
$$

Using the fact that $n_i \geq 1$ for all i we get

$$
n_0 \leq \frac{(d_m - 2)\nu + 2 - \sum_{i=1}^{m-1}(d_m - d_i)}{(d_m - 1)}.
$$

This establishes the upper bound.

Similarly observe that for $2 \leq j \leq m-1$ we can also write

$$\sum_{i=j}^{m} n_i(d_i - d_{j-1}) = (d_j - d_{j-1})(\nu - \sum_{i=0}^{1} n_0) + \sum_{i=j}^{m-1} n_i(d_i - d_j).$$

Consequently,

$$\begin{aligned} F &= \sum_{i=2}^{m} n_i(d_i - d_1) \\ &= (d_m - d_1)(\nu - n_0 - n_1) + \sum_{i=2}^{m-1} n_i(d_i - d_m) \\ &= (d_m - d_1)(\nu - n_0 - n_1) + (d_{m-1} - d_m)(\nu - n_0 - n_1 - n_m) + \\ &\quad (d_{m-2} - d_{m-1})(\nu - n_0 - n_1 - n_m - n_{m-1}) + \\ &\quad (d_{m-3} - d_{m-2})(n_{m-3} + \ldots + n_2) + \ldots + n_2(d_2 - d_3) \\ &= (d_2 - d_1)(\nu - n_0 - n_1) + \sum_{i=3}^{m} n_i(d_i - d_2). \end{aligned} \tag{4.12}$$

Now (4.9) and (4.12) yield

$$n_0(d_1 - 1) = (d_1 - 2)\nu + 2 + (d_2 - d_1)(\nu - n_0 - n_1) + \sum_{i=3}^{m} n_i(d_i - d_2).$$

That is,

$$n_0(d_1 - 1 + d_2 - d_1) = (d_1 - 2 + d_2 - d_1)\nu + 2 + n_1(d_1 - d_2) + \sum_{i=3}^{m} n_i(d_i - d_2),$$

or

$$\begin{aligned} n_0(d_2 - 1) &= (d_2 - 2)\nu + 2 + n_1(d_1 - d_2) + \sum_{i=3}^{m} n_i(d_i - d_2) \\ &= (d_2 - 2)\nu + 2 + (\nu - n_0 - n_2 - \ldots - n_m)(d_1 - d_2) + \\ &\quad \sum_{i=3}^{m} n_i(d_i - d_2). \end{aligned}$$

Hence,

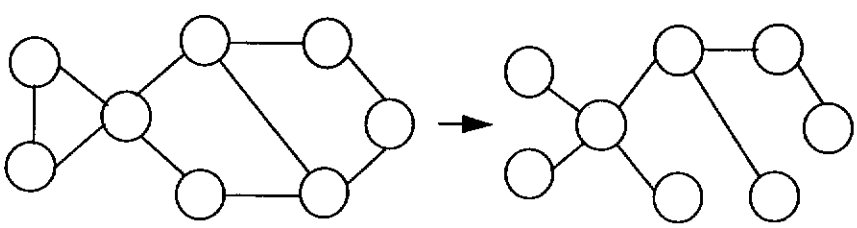
$$\begin{aligned} n_0(d_2 - 1 + d_1 - d_2) &= (d_2 - 2 + d_1 - d_2)\nu + 2 + (n_2 \ldots + n_m)(d_2 - d_1) + \\ &\quad \sum_{i=3}^{m} n_i(d_i - d_2) \\ n_0(d_1 - 1) &= (d_1 - 2)\nu + 2 + (d_2 - d_1)\sum_{i=2}^{m} n_i + \sum_{i=3}^{m} n_i(d_i - d_2) \\ n_0 &= \frac{(d_1 - 2)\nu + 2 + (d_2 - d_1)\sum_{i=2}^{m} n_i + \sum_{i=3}^{m} n_i(d_i - d_2)}{(d_1 - 1)}. \end{aligned}$$

Using the fact that $n_i \geq 1$, we get

$$n_0 \geq \frac{(d_1 - 2)\nu + 2 + (d_2 - d_1)(m - 1) + \sum_{i=3}^{m}(d_i - d_2)}{(d_1 - 1)}.$$

This establishes the lower bound and completes the proof. □

We show with the following example that these bounds are achievable.



Graph G Tree T

For the lower bound:

$$\frac{(d_1 - 2)\nu + 2 + (d_2 - d_1)(m - 1) + \sum_{i=3}^{m}(d_i - d_2)}{(d_1 - 1)}$$

$$= \frac{0+2+(1)(2)+1}{1}$$
$$= 5.$$

For the upper bound:

$$\frac{(d_m-2)\nu+2-\sum_{i=1}^{m-1}(d_m-d_i)}{(d_m-1)}$$
$$= \frac{(2)(8)+2-1-2}{3}$$
$$= 5.$$

As a corollary to Theorem 4.1.8 we have:

Corollary 4.1.9 *Let G) be a graph with maximum degree k containing a $(d_0 = 1, d_1, \ldots, d_{m-1}, d_m = k)$-tree T with $d_0 < d_1 < \ldots < d_m$ and $m \geq 3$. Then*

$$1 \leq n_i \leq \nu - \frac{(d_1-2)\nu+2+(d_2-d_1)(m-1)+\sum_{i=3}^{m}(d_i-d_2)}{(d_1-1)} - (m-1)$$

where n_i is the number of vertices of degree d_i and $1 \leq i \leq m$.

PROOF: We know that

$$n_1 + \ldots + n_m = \nu - n_0.$$

Using the results of Theorem 4.1.8 we get,

$$\nu - \frac{(d_1-2)\nu+2+(d_2-d_1)(m-1)+\sum_{i=3}^{m}(d_i-d_2)}{(d_1-1)} \leq$$
$$n_1 + \ldots + n_m \leq \nu - \frac{(d_m-2)\nu+2-\sum_{i=1}^{m-1}(d_m-d_i)}{(d_m-1)}.$$

We know there has to be at least one n_i vertex, and for any particular i, $1 \leq i \leq m$, and assume there are only one vertex of the other degrees, we get

$$1 \leq n_i \leq \nu - \frac{(d_1-2)\nu + 2 + (d_2-d_1)(m-1) + \sum_{i=3}^{m}(d_i-d_2)}{(d_1-1)} - (m-1).$$

□

4.2 Algorithms for the $(1,k)$-tree Problem

The $(1,k)$-tree problem is easily formulated as a Mixed Integer Linear Programming (MILP) problem. One formulation is:

$$Minimise \sum_i \sum_j c_{ij} x_{ij} \tag{4.13}$$

subject to

$$\sum_{i,j} x_{ij} = n - 1 \tag{4.14}$$

$$\sum_{i,j \in V'} x_{ij} \leq |V'| - 1, \qquad \forall \quad \emptyset \neq V' \subseteq V \tag{4.15}$$

$$\sum_{j=1, i \neq j}^{n} x_{ij} - (k-1) l_i = 1, \qquad 1 \leq i \leq n \tag{4.16}$$

$$x_{ij} = 0 \, or \, 1, \qquad 1 \leq i \neq j \leq n \tag{4.17}$$

$$l_i = 0 \, or \, 1, \qquad 1 \leq i \leq n \tag{4.18}$$

(4.13) - (4.18) is a valid formulation of the $(1,k)$-tree problem and can be established using a straightforward argument. The objective function is to find the minimum cost tree. For our purposes the edges all have costs $c_{ij} = 1$. Constraint (4.14) ensures that $(n-1)$ edges are chosen. Constraints (4.15) eliminate cycles whilst constraints (4.16) restricts the degrees in the tree to either 1 or k. Variable x_{ij} takes on the value 1 if the edge (i,j) is included in the tree and 0 otherwise.

Exact methods for solving the above problem would be useful only for small size problems. We now propose two simple heuristics for finding, in a given connected graph G of maximum degree k, a $(1,k)$-tree. Our heuristics first check the graphs for certain properties. The graphs have to have the following attributes:

- connected,
- maximum degree k,
- $\frac{(\nu-2)}{(k-1)}$ has to be an integer (from Lemma 4.1.6), and
- $\frac{(\nu-2)}{(k-1)} \geq n_k$ (from Lemma 4.1.6).

If the graph does not satisfy any of the above properties we know the graph does not have a $(1,k)$-tree. Heuristics 4.2.1 and 4.2.2 presented below have gone through the property checking condition as a preprocessing step.

Our first heuristic builds up the $(1,k)$-tree starting with a vertex of degree k. We know that all the edges incident to this vertex have to be in the tree hence we include these edges and its neighbouring vertices in the tree. We then look at the included neighbouring vertices of degree k and repeat the process of including all its incident edges and neighbours in the tree provided their

inclusion does not form a cycle. The heuristic is presented as follows:

Heuristic 4.2.1:

Step 1. Select a vertex of degree k.

Step 2. Include all its available incident edges and neighbouring vertices in the tree.

Step 3. Repeat

For all the selected neighbouring vertices of degree k include its remaining $(k-1)$ available edges in the tree if their inclusion does not form a cycle

until

all the available edges incident to all the selected neighbouring vertices of degree k are included in the tree.

If a $(1,k)$-tree is found

stop

else if no more available edges can be added

clear the tree and start again at Step 1 by choosing another vertex of degree k not previously chosen as the starting vertex to build up the tree.

Step 4. If all the degree k vertices have been used as a starting vertex stop. The heuristic has failed to find a $(1,k)$-tree.

Our second heuristic partitions the vertices into two sets with one set consisting only of the vertices degree k. It uses the fact that all the vertices in the second set have to be of degree one in the tree and hence deletes all the internal edges in that set. Consequently, if there is a vertex of degree one, we know that its neighbour is in the first set and has to be of degree k in the tree. The heuristic uses this idea to build up the $(1,k)$-tree sequentially. More precisely:

Heuristic 4.2.2:

Step 1. Partition the vertices of graph G into two Sets: (1) vertices of degree k, and (2) vertices of degree $\leq (k-1)$.

Step 2. Delete the internal edges in Set 2.

Step 3. If there is no vertex of degree 1 in the Set 2 go to Step 5.

Step 4. For a vertex of degree 1 in Set 2 include it and its incident edge in the tree. Then include its neighbour that is in Set 1 in the tree. Further, for this included vertex in Set 1 include the remaining $(k-1)$ available edges in the tree. Keep repeating this step until all the degree 1 vertices in Set 2 have been included in the tree. If a $(1,k)$-tree has been found stop.

Step 5. Find the vertex in Set 2 that has the least number of incident edges connecting it to vertices in Set 1. If no such vertex is found go to Step 6, else select one of its available edges and include it in the tree. For its connected vertex in Set 1 include it and its remaining $(k-1)$ edges in the tree. Then repeat Step 4 until all the degree 1 vertices in Set 2 are in the tree.

Step 6. Find a vertex in Set 1 with the least number of incident edges included in the tree. Let this vertex be a degree 1 in the tree by putting it in Set 2. Let its neighbour that is in Set 1 be of degree k in the tree by including its remaining $(k-1)$ edges in the tree. Repeat Step 6 until no more available edges can be added.

Step 7. If a $(1,k)$-tree is found stop. Else clear the subgraph obtained from Steps 5 and 6 and select another available edge in Step 5 of the vertex in Set 2 that has the least number of incident edges connecting it to vertices in Set 1. Repeat Steps 5, 6 and 7 until all the available incident edges in Step 5 have been used as the edge to build up the

tree. If after that no $(1,k)$-tree is found Heuristic 4.2.2 fails.

Remark: Heuristic 4.2.2 cannot be applied to regular graphs since the vertices cannot be partitioned into sets of degree k and $\leq (k-1)$.

We implemented Heuristics 4.2.1 and 4.2.2 in the C programming language on a SUN SPARC 2 workstation operating at 28.5 MIPS, the same way as the implementation in Chapter 3. We simulated connected random graphs that have maximum degree k with $\frac{(\nu-2)}{(k-1)}$ being an integer $\geq n_k$. For the number of vertices in the graphs we followed Table 4.1 (a result from Lemma 4.1.6).

The problem we faced with those random graphs was that a huge portion of them do not have a $(1,k)$-tree and our heuristics were unsuccessful in this sense. To get around this problem we implemented a random $(1,k)$-tree generator. What this generator does is that given n, the number of vertices in the graph, together with the values of k and n_k, it first generates the required $(1,k)$-tree and then randomly adds edges to the graph until it satisfies the required density. This way all the generated graphs have a $(1,k)$-tree. For each order we generated 100 such random graphs.

Our results showed that Heuristic 4.2.2 managed to produce a $(1,k)$-tree about 90% of the time with average minimum and maximum CPU times (secs) of 0.0008 and 0.2525. Heuristic 4.2.1, however, did not fare as well with only 71% success rate and average minimum and maximum CPU times of 0.0002 and 0.2029.

k \ n_k	3	4	5	6	7	8	9	10	11
3	8	10	12	14	16	18	20	22	24
4	11	14	17	20	23	26	29	32	35
5	14	18	22	26	30	34	38	42	46
6	17	22	27	32	37	42	47	52	57
7	20	26	32	38	44	50	56	62	68
8	23	30	37	44	51	58	65	72	79
9	26	34	42	50	58	66	74	82	90
10	29	38	47	56	65	74	83	92	101
11	32	42	52	62	72	82	92	102	112
12	35	46	57	68	79	90	101	112	123
13	38	50	62	74	86	98	110	122	134
14	41	54	67	80	93	106	119	132	145
15	44	58	72	86	100	114	128	142	156
16	47	62	77	92	107	122	137	152	167
17	50	66	82	98	114	130	146	162	178
18	53	70	87	104	121	138	155	172	189
19	56	74	92	110	128	146	164	182	200
20	59	78	97	116	135	154	173	192	211
21	62	82	102	122	142	162	182	202	222
22	65	86	107	128	149	170	191	212	233
23	68	90	112	134	156	178	200	222	244
24	71	94	117	140	163	186	209	232	255
25	74	98	122	146	170	194	218	242	266
26	77	102	127	152	177	202	227	252	277
27	80	106	132	158	184	210	236	262	288
28	83	110	137	164	191	218	245	272	299
29	86	114	142	170	198	226	254	282	310
30	89	118	147	176	205	234	263	292	321

Table 4.1: $\nu=(k-1)n_k+2$

Chapter 5

Graph Partitioning Problem

In this chapter we investigate the Graph Partitioning problem which is: given a weighted graph G partition the vertices into b sets in such a way that the total cost of the edges between sets is minimised. This problem has generated a tremendous amount of interest since Kernighan and Lin (1970) developed an efficient heuristic to partition vertices of the graphs into two sets. Since then some authors (Gilbert and Zmijewski 1987, Hendrickson and Leland 1995a) have tried to improve upon this heuristic by applying some additional refinements to it. Others have also approached the problem from the point of eigenvalues (Falkner et al. 1994, Pothen et al. 1990) and simulated annealing (Johnson et al. 1989). This problem has also attracted the interest of computer scientists because of its wide application in pattern recognition (Blake 1994) and parallel processing (Barnard and Simon 1993, Diniz et al. 1995, Gilbert and Zmijewski 1987, Hendrickson and Leland 1995b, Savage and Wloka 1991, Simon 1991, Vanderstraeten et al. 1995, Walshaw et al. 1995).

The Graph Partitioning problem can be formulated using a Mixed Integer Linear Programming (MILP). One formulation is:

$$Minimise \sum_i \sum_j c_{ij} x_{ij} \tag{5.1}$$

subject to

$$\sum_i y_{ij} \leq \frac{n}{b}, \qquad j = 1, 2, \ldots, b. \tag{5.2}$$

$$\sum_j y_{ij} = 1, \qquad i = 1, 2, \ldots, n. \tag{5.3}$$

For each edge (i, j) of G,

$$x_{ij} \leq (1 - y_{ik}) + (1 - y_{jk}), \qquad 1 \leq k \leq b. \tag{5.4}$$

$$x_{ij} \geq y_{ik} + y_{jl} - 1, \qquad 1 \leq k \neq l \leq b. \tag{5.5}$$

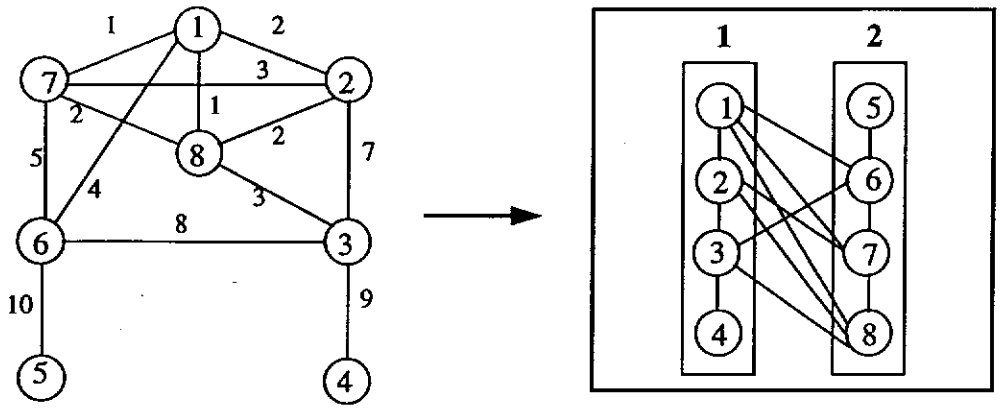
$$x_{ij},\ y_{ij} = 0 \, or \, 1, \qquad i = 1, \ldots, n,\ \ j = 1, \ldots, n.$$

The objective of the Graph Partitioning problem is to find an optimal partitioning for all the vertices in the graph in such a way that the cost of the edges that have end-points in different sets is minimum. Constraints (5.2) ensure that the vertices are divided into no more than b sets, with each set having at most $\frac{n}{b}$ vertices. Constraints (5.3) make sure that vertex i is only in one set. Constraints (5.4) and (5.5) restrict the edges to only those on the cut sets.

The following example (graph on the next page) illustrates the MILP. We use the graph in Figure 4.3 and partition it into two sets. We have assigned weights to the edges of the graph to show the partitions.

The objective function (5.1) is now

$$Minimise \sum_i \sum_j c_{ij} x_{ij}.$$



Constraints (5.2) is now

$$\sum_{8} y_{ij} \leq \frac{8}{2} = 4, \qquad j = 1, 2$$

which means there are no more than four vertices in each of the two sets.

Constraints (5.3) is

$$\sum_{2} y_{ij} = 1, \qquad i = 1, 2, \ldots, 8$$

which means vertex i can only be in either one of the two sets.

For each of the (i, j) edge of G, constraints (5.4) and (5.5) are

$$x_{ij} \leq (1 - y_{ik}) + (1 - y_{jk}), \qquad 1 \leq k \leq 2$$

and

$$x_{ij} \geq y_{ik} + y_{jl} - 1, \qquad 1 \leq k \neq l \leq 2$$

which means x_{ij} is 1 if it is an edge with one end-point in set 1 and the other in set 2, and 0 otherwise. As for our example the vertices are partitioned into two sets with the objective function having a value of 22. Observe that this is not the optimum partition.

The Graph Partitioning problem is equivalent to the problem when the objective is to maximise the weights of the edges in the same set. We can write such a MILP formulation as:

$$Maximise \sum_i \sum_j c_{ij} x_{ij} \tag{5.6}$$

$$\sum_i y_{ij} \leq \frac{n}{b}, \qquad j = 1, 2, \ldots, b.$$

$$\sum_j y_{ij} = 1, \qquad i = 1, 2, \ldots, n.$$

$$x_{ij} \leq (1 - y_{ik}) + (1 - y_{jl}), \qquad 1 \leq k \neq l \leq b. \tag{5.7}$$

$$x_{ij} \geq y_{ik} + y_{jk} - 1, \qquad 1 \leq k \leq b. \tag{5.8}$$

$$x_{ij},\ y_{ij} = 0 \, or \, 1, \qquad i = 1, \ldots, n,\ \ j = 1, \ldots, n.$$

This is so because the total cost of all the weights of the edges in any graph is always constant. The objective function (5.6) now maximises the cost of the edges in the same set shown in constraints (5.7) and (5.8).

Furthermore the problem is also equivalent to maximising the negative costs of the weights of the edges in the cut sets. This is the complement of objective function (5.1) and can be written as:

$$Maximise \quad -(\sum_{i}\sum_{j} c_{ij}x_{ij})$$

Garey and Johnson (1979) defined the problem with the additional vertex weight bound and edge weight bound. Their objective is to find the optimum set such that the sum of the vertex weights in each set is less than or equal to the vertex weight bound and the sum of the weights of the edges in the cut set is less than or equal to the edge weight bound. If the vertex weight is unitary then we are back to our original problem of partitioning vertices into b sets with each set having not more than $\frac{n}{b}$ vertices. Without loss of generality, we restricted each set to exactly $\frac{n}{b}$ vertices and conducted our testings on graphs with n divisible by b since vertices and edges of weights zero can always be added to the graphs if n is not divisible by b.

Garey and Johnson (1979) showed that the Graph Partitioning problem is NP-complete for fixed $b \geq 3$ and remains so even if the vertex and edge weights are unitary. It is only for $b = 2$ that it can be solved in polynomial time since it is equivalent to the weighted matching problem for bipartite graphs whose aim is to find a matching of minimum weight saturating edges in the cut set. Due to the wide application and practical importance of the Graph Partitioning problem, researchers opted for heuristics. Many have focused on variations including when the number of sets is just either two or three and when the number of vertices in each set is small.

Kernighan and Lin (1970), Barnard and Simon (1993) and Falkner et al. (1994) developed methods to partition the vertices into just two sets. Even though their methods can be reapplied over and over again to achieve other desired number of sets, their quality deteriorates as the number of sets increases. Feo

and Khellaf (1990) have approached the problem from the view of partitioning it according to the required number of vertices in a set. They presented several heuristics utilising the idea of matchings to produce sets with four vertices in each set.

The aim of this chapter is to develop heuristics that can partition vertices into b user-specified equal sets. This also means that the heuristics can be applied to the case when the user-specified criteria is the number of vertices in each set since this determines b and vice versa. However, it does not suggest that the heuristics are only restricted to the case when the number of vertices in the graph is divisible by b since vertices and edges of weight zero can always be added to the graph without any loss of generality.

Throughout we will approach the problem using the second formulation, that is, to maximise the objective function of the edges in the same sets. We implemented our heuristics together with the classic Kernighan and Lin's heuristic (1970) and Feo and Khellaf's heuristic (1990) as a means of evaluating the quality of our obtained solutions.

5.1 Heuristics for the Graph Partitioning Pro -blem

Our main idea is based on the approach used by Kernighan and Lin (1970) where edges are swapped between sets. We first outline Kernighan and Lin's approach (1970). Having obtained their two initial sets A and B, they calculate the difference, Df, for each vertex in the two sets. Recall that for $x \in V, Df(x)$ is the sum of its external edge weights (edges in the cut $(V, \overline{V})$) less the sum of its internal edge weights (edge in (V, V)). For each edge (a_i, b_i) in (A, B)

they proceed to work out the gains by the following:

$$\text{gain} = (Df \text{ of vertex } a_i \text{ in set } A) + (Df \text{ of vertex } b_i \text{ in set } B) - 2\times c_{a_i,b_i}$$

The edge (a_1, b_1) that produces the maximum gain is chosen and its end-points are removed from the two sets. The process is now repeated with the new difference for the remaining vertices in the two sets calculated as:

$$\begin{aligned}
&\text{new } Df \text{ of vertex } a_i \text{ in set } A = \\
&\qquad (\text{old } Df \text{ of vertex } a_i \text{ in set } A) \\
&\qquad + 2 \times c_{xa_1} - 2 \times c_{xb_1}, \text{ where } x \in A - \{a_1\} \\
&\text{new } Df \text{ of vertex } b_i \text{ in set } B = \\
&\qquad (\text{old } Df \text{ of vertex } b_i \text{ in set } B) \\
&\qquad + 2 \times c_{yb_1} - 2 \times c_{ya_1}, \text{ where } y \in B - \{b_1\}
\end{aligned}$$

The new gain is then computed as:

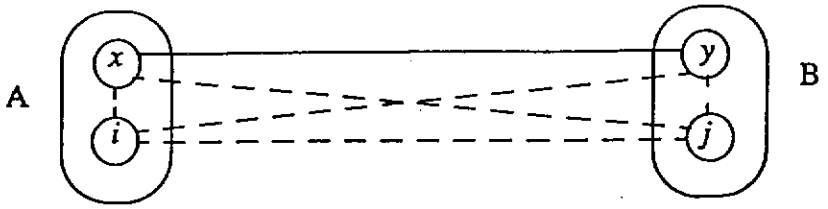
$$\text{gain} = (\text{ new } Df \text{ of vertex } a_i \text{ in set } A) + (\text{ new } Df \text{ of vertex } b_i \text{ in set } B) - 2\times c_{a_i,b_i}$$

Similarly, the edge (a_2, b_2) that produces the maximum gain is chosen and its end-points deleted from the two sets. This represents the gain that is obtained when vertex a_1 is swapped with vertex b_1 and vertex a_2 with vertex b_2. The k that maximises the partial sum of $\sum_{i=1}^{k} g_i$ is selected and the first k edges, $(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)$, are swapped. The procedure is then repeated until no possible gains can be achieved.

Our heuristic uses Kernighan and Lin's idea (1970) of swapping edges with the gains calculated in a slightly different manner. The idea is on the following page which we shall refer to as **Procedure 1.**

Procedure 1:

Given the following graph with two sets A and B: If



Df_x = ((weights of external edges of vertex x)
- (weights of internal edges of vertex x)) > 0

Df_y = ((weights of external edges of vertex y)
- (weights of internal edges of vertex y)) > 0

Df_i = ((weights of external edges of vertex i)
- (weights of internal edges of vertex i)) > 0

Df_j = ((weights of external edges of vertex j)
- (weights of internal edges of vertex j)) > 0

and

$$g_1 = Df_x + Df_j - 2 * c_{xj} > 0$$

$$g_2 = Df_y + Df_i - 2 * c_{yi} > 0$$

and

$$g_1 > g_2$$

then there is a gain of g_1 if vertex x is swapped with vertex j.

Without loss of generality we assume our graphs are complete as we always add edges of zero weight.

Our first heuristic uses Procedure 1 to calculate the gains given an initial b set. It then swaps the edges according to the calculated gains. More precisely:

Heuristic 5.1.1:

Step 1. Partition the vertices into b arbitrary sets.

Step 2. Repeat

Find the most expensive edge in the cut set and calculate the gain it will achieve with all the other vertices in the other set using Procedure 1 and record the maximum gain (only the positive gain is recorded).

Swap the end-points of the edge that produces the maximum gain.

until

no more gain can be achieved.

Heuristic 5.1.2 builds up each set sequentially according to edge weights. It always looks for the most expensive edge and attempts to include it in a set until the set has the required number of vertices. We refine this heuristic by applying Procedure 1 at the end of it. The heuristic is presented as follows:

Heuristic 5.1.2:

Step 1. Sort the edges by decreasing weight.

Step 2. Repeat

Repeat

Find the most expensive edge and put it in a set if
the set does not have the required number of vertices.
If the current set has $(\frac{n}{b} - 1)$ vertices
look for the most expensive edge that is incident
to any one of the vertices already in the current
set that has its other end-point not in any
set and put it in the current set.

until

the current set has $\frac{n}{b}$ number of vertices.

until

all the vertices of the graph are in a set.

Step 3. Refine the obtained sets by applying Procedure 1.

Our next heuristic, Heuristic 5.1.3, builds up the sets using Prim's algorithm (1957) except that it selects the most expensive incident edge. It repeatedly does this until the set has $\frac{n}{b}$ vertices. Again we apply Procedure 1 to this heuristic as a refinement step.

Heuristic 5.1.3:

Step 1. Sort the edges by decreasing weight.

Step 2. Repeat

Repeat

Select the most expensive edge whose end-points are not
in any set and put it in a set. Then apply Prim's
algorithm by selecting the most expensive edge incident
to any one of the vertices in that set and include it
in the same set.

until

the current set has $\frac{n}{b}$ number of vertices.

until

all the vertices of the graph are in a set.

Step 3. Refine the obtained sets by applying Procedure 1.

Our next three heuristics use the idea of selecting edges whose weight is greater than some critical value CV. Heuristic 5.1.4 uses CV as $\alpha\times$(average edge weight in the graph). It builds up the sets using Prim's algorithm (1957) and selectively includes edges into the set if the edge weight is greater than CV. Again Procedure 1 is applied to it as a refinement step. More specifically:

Heuristic 5.1.4:

Step 1. Set CV = $\alpha\times$(average edge weight in the graph).

Step 2. Repeat

Repeat

Find the vertex with the largest number of incident edges
and include its most expensive edge in the current set.
Then build up the set using Prim's algorithm by
selecting the most expensive incident edge that is
greater than the critical value, CV.
If no more incident edges can be added and the set
does not have the required number of vertices

repeat Step 2.

until

the current set has $\frac{n}{b}$ number of vertices.

until

all the vertices of the graph are in a set.

Step 3. Refine the obtained sets by applying Procedure 1.

Heuristic 5.1.5 uses the same depth-first search idea as used in Heuristic 3.1.5 of the DCMWST-problem with the same critical value as in Heuristic 5.1.4. We also refine this heuristic using Procedure 1.

Heuristic 5.1.5:

Step 1. Set CV = $\alpha\times$(average edge weight in the graph).

Step 2. Sort the edges $e_1 \geq e_2 \geq ... \geq e_j$ by decreasing weight.

Step 3. For $i = 1$ to j do

if e_i is already in a set then

discard this edge

else

put this edge $e_i = (u, v)$ in a set.

If $d(u) \geq d(v)$ then

push v on top of the stack followed by u

else

push u on top of the stack followed by v.

While stack $\neq \emptyset$ and the current edge does not have the required number of vertices do

pop the element from top of the stack. Try to include as many of its incident edges in the set if the edge weight is greater than the critical value.

For the newly selected vertex/vertices push the vertex of higher degree on top of the stack followed by the vertex of lower degree.

Stop when all the vertices of the graph are in a set.

Step 4 Refine the obtained sets by applying Procedure 1.

Heuristic 5.1.6 is similar to Heuristic 5.1.5 except it uses a different critical value. This value is found by summing up the most expensive $(((\frac{n}{b}(\frac{n}{b} - 1))/2) \times b)$ edges and assigning CV the average of that weight. This can be done since our graphs are complete. Hence in any set we know there are $((\frac{n}{b}(\frac{n}{b} - 1))/2)$ edges. We want to partition the vertices into b sets so there will be $(((\frac{n}{b}(\frac{n}{b} - 1))/2) * b)$ edges in all the sets.

Remark 1: In the computational experimentation for Heuristics 5.1.4, 5.1.5 and 5.1.6 we used critical value α ranging from 0.01 to 2.00.

The next heuristic, Heuristic 5.1.7, uses the idea of looking for the neighbouring vertex that will give the most gain when added to the current set. Again the heuristic is refined by Procedure 1. More precisely:

Heuristic 5.1.7:

Step 1. Sort the edges by decreasing weight.

Step 2. Repeat

Select the most expensive edge whose end-points are not in any set and put it in a set.

Repeat

Include the neighbour (not in any set) that gives the most gain to the edge weight in the set when added to the set.

until

the current set has $\frac{n}{b}$ number of vertices.

until

all the vertices of the graph are in a set.

Step 3. Refine the obtained sets by applying Procedure 1.

Our first seven heuristics presented can be used to partition vertices into any b sets, with $b \geq 2$. We now present heuristics which can only be used to partition vertices into sets with four vertices. This is also the case that was considered by Feo and Khellaf (1990). We have also programmed Feo and Khellaf's heuristic (1990) as well as refine their work.

Heuristic 5.1.8 is Feo and Khellaf's Heuristic 2 (1990) that uses the idea of matching. We have modified their starting greedy algorithm of obtaining the matching. Their greedy algorithm consists of selecting a vertex i at random and choosing its most expensive incident edge (i, j) to be included in the matching. Then vertices i and j are removed together with all their other incident edges and the process is repeated until no vertices are left. Instead of picking a vertex at random we repeatedly look for the most expensive edge to include in the matching. Our computational results show that this method of obtaining the initial matching produced better results. Heuristic 5.1.8 is presented as follows:

Heuristic 5.1.8:

Step 1. Sort the edges by decreasing weight.

Step 2. Repeat

Find the most expensive edge (i, j) whose end-points are not already in the matching and put it in the matching. Remove all the other incident edges beside vertices i and j.

until

all the vertices are in the matching.

Step 3. Contract the matching and merge parallel edges by summing their

weights.

Step 4. Find the new matching on this contracted graph using Step 2.

Step 5. Contract the matching and merge parallel edge by summing their weights.

The result obtained by Heuristic 5.1.8 is a partitioned graph with b sets, four vertices in each set. The results can also be refined by applying Procedure 1 after Heuristic 5.1.8 is completed. We shall call Heuristic 5.1.8 with such a refinement Heuristic 5.1.9.

Another simple approach to partition vertices into sets of four would be to partition according to the edge weights. Look for the most expensive edge and include it in the first set. Look for the next most expensive edge whose end-points are not in any set and include it in the second set. Keep doing so until all the b sets have one edge. Then look for the next most expensive edge and include it in the last set. Then include the next most expensive edge in the second last set and so on. We present this idea in Heuristic 5.1.10.

Heuristic 5.1.10

Step 1 Sort the edges by decreasing weight.

Step 2 Repeat

Find the most expensive edge whose end-points are not in any set and put it in the first set. Find the next most expensive edge whose end-points are not in any set and put it in the second set.

until

there is an edge in all of the b sets.

Step 3 Repeat

Find the most expensive edge whose end-points are not in any set and put it in the last set. Find the next most expensive edge whose end-points are not in any set and put it in the second last set and so on.

until

there are two edges in all the b sets.

If Feo and Khellaf's Heuristic 2 is reapplied to itself then what we get is a heuristic that partitions graphs into sets with eight vertices. We reapplied their heuristic to develop such a heuristic, Heuristic 5.1.11, and further refined this heuristic using Procedure 1 to produce Heuristic 5.1.12.

As mentioned, the classic Kernighan and Lin's heuristic (1970) is very efficient for partitioning vertices into just two sets. We have implemented their heuristic as a means of comparing the quality of our heuristics. We shall refer to Kernighan and Lin's heuristic (1970) as Heuristic 5.1.13. The refinement Procedure 1 when added to Heuristic 5.1.13 gives us our Heuristic 5.1.14.

Kernighan and Lin's heuristic (1970) can be reapplied to itself to partition vertices into four sets. We extend this in Heuristic 5.1.15. Heuristic 5.1.16 is Heuristic 5.1.15 with the refinement procedure added. Heuristic 5.1.17 takes Kernighan and Lin's heuristic (1970) another step further by partitioning vertices into eight sets. We also add the refinement procedure to Heuristic 5.1.17 to produce Heuristic 5.1.18.

We implemented our heuristics in the C programming language on a SUN SPARC 2 workstation operating at 28.5 MIPS. We tested the heuristics according to the number of vertices, n, in graph G. Only heuristics that are

suitable for the individual cases are tested. For example, for graphs with $n = 32$ and $b = 8$ we did not test Kernighan and Lin's heuristic (1970) and its refinements as these heuristics are only suitable for $b = 2$. Therefore, we tested the heuristics according to the number of vertices in a set. For $\frac{n}{b} = 4$ we generated random graphs with $n = 16$, 24, 28, 32, 36, for $\frac{n}{b} = 8$ we looked at graphs with $n = 16$, 24, 32, 40, 48 and for $b = 2$ we tested random graphs with $n = 50$, 100, 150, 200. The edge weights are real numbers in the range [1,20]. For each order we generated 30 random graphs of probability 0.10, 0.20, 0.25, 0.30, 0.40, 0.50, 0.60, 0.70, 0.75, 0.80, 0.90 and 1.00. We recorded the weight of the edges which are in the same sets after applying the heuristics. We present the average weight of the 30 graphs together with their average speed (in terms of CPU time (secs)).

p	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H15	H16
0.10	123.96	112.26	134.25	127.38	135.01	134.56	133.91	124.97	140.21	122.54	127.97	135.40
0.20	141.24	142.97	152.91	154.64	154.67	154.84	150.27	141.00	158.00	140.43	146.06	155.44
0.25	153.70	151.35	167.70	164.42	168.96	168.79	165.24	155.36	176.53	157.40	161.59	172.29
0.30	179.13	176.82	183.13	180.02	184.73	184.69	183.46	177.30	196.30	180.66	180.29	190.94
0.40	210.49	211.50	219.83	219.34	222.14	222.30	219.93	207.32	226.85	218.22	206.19	230.06
0.50	247.26	248.27	254.75	252.21	255.54	255.51	248.36	238.82	257.89	254.37	238.90	259.28
0.60	275.47	266.35	273.04	278.08	277.64	277.54	276.11	269.76	283.76	270.79	258.64	275.62
0.70	296.32	290.47	291.29	297.25	295.71	295.71	296.58	280.62	294.25	286.69	286.56	301.60
0.75	307.12	307.96	308.72	312.07	311.74	311.74	305.59	302.83	305.07	309.01	297.58	314.29
0.80	317.22	315.73	318.71	325.92	324.08	324.88	318.89	300.31	324.71	318.23	303.19	322.14
0.90	331.65	329.85	330.74	330.47	332.81	331.34	329.52	324.00	331.06	327.03	321.59	333.91
1.00	353.32	352.84	351.36	354.47	356.24	355.78	352.55	342.86	350.39	353.42	340.82	354.73

Table 5.1: Results for n=16 with $\frac{n}{b}$=4

p	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H15	H16
0.10	0.02467	0.02400	0.02200	0.02133	0.02433	0.02633	0.02600	0.01833	0.02100	0.02467	0.03133	0.04767
0.20	0.02800	0.02600	0.02467	0.02300	0.02800	0.02833	0.02800	0.01867	0.02133	0.02633	0.03233	0.04833
0.25	0.02733	0.02700	0.02433	0.02400	0.02700	0.03367	0.02800	0.01633	0.02033	0.02600	0.03133	0.04767
0.30	0.03200	0.02967	0.02800	0.03033	0.02800	0.03033	0.02700	0.01700	0.01967	0.03167	0.03333	0.04767
0.40	0.03167	0.03300	0.02500	0.02867	0.02900	0.03233	0.03267	0.01633	0.01900	0.03367	0.03167	0.05067
0.50	0.03633	0.03233	0.03233	0.02967	0.03300	0.03600	0.03500	0.01400	0.01900	0.03433	0.03167	0.04833
0.60	0.03533	0.03233	0.02733	0.02967	0.02867	0.03233	0.03333	0.01567	0.01800	0.03200	0.03233	0.04867
0.70	0.03700	0.03300	0.02933	0.03200	0.03200	0.03133	0.03533	0.01333	0.01733	0.03233	0.03033	0.05067
0.75	0.03667	0.03300	0.02867	0.03367	0.03400	0.03600	0.03467	0.01600	0.01467	0.03233	0.03100	0.04833
0.80	0.03733	0.03333	0.03167	0.03067	0.03100	0.03367	0.03500	0.01300	0.01700	0.03433	0.03300	0.05233
0.90	0.04133	0.03467	0.02933	0.03600	0.03433	0.03533	0.03667	0.01200	0.01500	0.03200	0.03267	0.04933
1.00	0.03667	0.03600	0.02900	0.03233	0.03300	0.03367	0.03667	0.01000	0.01500	0.03533	0.03033	0.05200

Table 5.2: Average CPU Time for n=16 with $\frac{n}{b}$=4

p	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
0.10	183.21	180.58	207.87	196.94	206.86	207.25	205.57	195.40	219.90	178.06
0.20	253.23	240.30	259.84	254.78	257.85	257.46	254.78	250.96	279.16	240.75
0.25	270.76	267.76	290.38	284.91	290.71	289.58	283.62	287.00	313.90	276.63
0.30	307.05	297.68	324.96	316.49	321.69	321.69	320.40	316.38	336.45	305.01
0.40	364.28	362.32	368.46	372.87	373.45	373.03	368.10	367.59	393.79	360.67
0.50	407.42	411.13	406.99	417.66	415.23	415.10	412.26	412.88	427.91	412.93
0.60	449.10	448.05	450.34	450.89	455.62	457.57	449.60	454.70	461.90	444.57
0.70	479.59	481.59	486.40	486.00	490.47	489.49	485.63	475.11	491.01	485.54
0.75	495.16	501.51	502.77	500.24	502.51	501.46	495.49	493.76	507.13	499.51
0.80	516.75	519.01	517.36	520.32	522.21	521.60	517.87	511.95	516.99	512.20
0.90	533.67	535.36	534.00	539.43	536.53	536.73	534.38	529.35	541.09	534.47
1.00	555.94	557.54	554.30	560.77	560.53	560.53	556.09	553.93	563.09	558.87

Table 5.3: Results for n=24 with $\frac{n}{b}$=4

p	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
0.10	0.11167	0.07533	0.07633	0.07800	0.09033	0.09600	0.08733	0.06133	0.07633	0.08433
0.20	0.13867	0.12500	0.08867	0.09167	0.10100	0.10033	0.10467	0.06633	0.07500	0.10033
0.25	0.14700	0.15133	0.09933	0.10500	0.10267	0.11633	0.11467	0.06633	0.07600	0.11467
0.30	0.16833	0.17167	0.10833	0.11367	0.12167	0.13333	0.12800	0.06233	0.07867	0.13433
0.40	0.17600	0.19500	0.11600	0.12067	0.11667	0.14300	0.14233	0.06067	0.07467	0.14567
0.50	0.16867	0.17833	0.11867	0.13733	0.12367	0.12500	0.13867	0.05967	0.07133	0.14633
0.60	0.17333	0.17033	0.12100	0.13267	0.12533	0.13900	0.14267	0.05700	0.06433	0.13833
0.70	0.16133	0.16300	0.11600	0.16567	0.14600	0.14300	0.14400	0.05733	0.06500	0.15033
0.75	0.16167	0.15233	0.14133	0.14900	0.14200	0.12933	0.14700	0.05567	0.06800	0.14767
0.80	0.16633	0.16133	0.14500	0.17100	0.12933	0.14533	0.15500	0.05300	0.06100	0.13967
0.90	0.18333	0.16967	0.12467	0.14667	0.15033	0.14233	0.15967	0.05033	0.06233	0.15600
1.00	0.18933	0.17200	0.12867	0.18800	0.14133	0.13933	0.18300	0.04600	0.05633	0.16000

Table 5.4: Average CPU Time (secs) for n=24 with $\frac{n}{b}$=4

p	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
0.10	204.03	205.34	244.25	227.04	243.00	243.17	237.17	234.22	266.15	218.41
0.20	304.24	288.70	327.52	317.50	326.64	327.00	321.45	323.84	349.54	297.87
0.25	339.92	330.87	355.43	353.83	363.58	361.46	357.49	365.04	386.37	340.20
0.30	371.64	367.61	387.77	382.40	393.20	392.37	382.29	397.57	423.04	367.85
0.40	444.39	436.74	446.55	450.31	450.74	449.81	441.04	453.25	482.42	443.97
0.50	502.93	501.99	505.13	509.12	510.19	510.13	502.41	509.43	528.90	500.67
0.60	542.10	536.33	541.75	542.86	546.20	546.20	542.59	541.30	563.36	543.64
0.70	580.72	581.51	580.66	582.58	585.98	585.52	575.99	581.36	597.73	579.12
0.75	599.40	591.49	598.65	599.30	600.97	600.97	594.69	594.93	616.19	593.85
0.80	618.41	611.23	616.50	619.32	622.93	622.93	611.02	624.03	626.00	610.55
0.90	635.70	636.76	640.25	638.75	642.53	641.40	633.61	642.89	652.39	633.31
1.00	653.11	658.79	658.85	659.76	661.91	661.52	658.50	665.87	670.84	654.76

Table 5.5: Results for n=28 with $\frac{n}{b}$=4

p	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
0.10	0.20633	0.12900	0.12067	0.11400	0.11700	0.14167	0.13533	0.10633	0.12967	0.14433
0.20	0.27633	0.19767	0.16333	0.16200	0.16633	0.18867	0.17933	0.10467	0.12967	0.20033
0.25	0.31367	0.22000	0.17833	0.18100	0.19333	0.21067	0.20200	0.10600	0.13100	0.22967
0.30	0.29667	0.21933	0.17700	0.32567	0.19000	0.19667	0.21333	0.10367	0.12667	0.23733
0.40	0.32933	0.25900	0.19200	0.21867	0.19833	0.21533	0.24233	0.10000	0.11867	0.24667
0.50	0.30500	0.23400	0.20600	0.22067	0.20333	0.23300	0.24867	0.09933	0.11233	0.24800
0.60	0.29633	0.25467	0.20333	0.25667	0.23533	0.24500	0.24933	0.09600	0.11033	0.24900
0.70	0.31400	0.24467	0.20367	0.24367	0.22700	0.22967	0.25033	0.09300	0.10567	0.24800
0.75	0.31200	0.22633	0.20933	0.22000	0.19800	0.20900	0.26033	0.09200	0.10600	0.24900
0.80	0.30133	0.24167	0.21433	0.23567	0.24900	0.25667	0.27600	0.08967	0.10533	0.26333
0.90	0.30167	0.24500	0.21733	0.26000	0.23733	0.24100	0.26333	0.08700	0.10167	0.25233
1.00	0.29067	0.27533	0.21533	0.24167	0.23533	0.24667	0.29967	0.07633	0.09100	0.24967

Table 5.6: Average CPU Time (secs) for n=28 with $\frac{n}{b}$=4

p	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H17	H18
0.10	239.17	236.68	288.78	266.98	287.26	287.39	279.49	267.63	304.57	250.41	278.13	290.89
0.20	367.38	348.20	387.64	379.26	394.18	391.89	378.31	389.16	430.99	365.71	382.22	400.32
0.25	410.21	402.86	420.78	421.80	431.57	429.75	412.19	434.96	470.36	399.54	411.33	440.23
0.30	441.75	441.55	466.00	462.41	464.97	464.32	458.93	472.57	504.87	451.51	449.29	475.33
0.40	518.23	533.32	529.70	534.86	542.35	542.09	532.51	557.59	570.99	515.47	510.38	543.01
0.50	579.70	584.37	586.48	592.12	598.07	598.10	588.44	593.45	624.15	589.22	571.37	604.48
0.60	623.47	639.09	639.10	633.13	640.10	637.48	627.59	647.74	670.11	630.51	612.12	650.01
0.70	674.98	675.54	673.20	679.48	685.78	684.87	674.92	691.40	710.07	672.76	658.22	687.75
0.75	691.22	691.65	693.50	701.01	700.87	702.04	696.63	711.37	724.13	690.14	671.70	697.79
0.80	715.67	719.99	715.36	719.13	718.76	717.25	714.52	714.83	739.33	708.90	690.98	721.93
0.90	737.29	739.72	736.51	738.28	739.73	739.66	732.03	741.74	764.31	739.48	709.43	740.58
1.00	762.40	762.08	767.17	766.57	770.63	767.48	761.20	774.18	788.55	763.04	737.16	768.83

Table 5.7: Results for n=32 with $\frac{n}{b}$=4

p	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H17	H18
0.10	0.32633	0.21167	0.18100	0.18367	0.18400	0.20300	0.19833	0.17433	0.19933	0.22733	0.33967	0.46433
0.20	0.42233	0.29400	0.21333	0.27833	0.24267	0.27033	0.28533	0.17200	0.19967	0.34033	0.32433	0.49500
0.25	0.53267	0.37467	0.22800	0.32967	0.26900	0.30067	0.32567	0.17267	0.20033	0.37067	0.33067	0.53500
0.30	0.52933	0.40667	0.29833	0.38200	0.29800	0.34733	0.33933	0.16900	0.20000	0.40167	0.33133	0.52800
0.40	0.53867	0.44067	0.30700	0.41667	0.34733	0.38233	0.40900	0.17200	0.19433	0.41967	0.36067	0.58600
0.50	0.51333	0.43767	0.33867	0.33800	0.33300	0.35233	0.41100	0.16033	0.18533	0.38633	0.35667	0.55800
0.60	0.53100	0.40033	0.32000	0.35800	0.33933	0.35100	0.41567	0.16133	0.17700	0.41700	0.34100	0.54967
0.70	0.51133	0.39967	0.32633	0.33267	0.33733	0.35133	0.40900	0.15367	0.17500	0.41667	0.32800	0.51933
0.75	0.48733	0.42200	0.33200	0.33700	0.35667	0.37067	0.45033	0.15100	0.17433	0.39667	0.33067	0.53533
0.80	0.48067	0.43333	0.35967	0.39533	0.37133	0.40500	0.42733	0.14700	0.17167	0.38233	0.34233	0.58967
0.90	0.49500	0.38367	0.38433	0.38567	0.39100	0.41933	0.47400	0.14467	0.16600	0.42700	0.34700	0.58433
1.00	0.49067	0.44000	0.35567	0.38333	0.37033	0.38867	0.47267	0.12933	0.15233	0.41267	0.32367	0.59433

Table 5.8: Average CPU Time (secs) for n=32 with $\frac{n}{b}$=4

p	H1	H3	H5	H6	H7	H8	H9	H10
0.10	287.75	336.76	335.37	334.78	330.54	322.84	364.64	298.34
0.20	429.27	449.93	453.42	449.92	436.19	476.45	493.45	427.20
0.25	480.14	491.05	498.72	498.72	475.73	519.96	548.52	471.68
0.30	521.25	536.86	540.33	540.26	523.89	561.81	590.64	521.64
0.40	614.27	624.94	627.91	628.25	611.70	643.83	674.10	622.18
0.50	675.20	671.71	680.08	669.92	675.16	697.99	729.09	671.56
0.60	720.33	731.65	736.11	737.33	731.58	747.93	775.93	718.73
0.70	773.39	780.69	782.88	783.98	779.13	795.98	821.28	775.46
0.75	792.83	790.86	799.32	799.32	793.95	810.24	835.07	788.88
0.80	813.11	813.82	819.86	816.82	808.87	827.87	853.43	812.45
0.90	834.09	842.95	844.92	843.89	845.54	868.01	881.28	839.63
1.00	871.61	871.73	874.73	873.47	874.17	894.50	908.97	867.37

Table 5.9: Results for n=36 with $\frac{n}{b}$=4

p	H1	H3	H5	H6	H7	H8	H9	H10
0.10	0.52033	0.23133	0.25233	0.28933	0.28500	0.25167	0.29633	0.28367
0.20	0.71100	0.35733	0.37433	0.39467	0.44900	0.24967	0.29133	0.56433
0.25	0.78500	0.36267	0.46767	0.49733	0.52500	0.24833	0.29567	0.62267
0.30	0.84667	0.45533	0.50967	0.52500	0.55533	0.24633	0.29000	0.61333
0.40	0.88233	0.49967	0.55033	0.58667	0.62033	0.23867	0.28033	0.67400
0.50	0.83133	0.54467	0.56567	0.53133	0.65367	0.23800	0.27000	0.64967
0.60	0.74667	0.47533	0.50633	0.55200	0.63467	0.23267	0.25900	0.57833
0.70	0.73967	0.53867	0.54000	0.55700	0.68533	0.22467	0.25100	0.63000
0.75	0.73133	0.46267	0.51633	0.57233	0.69667	0.21700	0.25400	0.64300
0.80	0.73833	0.46967	0.49067	0.54933	0.71900	0.21700	0.24467	0.62533
0.90	0.78467	0.51300	0.54033	0.55500	0.71767	0.20167	0.24033	0.62833
1.00	0.77733	0.54467	0.54233	0.58600	0.74500	0.18067	0.22500	0.68367

Table 5.10: Avearge CPU Time (secs) for n=36 with $\frac{n}{b}$=4

p	H1	H3	H5	H6	H7	H11	H12	H13	H14
0.10	157.78	168.77	171.39	172.02	168.33	159.37	171.01	176.98	176.98
0.20	210.06	207.28	212.65	211.80	207.62	199.00	204.74	215.78	215.89
0.25	226.14	232.82	235.77	237.79	231.86	221.29	236.59	245.23	245.23
0.30	255.22	261.62	266.29	266.64	264.37	257.00	272.33	277.33	277.56
0.40	332.55	331.04	337.31	336.91	335.81	323.89	334.15	345.22	346.60
0.50	406.11	406.64	412.43	411.71	412.19	386.26	407.66	422.69	423.01
0.60	464.60	458.18	473.12	472.89	460.71	449.73	466.50	479.58	480.05
0.70	513.76	517.90	522.00	522.00	518.51	487.45	506.22	528.33	528.62
0.75	544.59	549.89	552.01	554.26	553.05	529.13	541.60	561.35	561.41
0.80	576.49	581.02	586.21	586.21	583.11	558.01	577.38	593.33	593.55
0.90	627.70	622.93	626.19	627.23	623.28	601.46	613.13	634.33	634.47
1.00	681.77	685.35	686.68	684.92	683.38	667.59	677.06	688.30	688.81

Table 5.11: Results for n=16 with $\frac{n}{b}$=8

p	H1	H3	H5	H6	H7	H11	H12	H13	H14
0.10	0.02033	0.02167	0.02167	0.02600	0.02200	0.01867	0.02267	0.02200	0.03100
0.20	0.02100	0.02100	0.02200	0.02533	0.02633	0.01900	0.02167	0.02133	0.03200
0.25	0.01867	0.02167	0.02300	0.02700	0.02400	0.01733	0.02200	0.01933	0.02967
0.30	0.02000	0.02300	0.02333	0.02700	0.02400	0.01700	0.02133	0.02167	0.03233
0.40	0.02033	0.02333	0.02367	0.02767	0.02467	0.01800	0.02133	0.02067	0.03200
0.50	0.02167	0.02267	0.02400	0.02733	0.02633	0.01667	0.02100	0.02000	0.03100
0.60	0.01900	0.02200	0.02200	0.02500	0.02367	0.01700	0.01833	0.02100	0.03067
0.70	0.02200	0.01967	0.02167	0.02433	0.02567	0.01500	0.01767	0.02067	0.03133
0.75	0.02300	0.02233	0.02200	0.02667	0.02700	0.01533	0.01733	0.02000	0.03133
0.80	0.02267	0.02367	0.02333	0.02567	0.02667	0.01433	0.01733	0.02033	0.03133
0.90	0.02300	0.02200	0.02300	0.02933	0.02567	0.01200	0.01800	0.01900	0.02933
1.00	0.02233	0.02233	0.02233	0.02600	0.02833	0.01333	0.01500	0.01900	0.03200

Table 5.12: Average CPU Time (secs) for n=16 with $\frac{n}{b}$=8

p	H1	H3	H5	H6	H7	H11	H12
0.10	246.03	258.51	265.29	264.57	261.77	254.03	269.66
0.20	353.92	354.24	367.40	367.70	354.01	343.42	373.62
0.25	410.99	417.33	422.04	424.28	411.85	410.03	428.83
0.30	471.20	474.35	484.84	485.30	471.74	465.55	483.75
0.40	583.13	584.68	593.59	592.38	586.86	563.79	588.52
0.50	684.12	695.40	698.16	693.99	699.14	662.11	692.14
0.60	780.25	780.95	791.08	791.06	778.88	754.56	760.90
0.70	865.18	857.95	871.29	870.07	865.43	829.11	847.38
0.75	908.15	912.00	913.74	911.91	912.57	875.63	888.09
0.80	961.91	964.48	968.42	968.33	961.36	931.45	933.85
0.90	1018.73	1023.59	1029.53	1026.99	1018.81	986.59	1011.99
1.00	1088.47	1089.81	1098.05	1097.01	1094.90	1061.99	1069.23

Table 5.13: Results for n=24 with $\frac{n}{b}$=8

p	H1	H3	H5	H6	H7	H11	H12
0.10	0.08867	0.07767	0.08133	0.09200	0.08433	0.07133	0.08467
0.20	0.09367	0.07433	0.09367	0.09433	0.09900	0.06800	0.07833
0.25	0.10100	0.08300	0.08933	0.09567	0.10000	0.06600	0.07967
0.30	0.10467	0.09267	0.10400	0.12100	0.11433	0.06467	0.08000
0.40	0.11433	0.09933	0.10800	0.11833	0.12233	0.06533	0.07600
0.50	0.10767	0.09933	0.10633	0.11433	0.11733	0.06433	0.07267
0.60	0.11367	0.11000	0.11133	0.11367	0.11600	0.05967	0.07100
0.70	0.10667	0.10067	0.10100	0.10833	0.12433	0.05833	0.06867
0.75	0.10833	0.10867	0.10233	0.11900	0.12333	0.05733	0.06000
0.80	0.11433	0.10367	0.10867	0.11300	0.11733	0.05467	0.06567
0.90	0.11533	0.10767	0.11033	0.12233	0.12100	0.05300	0.06167
1.00	0.12033	0.10667	0.12133	0.12700	0.13533	0.04833	0.05867

Table 5.14: Average CPU Time (secs) for n=24 with $\frac{n}{b}$=8

p	H1	H3	H5	H6	H7	H11	H12	H15	H16
0.10	340.79	361.09	364.89	365.87	359.58	339.57	374.93	380.83	386.71
0.20	553.61	543.52	562.20	558.17	544.93	541.26	575.22	574.27	582.88
0.25	620.92	627.72	642.80	642.92	626.50	621.26	659.16	657.52	670.90
0.30	706.62	706.76	720.15	719.08	701.33	689.40	730.24	734.35	749.09
0.40	867.72	869.70	874.66	878.23	867.12	854.22	879.59	880.84	894.26
0.50	988.94	995.67	1012.42	1008.80	995.90	962.95	1001.75	1017.17	1031.33
0.60	1108.45	1118.74	1127.64	1125.64	1129.67	1088.44	1114.38	1130.15	1153.31
0.70	1233.59	1230.20	1242.55	1245.10	1230.14	1203.30	1215.10	1241.78	1260.71
0.75	1297.31	1291.46	1300.30	1300.24	1280.30	1245.87	1267.23	1298.18	1320.25
0.80	1340.38	1345.52	1357.08	1356.63	1342.96	1295.53	1322.19	1351.27	1377.74
0.90	1423.10	1425.42	1434.44	1432.82	1430.07	1368.74	1404.61	1423.63	1445.40
1.00	1507.74	1508.27	1518.80	1519.40	1515.17	1472.91	1498.08	1513.45	1530.78

Table 5.15: Results for n=32 with $\frac{n}{b}$=8

p	H1	H3	H5	H6	H7	H11	H12	H15	H16
0.10	0.24133	0.18000	0.20333	0.21967	0.20167	0.18800	0.21600	0.25367	0.36867
0.20	0.32733	0.24267	0.28933	0.26900	0.28300	0.18167	0.21567	0.24433	0.38167
0.25	0.35267	0.24500	0.30600	0.31633	0.31433	0.18300	0.21300	0.24867	0.37100
0.30	0.37733	0.25567	0.33133	0.34233	0.35767	0.18133	0.21033	0.24900	0.37733
0.40	0.40333	0.29033	0.34800	0.36133	0.38100	0.17833	0.20667	0.25733	0.38100
0.50	0.37133	0.33633	0.32433	0.34200	0.35033	0.17333	0.20133	0.26533	0.37733
0.60	0.35800	0.30200	0.32833	0.32100	0.39000	0.17100	0.19233	0.28067	0.43900
0.70	0.36400	0.30000	0.32467	0.33200	0.33867	0.16700	0.18700	0.26667	0.40267
0.75	0.35600	0.30767	0.32667	0.33000	0.36033	0.16233	0.18633	0.25133	0.38867
0.80	0.38167	0.31167	0.32500	0.33667	0.37133	0.16167	0.18067	0.25700	0.39467
0.90	0.38167	0.31600	0.36067	0.36267	0.38133	0.15567	0.17800	0.25867	0.39200
1.00	0.35267	0.32267	0.33133	0.33867	0.37067	0.13833	0.16500	0.23833	0.38167

Table 5.16: Average CPU Time (secs) for n=32 with $\frac{n}{b}$=8

p	H1	H3	H5	H6	H7	H11	H12
0.10	483.24	508.49	510.55	510.52	490.66	483.69	531.09
0.20	738.29	739.69	755.25	764.82	725.48	750.95	791.55
0.25	835.01	841.98	864.67	865.81	841.25	851.81	893.85
0.30	944.57	934.50	960.71	957.91	937.57	937.54	970.27
0.40	1137.35	1147.78	1161.28	1160.45	1146.40	1134.05	1179.19
0.50	1301.61	1295.94	1315.11	1315.68	1304.10	1269.83	1304.46
0.60	1460.08	1474.50	1480.88	1479.80	1467.10	1439.41	1463.18
0.70	1603.44	1605.88	1618.07	1616.08	1611.88	1571.16	1610.39
0.75	1671.51	1672.35	1684.62	1684.40	1680.91	1624.44	1658.64
0.80	1743.02	1725.35	1736.23	1730.45	1730.43	1692.38	1717.51
0.90	1830.41	1827.81	1844.53	1838.76	1834.44	1794.87	1826.64
1.00	1945.55	1946.07	1960.27	1957.67	1950.50	1915.11	1939.95

Table 5.17: Results for n=40 with $\frac{n}{b}$=8

p	H1	H3	H5	H6	H7	H11	H12
0.10	0.64667	0.37900	0.43800	0.50533	0.47667	0.43733	0.50133
0.20	0.89267	0.57367	0.64267	0.70367	0.70233	0.43667	0.50400
0.25	0.90267	0.67867	0.79733	0.86800	0.83467	0.43633	0.49967
0.30	1.02100	0.68967	0.83167	0.85333	0.89167	0.43667	0.50100
0.40	1.02567	0.78267	0.87233	0.91767	0.98500	0.42900	0.48300
0.50	0.96900	0.76233	0.77867	0.88933	0.96700	0.42967	0.47667
0.60	0.89033	0.82067	0.76600	0.76067	0.89567	0.41600	0.45967
0.70	0.89700	0.75600	0.79533	0.82800	0.88233	0.40100	0.44667
0.75	0.92933	0.77900	0.77533	0.80533	0.94300	0.40000	0.43867
0.80	0.88733	0.68900	0.71733	0.75600	0.90467	0.39400	0.43800
0.90	0.90567	0.69600	0.74033	0.79933	0.90500	0.38433	0.41167
1.00	0.96967	0.80067	0.79533	0.78267	1.00033	0.35800	0.39467

Table 5.18: Average CPU Time (secs) for n=40 with $\frac{n}{b}$=8

p	H1	H3	H5	H6	H7	H11	H12
0.10	605.37	637.36	649.76	646.12	624.72	632.05	689.85
0.20	940.21	942.59	969.77	969.85	952.52	966.15	1019.2
0.25	1069.1	1058.8	1088.8	1090.8	1057.9	1080.0	1127.4
0.30	1188.7	1190.7	1218.1	1214.7	1188.0	1202.0	1246.0
0.40	1450.9	1432.2	1460.9	1460.9	1431.4	1431.3	1465.2
0.50	1633.4	1638.0	1642.7	1644.6	1618.8	1618.9	1672.5
0.60	1820.6	1807.2	1823.4	1821.5	1818.2	1773.8	1825.5
0.70	1984.7	1990.7	1995.9	1990.7	1986.2	1935.4	1987.0
0.75	2060.4	2062.2	2078.0	2072.2	2053.0	2027.9	2071.8
0.80	2123.6	2122.7	2142.4	2143.2	2135.2	2094.5	2135.2
0.90	2249.8	2261.0	2263.8	2266.9	2256.2	2216.7	2259.2
1.00	2369.9	2378.1	2386.8	2382.3	2387.1	2355.4	2396.3

Table 5.19: Results for n=48 with $\frac{n}{b}$=8

p	H1	H3	H5	H6	H7	H11	H12
0.10	1.47867	0.74733	0.90700	0.98433	1.03633	0.91533	1.03400
0.20	2.07367	1.36400	1.60467	1.58633	1.60600	0.90800	1.03233
0.25	2.17467	1.37067	1.64900	1.70600	1.66367	0.91233	1.01600
0.30	2.26700	1.60533	1.77900	1.83767	2.02300	0.89667	1.01767
0.40	2.55167	1.83000	1.95233	2.00900	2.23433	0.90067	1.00133
0.50	2.29433	1.72600	1.88533	1.81167	2.05767	0.89067	0.98767
0.60	2.16167	1.63600	1.64667	1.73700	2.08267	0.90500	0.96700
0.70	2.08333	1.63600	1.64500	1.60400	1.91200	0.85900	0.95267
0.75	1.94900	1.60600	1.62133	1.70500	1.88367	0.85333	0.95800
0.80	2.00833	1.61867	1.64567	1.77767	1.97333	0.82633	0.95467
0.90	1.92167	1.68700	1.75000	1.72533	2.05733	0.78233	0.95400
1.00	1.97667	1.67967	1.71300	1.74567	2.19233	0.74700	0.94233

Table 5.20: Average CPU Time (secs) for n=48 with $\frac{n}{b}$=8

n	p	H1	H3	H5	H6	H7	H13	H14
50	0.05	617.98	642.13	650.66	646.35	631.78	674.34	674.34
	0.25	2104.43	2098.57	2143.31	2142.87	2127.38	2191.16	2191.79
	0.50	3811.02	3833.45	3858.56	3863.52	3838.79	3935.92	3937.09
	0.75	5442.47	5438.27	5473.51	5475.61	5461.14	5522.36	5524.49
	1.00	6884.36	6881.37	6899.80	6898.26	6878.68	6933.44	6934.60
100	0.05	2147.93	1988.63	2062.15	2060.18	2042.34	2126.12	2126.12
	0.25	8359.35	8016.24	8123.20	8126.47	8053.11	8272.03	8272.18
	0.50	15411.68	14855.88	14975.68	14981.11	14838.92	15151.61	15152.64
	0.75	21824.67	21355.29	21502.88	21477.78	21450.62	21611.74	21612.60
	1.00	27873.52	27534.78	27535.89	27535.89	27487.71	27613.63	27613.86
150	0.05	4299.28	4207.47	4378.46	4385.92	4352.97	4526.07	4526.07
	0.25	17628.79	17639.27	17753.03	17753.03	17666.43	18014.84	18015.26
	0.50	32927.12	32921.20	33036.26	32998.30	32961.46	33361.69	33361.87
	0.75	47700.95	47717.98	47802.52	47772.37	47726.30	48033.30	48034.66
	1.00	61542.62	61536.64	61631.13	61603.91	61542.46	61776.43	61777.41
200	0.05	7763.79	7180.54	7386.29	7387.05	7353.06	7620.58	7620.58
	0.25	31372.82	30836.78	30899.28	30895.92	30801.91	31291.80	31293.44
	0.50	58685.86	57849.07	57992.83	58042.49	57888.70	58488.74	58488.74
	0.75	85222.88	84209.47	84348.01	84355.09	84174.03	84677.58	84677.58
	1.00	109605.77	109039.20	109088.73	109114.49	109080.91	109371.80	109374.97

Table 5.21: Results for b=2

n	p	H1	H3	H5	H6	H7	H13	H14
50	0.05	0.83267	0.77033	0.69900	0.77533	0.76667	0.69900	1.00900
	0.25	0.98233	0.89467	0.99000	1.08667	1.12800	0.68733	1.00333
	0.50	0.94667	0.95000	0.95267	1.01667	1.15633	0.74500	1.09633
	0.75	0.88567	0.83333	0.86167	0.91100	1.04367	0.75333	1.12767
	1.00	0.98567	0.88467	0.94400	0.96000	0.94567	0.55100	1.14400
100	0.05	9.48800	9.35733	7.63167	8.18467	8.31133	7.55233	10.70167
	0.25	13.39267	12.93900	13.69467	14.30700	14.32033	7.29733	10.29267
	0.50	12.34267	12.13900	11.95133	12.51900	13.72333	7.38867	10.83333
	0.75	10.12866	9.91633	9.09300	9.70467	11.33766	7.33534	10.44100
	1.00	9.86500	9.53000	9.24066	9.56700	10.87000	6.44367	10.61467
150	0.05	41.15533	44.23034	38.06666	39.03233	38.94333	24.75967	35.34800
	0.25	67.07333	69.67467	66.04700	70.23466	75.11800	26.75667	37.86800
	0.50	55.03400	55.05601	55.97066	56.97133	63.49467	25.67300	37.18234
	0.75	40.11933	41.81334	39.51967	41.22134	46.20234	27.50433	39.30534
	1.00	34.99333	34.52300	33.74200	34.77734	41.49667	26.03800	40.02000
200	0.05	117.92501	122.13567	103.79066	113.28035	114.91002	59.38866	85.39900
	0.25	223.73167	226.50798	218.66602	225.74831	246.86433	66.49633	95.70366
	0.50	164.17502	172.74535	168.30165	175.64363	183.83763	73.12434	102.16067
	0.75	116.54765	110.84132	109.33366	111.71933	130.97697	75.52135	104.13733
	1.00	94.05700	92.42664	87.76232	88.49165	107.94099	63.08266	98.42999

Table 5.22: Average CPU Time (secs) for b=2

For the case when $n = 16$ and $\frac{n}{b} = 4$ Heuristic 5.1.16, the refinement of Kernighan and Lin's heuristic, seemed to perform best even though the CPU time is slightly higher than others. This is then followed by Heuristic 5.1.5. Heuristic 5.1.6 performed better than the refinement of Feo-Khellaf's heuristic, Heuristic 5.1.9, when the density of the graph is higher than 0.70. Heuristics 5.1.3, 5.1.2, and 5.1.10 seemed to produce better results than Kernighan and Lin's heuristic (1970). As for Heuristic 5.1.15 its results are better than that of Heuristic 5.1.10 and 5.1.1. Among the heuristics, Feo and Khellaf's heuristic (1990) is the worst. As for the CPU time (secs) Heuristic 5.1.8 is the fastest and Heuristic 5.1.5 is the slowest. All the other heuristics have average times which are relatively fast.

For the other cases of $\frac{n}{b} = 4$ the refinement of Feo and Khellaf's heuristic, Heuristic 5.1.9, is always the best. This is normally followed by Heuristics 5.1.5 and 5.1.6. Heuristic 5.1.3 is comparable with Heuristic 5.1.4. On average Heuristics 5.1.1, 5.1.2 and 5.1.10 are always producing results which are close to each other. For the case $n = 32$ when the refinement of Kernighan and Lin's heuristic, Heuristic 5.1.9, is tested it always outperforms the others. Feo and Khellaf's (1990) heuristic is always worse than Heuristic 5.1.3. All the heuristics have similar times with Heuristics 5.1.8 and 5.1.9 the fastest.

When $\frac{n}{b} = 8$, for the case when $n = 16$, Heuristic 5.1.14, the refinement of Kernighan and Lin's heuristic, is the best followed by Kernighan and Lin's (1970) heuristic, Heuristic 5.1.13. For the other cases, most of the time Heuristic 5.1.5 produced results that are the best followed by Heuristic 5.1.6 and 5.1.7. The extension of Feo and Khellaf's heuristic (1990) did not perform so well when $\frac{n}{b} = 8$. Heuristic 5.1.7 is most of the time better than Heuristic 5.1.3 which is most of the time better than Heuristic 5.1.1.

As for the CPU time (secs) for the case $\frac{n}{b} = 8$, although Heuristic 5.1.11 does not perform so well, it is in fact the fastest among the heuristics. This is followed by Heuristics 5.1.12, 5.1.3, 5.1.5, 5.1.6 and 5.1.1.

When $b = 2$, on average, Heuristic 5.1.1 produced the best results. Heuristic 5.1.14, the refinement of Kernighan and Lin's heuristic, is the second best followed by Kernighan and Lin's heuristic (1970). On average, Heuristic 5.1.5 and 5.1.6 are very close followed by Heuristics 5.1.7 and 5.1.3. In terms of CPU time (secs) Kernighan and Lin's heuristic (1970), Heuristic 5.1.13, is the fastest followed by Heuristic 5.1.14. This is followed by Heuristics 5.1.5, 5.1.1, 5.1.6, 5.1.3 and finally 5.1.7. The time for Heuristic 5.1.7 increases as the graph gets bigger.

From our tests we conclude that Kernighan and Lin's heuristic (1970) is very good when $b = 2$. Feo and Khellaf's heuristic (1990) did not perform as well as some of our heuristics. When further extended, Kernighan and Lin's heuristic (1970) to solve for $b = 4$ and 8, the quality of the results deteriorated. Similarly, when Feo and Khellaf's heuristic (1990) was extended to solve cases when $\frac{n}{b} = 8$ it did not perform that well.

Our aim is to find heuristics that are able to partition vertices of a graph into b user specify sets. Our computational work indicates that the two best performing heuristics (5.1.5 and 5.1.6) are those that select edges according to a prescribed critical value, CV. Heuristic 5.1.5 performed the best most of the time and is closely followed by Heuristic 5.1.6. The CPU time (secs) of Heuristic 5.1.5 is also on average better than that of Heuristic 5.1.6.

Chapter 6

Concluding Remarks

In this thesis we developed heuristics for the Degree Constrained Minimum Spanning Tree problem (DCMWST) as well as for the Graph Partitioning problem. The problems were looked at with a view of finding fast and efficient heuristics that are easy to implement. There are several possible directions for future work on these two problems.

In Section 3.1 we analysed the DMWST problem where all the vertices in the graph have the same degree restriction. A variant of the problem would be to define a set R_i of allowable degrees and find such a degree constrained tree. The problem can be stated as:

> *Given an edge weighted graph* $G = (V, E)$ *with* $V = \{1, 2, \ldots, n\}$ *and a set* $R_i = \{r_1, r_2, \ldots, r_n\}$ *of allowable degrees, find a spanning tree* T *of* G *of minimum weight such that the degree,* $d_T(i)$*, in* T *of vertex* i *is from the set* R_i*.*

For the non-weighted case it is in fact the problem we considered in Section 4.2 of this thesis where the spanning trees have to have degrees from a specified degree spectrum. We know the problem remains NP-complete even for planar graphs of maximum degree 3. Hence it would be interesting to develop good

quality heuristics to solve this problem for the weighted case.

We can also extend this problem to the Steiner case (trees whose set of vertices is a subset of V). The problem then becomes:

> *Given an edge weighted graph $G = (V, E)$ with $V' \subseteq V$ and a set $R_i = \{r_1, r_2, \ldots, r_n\}$ of allowable degrees, find a spanning tree T consisting of all the vertices of V' such that the degree, $d_T(i)$, in T of vertex i is from the set R_i.*

Another possible avenue that we can look at is to solve the DCMWST with an additional constraint such as the flow, shortest path and Steiner constraints. This just opens up a multitude of problems since there is no limit as to which two constraints should be combined. In view of the wide application of the DCMWST problem in telecommunication networks we propose the addition of the flow constraint.

The DCMWST with the additional flow constraint can be stated as:

> *Given an edge weighted graph $G = (V, E)$ with a source s and a sink t, find a minimal spanning tree T for transferring commodity from the source to the sink such that the degree, $d_T(i)$, in T of vertex i is at most r_i, $1 \leq i \leq n$ and each arc and each vertex satisfies its capacity restriction.*

This problem is similar to the Capacitated Minimum Spanning Tree problem with the exception of having just a sink as opposed to one source with all the other vertices as sinks. This problem would be interesting to solve if not optimally, with heuristics. The problem can certainly be extended to the Steiner case with a subset of vertices satisfying the specified flow and degree

constraints.

The flow constraint can also be added to our first problem mentioned in this chapter. The problem thus becomes:

> *Given an edge weighted graph $G = (V, E)$ with a source s, a sink t and a set $R_i = \{r_1, r_2, \ldots, r_n\}$ of allowable degrees, find a spanning tree T of minimum weight for transferring commodity from the source to the sink such that the degree, $d_T(i)$, in T of vertex i is from the set R_i and each arc and each vertex satisfies its capacity restriction.*

This problem can be approached from the point of heuristics. We can also analyse the Steiner case of this problem.

Thus the DCMWST problem can be extended to many different new problems. The aim is always to develop efficient methods (exact and heuristics) to solve these problems.

For the case of the $(1, k)$-tree problem in Chapter 4, future work that we propose would be to extend it to the case of weighted graphs. This problem is similar to the first problem we define in this chapter where the set of allowable degrees is now $R_i = \{1, k\}$ with the exception that the maximum degree in G is k.

For the case of Theorem 4.2.8 where the degrees of the spanning tree are from a specified degree spectrum $d = (d_0, d_1, \ldots, d_m = k)$, when we extend this problem to the weighted case we get the first problem of this chapter where $R_i = \{d\}$. Again, the exception would be in the maximum degree of k. We

can also add the flow constraint to this problem and produce the following:

> *Given an edge weighted graph $G = (V, E)$ with maximum degree k together with a source s, a sink t and a set $R_i = (d_0, d_1, \ldots, d_m = k)$ of allowable degrees, find a spanning tree T of minimum weight for transferring commodity from the source to the sink such that the degree, $d_T(i)$, in T of vertex i is from the set R_i and each arc satisfies its capacity restriction.*

As for the case of the Graph Partitioning problem (Chapter 5), we have analysed the problems with a view of developing heuristics for partitioning the vertices into b user-specified sets. Future work that can be carried out includes the development of fast exact algorithms to solve problems with b user-specified sets.

Another avenue the Graph Partitioning problem can take, is to partition the vertices into unequal b user-specified sets. The number of vertices in the sets could be in increasing order starting from 1 in increment of 1 or more. Another variant could be for the user to specify which set gets how many vertices. Of course future work in developing exact algorithms for such problems should be considered.

The Graph Partitioning problem can be studied with the additional vertex weight constraint. This is the same as the following problem in Garey and Johnson (1978):

> *Given an edge and vertex weighted graph $G = (V, E)$, partition the vertices into b sets such that the cost of the edges in the cut set is minimum and the sum of the vertex weights in each set is less than or equal to the vertex weight bound.*

This problem is NP-complete. Of course, the b sets could be of equal or unequal size. It would be interesting to develop good algorithms for this problem.

In conclusion, there are many different avenues the Restricted Spanning Tree and the Graph Partitioning problems can take.

Bibliography

- Achuthan, N.R., Caccetta, L., Caccetta, P. and Geelen, G.F. (1994). Computational Methods for the Diameter Restricted Minimum Weight Spanning Tree Problem. *Australasian Journal of Combinatorics*, 10 pp. 51-71.

- Ahuja, R.K., Magnanti, T., Reddy, M.R. and Orlin, J.B. (1995). Applications of Network Optimisation. *Chapter 1 of the Handbooks in Operations Research and Management Science, Volume 7: Network Models (Ball, M.O., Magnanti, T.L., Monma, C.L. and Nemhauser, G.L. eds.)*, Elsevier, North Holland, pp. 1-84.

- Ali, A.I. and Huang, C. (1991). Balanced Spanning Forests and Trees. *Networks*, 21 pp. 667-687.

- Applegate, Bixby, R., Chvatal, V. and Cook, W. (1995). Finding Cuts in the TSP (A Preliminary Report). *DIMACS Technical Report*, pp. 95-105.

- Arbib, C. (1987/88). A Polynomial Characterisation of some Graph Partitioning Problems. *Information Processing Letters*, 26 pp. 223-230.

- Abdalla, A., Deo, N., Kumar, N. and Terry, T. (1997). Parallel Computation of a Diameter-Constrained MST and Related Problems. *Congressus Numerantium*, 126 pp. 131-155.

- Barnard, S.T. and Simon, H.D. (1993). A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems, in *Proceedings to the* 6^{th} *SIAM Conference on Parallel Processing for Scientific Computing*, SIAM pp. 711-718.

- Blake, R.E. (1994). Partitioning Graph Matching with Constraints, *Pattern Recognition*, 27(3) pp. 439-446.

- Bondy, J.A. and Murty, U.S.R. (1976). Graph Theory with Applications. The Macmillan Press, London.

- Caccetta, L. (1989). Graph Theory in Network Design and Analysis, in *Recent Studies in Graph Theory, (Kulli, V.R. ed.)*, Vishwa International, India, pp. 29-63.

- Caccetta, L. and Lam, B.K. (1998). Spanning Trees with Prescribed Degrees. (submitted for publication).

- Caccetta, L., Lam, B.K. and Hill, S.P. (1998). Heuristics for the Degree Restricted Spanning Tree Problem. (submitted for publication).

- Caccetta, L. and Hill, S.P. (1997). A Branch and Cut Method for the Degree Constrained Minimum Spanning Tree Problem. (submitted for publication).

- Caccetta, L. and Vijayan, K. (1987). Applications of Graph Theory, *Ars Combinatoria*, 23(B) pp. 21-77.

- Camerini, P.M., Galbiati, G. and Maffioli, F. (1980). Complexity of Spanning Tree Problems: Part 1. *European Journal of Operational Research*, 5 pp. 346-352.

- Camerini, P.M., Galbiati, G. and Maffioli, F. (1983). On The Complexity of Finding Multi-Constrained Spanning Trees. *Discrete Applied Mathematics*, 5 pp. 39-50.

- Cayley, A. (1874). On the Mathematical Theory of Isomers. *Philosophical Magazine*, 47(4) pp. 444-446.

- Chandy, K.M. and Lo, T. (1973). The Capacitated Minimum Spanning Tree. *Networks*, 3(2) pp. 173-181.

- Chandy, K.M. and Russell, R.A. (1972). The Design of Multipoint Linkages in a Teleprocessing Tree Network. *IEEE Transactions on Computers*, 21(10) pp. 1062-1066.

- Cheriton, D. and Tarjan, R.E. (1976). Finding Minimum Spanning Trees. *SIAM Journal of Computing*, 5(4) pp. 724-742.

- Christofides, N. (1976). Worst-case Analysis of a New Heuristic for the Travelling Salesman Problem. Carnegy-Mellon University, Pittsburg.

- Dantzig, E.W. (1961). The Decomposition Algorithm for Linear Programming. *Econometrica*, 29(4) pp. 101-11.

- Deo, N. and Kumar, N. (1997). Computation of Constrained Spanning Trees: A Unified Approach. *Lecture Notes on Economics and Mathematical Systems 450 (Paralos, M.P., Hearns, D.W. and Hager, W.W. eds)*, Springer, pp. 194-220.

- Dijkstra, E.W. (1957). A Note on Two Problems in Connection in Graphs. *The Bell System Technical Journal*, 36 pp. 269-271.

- Diniz, P., Plimpton, S., Hendrickson, S. and Leland, R. (1995). Parallel Algorithms for Dynamically Partitioning Unstructured Grids. *Proceedings to the 7^{th} SIAM Conference on Parallel Processing for Scientific Computing, SIAM*, pp. 615-620.

- Douglas, R.J. (1992). NP-Completeness and Degree Restricted Spanning Trees. *Discrete Mathematics*, 105 pp. 41-47.

- Eppstein, D. (1994). Offline Algorithms for Dynamic Minimum Spanning Tree Problems. *Journal of Algorithms*, 17 pp. 237-250.

- Esau, L.R. and Williams, K.C. (1966). On Teleprocessing System Design. *IBM Systems Journal*, 5(3) pp. 142-147.

- Falkner, J., Rendl, F. and Wolkowicz, H. (1994). A Computational Study of Graph Partitioning. *Mathematical Programming*, 66 pp. 211-239.

- Feo, T.A. and Khellaf, M. (1990). A Class of Bounded Approximation Algorithms for Graph Partitioning. *Networks*, 20 pp. 181-195.

- Fernandes, L.M., and Gouveia, L. (1998). Minimal Spanning Trees with a Constraint on the Number of Leaves. *European Journal of Operational Research*, 104 pp. 250-261.

- Fürer, M. and Raghavachari, B. (1992). Approximating the Minimum Degree Spanning Tree to within One from the Optimal Degree. *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, Orlando, Florida, pp. 317-324.

- Gabow, H.N. (1977). Two Algorithms for Generating Weighted Spanning Trees in Order. *SIAM Journal of Computing*, 6(1) pp. 139-150.

- Gabow, H.N. (1978). A Good Algorithm for Smallest Spanning Trees with a Degree Constraint. *Networks*, 8 pp. 201-208.

- Gabow, H.N., Galil, Z., Spencer, T. and Tarjan, R.E. (1986). Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs. *Combinatorica*, 6(2) pp. 109-122.

- Galbiati, G., Maffioli, F. and Morzenti, A. (1994). A Short Note on the Approximability of the Maximum Leaves Spanning Tree Problem. *Information Processing Letters*, 52 pp. 45-49.

- Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability, A guide to the Theory of NP-Completeness*, Freeman, San Francisco.

- Garey, M.R., Johnson, D.S. and Stockmeyer, L. (1976). Some Simplified NP-Complete Graph Problems. *Theoretical Computer Science*, 1 pp. 237-267.

- Gavish, B. (1982). Topological Design of Centralised Computer Networks-Formulation and Algorithms. *Networks*, 12 pp. 355-377.

- Gavish, B. (1983). Formulations and Algorithms for the Capacitated Minimal Directed Tree Problem. *Journal of the Association for Computing Machinery*, 30(1) pp. 118-132.

- Gilbert, J.R. and Zmijewski, E. (1987). A Parallel Graph Partitioning Algorithm for a Message-passing Multiprocessor. *International Journal of Parallel Programming*, 16 pp. 498-513.

- Glover, F. and Klingman, D. (1974). Finding Minimum Spanning Trees with a Fixed Number of Links at a Node. *Colloquia Mathematica Societatis János Bolyai 12, Progress in Operations Research*, Eger (Hungary), pp. 425-439.

- Golden, B., Ball, M. and Bodin, L. (1981). Current and Future Research Directions in Network Optimisation. *Computer and Operations Research*, 8 pp. 71-81.

- Golden, B.L. and Magnanti, T.L. (1977). Deterministic Network Optimisation: A Bibliography. *Networks*, 7 pp. 149-183.

- Gouveia, L. (1995). A $2n$-constraint formulation for the capacitated minimal spanning tree problem. *Operations Research*, 43 pp. 130-141.

- Griggs, J.R. and Wu, M. (1992). Spanning Tree in Graphs of Minimum Degree 4 or 5. *Discrete Mathematics*, 104 pp. 167-183.

- Hendrickson, B. and Leland, R. (1995a). A Multilevel Algorithm for Partitioning Graph. *Proceedings of the 1995 ACM/IEEE Supercomputing Conference.*

- Hendrickson, B. and Leland, R. (1995b). An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations. *SIAM Journal of Scientific Computing*, 16.

- Hwang, F.K. (1979). An O(n log n) Algorithm for Rectilinear Minimal Spanning Trees. *Journal of the Association of Computing Machinery*, 26(2) pp. 177-182.

- Johnson, D.S. (1985). The NP-Completeness Column: An Ongoing Guide. *Journal of Algorithms*, 6 pp. 45-159.

- Johnson, D.S., Aragon, C.R., McGeoch, L.A. and Schevon, C. (1989). Optimisation by Simulated Annealing: An Experimental Evaluation; Part 1, Graph Partitioning. *Operations Research*, 37(6) pp. 865-892.

- Johnson, D.B. and Metaxas, P. (1995). A Parallel Algorithm for Computing Minimum Spanning Trees. *Journal of Algorithms*, 19 pp. 383-401.

- Kernighan, B.W. and Lin, S. (1970). An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49 pp. 291-307.

- Kershenbaum, A. (1974). Computing Capacitated Minimal Spanning Trees Efficiently. *Networks*, 4 pp. 299-310.

- Kershenbaum, A. and Boorstyn, R.R. (1983). Centralised Teleprocessing Network Design. *Networks*, 13 pp. 270-293.

- Khellaf, M. (1987). *On the Partitioning of Graphs and Hypergraphs.* Ph.D. Dissertation, Dept IEOR, University of California, Berkeley.

- Kleitman, D.J. and West, D.B. (1991). Spanning Trees with Many Leaves. *SIAM Journal of Discrete Mathematics*, 4(1) pp. 99-106.

- Krishnamoorthy, M., Ernst., A.T. and Sharaiha, Y.M. (1998). Algorithms for the Degree Constrained Minimum Spanning Tree. *Proceedings to the International Conference of Optimisation Techniques and Applications*, pp. 859-866.

- Kruskal, J.B. (1956). On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem. *Proceedings of the American Mathematical Society*, 7 pp. 48-50.

- Kučera, L. (1995). Expected Complexity of Graph Partitioning Problems. *Discrete Applied Mathematics*, 57 pp. 193-212.

- Lin, S. (1975). Heuristic Programming as an Aid to Network Design. *Networks*, 5 pp. 33-43.

- Malik, K. and Yu, G. (1993). A Branch and Bound Algorithm for the Capacitated Minimum Spanning Tree Problem. *Networks*, 23 pp. 525-532.

- Malyshko, V.V. (1985). An Effective Approach to Some Practical Capacitated Tree Problems. *Graph Theory with Applications to Algorithms and Computer Science (Y. Alavi, et. al. eds.)*, John Wiley and Sons, New York, pp. 531-542.

- Martin, J. (1967). *Design of Real-Time Computer Systems*, Englewood Cliffs, New Jersey, Prentice-Hall.

- Mao, L.J., Deo, N., Kumar, N. and Lang, S.D. (1997). A comparison of Two Parallel Approximate Algorithms for the Degree-Constrained Minimum Spanning Tree Problem. *Congressus Numerantium*, 123 pp. 15-32.

- Matousek, J. and Nesetril, J. (1996). *Kapitoly z Diskretni Matematiky*, MATFYZ-Press Charles University.

- Minoux, M. (1989). Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications. *Networks*, 19 pp. 313-360.

- Narula, S.C. and Ho, C.A. (1980). Degree-Constrained Minimum Spanning Tree. *Computers and Operations Research*, 7 pp. 239-249.

- Papadimitriou, C.H. (1978). The Complexity of the Capacitated Tree Problem. *Networks*, 8 pp. 217-230.

- Papadimitriou, C.H. and Yannakakis, M. (1982). The Complexity of Restricted Spanning Tree Problems. *Journal of the Association for Computing Machinery*, 29(2) pp. 285-309.

- Pothen, A., Simon, H. and Liou, K. (1990). Partitioning Sparse Matrices with Eigenvectors of Graphs. *SIAM Journal of Matrix Analysis*, 11 pp. 430-452.

- Prim, R.C. (1957). Shortest Connection Networks and Some Generalisations. *The Bell System Technical Journal*, 36 pp. 1389-1401.

- Prüfer, H. (1918). Neuer beweis eines satzes ber permutationen. *Arch. Math. Phys*, 27 pp. 742-744.

- Ravi, R., Sundram, R., Marathe, M.V., Rosenkrantz D.J. and Ravi, S.S. (1996). Spanning Trees- Short and Small. *SIAM Journal of Discrete Mathematics*, 9(2) pp. 178-200.

- Rendl, F. and Wolkowicz, H. (1990). *A Projection Technique for Partitioning the Nodes of a Graph.* Technical Report CORR 90-20, University of Waterloo, Waterloo, Canada.

- Savage, J.E. and Wloka, M.G. (1991). Parallelism in Graph-Partitioning. *Journal of Parallel Distance Computing*, 13 pp. 257-272.

- Savelsbergh, M. and Volgenant, A. (1985). Edge Exchange in the Degree-Constrained Minimum Spanning Tree Problem. *Computers and Operations Research*, 12 pp. 341-348.

- Shamos, M.I. and Hoey, D. (1975). Closest-Point Problems. *Proceedings to the 6th Annual Symposium on Foundations of Computer Science*, pp. 151-162.

- Sharaiha, Y.M., Gendreau, M., Laporte, G. and Osman, I.H. (1997). A Tabu Search Algorithm for the Capacitated Shortest Spanning Tree Problem. *Networks*, 29 pp. 161-171.

- Shogan, A.W. (1983). Constructing a Minimal-Cost Spanning Tree Subject to Resource Constraints and Flow Requirements. *Networks*, 13 pp. 169-190.

- Silver, E.A., Vidal, R.V.V. and de Werra, D. (1980). A Tutorial on Heuristic Methods. *European Journal of Operational Research*, 5 pp. 153-162.

- Simon, H.D. (1991). Partitioning of Unstructured Problems for Parallel Processing. *Proceedings to the Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications.* Pergammon Press.

- Smolenski, W.F. (1964). Application of the Theory of Graphs to Calculations of the Additive Structural Properties of Hydrocarbons. *Russian Journal of Physical Chemistry*, 38 pp. 700-703.

- Vanderstraeten, D., Keunings, R. and Farhat, C. (1995). Optimisation of Mesh Partitions and Impact on Parallel CFD. *Parallel Computational Fluid Dynamics, New Trends and Advances (Ecer, A., Hauser, J., Leca,*

P. and Periaux, J. eds.), Elsevier, pp233-239. (Also in Proc. Parallel CFD '93).

- Volgenant, A. (1989). A Lagrangean Approach to the Degree-Constrained Minimum Spanning Tree Problem. *European Journal of Operational Research*, 39 pp. 325-331.

- Volgenant, T. and Jonker, R. (1983). The Symmetric Travelling Salesman Problem and Edge Exchanges in Minimal 1-Trees. *European Journal of Operational Research*, 12 pp. 394-403.

- Walshaw, C., Cross, M. and Everett, M. (1995). A Parallelisable Algorithm for Optimising Unstructured Mesh Partitions. *Technical Report 95/IM/03, University of Greenwich*, London SE18 6PF, UK. (submitted for publication).

- Weiner, P. (1975). Heuristics. *Networks*, 5 pp. 101-103.

- Zhou, G. and Gen, M. (1997). A Note on Genetic Algorithms for Degree-Constrained Spanning Tree Problems. *Networks*, 30 pp. 91-95.

- Zhou, G. and Gen, M. (1998). An Effective Genetic Algorithm Approach to the Quadratic Minimum Spanning Tree Problem. *Computers and Operations Research*, 25 pp. 229-237.