The use of partitioning strategies in local access telecommunication network problems and other applications

D.J. van der Merwe

Thesis submitted for the degree Doctor of Philosophy at the Potchefstroom Campus of the North-West University

Promotor: Prof. J.M. Hattingh

November 2007

Abstract

Telecommunications networks can usually hierarchically be divided into three parts. The local access telecommunications network (LATN) connects individual subscribers to the network and makes a significant contribution to the total telecommunications network cost. The Tree Knapsack Problem (TKP) and Extended Tree Knapsack Problems (ETKP) can be used to model some aspects of the LATN design problem. Efficient solution methods for these models can improve planning of LATN's as it may be necessary to solve many such problems for different scenarios.

The TKP and ETKP models are investigated and solution methods found in the literature are discussed. New exact solution methods for the TKP and ETKP models are proposed. These solution methods are based on enhanced modeling and partitioning strategies and use standard off the shelf software. The idea is to produce a solution strategy that utilises the strengths of modern optimisation software and hardware platforms to solve problems of realistic size.

The algorithms were evaluated by using planned empirical methods. Results of the experimental work and comparisons are presented. The suggested partitioning and enhanced modeling approaches performed well when compared to other solution approaches, in the sense that more stable solution times were observed as well as better capabilities with larger problem instances. It appears to be feasible to use enhanced modeling and partitioning to solve TKP and ETKP problems.

Uittreksel

Telekommunikasie netwerke kan in die algemeen in drie hierargiese dele verdeel word. Gebruikers word deur Plaaslike Toegang Telekommunikasie Netwerke (PTTN) verbind aan die telekommunikasie netwerk. Die Boomknapsakprobleem (BKP) en Uitgebreide Boomknapsakprobleem (UBKP) kan gebruik word as model vir dele van die PTTN. Dit is soms nodig om verskeie van hierdie probleme op te los vir verskillende scenario's. Dit is dus noodsaaklik om effektiewe oplossingsmetodes vir hierdie probleme te hê.

'n Ondersoek is gedoen op die BKP en UBKP modelle en gepubliseerde oplossingsmetodes vir hierdie modelle word voorgelê. Nuut ontwikkelde eksakte metodes vir die BKP en UBKP word voorgestel. Hierdie metodes maak gebruik van standaard kommersiele sagteware en is gegrond op verbeterde modellerings- en partisioneringskonsepte. Die idee is om die krag van moderne sagteware en hardeware platforms te gebruik om realistiese grootte probleme op te los.

Die ontwikkelde algoritmes is geëvalueer deur gebruik te maak van sistematiese empiriese toetse. Die resultate van hierdie toetse en vergelykings is voorgelê. Die werkverrigting van die verbeterde modellering en partisionerings algoritmes is goed indien dit vergelyk word met ander oplossingsmetodes vir die BKP en UBKP probleme.

Acknowledgements

I would not have been able to complete this study and thesis, without the help and support from a few people. I would like to thank everyone, friends and family alike for all the support, encouragement and patience. I would especially like to express my gratitude to the following people:

- My promotor, Prof Hattingh for introducing me to the field of Operations Research, suggesting a topic and his assistance during my studies. Thank you for all your help and work in compiling this thesis.
- My parent parents and my family for all their support during this and my other studies. Without the support and encouragement of my parents, I would not have been able to complete this work.
- Mrs. Anriette Pretorius, the librarian, for her kind assistance with the gathering of literature.
- Bouke Spoelstra for the language editing of the thesis.
- Mrs. Anne Mans and Mrs. Klaasje Benadie for their assistance with the computing platforms and software used in this study.
- The CSIR, especially Mr. Hans Ittman for the financial assistance. Without this I would not have been able to complete this study.
- My fiancée, Elmarie Pienaar for her encouragement, patience and support during this study, thank you for your understanding and love.

My Heavenly Father, for the ability, opportunities and strength to complete this undertaking.

Table of Contents

List of figures

Chapter 1 Introduction

According to George Gilder (2002:2) a new age is dawning on man. This is an era in which communication is of paramount importance, it may even surpass the importance of computing. The importance of computing is in creating and processing information, but communication is more fundamental to humanity than computing. As such, the telecommunications sector is very competitive and ever changing. Before attempting to discuss telecommunication planning issues, which will be the main focus of this study, it is first necessary to discuss the broader telecommunications field.

1.1 Background

1.1.1 Background on telecommunication

To put telecommunication networks into perspective, a brief history of telecommunications networks is presented to highlight the importance of planning and also the role technological advances play in the field of telecommunication. This discussion is based on work presented by Shepard (2002). As precursor to modern telecommunication networks, simple telephone networks started out very humbly. As early as 1677, inventor Robert Hooke showed that sound vibrations could be transmitted via a medium and interpreted at the other end of the medium. After this initial revelation, many years passed before the invention of the telegraph in the 1830's made it possible to transmit information over long distances. Later in the nineteenth century, a lot of work was done to create the first machine that could transmit voice electronically along a wire. On 14 February 1876, Alexander Graham Bell filed a notice of invention, the first device to send voice across a simple network. Arguments exist that another inventor, Elisha Gray, the inventor of the telegraph, filed a notice first, but this is not important in this exposition. As is the case in the telecommunications world, a legal battle ensued over patent infringements, which was settled in 1879. After this, telephone services moved along at a rapid rate, according to Shepard (2002:133).

By spring 1880, the United States had 138 exchanges and 30 000 subscribers which grew to 150 000 subscribers by 1887. As no switching mechanisms were available yet, this meant that phones were sold in pairs and connected by a single continuous wire. This implied that for two people to talk, a dedicated line was needed between them. If the same person wanted to talk to more than one person, more than one wire was required. This was a very limiting

factor and was solved by the concept of a central office. All the subscribers in a certain area were connected to a central office that would connect specific users upon request. This concept was refined further into switches that could do the work automatically. As the first central offices were operated by hand and this limited the number of calls a switching station could handle per time interval, electric and electronic switches increased the number of calls that could be handled vastly. Even at that stage planning the location of a central office or switching centre was important, as customer satisfaction and revenues were dependent on the location of the central office.

Telecommunication networks have evolved tremendously since the first telephone networks of the late nineteenth and early twentieth century, but many of the concepts do still apply. When looking at a modern, fixed line telecommunications network the basic structure still holds. Customers are connected to the network in some way or other, for example by copper wires. Users are grouped together in some way and connected to a central office or switches. These switches have limited capabilities and can only connect relatively small numbers of users. To overcome this problem, switches are connected to a higher level of switches, known as long distance switches or inter office switching centres. These long distance switches are in turn connected to gateway nodes. This sort of topology is also discussed in Balakrishnan *et al.* (1991:242). A phone call is placed in this network configuration as follows: If a user places a call to another user belonging to the same local switch, the call is rooted directly through the switching centre. If the user being called belongs to a different switching centre, the switching centre passes the call to the relevant inter office switching centre to which it belongs, to complete the call. The inter office switching centre calls the switching centre to which the user being called belongs to and the call gets completed by this switching centre. The users belonging to a specific switching centre forms a so-called local access telecommunications network. This is a simple view of how phone calls are placed in a fixed line telecommunications network. Complex accounting and billing is also built into telecommunications networks. Traditionally phone call costs are dependent on the distance and duration of the call being made.

1.1.2 Telecommunication company challenges

At present the telecommunications industry is riddled with acronyms, abbreviations, terms and concepts foreign to many end users. This can be observed by looking at advertising material from telecommunications companies. Users are continually bombarded with new

technologies, words, terms and acronyms. End-users of telecommunications services often do not know what all these terms mean, nor do they seem to care. They are mostly interested in communicating, gathering information, shopping etc. and doing this in the most cost effective way possible. This causes a dilemma for the telecommunications companies, as the companies want to introduce new services and connect new subscribers to their network while maximising profits. The problem is compounded by the added pressures of competition amongst different telecommunications companies and the effect of disruptive technologies on existing networks and business models. Thus, telecommunications companies need to make intelligent strategic decisions in order to survive and more importantly, to remain profitable. This leads to the importance of planning the telecommunications networks. Planning in this sense, means that hardware should be placed so that customers can be served at the lowest cost possible. An alternative formulation, making the problem even more daunting, is to state that the network should be built to maximise profit. In principle, this seems simple enough, but in practice, this is an extremely difficult problem. As stated by Balakrishnan *et al.* (1991:242) the ideal would be to simultaneously plan all the levels of the telecommunications network, as the levels are interdependent. Here the local access telecommunications network forms one level, which is connected to the inter office switches, which form a second level. The inter office switching is connected to a next level known as the long distance network. These levels are generally considered individually and as such the local access telecommunications network is considered to be a very important design problem. The reason for this is that according to Balakrishnan *et al.* (1991:240), the local access network constitutes approximately 60% on average of the total investment in telecommunications facilities. It is thus important that great care is taken when planning the local access part of the network as building these networks require a large portion of the total expenditure on the network and there is usually pressure to deliver services as cost effectively as possible. The fact that the total telecommunications network can not be feasibly designed as a whole, already hints at the difficulty involved when trying to solve network design problems.

1.1.3 Local access telecommunication network

It is important to discuss the local access telecommunication network (referred to as LATN in the rest of the text), in more detail. As discussed previously the LATN had humble beginnings where a separate wire connected every two people who wanted to be connected to the modern setup we have today. Various ways exist in which users can be connected to the telecommunications network; also, multiple configuration patterns exist in which users can be

connected. The pattern in which the users are connected, with the connections being referred to as links, is referred to as the topology of the local access network. Refer to Gavish (1991) for a description of some of these topologies.

The LATN has gone through many stages of evolution, before it reached the topology it has today. Users are first connected to distribution points. At these points traffic, as the calls or data being transmitted may be referred to, may be compressed in order to make the transmission more cost effective. The idea is that compressing data or simply aggregating traffic leads to more cost effective solutions. In principle multiple users may be connected to a single distribution point from where only a single transmission line is needed to the higher levels of the network. The distribution points are in turn connected to switching centres where additional compression or aggregation may take place before traffic (if needed) is passed on to the higher levels of the network. As economy of scale is important; the quest for the planner of the LATN is located in placing the distribution and switching centres. This problem is of combinatorial nature and may be very difficult to solve, as the aim is to place these facilities in such a way that costs are minimised while profits are maximised. As many configurations are possible and multiple ways exist in which a single end user may be connected, the problem of connecting users to a telecommunications network via a LATN can quickly become very difficult.

The LATN design problem may be seen as finding the locations to place hardware in order to serve customers under certain constraints. These constraints may, for example, be that only a certain traffic capacity may be served from a switching centre, or some other hardware constraints. Various models exist for representing the LATN, see for example Balakrishnan *et al.* (1991), Costmagna *et al.* (1998) and Shaw *et al.*(1997). After modeling the problem, the model still has to be solved and the results interpreted. This is where optimisation plays an important role. New regulatory advances like Local Loop unbundling, (see Bourreau and Dogan (2005)), where the incumbent operators are being forced to allow competitors to use their networks, place even more importance on the planning and the costing of the local area telecommunications networks.

1.1.4 Optimisation

In order to solve planning problems, a model is required first, which has to be solved. A model according to Bierman *et al.* (1991) is "a simplified representation of an empirical

situation. Ideally, it strips a natural phenomenon of its bewildering complexity and duplicates the essential behaviour of the natural phenomenon with a few variables that are simply related". After a model has been chosen, an optimisation procedure has to find a solution for the model. According to Gottfried and Weisman (1973:3) classical optimisation may be defined as an art for creating policies to satisfy certain objectives while satisfying certain constraints. According to Wilde and Beightler (1967:1) man's longing for perfection find expression in the theory of optimisation. They state that once a person knows how to measure good or bad, one can search for the best. In many instances optimal may be used interchangeably for best. According to them "optimisation theory encompasses the quantitative study of optima and methods for finding them". An important fact to note here is the notion of best solution. It may be possible to find various feasible solutions, but the notion of "best solution" means that the goal is to obtain "the best" solution while not violating the constraints, which is a more difficult problem than simply finding "a solution". More modern definitions may be found, but looking at these classic definitions, it is evident that optimisation may assist greatly in solving the design problem of LATN's. Finding such optimised solutions for the design of LATN's will be the focus of this thesis.

1.2 Goals

After broadly discussing telecommunication networks to put the LATN into perspective and after giving a succinct discussion on optimisation it is now possible to state the objectives for the study. The main objectives are to:

- Investigate the attributes of local access telecommunications networks.
- Investigate selected models that are used during the design of LATN's and solution methods for these models.
- Develop new solution methods for some of the problems associated with LATN design based on enhanced modeling of the problem and partitioning of the search space,
- Investigate the feasibility of using standard software combined with enhanced modeling techniques to solve models used during LATN design.
- Make comparisons between different solution methods researched for the models investigated. This will be done by means of empirical work.

1.3 Thesis overview

This chapter serves as a brief description of telecommunications networks and gives a brief view on the aims of optimisation. This is done to familiarise the reader with the environment without being overly technical.

Chapter 2 gives an overview of graphs, graph representations and relevant ideas. After the representation of graphs is presented Chapter 3 deals with the complexity of optimisation problems This will be followed by a chapter on solution strategies. It is not feasible to merely know that a problem is difficult, it still needs to be solved!

Chapter 5 deals with various general telecommunications applications and models and gives a more detailed discussion of the broader telecommunications field. In Chapter 6 the focus falls on models specifically investigated with regard to the LATN design problems. Here attention is also given to modeling requirements. Chapter 7 gives an exposition of the solution strategies developed in the course of this study. This presents the bulk of the contribution made in this study.

Chapter 8 presents the empirical work undertaken and in Chapter 9 conclusions and recommendations are presented. Chapter 10 is a bibliography of the sources used in the study.

Chapter 2 Modeling

In the previous chapter a brief introduction is presented on the evolution of telecommunications networks and on optimisation. One question arising is "How can a "relationship" be formed between the real world and the "world" found in optimisation?" In this regard the concepts of models and of modeling will be discussed. For the purpose of this thesis, a model can be seen informally as a representation of some real world system. There exists an entire field of research concerned with the meaning of systems and in which way they can be perceived, namely systems thinking. This, however, falls outside the scope of this study. The aim of this chapter is to present intuitive definitions of what models are and of the use of modeling in optimisation of network design problems.

2.1 Mathematical models

Numerous books have been written on modeling and models, also, the field of systems thinking is concerned with systems and how systems are perceived. Due to this, various definitions may be presented for what a model is and exactly what it represents. It is not the aim of this thesis to contribute to the field of systems thinking. It is nevertheless important to understand what models are. Specific focus is placed on models used for optimisation: generally the models of interest are referred to as mathematical models.

To understand where mathematical models fit in an operations research methodology, look at the graphical representation given in figure 2-1 of a possible operations research methodology as presented by Winston (1994:1). In this iterative methodology, the mathematical model is central in the solution process. In the rest of this thesis, the terms model and mathematical model will generally be used interchangeably. Another operations research process is presented in Rardin (1998:3), which is similar to the one presented by Winston, but which simplifies the process somewhat. Atamturk *et al.* (2000:847) present something referred to as the modeling life cycle, which aims to generate solutions to real world problems. It has the following steps:

- Develop a model.
- Generate an instance of the model, which normally refers to the collection of data of a specific problem.
- Solve the instance (or solve the model for the specific instance to be more precise).

- Validate the solution and the model.

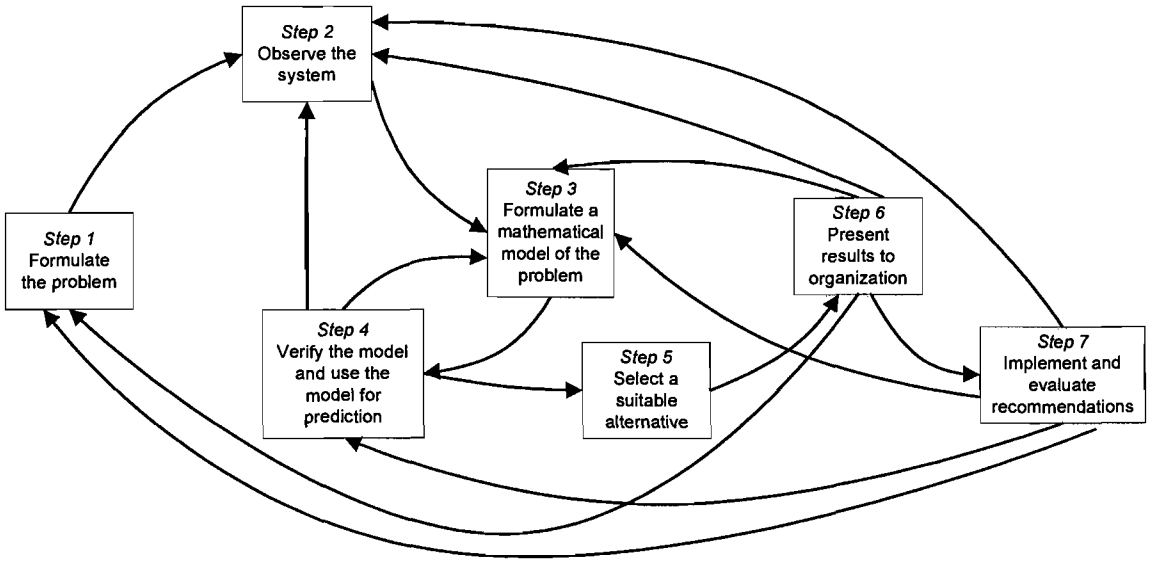

Figure 2-1 An operations research methodology according to Winston (1994:2)

Given these methodologies, it is important to present a definition for what a model is. This is nevertheless not an easy task. The models holding special interest will be used for optimisation, which as described by Gottfried and Weisman (1973:3), is a technique for obtaining best decisions while satisfying predefined objectives. This is a very simple definition for optimisation but it captures the essence of the goals of optimisation. Generally the aim of optimisation is to find the "best" solution for a problem, while not violating certain conditions that the solution has to conform to.

Bender (1978:2) makes the point that various people would offer different definitions for what a mathematical model is, but offers a starting point by saying that it is "an abstract, simplified, mathematical construct related to a part of reality and created for a specific purpose". Examining this definition, it seems to yield a very good notion of what mathematical models are. A model represents some real world situation; this is sometimes referred to as a system. It is necessary to point out that other types of modeling also exist like simulation, or geometric construction, as mentioned in Bellomo and Preziosi (1994:2). According to Giordano and Weir (1983:32) mathematical models can further be divided into models that refer to processes that exist in nature and models that can be constructed or selected to represent a real world problem or system, where a system can be defined as a collection of objects that share some interaction with, or dependence on one another (Giordano and Weir (1983:30)).

A more formal definition for mathematical models is presented by Bellomo and Preziosi (1994:2). This definition tries to represent a mathematical model in terms of independent variables, which is divided into two broad classes. The first class of independent variables is generally used to represent "space" while the second class can be used to represent time. The goal is to obtain a function that transforms these independent (or decision) variables to produce a state variable, alternatively called a dependent variable. This definition tries to map from the physical world to a modeling world. Consequently, the concept of parameters is offered to characterise the system which is being modeled. A state variable is used to represent the physical state of the system being represented by the model, after a possible evolution of the independent variables have taken place. Using the terms defined in this paragraph, a possible formal definition for a mathematical model is the following definition presented by Bellomo and Preziosi (1994:4):

> *A mathematical model is an equation, or a set of equations. The solution of the equations provides a transformation of the state (or independent) variables to represent in a mathematical sense the behavior in a physical world.*

In literature many similar definitions for mathematical models can be found, even though each person may have his/her own definition of mathematical models, but all models share the representation of a system of some sort as some mathematical equation or set of equations. An important aspect is to know what is represented by a model and what is not. Bender (1978:2) identifies three aspects of reality models are concerned with, namely:

- Aspects which have effects which are neglected,
- aspects which influences the model, but which the mathematical model is not designed to model, and
- aspects that the mathematical model should represent.

Most of the other literature have a similar type of separation, as it is sometimes impossible to develop a model that takes all factors into account. This may not be a shortcoming, as the model may have been developed to model only certain aspects of the total system.

Giordano and Weir (1983:33) identifies another set of traits associated with models, which holds not only for mathematical models, but for other types of models as well. These traits are precision, the cost of the model and flexibility. Precision has to do with how accurately the model represents the system. Different models may have different precision, for instance

a first model being developed may not be as precise as subsequent and better refined versions of the same model. Cost has to do with how expensive it is to develop a certain model. If more refinements are made to a standard mathematical model, for instance if extra restrictions are placed on a certain source of commodity, more time and effort may go into the model construction phase. This adds additional costs to the development of the model. The last trait mentioned has to do with flexibility. Some models are more difficult to change than others. This means that it may be preferable to model a certain problem in a certain way that allows other aspects to be modeled more easily. Another manifestation of this is where simulation is contrasted against mathematical modeling. In some instances it is easier to change the a simulation model, than it is to change a mathematical model, due to the way in which simulation models are generally constructed.

The two previous paragraphs lead ultimately to two intriguing questions, to which many answers exist. The questions are: "How are models constructed" and "How can the best model be selected"? Fowkes and Mahony (1994:2) is of opinion that modeling is as much an art as a science and also that the choice of model is the most important decision a modeler can make. The reason for this is that various models can be developed for the same system or problem being investigated. Looking a figure 2-1 (a possible operations research methodology as presented by Winston (1994:1)), it is evident that even a single model can go through various steps of refinement. The problem is that not all models developed for a specific problem or system is necessarily useful, or even applicable to the problem being investigated. For this reason, most model construction methodologies include a step of model verification, see Giordano and Weir (1983:36), Bellomo and Preziosi (1994:4) and Bender (1978:7). This is necessary to validate that the model developed is applicable and usable. Another facet of this problem is to know the ranges of validity for the variables used in the mathematical model. This may be due to the fact that the function or formulas used to describe the model only hold true for certain ranges of values and that other formulas or equations may be needed to describe the behaviour in different ranges. If a model is deemed inappropriate during the validation phase, refinements can be made to the model or another model may be developed altogether. Most model development methods allow for this. The previous discussion still does not address the question of which model is the best model for a certain system or problem. This is a rather complex question and does not have a simple answer. In the literature, for instance Giordano and Weir (1983:34) alludes to the fact that the

choice of type of model to use depends on the resources available and the accuracy and flexibility required, but that choosing a best model is still a daunting task.

In conclusion, various ways of constructing models exist and multiple definitions for mathematical models can be presented. For the purpose of this thesis, a model will be a mathematical representation of some real world system, with known assumptions on the inputs and required outputs of the model. To obtain the best solution for a model instance optimisation is required. Different techniques for optimisation will be discussed at a later stage.

2.2 Graph representation

Of particular interest for network design and optimisation in the design process is the representation of networks as graphs. Here a graph will be used as a model for the real world network. The topic discussed now is the representation of graphs in terms of computers and data structures. This is necessary as certain algorithms are required to do certain calculations and operations on the representations. An example of a simple operation that can be performed on a graph is the traversal of a graph from one point to another. This is for instance important when determining the distance between two points on the graph. The function used to define the distance can differ from application to application, but the general idea is the same. An indication of how important graph representation is, is evident from the fact that most modern text books on data structures devote an entire chapter to this representation, see for instance (Malik (2004) and Preiss (1999)).

Before describing the methods that can be used to represent graphs, a definition of a directed graph as found in Preiss (1999:539) is presented.

> *A directed graph or digraph is an ordered pair* $G=(V,E)$ *with the following properties:*
>
> 1. *The first component* V*, is a finite, non-empty set. The elements of* V *are called the vertices of* G*.*
> 2. *The second component,* E*, is a finite set of ordered pairs of vertices, that is* $E=V\times V$*. The elements of* E *are called the edges of* G*.*

This definition can be modified for the case where the vertices are not directed, which are referred to as arcs. In the following section directed arcs or vertices will be used unless otherwise indicated.

The question on how graphs can be represented on a computer will now be addressed. This will be answered in two parts. The first part will deal with static data structures. The second part will focus on more dynamic data structures, which is a newer approach. The following section is by no means complete or exhaustive, but merely gives an indication of how graphs may be represented. During implementation, many of these techniques are modified to suit the specific application. According to Gondran and Minoux (1984:8), not all graph representations are equivalent in terms of efficiency. This efficiency can be in terms of algorithmic performance, storage of data or other aspects.

2.2.1 Static data structures

For the purposes of the thesis, static data structures will be considered to mean structures using static memory allocation structures, like arrays and vectors. This means that once memory has been allocated, it cannot be resized or reallocated until the end of the program. The section will be divided into two parts, the first dealing with matrix based representations and the second with vector based representations.

2.2.1.1 Vertice-arc incidence matrix based representations

Some of the most easily understood graph representations can be made using the so-called vertice-arc incidence matrix-type representations. In this representation of directed arcs, edges are represented as either entering or leaving a vertice or if no arc exist between a set of nodes. Formally it is defined by a matrix $A=\left(a_{ij}\right), i=1,\ldots,N; j=1,\ldots,M$ where $a_{ij} \in \{-1,0,1\}$; N is the number of arcs and M is the number of vertices of the problem. Each column represents an arc and contains a 1 in the row corresponding to the vertice from which the edge is directed and a -1 in the row corresponding to the vertice to which the edge is directed, all other entries in the column are 0.

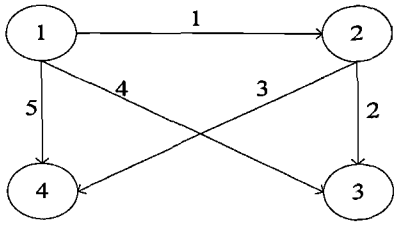

Figure 2-2 Sample graph used to illustrate graph representation

Figure 2-2 contains a graph used for illustrative purposes. The vertice-edge incidence matrix for the graph is as follows:

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ -1 & 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 \end{bmatrix}$$

This is one form of graph representation for directed graphs. For undirected graphs a similar representation can be used, with the difference that since no direction is associated with the arcs only 1's are used in the representation to indicate which two nodes are connected. The values of 1 and -1 can be substituted for values that indicate the weights of the edges, for example if the graph is used to represent distances measured between points.

Assume the case where each vertice is only connected to a maximum of one other vertice, and no vertice is connected to itself. In this case $M = N$ and according to Preiss (1999:545) a possible problem with this type of representation is then that the amount of space needed to represent a graph is independent of the number of edges of the graph. The problem is that storage space proportional to N^2 is needed to represent a graph. According to Gondran and Minoux (1984:8), in general, if more arcs are allowed between two vertices the storage space is proportional to N^2 and if M is a lot smaller than N, this leads to a sparse matrix with possibly a lot of wasted storage. Also, no arcs can be represented to flow from a vertice to itself. In order to overcome these two issues the type of data structures presented in the following section was proposed.

2.2.1.2 Adjacency matrix based representations

An adjacency matrix for a graph is a matrix representing whether an arc exist between two vertices, the general form being a matrix $A = \left(a_{ij}\right)\ i = 1,\ldots N; j = 1,\ldots,M$ where $a_{ij} = 1$ if and

only if $(i,j) \in E$, otherwise $a_{ij} = 0$. This representation works if there is no more than one arc between two nodes with the possibility of one arc connecting a specific vertice to itself.

The adjacency matrix representation can be made more compact by using two tables, possibly implemented as vectors in computing terms. Gondran and Minoux (1990:10) call these two vectors α and β. Assume that the entries of the two tables are labelled starting from 1. The table β contains the endpoints of arcs while the entries of table α indicate where the endpoint of arcs leaving a vertice can be found in table β. This means that for vertice i the endpoints of arcs leaving vertice i can be found in table β from entries $\alpha[i]$ to entries $\alpha[i+1]-1$. For example the graph presented in figure 2-2 is the following:

$\alpha =$

1	2	3	4	5
1	4	6	7	8

and $\beta =$

1	2	3	4	5	6	7	8
2	3	4	3	4	-1	-1	-1

In this representation the -1 indicates that no arcs are leaving a specific vertice. Note that even though there are only 4 vertices, a fifth entry is made in table α. This representation is more compact and in general uses less storage space to present the same graph. It also allows multiple arcs to be represented per vertice. This representation can be extended by adding another table to represent for example the weights of the arcs, or to indicate the number of the arc, if the arcs are labeled.

Many variations of the above mentioned techniques exist. However, it is not the aim of this thesis to present a complete and exhaustive list of graph representations. Only an introduction is presented and as such only these two static memory based representations are offered. The next section deals with dynamic data structures used for graph representation.

2.2.2 Dynamic data structures

With the advent of object oriented programming new methods of representing graphs became available to programmers. Authors of textbooks like Preiss (1999) and Malik (2004) devote entire chapters to this topic. With object oriented programming, data and the operations on the data are combined, leading to a concept called data encapsulation, Preiss (1999:2). This allows for software reusability and quicker program development. It also allows for enhanced operations on data being represented. For graphs, this means that algorithms such as the shortest path algorithms, can be coupled closely with the graph representation. It also allows

better memory management, as no large blocks of memory is allocated at runtime. Memory is allocated dynamically for each vertice and arc as is necessary.

In terms of object oriented programming vertices and arcs can be represented with objects. This allows more information to be stored about each vertice and arc. It allows programmers to enhance the reuse of existing representations through a process called inheritance. Object oriented programming is discussed here as it allows methods for dynamically adding vertices and arcs to a graph representation at the run time of the program, giving more flexibility. Another advantage is that all the vertices and edges form part of a graph object, representing the total representation of the graph. Using the graph object algorithms can be implemented that perform operations on the representation through a single well defined interface.

Object oriented programming allows operations like getting a set of all the arcs leaving a specific vertice. This set is called a container in object oriented programming terms. Working through this container presents a method to enumerate all the arcs leaving a vertice.

Using object oriented programming techniques allows programmers to use computing resources more efficiently when representing graphs and appears to be gaining popularity when representing graphs. For a more complete discussion on this, see Preiss (1999) Chapter 16 or Malik (2004) Chapter 21.

2.3 Conclusion

Models can be used to represent real world systems, as such modeling can be considered both an art and a science. It is important to note that a good model presents a representation of some part of reality allowing manipulation of the model to obtain solution that may be ported to the real world system. This process of obtaining a solution will be dealt with in a subsequent chapter. The last part of this chapter is devoted to ways of representing graphs. This is important as graphs will be used to model telecommunications networks in parts of this thesis.

The next section of the thesis is devoted to complexity theory, which aims to show that not all problems that can be modeled are of equal difficulty to solve.

Chapter 3 Complexity

When investigating problems, it is necessary to know how difficult the problem under investigation is. The field of complexity study is concerned with how difficult (or easy) a specific problem is to solve. This chapter gives an introduction to the complexity of problems, but the aim of the study was not to uncover new knowledge in the field of complexity, and therfor the introduction is very brief. It does however aim to introduce the reader to an identification of which classes of problems are difficult. This is needed to put the models investigated later into perspective.

3.1 Introduction

An important issue when solving a specific problem is how easy (or difficult) it is to solve an instance of a specific problem. It may be that a certain problem is very difficult to solve, while a problem with a similar type of formulation may be far easier to solve. It is worthwhile to know that problems can be divided into different classes of difficulty. It is also true that simply knowing that a problem may be difficult to solve is not enough. The problem still needs a solution, if not the exact optimal solution, then at least to a reasonably good solution in an acceptable time frame.

An interesting illustrative example of this is given by Garey and Johnson (1979:1). It is stated that a developer of algorithms was tasked with developing an efficient algorithm for a specific problem. The algorithm developer could not find an efficient algorithm for the specified problem and had three possible avenues to pursue. The person could inform his boss that:

- He could not find an efficient algorithm and thereby run the risk of losing his job.
- The problem is inherently difficult and that no efficient algorithm exist. Proving that there are no efficient algorithms may be a daunting task.
- By applying complexity theory, he could state that he could not obtain a solution process, but that other famous people could also not find efficient algorithms to similar problems.

The last approach depends on being able to prove that a specific problem belongs to a certain class of problems. The most difficult class is known as NP-complete problems. The aim often is to show that a problem has an "equivalent" difficulty to NP-complete problems as described by Garey and Johnson (1979). A lot of theoretical work has been done on NP-complete problems, as is for instance discussed by Papadimitriou (1995). As stated by Hromkovic (2003:121), the objective of complexity theory is to find a specification for the

class of practically solvable problems and to develop methods to classify problems according to their membership of this class. But, being able to prove that a problem belongs to the class of NP-complete problems should not be the end, but rather be the beginning of the search for an efficient solution process or algorithm.

In the next section more information will be presented on the class of NP-complete problems and also the measuring of complexity.

3.2 Measuring complexity

Before we can measure complexity, it is first necessary to introduce certain important concepts. The first important concept to be defined is the notion of an algorithm. An algorithm as defined by Deitel and Deitel (2003:72), can be seen as a procedure to solve a problem in terms of:

1. The actions to perform;
2. the order in which the actions are to be performed.

It is thus important not only which actions are to be executed, but also in which order they are to be executed. Performing an incorrect action at a specific point in time will not solve a problem correctly.

It is also important to note that several "quantities" may be investigated when determining whether or not an algorithm solves a problem efficiently. An algorithm may be considered efficient if it solves problem instance to optimality in reasonable amounts of time. This same algorithm may be very inefficient when considering its memory requirements or its use of some other form of computing hardware. Hence it is important to define what will be used as measure of efficiency. In the following exposition the time taken by an algorithm to solve problems will be used as a measure of its efficiency.

3.3 Complexity functions

In order to measure the efficiency of an algorithm, it is first necessary to introduce notation. Firstly, the concept of a complexity time function is introduced. The complexity time function for an algorithm indicates the time requirements, for each specific problem size, as the largest time required for solving the specific problem size (Garey and Johnson (1979:6)). It can be seen as a "mapping" of problem size to the maximum time needed to solve a problem instance of a specific size.

In complexity theory complexity functions from $\mathbb{N}$ to $\mathbb{N}$ are considered. As argument for these functions the letter n is used. Even though a mapping function f may produce values that are non-integer or negative, the numbers will always be considered to be non negative integers. This means that the function denoted by $f(n)$ will usually mean $\max\{\lceil f(n)\rceil, 0\}$. One of the aims of complexity theory is to represent the order of the complexity time function, denoted by O, and this will give an indication of possible performance of an algorithm.

In complexity theory it is said that $f(n) = O(g(n))$ if there exist positive integers c and n_0 such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$. This is a definition found in various basic data structure text books, for example Preiss (1999:36). This is also known as the big "O" notation. It means that the function f grows as fast a g, or slower. Another way of looking at this is that the maximum time taken by function f, will be less than the function g evaluated as a function of the problem size to which the algorithm is applied. For example, $f(n) = \mathrm{O}(n^2)$ means that the maximum time needed to execute the algorithm increases as quadratic function of the problem size. This means that a doubling of the problem size would result in a four-fold increase in the maximum time needed to solve a problem.

Another notation commonly defined is the opposite to that described above and is written as $f(n) = \Omega(g(n))$, meaning that $g(n) = O(f(n))$. This gives an indication of the least amount time needed to solve problem instances with an algorithm. Lastly, it can be written that $f(n) = \theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, meaning that f and g has the same growth rate. This is an exact measure of how long an algorithm will take, meaning that it will take no longer or shorter than a function of g. This would seem to be an ideal measure, but it is not often that this measure can be computed. On many oocasions the data to which the algorithm is applied may play a role, especially for the best time analysis and for this reason the θ norm is not used often. Generally an algorithm is considered to be efficient if the worst time it would take is considered to be "acceptable".

The question now is, when can an algorithm be considered to have a good complexity function? This can be answered by using the notation defined above to define a polynomial time algorithm. A *polynomial time algorithm* is defined as an algorithm with a time complexity function of $O(p(n))$ for some polynomial function p where n is a measure of the size of the problem. An algorithm where the time complexity function can not be bounded by a polynomial function is usually referred to as an *exponential time algorithm* (Garey and Johnson (1979:6)). To illustrate the effect of different time complexity functions, table 3-1 is quoted from Garey and Johnson (1979:7).

	Size n					
Time complexity function	10	20	30	40	50	60
n	0.00001 seconds	0.00002 seconds	0.00003 seconds	0.00004 seconds	0.00005 seconds	0.00006 seconds
n^2	0.0001 seconds	0.0004 seconds	0.0009 seconds	0.0016 seconds	0.0025 seconds	0.0036 seconds
n^3	0.001 seconds	0.008 seconds	0.027 seconds	0.064 seconds	0.125 seconds	0.216 seconds
n^5	0.1 seconds	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	0.001 seconds	1.0 seconds	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	0.059 seconds	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

Table 3-1 Comparison of execution times for some polynomial and exponential time complexity functions

By examining table 3-1 it may appear that in general polynomial time algorithms should be preferred to exponential time algorithms. Generally, for the same input will polynomial time functions outperform algorithms with exponential time complexity functions? This may not always be true, as only a theoretical worst case is examined. In general some algorithms with exponential time complexity functions perform quite well. As an example, the simplex method can be taken. It was known to have an exponential time complexity function for many years, as was for example shown by Zadeh (1973) and by Parker and Rardin

(1988:108), but in practice it performs quite well. It is still used in most linear programming software on the market today, for example CPLEX from ILOG (ILOG (2002:158)).

3.4 Complexity classes

An important concept that needs to be defined at this point is the definition of a *decision problem* or alternatively the *language recognition* problem (Parker and Rardin (1988:23)). These problems may be solved correctly by either a yes or no response to any well formed input for the problem. This is in contrast to other (optimisation) problems that require a full solution vector or an optimal value.

There is, however, a relationship between these problems. It is generally possible to transform an optimisation problem into a decision problem. A classic example presented by Parker and Rardin (1988:23) and also by Garey and Johnson (1979) is the travelling salesman problem. The optimisation formulation of the problems aims to find a closed cycle (without any subtours) through a given set of vertices in a graph that has a minimum aggregate edge weight. Analogously, the decision problem may be formulated as follows:

Given a graph $G=(V,E)$, with weights w_{ij} for vertex pairs (i,j) and w, does there exist a closed cycle in G meeting every vertex of V having a total edge weight no greater than w?

Generally the edge weight used for the travelling salesman problem is the distance between vertices.

The previous description forms part of the exposition on complexity classes as complexity classes mostly refer to decision problems and not to optimisation problems. As optimisation problems can be transformed to decision problems, it holds true that if a decision problem is easy to solve the same should be true for the optimisation problem. It is important to note that the aim is to categorise problems and not necessarily to classify algorithms. In this sense it is noteworthy to present a definition for intractability, as defined by Garey and Johnson (1973:8): (We) "refer to a problem as intractable if it is so hard that no polynomial time algorithm can possibly solve it".

This specific field has generated a lot of interest in the past, and most text books on algorithms and optimisation devote time to discuss the problem of complexity classes (see for instance Hromkovic (2003) or Parker and Rardin (1998)). For this thesis the goal is not to

present this work in detail, but rather give an intuitive discussion of the subject. Various complexity classes exist, but the ones we are interested in are the classes P and NP. Informally the class P can be seen as the class of problems solvable by polynomial time computations or algorithms. This concept is closely coupled to the concept of a representation of a problem instance, also called a language or encoding scheme (for more information see, Garey and Johnson (1979:20)). For the purpose of this thesis, it will suffice to say that an encoding scheme is a representation of a problem instance and that not all representations are equal. For instance, 1 000 000 can also be represented as 111111...111 where the number of 1's equal 1 million. For this exposition "reasonable" encoding schemes are required, even though this may also be the subject of conjecture. To summarise, problems belonging to class P can generally be solved with polynomial time algorithms. But generally real life problems are much harder to solve and for this the class NP exist. These problems are generally more difficult to solve and it is even difficult to prove that a problem belongs to this class. It is generally hard to prove that a problem is of the class NP (Garey and Johnson (1979:45)), and as such, the problem to identify a problem as belonging to the class NP constitutes the transformation of the problem to be classified to a problem that is known to be of the class NP. The transformation process should be of polynomial time, if it is to be usable. For a more detailed description of this process, see Garey and Johnson (1979:46). The problems of interest for this study belong to the class NP. In the next chapter, solution methods for these types of problems will be discussed.

Chapter 4 Solution strategies

Once a model has been formulated, it becomes necessary to investigate algorithms to "solve" the model. The previous chapter on complexity showed that many problems may be too complex to solve by simply enumerating all the possible combinations in a brute force manner. The simplest way to solve almost any problem is to explicitly enumerate all possible combinations and for each combination to test if all the constraints hold. For each feasible combination an objective function value is calculated. The combination with the best objective function value is chosen as the optimal solution. This is not a very tractable solution strategy, especially for instances involving large scale real-world problems. Another problem with this approach is that that no combination may exist that does not violate one or more of the constraints.

To solve the problem as stated above, i.e. to obtain solutions for a problem instance, optimisation methods are employed. The goal of many optimisation methods is to implicitly enumerate all possible combinations. This means that not all possible combinations are generated, but that the generation of combinations is done in such a way that a statement can be made regarding all possible combinations. This chapter is devoted to some solution strategies for optimisation models, specifically the methods prevalent in telecommunications networks design planning. This chapter is only a very brief introduction to solution methods available to practitioners in the field of optimisation. The aim is not to present an exhaustive list of solution strategies, but rather to give a taste of some of the methods. More specifically some of the methods used in telecommunication network planning problems are discussed. An important observation is that not all solution methods are applicable to mathematical models of a specific type. For instance, some solution methods can only be applied to models with linear objectives and constraints.

4.1 Introduction

The idea of optimisation of a model is to obtain a solution that satisfies all constraints required in the model. The aim is more than just finding a solution , it is actually to obtain a solution that is considered best. This process of searching for solutions can be referred to as a solution strategy. This type of definition for optimisation can be found in various books on the subject of optimisation, one of these definitions presented by Rardin (1998:4) is:

"Optimisation models (also called mathematical programs) represent problem choices as decision variables and seek values that maximise or minimise objective function variables subject to constraints on variable values expressing the limits on possible decision choices."

Winston (1995:1) states that in a mathematical programming problem a decision must be made to choose decision variable values to maximise or minimise an objective function, while satisfying the constraints included in the formulation. A set of decision variable values that does not violate any of the constraints of the formulation is known as a feasible solution for the data instance. This is coupled with the idea of a feasible region, which is a collection of alternatives that satisfy all constraints. Rardin (1998:33) states that an optimal solution is a choice of feasible decision variables with an objective function value at least equal to that of any other set that satisfies all constraints. The objective function value of the optimal solution is sometimes referred to as the optimal value. It is possible that a problem instance can have alternative optimal values, there is not necessarily only one optimal value. It is also possible that a problem instance may be unbounded if the feasible region is unbounded. Another possibility it that the problem instance is infeasible, this happens when no choice of decision variables satisfies all constraints. In this section the use of the word model can also be seen as a data instance of a certain problem.

Choosing the decision variables in such a way that an optimal solution is obtained is not a trivial task. The aim of this chapter is to present commonly used methods to obtain feasible solutions for problem instances, preferably optimal solutions. For the purpose of this thesis, an exact method will be a method that produces an exact optimal solution. An exact optimal solution is a solution can be proven to be at least as good as any other feasible solution. Some of the methods presented may also prove that no feasible solution exist for a specific data instance. According to Parker and Rardin (1988:358) an informal description of a nonexact method is that it may produce feasible solutions but it cannot guarantee optimality of the solutions generated. For this thesis generally models will be formulated in the following way:

$$\begin{aligned} &\min \textit{or} \max \left(\text{Objective function or multiple objective functions}\right) \\ &\textit{s.t.}\ \text{Constraints} \end{aligned} \tag{4.1}$$

where *s.t.* means subject to. Generally constraints are represented as equations that define the feasible set, but the constraints may also be formulated in terms of sets as well.

4.2 Linear and Integer Linear Programming problems

The first important types of models for which solution strategies are presented, is the Linear and Integer Linear Programming models. These types of models are deterministic with certain specific assumptions. These models are generally formulated as:

$$\min \sum_{j=1}^{n} c_j x_j \tag{4.2}$$

$$s.t. Ax \leq b \tag{4.3}$$

where $A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$ is known as the constraint matrix and $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ can be referred to as a decision variable vector. The vector $b^T = \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix}$ stores the bounds on the constraints. Some assumptions for these type of models is proportionality and additivity. For a more comprehensive list of assumptions for linear programming refer to Bazaraa *et al.* (1990:3). If all the variables in x may take on real values $x_j \in \mathbb{R}$ $j = 1,2,\ldots,n$, the model will be referred to as a linear program (LP). If all of the variables $x_j, j = 1,2,\ldots,n$ may only be integer valued $x_j \in \mathbb{Z}$ $j = 1,2,\ldots,n$, the problem is sometimes referred to as an Integer Linear Program (ILP). If some of the x_j's are integer valued and some real valued, the problem is sometimes referred to as and Mixed Integer Linear Program (MILP). If the variables x_j may have values 0 or 1, $x_j \in \{0,1\}, j = 1,2,\ldots,n$, the problem is sometimes referred to as 0-1 Integer Linear Program. Nemhauser and Wolsey (1988:3) also introduce the concept of combinatorial optimisation where an objective function is minimised or maximised subject to constraints (which can be equality or inequality constraints) with some of the variables restricted to having integer values. This is also referred to as discrete optimisation problems, as the variables may only take on discrete values.

Linear models are one of the models that has been investigated extensively and numerous solution methods are available to solve this type of problems. For example by using the well known simplex method, interior point methods, etc. The LP, ILP and MILP models were used extensively in this study. Standard off the shelve software like CPLEX from ILOG are available to solve problems of these types.

4.3 Branch and bound methods

Branch and bound methods use an implicit enumeration scheme to obtain solutions to integer programming type models. A discussion is presented by Johnson and Nemhauser (1992:84). A more formal discussion presented in Salkin and Mathur (1989:245). The basis is to use a linear programming relaxation, (e.g. $0 \leq x_j \leq 1$ rather than $x_j \in \{0,1\}$ for the binary case), and to consider a partitioned search space. Branch and bound methods are widely used in commercial software applications and in custom solution strategies for certain problem.

The basic idea of the branch and bound algorithm is as follows. Assume that a maximisation problem being investigated is an ILP model with 0-1 variables. Assume the linear programming relaxation, of this problem is denoted by ILPR. The idea is to solve the relaxation ILPR, if it produces an integer valued solution, this solution will be an optimal solution to the ILP model as well. If there is no solution, then the ILP model has no solution either. In many cases the solution to ILPR produces solutions with variables that do not have integer valued values. The algorithm now chooses one of the non-integer valued variables and creates two sub-problems. One sub-problem will set the variable to value 1 and the other sub-problem will set the variable to value 0. The optimal solution to the ILP model will be obtained by solving both sub-problems and taking the best solution generated by the sub-problems. The process of dividing into further sub-problems may need to be repeated numerous times.

By dividing problem instances, the complete problem is enumerated. The process of dividing a problem into subparts is known as branching. The number of sub-problems created may grow exponentially for complex problems and bounds are used to limit growth. A sub-problem is not divided further once it produces an integer valued solution or if it is infeasible. If a sub-problem has a feasible integer solution, this solution serves as a lower bound for the optimal value of the IP problem. A best lower bound is kept and updated as necessary during the solution process whenever new integer valued solutions are produced. This lower bound may also show that a certain sub-problem cannot produce a better lower bound, i.e. the ILPR optimal solution for the sub-problem is lower than the best lower bound known. In such a case the sub-problem is not investigated further and is discarded. This is known as pruning. The creation of sub-problems can be represented as a binary tree. This can be visualised as two branches that are created when a sub-problem is divided into two additional sub-

problems. Figure 4-1 gives visualisation of a branch and bound process. The circle represents a sub-problem. The top circle represents a case where the ILPR sub-problem produced a solution where the variable x_j has a value of say 0.5. Two sub-problems are created, one where x_j is constrained to a value of 0 and one where it is constrained to a value of 1, this is indicated by the link between the nodes. Pruning is when one or more of the branches is removed from the binary tree. This can be done if a sub-problem is infeasible or if it produces a solution value smaller than the current lower bound. It is also possible to remove existing branches when the current lower bound is updated.

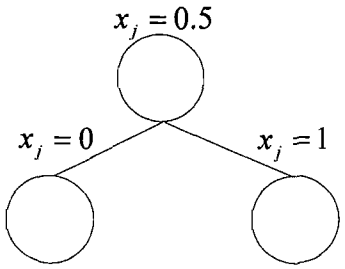

Figure 4-1 Tree representation of a branch and bound process

The discussion of the branch and bound algorithm presented above is on a very simplistic, but gives some indication of the mechanisms in of the algorithm. There are questions that have not been answered yet. For example, which sub-problem should be chosen to create additional sub-problems? Various heuristic strategies can be used, a few of them are:

- Depth first search. The search tree can be expanded in a depth first search manner.
- Choose the sub-problem with the most promising objective function value.
- Determine a heuristic value that identifies potential good sub-problems. In this case choose the sub-problem with the most promise.

Also, once a sub-problem has been chosen to divide further, which variable should be used to guide the branching process? A possible choice may be to choose the variable with the highest likelihood of producing an integer solution, how this is done is beyond the scope of this document. For more branching and variable selection rules, refer to Linderoth and Savelsberg (1999) and Johnson and Nemhauser (1992). The discussion above is presented for 0-1 variables, branch and bound methods can also be used for general integer valued variables, see for instance Salkin and Mathur (1988:245).

Other ways of solving sub-problems other than linear programming can also be used, an example of this with an application in the telecommunications field can be seen in Shaw and Cho (1996).

Branch and bound methods are very important and many ILP commercial software packages are based on branch and bound methods.

4.4 Branch and Cut

According to Alevras and Padberg (2001:325) the most successful approach to solving MILP problems is a branch and cut approach. This approach combines a cutting plane approach with a branching to partition the search space. Cutting plane approaches aim to add cuts to problem to identify the solution without having to use a branching strategy. Alevras and Padberg (2001:325) note that computational results are at best mixed for cutting planes approaches.

Alevras and Padberg (2001:327) present a flow chart of a typical branch and cut problem solver. It has four major building blocks:

- Presolver: Here the problem is investigated and if possible the current formulation is improved.
- LP Solver: This is solver used to solve all ILPR problems encountered during the solution process.
- Heuristic: Here attempts are made to obtain good feasible solutions. This can be done via a separate procedure or by using the ILPR relaxation of the problem.
- Constraint generator: Here cuts are generated and added to the LP formulation. This new formulation is then optimised again. Redundant constraints may also be removed in this procedure. The constraint generator looks for violated constraints, if none exist, branching is done, otherwise constraints are added to the current formulation.

Some notation is required for the flow chart. The objective function value of the best known integer valued solution is referred to as z^*. The objective function value of the current ILPR relaxation is referred to as z_{LP}. During the fixing step, an attempt is made to set non-integer valued variables to their upper or lower bounds, without changing the objective function value. This may be done with reduced cost information provided by the ILPR, or by some other technique. In this discussion adding constraints could be seen as being equivalent to adding cuts or inequalities to the formulation. Various families of inequalities exist, Fischetti

et al. (2000:246) lists cover and clique inequalities. For more information the interested reader is referred to Padberg and Rinaldi (1990) and Padberg and Rinaldi (1991).

An example of a branch and cut algorithm is found in Fischetti *et al.* (2000), where a branch and cut algorithm is presented for frequency allocation in mobile radio services.

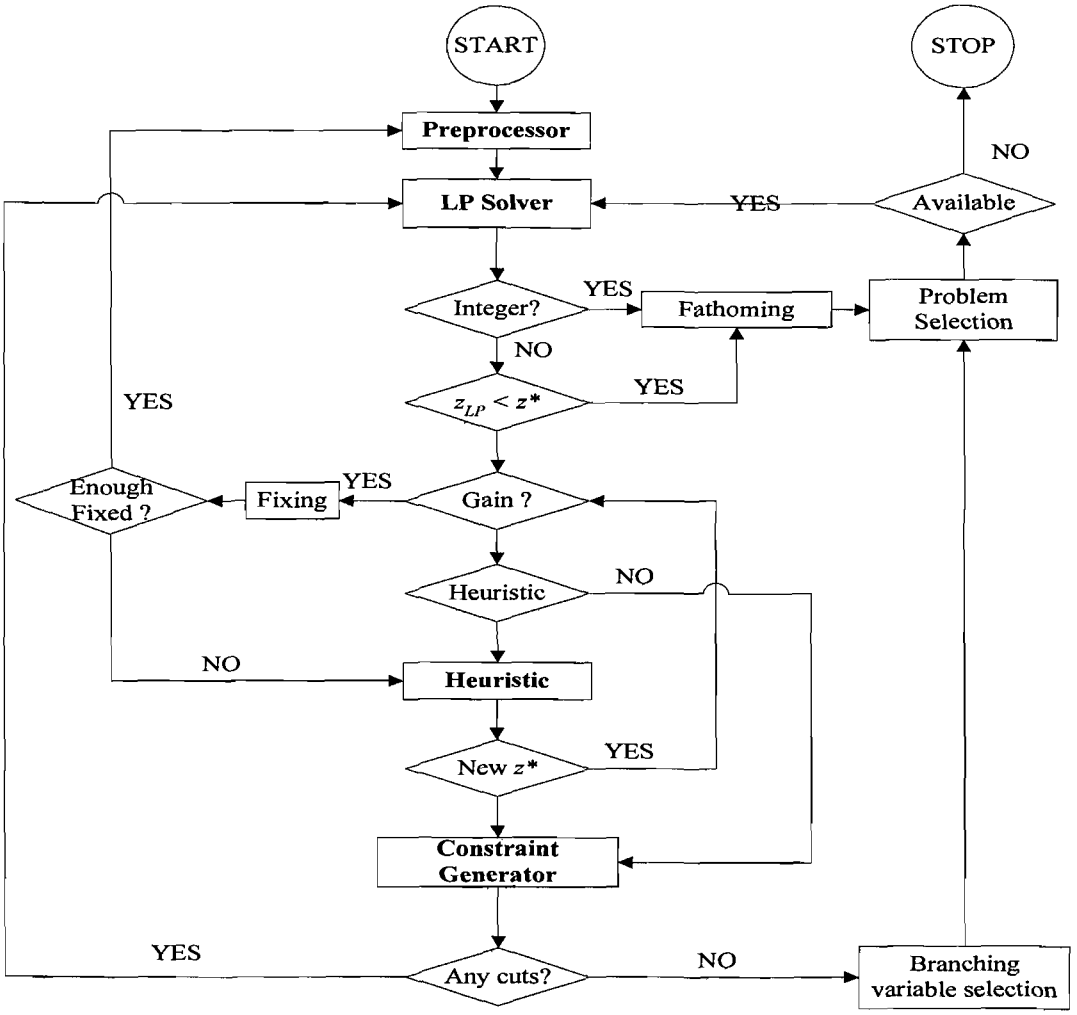

Figure 4-2 Flow diagram for a branch and cut solver (Alevras and Padberg (2001:327))

4.5 Dynamic programming

Dynamic programming is an implicit enumeration process, that is still complete, to obtain solutions to optimisation problems, Gondran and Minoux (1984:629). To apply a dynamic programming algorithm the problem needs to have a specific structure, sometimes referred to as being of sequential type. This means that the problem can be divided into smaller (nested) sub-problems that can be solved more easily than the original problem. The smaller sub-problems are used to determine the optimal solution of the larger problem. The sub-problems

are solved sequentially. Winston (1994:1010) gives the following characteristics that are common to most problems that can be solved using dynamic programming algorithms:

- The problem can be divided into stages or parts with a decision that is required at each stage.
- Each stage has a number of states associated with it. This refers to the decisions that should be made at each stage, to make an optimal decision for that stage.
- The decision at a specific stage describes how the current state is transformed to the state of the next stage.
- If a state is given, the optimal decision for the remaining stages must not be dependent on previously reached stages or prior decisions.
- For a problem divided into *N* stages, there must be way to translate the objective function value from one stage to the next. This is frequently done using recursion. An important step is to create the recursion function to do these calculations.

The second last characteristic is known as the optimality principle and is an integral part of dynamic programming algorithms. It ensures that sub-problems can be solved sequentially without fear of suboptimal solutions. An example of this, is that if city C is on the shortest path between cities A and D, then the path between cities C and D will also be of minimal length. Another way of looking at dynamic programming approaches is that it some cases it starts at the end of the problem and works to the beginning. In the case of the calculation of the shortest path, it starts with sub-problem of determining the shortest path between the last city and the cities directly adjacent to it. This is a simple calculation, if a direct path exist, then the distance is known and only one possible decision can be made. This type of calculation is used in some routing algorithms where the principle of optimality is used, see for example Tanenbaum (2003:352).

Winston (1994:1041) states that a possible problem with dynamic programming is that the search space that dynamic programming searches can become extremely large leading to very long solution times. Another problem is that in many cases, if the recursion steps of the different states is not entirely completed, that no solution, not even a feasible solution is generated.

4.6 Heuristics

Heuristic methods aim to obtain solutions to problems in a reasonable amount of time. A major drawback of many heuristics methods are that most heuristics are nonexact methods, meaning that feasible solutions are generated, but optimality cannot be guaranteed. Heuristic methods are often used when a problem becomes intractable due to growth in the size of the search space for certain models. Heuristics methods often tries to exploit some inherent structure of the problem under investigation. They generally give no qualification of how good (or bad) the solution is that they generate. These methods may sometimes find local maxima or minima in the search space rather than obtaining a global maximum or minimum.

4.7 Simulation

According to Winston (1994:1183) simulation is a technique that imitates real-world systems over time. In order to imitate a real world system, a simulation model is required. With the advances made in computer hardware and software simulation has become more and more attractive, as the computational aspect can be handled efficiently with computers. One of the advantages of simulation is that in most cases it is very straight forward and easy to set up. Simulation is done under certain assumptions, these assumptions can in most cases be changed easily, leading to simulation techniques being good for what-if analysis. A disadvantage is that some systems can be computationally intensive to simulate. When used for optimisation problems, it is usually not possible to guarantee optimal solutions. It is more often used a tool to explore the behaviour of some real-world system under varying conditions.

4.8 Tightening bounds

When solving IP and MILP problems, branch and bound type algorithms have been very successful. Linderoth and Savelsbergh (1999:173) note that to improve the efficiency of these algorithms research has focussed on two areas namely:

- Improving the search strategies. This may entail enhanced decision making on the node to branch on, and which variable to choose to create the sub-problems with. This leads to various strategies and rules to improve the speed of the branch and bound methods.
- Improving the bounds of the linear programming relaxation. This leads to smaller integrality gaps, where the integrality gap is the difference between the LP relaxation solution value and the best current integer solution of the problem. Johnson and

Nemhauser (1992:85) states that the success of branch and bound methods depends on how closely the LP relaxation approximates the MIP or MILP solution.

The interested reader is referred to Linderoth and Savelsbergh (1999) for a examination of search techniques found in branch and bound methods. They remark that most work have focussed on improving the bounds in recent times, rather than improving the search methods.

The way in which a problem is modeled can lead to different formulations. The formulations may have the same optimal solution value, but may have very different LP relaxation solution values. With branch and bound methods that rely on LP relaxations, the formulation with a better LP relaxation solution would be preferred. Johnson and Nemhauser (1992:86) presents some methods for improving the bounds. For example, disaggregation is discussed where one constraint is replaced by a number of constraints. In the example presented, it is shown that the constraint

$$x_1 + x_2 + \ldots + x_n \leq nx_0 \tag{4.4}$$

can be rewritten as

$$x_j \leq x_0 \text{ for } j = 1, 2, \ldots, n \tag{4.5}$$

Even though the second formulation has many more constraints, the formulation is better. The interested reader is referred to Johnson and Nemhauser (1992) for more information. Nemhauser and Wolsey (1988:14) states in ILP problems it is of vital importance to use a good formulation for a problem, as it has a very marked impact on the tractability of a problem. In this sense tractability is used to denote the solvability of problem instances. An example of two formulations for the same problem with one being better due to a better bound is also presented by Nemhauser and Wolsey (1988:15).

An additional method to improve the bounds is to add valid inequalities to the problem formulation. The idea of that the valid inequality must cut off or remove some feasible parts of the LP relaxation that are not feasible in the ILP model (Rardin (1998:644)). Some of these types of inequalities are also used in cut generation. Examples of these are found in Johnson and Nemhauser (1992), Padberg and Rindaldi (1990, 1991). These inequalities are not discussed in detail in this thesis, but it is important to note that adding these types on inequalities can lead to better formulations and improved solution times.

Nielson *et al.* (2004) present a practical example where a formulation with a better LP relaxation is used to obtain solutions quicker.

4.9 Conclusion

The methods presented in this chapter is but a few of the techniques that can be used for obtaining solutions to optimisation problems. The aim of this chapter is to present the reader with a few of the general methods used in the telecommunications planning field to solve problems. In the next chapter telecommunications networks will be discussed and some of the problems associated with telecommunication network design.

Chapter 5 Telecommunication network architecture and design problems

In the introduction given in Chapter 1, the tremendous evolution of telecommunication networks and telephony services is discussed very briefly. What has remained constant is the quest to implement telecommunication services efficiently. This efficiency is not a single facetted goal. Efficiency can be defined in terms of cost, implementation, technical requirements etc. In this regard, optimisation has a place in the planning and implementation of telecommunication services. Gavish (1992:154) states that a goal of network planning is to balance the investment made by the telecommunication company with the services and the quality of the service received by the customer. This chapter aims to introduce a few models used in this field. It is also important to note that optimisation can be done in various aspects of telecommunication networks. The focus of this thesis is on planning issues in network design and specifically in local access telecommunication network (LATN) design. Other planning issues include frequency allocation, routing of traffic, billing etc. An interesting list of combinatorial optimisation problems found in the telecommunication field is presented by Gavish (1991:31). The goal of this chapter is to introduce telecommunication networks more formally and to show a few of the problems associated with the planning and implementation of these networks. This is a very broad field and as such this is merely a sample of problems found in the telecommunications field. Most of the problems will not be presented as mathematical models, even though numerous mathematical models exist, but will be presented as a problem description.

5.1 Introduction

Most modern telecommunication networks have the same basic structure, as mentioned in the first chapter. More formally, telecommunications networks are generally divided hierarchically into the following three parts:

- Long distance networks: Typically connect areas long distances apart and basically connect switching centre networks. This type of network is sometimes referred to as a backbone network, with the resulting traffic being referred to as backhaul traffic. Traffic is mostly transmitted digitally on fibre optic connections.

- Switching centre networks: Normally connect switching centres within a town or city. On this level traffic is sometimes aggregated to use the network equipment more efficiently. This can be done using multiplexing techniques (Tanenbaum (2003:137)).
- Local access networks: Connect individual subscribers to the network through a switching centre. This part of the telecommunication network is also known as the local loop or last mile problem.

Figure 5-1 below shows the hierarchical structure of telecommunications networks as presented by Balakrishnan *et al.* (1991:243). Kershenbaum presented similar representations, see for example Kershenbaum (1993:11).

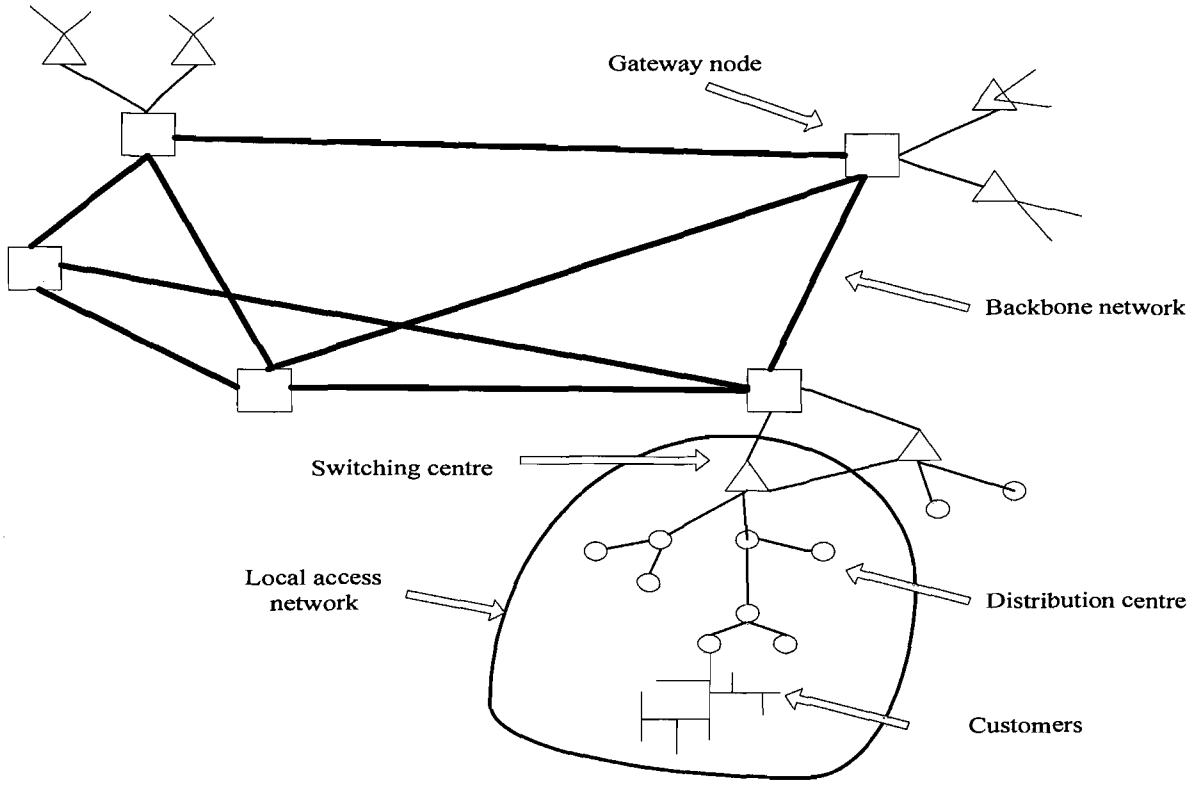

Figure 5-1 Representation of a hierarchical structure of telecommunication network

This is a broad classification that can be found in many sources, e.g. Balakrishnan *et al.* (1991:242), Tanenbaum (2003:121) and Mullet (2006:10). This is a very generic and high level view of telecommunication networks. However, when planning a network, in a lot of instances, the network is divided into two parts, namely the backbone network and the LATN (see for instance Gouveia and Lopes (1997:315) and Park *et al.* (2000:363)). These two parts of the telecommunication network will be discussed in the next sections. In the context of a two-level hierarchy, Gavish (1991:30) gives a generic network design process. This is an iterative process that aims to develop an acceptable network design as output. The steps of the process are:

- Determine locations for the backbone nodes. This is the location of hardware for the backbone network.
- Assign customer nodes to the backbone nodes, also sometimes referred to as clustering.
- Design the LATN.
- Interconnect the customer or user nodes and calculate the traffic requirements between the backbone nodes.
- Design the backbone network.
- Connect the backbone network, assigning link capacities and determining the routes for messages.
- Evaluate the network, using criteria such as response time, reliability etc.
- Make a decision on whether or not the design is acceptable. If the design is not acceptable, it may be necessary to return to the first step of the process to improve the network design until a satisfactory design is produced.

This is only a generic design process and some of the steps in the process could involve numerous sub design problems to be solved. Some of these steps will be covered in more detail in subsequent sections.

Each of the parts of the networks may use different technologies and has different planning issues associated with it. For instance, the switching network and long distance network are in most cases digital, whereas the local access part may still be analogue. The LATN has also evolved in terms of access media. It is no longer only copper wire connecting subscribers or users to the network. It is now possible to use wireless technologies to connect users to telecommunication networks (see for example Agrawal and Zeng (2006) who give a presentation on cellular and other systems used for telecommunication network access). The LATN is also no longer just used for telephony, it is also used for data and other services. A connectivity option gaining popularity is to extend the use of fibre optic connections to the customer's premises or very close to it (see for example Mateus and Patrocinio (2006:345)). Yoon *et al.* (2005:452) present some of the connection possibilities for fixed line connections. Pupillo and Conte (1998) present some possible architecture with for this type of network.

5.2 Local access telecommunication network (LATN)

The LATN connects individual users to the telecommunication network (Balakrishnan *et al.* (1991:242)). This can be done by means of cable or more recently by using wireless

technologies. The wireless technologies have evolved tremendously over the last few years, Agrawal and Zeng (2006:6) gives a brief introduction to this evolution. Generally speaking, the wire line version of the LATN uses copper to connect users or subscribers to the network. This is the method of interest in this study. Balakrishnan *et al.* (1991:250) present a good discussion on the planning of this type of network. The importance of the fixed wire line connection is that data services in the newer generation telecommunication networks can be implemented very efficiently. For a more complete discussion on access technologies refer to Shepard (2002), which has a whole chapter on access technologies or Tanenbaum (2003), which also covers numerous technologies.

At this point it is helpful to discuss how a call is made through the telecommunications network, as discussed by Tanenbaum (2003:129). If a call is made to another subscriber in the same local loop, the call is routed (or sent) through the switching centre that the two callers have in common. All subscribers are connected to a switching centre through a local loop. If the person called is not part of the same local loop, but still in close proximity, the call can be routed through the switching centre network. If the call is to a subscriber a relatively long distance away, the call will usually be routed through the long distance network or backbone network. For this discussion no specific technology is presented, but a generalised view of calls in the local access network is presented. This view holds for data or other types of traffic as well.

The above discussion makes it clear that certain aspects need more discussion. To use the switching centre network and long distance networks efficiently, some sort of data or call aggregation is required. This enables economy of scale to reduce costs. This aggregation can be done in various ways, via for example the use of multiplexers (Balakrishnan *et al.* (1991:246)). Aggregation can be done in the LATN to ensure cost reduction and economy of scale.

Gouveia and Lopes (1997:316) presented some of the optimisation problems present in the LATN. These include the determining of the number of concentrating equipment devices required and selecting locations for these devices. After this is done, nodes or customers should be assigned to the concentrating devices, this is referred to as a terminal clustering problem. Once it is know which terminals or nodes should be connected to a certain concentrating device, a layout must be created for the terminals. This is a general view of a

complete design problem for the LATN, but generally the problem is not solved in one step. Another example of this type of planning approach can be found in Park *et al.* (2000) or Balakrishnan *et al.* (1994).

The following is a partial list compiled from amongst others, Balakrishnan *et al.* (1991), Gavish (1991) and Carpenter and Luss (2006) on problems associated with local access design issues:

- Concentrator location: Decisions must be made on the placement of concentrators. In some applications these decisions also incorporate the size of the concentrators. Additionally the nodes that should belong to a specific concentrator must also be identified (Gouveia and Lopes (1997)). Filho and Galvao (1998) present a tabu search heuristic for the concentrator location problem.
- Terminal layout (or node location): This problem deals with the way that nodes will be arranged in relation to concentrators. Given a concentrator topology, how will the nodes be arranged and efficiently connected to the concentrators? See for instance Gouveia and Paixao (1991) for more information.
- Switching centre connection model: In this problem the concentrators must be connected to one another. As calls can be routed between switching centres, the topology and connections between switching centres are also important in the total network design.
- Minimal spanning tree problems: Spanning tree algorithms are used in LATN design to produce trees that connect a set of nodes at minimal cost while not violating capacity constraints of the concentrators or the links. A logical constraint present in many models is that traffic may only be concentrated once (for example see Corte-Real and Gouveia (2007:1143)).
- Capacity expansion: For many networks, it is necessary to expand the capacity of the links and/or concentrators, after a while. This can for instance be due to increases in traffic generated. This problem can be solved by adding concentrators, or increasing the capacity of the links connecting users and concentrators. Sample models for this problem can be found in Corte-Real and Gouveia (2007) and Balakrishnan *et al.* (1995).
- Layered network design: Balakrishnan *et al.* (1991:258) present a layered network design plan for the LATN. This is done to facilitate the use of differentiation between different transmission rates, different transport media etc. A layered model type can be used to

model different transmission speeds on different levels. Another variation is a multi level design, see for example Cruz *et al.* (1999).

- When looking at wireless and cellular local access networks the allocation of frequencies to base stations must also be considered. In some instances only a limited number of frequencies are available for use. When adjacent base stations have the same frequency, this could lead to interference and may result in a lower quality of service. To limit interference, frequency allocation should be done with care and optimisation methods may help to alleviate problems in this regard. For an example of this type of planning, refer to Fischetti *et al.* (2000).

The issues listed above are some of the problems associated with LATN design, but the list is by no means exhaustive. It is merely presented as an introduction to the wide range of problems where optimisation may be important to LATN design. Kershenbaum (1993) also presented some algorithms and models for problems in this class, the interested reader may locate more information there.

5.3 Backbone networks

In many instances two customers who want to make a call are situated a long distance apart. According to Shepard (2002:309), this is where "wider area networks" are used to connect geographically distant locations. In this context the backbone network is the carrier of information between relatively distant locations. Different protocols and transmission technologies are used in the backbone network. Fibre optic cables are frequently used in the backbone network for the cost saving benefits it possesses as well as for other technical advantages it has over copper cables. For example, fibre is not affected by power surges or electromagnetic interference. For more information, refer to Tanenbaum (2003:99). An advantage is that through the use of various wave division multiplexing makes very high bandwidths available. See for instance Shepard (2002:356).

Some of the issues found in backbone network design are:

- Location problems: Similar to the LATN, the location of hardware within the backbone network is essential to create cost efficient networks.
- Topological design of the backbone network: When starting with a new network, the topology of the network should be determined. For an example, refer to Cortes *et al.* (2001). Another related article (Lee *et al.* (2000)) deals with an optical ring network that has partitioning as well.

- Routing problems: A simple definition for routing is the decision that must be made regarding the path to use to transmit data through the network from a source to a destination (Tanenbaum (2003:351)). Various goals can be set for deciding how traffic should be routed. It is for instance preferable to route traffic such as not to overload certain network elements. This may lead to congestion in certain parts of the network which may in turn result in poor quality of service. Amiri and Pirkul (1996:98) state that using a good routing algorithm can lead to considerable improvements, in terms of the users experience of the network and the utilisation of network resources. According to Tanenbaum (2003:351) routing algorithms can broadly be divided into two classes namely:
  - Adaptive algorithms: These algorithms attempt to change the routing dynamically to suit network conditions. In a situation where a certain backbone node becomes overloaded, additional traffic may be routed through different backbone nodes to alleviate the overload and improve network performance.
  - Non adaptive algorithms: This class of routing algorithm is set up once and all traffic between specific nodes gets routed via a known path that is computed beforehand.
- Survivability: It is necessary to ensure network operation if one or more of the backbone nodes fail. This is necessary to ensure reliability of the network designed and allow the telecommunication network to offer high levels of service to its customers. According to Carlier *et al* (1997:65), the process of recovering from a failure is known as restoration or rerouting. Carlier *et al.* (1997:64) also identified three different types of failures, namely node failure, for instance when a router is no longer operational and then the second and third types which are closely related, namely link and cable failures. A cable may contain more than one link which can fail. When a cable fails, all the links on the cable become inoperational.

These are just a few optimisation type problems, that are found in the backbone network. Many of these problems have high complexity. A lot of research effort has been spent on trying to solve these problems more efficiently.

5.4 Other telecommunication applications

In Sections 5.2 and 5.3 various application of optimisation in the local access and backbone networks were listed and briefly discussed. In this section a few other issues are discussed that are important but do not necessarily fit into either of the previous sections.

5.4.1 Pricing

An important aspect of the offering of services is the price the telecommunication company will ask for the service rendered. Wang (2006:545) presents some of the issues with pricing of services. Initially telecommunication services were largely controlled by monopolies and pricing was based on certain policies. With deregulation came the requirement to be competitive in the market, while still generating enough capital to expand the network and introduce new services. A specific example of this is local loop unbundling, where incumbent operators are forced to lease their hardware in the local access network to competitors. The goal is to facilitate cheaper network access to customers. Bouerreau and Dogan (2005) presented work on the pricing local loop unbundling and present the goals of unbundling. Setting the prices badly can have a large impact on the overall telecommunication networks. If for example the price for the leased facilities are set too low, the incumbents will not be able to fund expansions. Valletti (1999) presents a discussion on the practice of pricing in the United Kingdom, discussing the various phases operators went through.

5.4.2 Steiner tree problems

According to Voss (2006:459) connecting a set of points at minimum cost is one of the most important problems in telecommunications. This is done in the LATN and backbone networks in various forms. This type of problem has been attributed to a Swiss mathematician, by the name of Steiner, and subsequently named after him. Various classes of Steiner tree models have been identified and a lot of research has been done on modeling and solving this problem.

This type problem can be used to model various problems in the telecommunications planning field. Gouveia and Pires (2001) for instance reports work on ring networks, that are sometimes used in the backbone network.

5.5 Conclusion

Optimisation techniques are used extensively and successfully in the design and operation of telecommunication networks. A lot of research has been focussed on improving models for telecommunications and on the solution processes for these models. This chapter presents a brief introduction to various areas in which optimisation is used in telecommunication network planning. For a more complete introduction refer to Resende and Pardalos (2006).

The next chapter contains specific models investigated in this study, for parts of the LATN design problem.

Chapter 6 Models investigated

Before solving a problem mathematically, a symbolic representation of the problem, referred to as a mathematical model, is required, as discussed previously. For a specific problem, it may be possible to create more than one representation. The choice of representation used may depend on the underlying assumptions made for the problem, the solution method to be used etc. This chapter deals with models that can be used to model LATN design problems.

6.1 Motivation for study of LATN design problems

It is worthwhile to discuss the necessity to efficiently model the LATN design problem and to be able to obtain a good solution to the problem in a reasonable time frame. According to Balakrishnan *et al.* (1991:239) telecommunication network planning have been a very active field for developing optimisation models. This is mainly due to two factors:

- Enormous investments in terms of time and money are made in telecommunications facilities;
- rapid technological and regulatory changes take place, providing various design alternatives and operating environments.

It is also claimed that the cost of the LATN accounts for approximately 60% of the total cost of the telecommunication network. For a long time copper has been the dominant delivery medium used in the LATN. According to Tanenbaum (2003:120), at some point up to 80% of AT&T's capital value was in the copper used for the LATN. Such a significant investment necessitates thorough planning.

Various models for the LATN are proposed in the literature. These models vary in complexity, underlying assumptions and tractability. Balakrishnan *et al.* (1991) proposed different models for the LATN and offered a good discussion on the various assumptions. Other formulations are presented by Costamagna *et al.* (1998) and Gavish (1991). As the assumptions of the network to be modeled and the telecommunication hardware differ, so do the various models used to represent them. A more general view of the LATN is taken in this study and hence an abstraction of the hardware is made. This means that the model is hardware independent on the network level. The Tree Knapsack Problem (TKP) and the Extended Tree Knapsack Problem (ETKP) will be used as parts of models for the LATN. The models will be discussed detail in subsequent sections. Solution methods to these problems found in the literature will be covered after the models have been presented.

6.2 Tree Knapsack Problem

As a very basic model for a part of the LATN design the so-called Tree Knapsack Problem (TKP) was considered. This model has been described by Shaw and Cho (1996) and also by Cho and Shaw (1997). This problem is alternatively referred to as a capacitated sub-tree of a tree problem, which has been described by Cho *et al.* (1997).

The Tree Knapsack Problem (TKP) may be seen as choosing a sub-tree of a tree. The goal is to choose the sub-tree that would realise the most profit while not violating a certain capacity constraint. For each node selected in the sub-tree a certain cost is incurred, while at the same time a profit is made from the inclusion of the node. According to Shaw and Cho (1996:206) the TKP is a NP-hard problem.

This problem corresponds to the delivery of service in a network. The nodes belonging to the sub-tree will receive the service, while the nodes not chosen will represent customer sites which will not receive the service. In the following paragraph notation and concepts will be introduced which are required to describe this problem mathematically.

Suppose we are given is an undirected tree $T=(V,E)$ with n nodes, rooted at node 0, where $V=\{0,1,2,\ldots,n-1\}$ is the set of nodes, labelled in a breadth or depth first manner, and E is the set of edges. Assume that the demand of a node is fully satisfied, or not at all. This is referred to as an *indivisible demand assumption.* For a node to be selected in the sub-tree, the parent node of the specific node must be included. This implies that all nodes on the path between a node and the root node must be included if a node is included in a sub-tree. This is known as the *contiguity assumption.*

The following notation is needed for the TKP model formulation:

d_j = The demand or capacity used by including node j in the sub-tree,

c_j = The profit gained by including node j in the sub-tree,

p_j = The predecessor or parent node of node j,

H = The capacity of the TKP.

Assume that d_j, c_j and H are positive integers. The TKP can be modeled mathematically as follows:

$$\max \sum_{j=0}^{n-1} c_j x_j \tag{6.1}$$

$$s.t. x_{p_j} \geq x_j, j = 1,2,3,\ldots,n-1 \tag{6.2}$$

$$\sum_{j=0}^{n-1} d_j x_j \leq H \tag{6.3}$$

$$x_j \in \{0,1\} \quad j = 0,1,2,\ldots,n-1. \tag{6.4}$$

If a node j is included in the sub-tree it has value $x_j = 1$ and if it is not included it has the value $x_j = 0$. Graphically, a sample TKP problem can be seen as in figure 6-1 containing 27 nodes, labeled in breadth first manner.

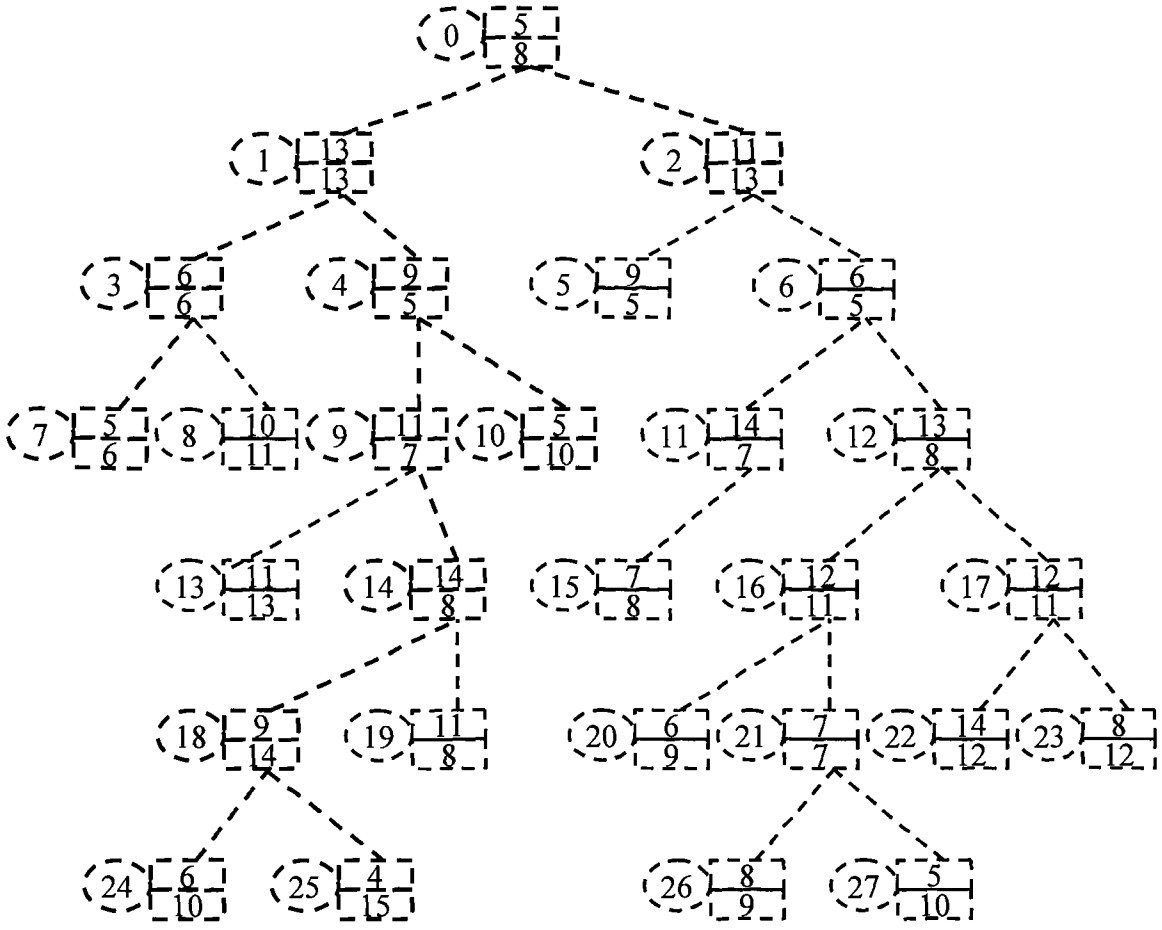

Figure 6-1 Sample TKP data instance

In figure 6-1 the node number is indicated in the oval shape. The top rectangle contains the profit for adding a node in a sub tree, while the lower rectangle contains the demand used up for including the node in a sub tree selected. This tree contains sample data and will later be

used as an illustrative example. In the figure the relationship between parent and child nodes can be seen. The dashed lines indicate that no sub-tree is selected. In subsequent paragraphs, a sub-tree selected will be indicated by solid lines.

6.3 Extended Tree Knapsack Problem

The Extended Tree Knapsack problem (ETKP) can be seen as a natural extension to the TKP. Similarly to the TKP the aim is to select a sub-tree from a tree that maximises the profit while not violating a specified capacity constraint. Similarly to the TKP problem, a profit is gained by including a node in the sub-tree while at the same time this node consumes some of the capacity available. This quantity of the total capacity used up by a node is called the demand of the node. The extension to the TKP is that in the ETKP flow is produced by nodes that are included in the sub-tree. These flows generated by the nodes may incur a further cost. The objective function value can be calculated as the sum of the profits gained by including nodes in the sub-trees minus the cost incurred by the flow generated. This problem is NP-complete according to Shaw *et al.* (1997:30).

A mathematical formulation for this problem is presented in Shaw *et al.* 1997. An important observation is that various cost functions may be used to represent the flow cost. For example, if a capacity expansion problem is investigated where certain capacity for flow in the network is available, a piece-wise linear function may be used to approximate the cost function. At first no expansion is needed for a certain link and hence there is no additional expenditure needed. Once the capacity of a link is exceeded, a fixed cost is incurred for upgrading infrastructure and a further variable cost proportional to the capacity expansion needed must be paid. This corresponds to an expansion of service problem.

A mathematical formulation for the ETKP is now presented. This is similar to the formulation presented by Shaw *et al.* (1997). Given an undirected tree, say $T=(V,E)$ rooted at node *0* with $V=\{0,1,2,\ldots,n-1\}$ the set of the nodes and *E,* the corresponding set of edges let y_j is the number of units of flow between node j and its predecessor node p_j. Assume that, similar to the TKP, d_j and c_j are positive integers respectively representing the demand generated by and profit realised by the inclusion of node j. Assume further that a non-negative integer H represents the capacity of the ETKP.

Assume that y_j units of flow is sent from node j to its predecessor node p_j. A function $f_j(y_j)$ is introduced to represent the flow cost generated by sending y_j units of flow from node j to node p_j. In the model presented by Shaw *et al.* (1997) each link has a certain capacity, say b_j, meaning that b_j units of flow can be sent from node j to its predecessor node. Before more than b_j units of flow can be sent on a link, certain expansion costs need to be paid as described earlier. Assume that when expansion is necessary, a fixed non-negative integer cost of F_j is paid along with variable cost of a_j times the units of flow over and above the level b_j. Mathematically the cable expansion cost $f_j(y_j)$ is defined as follows:

$$f_j(y_j) = \begin{cases} 0 & \text{if } y_j \le b_j, \\ F_j + a_j(y_j - b_j) & \text{otherwise.} \end{cases} \tag{6.5}$$

A graphical representation of the cost function is presented in figure 6-2 below. In this case the ETKP can be seen as a capacity expansion problem. On each link an existing capacity b_j is present. Flow below b_j does not produce additional cost, but that flow more than b_j units incur an additional variable cost. Examples of other cost functions are presented by Carpenter and Luss (2006:316). These functions depend on the underlying assumptions of the telecommunications hardware and other design requirements.

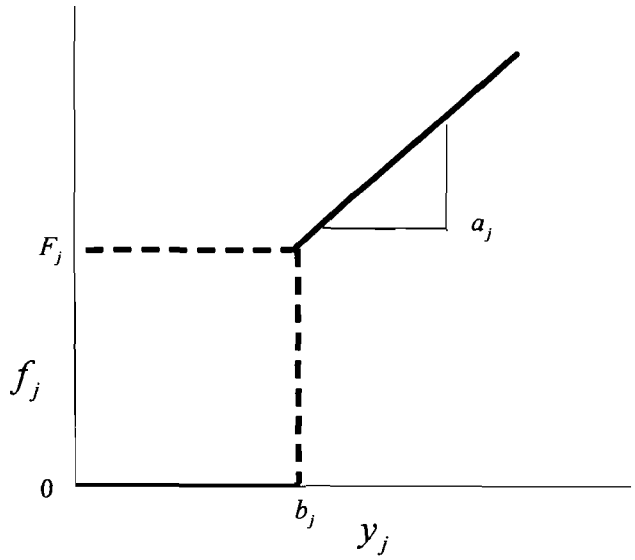

Figure 6-2 Graphical representation of the cost function used for the ETKP

It is assumed that indivisible demand (a node is completely served or not at all) and contiguity constraints (all nodes on the path between a node and the root node must be included for a node to be included in the sub-tree selected) apply to the ETKP.

In order to facilitate the mathematical representation of the ETKP, we define the following notation. Noting that $T=(V,E)$ is an undirected tree where $V=\{0,1,2,\ldots,n-1\}$ is the set of nodes, let $\hat{T}=(V,A)$ be a directed out-tree derived from T. Assume that $A=\{(p_j,j)\,|\,j\in V\}$ is the set of directed edges formed by creating $\hat{T}$. Define B to be a $(n-1)\times(n-1)$ node-arc incidence matrix of $\hat{T}$ with the root node being excluded. By definition, each column in B corresponds to a node and each row corresponds to an arc in the directed tree. This means that the ith column of B has entries of zero, except in row i and row p_i, which has values of 1 and -1 respectively. Let

$$x_j=\begin{cases}1 \text{ if node } j \text{ is selected to be in the sub-tree,}\\ 0 \text{ otherwise.}\end{cases} \tag{6.6}$$

Define a vector y of the flows sent from nodes to their respective parent nodes as $\mathbf{y}=(y_1,y_2,\ldots,y_{n-1})\in\mathbb{R}^{n-1}$ and the matrix D as $D=diag(d_j), j=1,2,3,\ldots,n-1$. Define the vector x as $\mathbf{x}=(x_1,x_2,\ldots,x_{n-1})$. Given a capacity H the ETKP can now be formulated as follows:

$$\max \sum_{j=0}^{n-1} c_j x_j - \sum_{j=1}^{n-1} f_j(y_j) \tag{6.7}$$

$$\text{subject to } x_j \le x_{p_j}, \qquad j=1,2,\ldots,n-1 \tag{6.8}$$

$$D\mathbf{x}-B\mathbf{y}=\mathbf{0}, \tag{6.9}$$

$$\sum_{j=0}^{n-1} d_j x_j \le H, \tag{6.10}$$

$$y\ge 0, \tag{6.11}$$

$$\text{and } x_j\in\{0,1\} \qquad j=0,1,2,\ldots,n-1 \tag{6.12}$$

where $\mathbf{0}$ denotes the n-dimensional zero vector.

The matrices B and D create a set of constraints that ensure flow balance in the ETKP. An illustration of this can be seen by noting that the matrix B has the following structure:

$$B = \begin{array}{c} \\ \\ p_j \\ \\ \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} p_j & & j & & \end{array} \\ \begin{bmatrix} \cdots & 0 & \cdots & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \cdots & -1 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & 1 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & 0 & \cdots \end{bmatrix} \end{array} \tag{6.13}$$

When looking at row p_j the following equation can be constructed from equation set (6.9):

$$y_{p_j} = d_{p_j} x_{p_j} + \sum_{\{i \mid p_i = p_j\}} y_i \tag{6.14}$$

This implies that the flow generated and sent from node p_j to its parent node should be equal to the flow generated in the node itself plus the flow being sent from its child nodes. In this manner flow balance is ensured in the ETKP.

The mathematical programming problem defined in (6.7) to (6.12) can be formulated as an MILP for the case where the functions f_j are in the form (6.5) by introducing new variables as follows:

Firstly the y_j variables are divided into two parts, such that $y_j = y_{j1} + y_{j2}$, where y_{i1} is the part of the flow less than the current capacity ($y_{j1} \le b_j$) and y_{j2} the part of the flow greater than the current capacity of the link.

Secondly, introduce new 0-1 variables, $\delta_j, j = 1,2,3,\ldots,n-1$ and logical constraints of the form $y_{j2} \le H\delta_j$ to force y_{j2} to zero if $\delta_j = 0$.

Define $\mathbf{y}_1 = \left(y_{11}, y_{21}, \ldots, y_{(n-1)1}\right)$ and $\mathbf{y}_2 = \left(y_{12}, y_{22}, \ldots, y_{(n-1)2}\right)$

The ETKP can now be formulated as the following MILP:

$$\max \sum_{j=0}^{n-1} c_j x_j - \sum_{j=1}^{n-1} \left(F_j \delta_j + a_j y_{j2}\right) \tag{6.15}$$

$$\text{s.t. } x_j - x_{p_j} \le 0 \quad j = 1,2,\ldots,n-1 \tag{6.16}$$

$$x_0 = 1 \tag{6.17}$$

$$D\boldsymbol{x} - B\left(\mathbf{y}_1 + \mathbf{y}_2\right) = \mathbf{0}, \tag{6.18}$$

$$\boldsymbol{d}^{\mathrm{T}} \boldsymbol{x} \le H, \tag{6.19}$$

$$y_{i2} \leq H\delta_i, \tag{6.20}$$

$$0 \leq y_{j1} \leq b_j \qquad j = 1,2,\ldots n-1 \tag{6.21}$$

$$y_{j1}, y_{j2} \geq 0 \qquad j = 1,2,\ldots n-1, \tag{6.22}$$

$$x_j \in \{0,1\} \qquad j = 0,1,\ldots n-1, \tag{6.23}$$

$$\delta_j \in \{0,1\} \qquad j = 1,2,3,\ldots,n-1. \tag{6.24}$$

The ETKP can be see as a natural extension to the TKP and captures some of the complexities of the LATN design problem more accurately as it takes into account the flow generated by including nodes in the sub-tree.

6.4 Solution methods for the TKP and ETKP

Generally numerous solution methods are available to solve optimisation problems. This section is devoted to solution methods specifically developed for TKP and ETKP problems. Being formulated as integer and mixed integer linear problems it is possible to solve these problems with general purpose MILP solvers. This is an approach that is most readily available and general purpose optimisation software has improved vastly in the last couple of years. This section, however, covers tailor made algorithms found in literature.

The necessity of finding efficient solution methods for the TKP and ETKP lies in the fact that the problems are NP-complete as previously indicated. When the TKP and ETKP are used in LATN design, it may be necessary to solve several problem instances as part of a larger design algorithm, as it may not be possible to obtain a solution to the complete telecommunication network design problem in one step. A speedup in the solution of the TKP and ETKP problem instances will enhance the solution efficiency of the main design algorithm considerably. Solution methods for the TKP are presented first, followed by solution methods for the ETKP.

6.4.1 The dynamic programming approach of Johnson and Niemi

Johnson and Niemi (1983) worked on the so-called partially ordered knapsack problem and the tree partitioning problem. These can be seen as equivalent names for the TKP. The authors point out that the restriction imposed when using trees opens up the possibility of several different solution methods, in particular pseudo polynomial time algorithms as well as polynomial time approximation schemes. The work done by Johnson and Niemi followed along the lines of work presented by Lukes (1974) and Ibarra and Kim (1978).

An example of the use of the partially ordered knapsack problem presented in this paper can be seen as the modeling of investments. In the tree the vertices represent possible investments. The weights of the nodes are the costs incurred for the investment while the values at the nodes are the expected profits. In this model certain investments can only be made after certain other investments have been made. For more examples see Johnson and Niemi (1983:1). The main focus of their paper was trees, specifically the TKP.

The dynamic programming approach proposed by Johnson and Niemi is a so-called left-to-right dynamic programming approach. The problem is divided into sub-problems that may be easier to solve. The order in which sub-problems are fathomed is from the left to the right. The premise is that if a solution to a sub-problem is found, this solution is also a feasible solution to a larger problem, of which the sub-problem forms part. The so called left to right approach is an alternative technique to the bottom up technique used by other dynamic programming algorithms.

The complexity bound of this algorithm is $O(nC)$, where n is the number of nodes in the problem and C is the optimal objective function value. An interesting remark made by Cho and Shaw (1997:431) is that C is not part of the input data and as such a more realistic time complexity bound would be $O\left(n\sum_{i=0}^{n-1}c_i\right)$ where c_i is the objective function coefficients (profit) of node i with $i=0,1,2,\ldots,n-1$.

The tree needs to be labelled in a depth first manner in order to use the left-to-right approach as can be seen in figure 6-3.

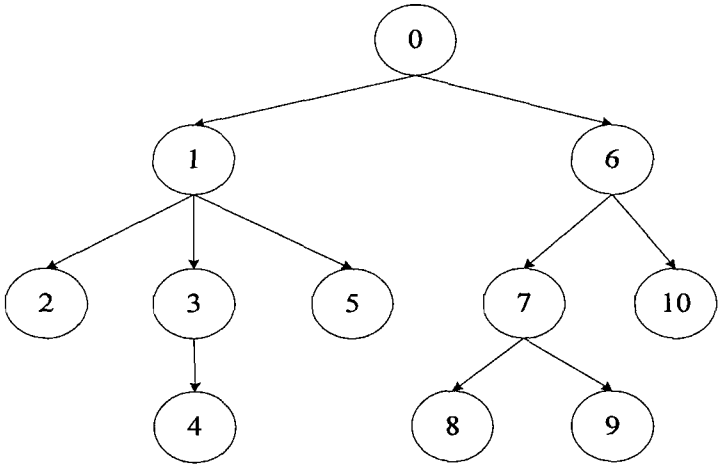

Figure 6-3 Example of depth-first labelled TKP

This dynamic programming approach can be seen as combining sub-trees to obtain the solution to the main problem. This can be done as the same constraints enforced for sub trees must hold for the main problem. The algorithm chooses sub-trees from left-to-right, for example the sub tree containing the root node 0, would be first sub-tree and moving left, the sub tree containing nodes 0 and 1 can be seen as the next left most sub-tree. For a full description of this algorithm see Johnson and Niemi (1983:3).

A negative aspect of this algorithm is that it ideally only gives an optimal objective function value and the optimal solution sub-tree still needs to be constructed once the algorithm terminates. This involves careful bookkeeping schemes. Thus, this algorithm has large storage requirements (see Johnson and Niemi (1983:10)) which may cause difficulties if large problem instances are attempted to be solved.

6.4.2 A Depth-first dynamic programming algorithm of Cho and Shaw for the TKP

Cho and Shaw (1997) published a depth-first dynamic programming algorithm for the TKP. This algorithm was developed as a variant of the algorithm presented by Johnson and Niemi (1983). The way in which sub trees are chosen to be fathomed is in a depth-first manner.

The algorithm tries to find a sub tree $T' = (V', E')$ so that the demand over V' does not exceed h for $h = 0,1,2,\ldots,H$ and the sum of the profits of the nodes included in T' is maximised. This is done by applying recursive rules. According to Cho and Shaw (1997:432) the time complexity of their algorithm is $\theta(nH)$ where n is the number of nodes in the tree and H is the capacity of the TKP.

6.4.3 A Branch and bound algorithm of Shaw and Cho for the TKP

An alternative solution strategy for the TKP presented by Shaw and Cho is a branch and bound algorithm (Shaw and Cho (1996)). A similar algorithm was presented by Cho *et al.* (1997). According to the authors, one of the main results of the paper is to present an improved upper bound for the TKP, which is based on a Lagrangian relaxation method.

An important part of the algorithm presented by Shaw and Cho hinges on the identification of the so-called critical item. A definition for the critical item is given by Martello and Toth

(1987:215) for the simple 0-1 knapsack. To define the critical item for a 0-1 knapsack, assume that all the items in the 0-1 knapsack are arranged in decreasing ratio of profit versus weight. Assume that for a knapsack with n items, node j has a non-negative profit c_j and a non-negative demand (or weight) d_j, then if

$$\frac{c_j}{d_j} \geq \frac{c_{j+1}}{d_{j+1}} \qquad j = 0,1,\ldots,n-2.$$

The critical item with an index s is now defined as

$$s = \min\left\{ j \mid \sum_{i=1}^{j} d_i > H \right\}$$

where H is the capacity of the knapsack. Alternatively, the critical item can be obtained in the following way. Iteratively add items to the knapsack, where the node added has the highest profit versus demand (or weight) of all nodes not already added to the knapsack. The node added to the knapsack that causes the capacity constraint to be violated, is the critical item.

The idea of a critical item does not transfer to the TKP without modification. The problem is that there is a precedence ordering amongst the nodes of the TKP, which is enforced through a set of constraints. Simply adding nodes with the best ratios of profit versus demand may not produce a valid sub-tree and will not be a feasible solution for the TKP. Consequently, Shaw and Cho defined a critical item for the TKP in a different way (Shaw and Cho (1996:211)). Their procedure works as follows, start by including the whole TKP and then iteratively delete the node with the poorest ratio of profit divided by demand. The first node deleted that forms a sub tree that does not violate the capacity constraint of the TKP, is defined as the critical item.

In their experience with empirical studies, their branch and bound algorithm outperforms the dynamic programming algorithm developed by them (Shaw and Cho (1996:205)). They also point out that branch and bound algorithms are very competitive and may in some instances be superior to dynamic programming approaches (Shaw and Cho (1996:206)). This stimulated us to the implement the algorithm from the given pseudo code, since this would yield (another) base for comparisons of algorithmic performance.

6.4.4 A Depth-first dynamic programming algorithm by Shaw *et al.*. for the ETKP

The ETKP model can be viewed as a natural extension to the TKP model. The ETKP contains the TKP as special case. If the flow costs are taken to be zero, the ETKP model simplifies to the TKP. This enabled Shaw *et al.* (1997) to extend their depth first dynamic programming algorithm from the TKP to the ETKP.

The ETKP has flow constraints and as such is a more complex problem. The algorithm proposed by Shaw *et al.* for the ETKP is of complexity $O(nH^2)$ and when the cost function is defined as the cable expansion cost as defined in equation (6.6) the complexity of the algorithm is $O(n\delta H)$. In these equations n is the number of nodes in the ETKP and H is the capacity of the ETKP. The value for δ is defined as $\delta = \max\{D, \max_j \{b_j \mid j = 1,2,\ldots,n-1\}\}$ where D is the depth of the tree and b_j is the existing cable capacity of arc j. Note that the ETKP is modeled in a depth first manner for the implementation of this algorithm.

Similar to the algorithm for the TKP, the depth-first algorithm for the ETKP uses recursive rules to build solutions to sub trees and to generate an optimal solution value for the problem. A negative aspect is that once an optimal objective function has been obtained, the optimal solution sub-tree and flow dimensioning is not known. An additional procedure is presented that can be used to obtain the optimal solution, after the optimal objective function value has been determined (Shaw *et al.* 1997:39).

Shaw *et al.* (1997) tested their algorithm on randomly generated test cases and presented some computational results. The trees were generated in the following way. Firstly the number of nodes in the tree was specified as n. Nodes were then added in a breadth-first manner. The number of child nodes for a specific parent was chosen as an integer from the interval $[0, \lfloor \log_2 n \rfloor]$, where $\lfloor x \rfloor$ denotes the floor value of x.

For the computational results the number of nodes ranged between 20 and 300. There were two classes of problem instances; in the one class the capacity H was set to 500 and in the other class it was set to 1 000. For the class where the capacity was set as H = 500, the

demand for each node was chosen as $d_j \in [1,50]$ and for the class where H = 1 000 demands were generated randomly from the interval $d_j \in [1,100]$. The arc capacity for a node j, b_j was generated randomly from the interval $[1,50]$.

Eight problem instances were generated for each pair of n and H, and computational results on these cases were presented. It is interesting to note that the choice of H remained fixed, even though the problem size increased. This leads to questions as the time complexity of the algorithm is dependant on the choice of H and as such the results may be a bit misleading. In later chapters we show results that suggest the deterioration in their algorithm performance when H is increased as n is increased.

6.5 Conclusion

In this chapter a brief introduction is given to the TKP en ETKP models. Some solution methods for these problems, as found in the literature, are presented. These models form part of the bigger telecommunication networks planning process and may need to be solved for several network configurations. This means that efficient solution methods may be required for these models. In the following chapter the solution method developed in this study is presented.

Chapter 7 Proposed Solution Method

In this chapter new solution methods for solving the TKP and ETKP are presented. The main goal of these methods are to use enhanced modeling and partitioning of the search space while using standard off the shelf software to solve the optimisation problems created during the solution process. The motivation for this approach will be discussed in the next paragraph.

7.1 Choice of solution approach

In the previous chapter the TKP and ETKP have been introduced and various solution strategies proposed by other researchers were shown. It was decided to develop a solution strategy that uses standard optimisation software as a building block in this study. The main reasons for not choosing tailor made dynamic programming or heuristic approaches were:

- **Generation of solutions:** Some dynamic programming solutions do not generate intermediary solutions. This may be troublesome if the algorithm fails to solve a problem instance to completion. This may lead to a case where no feasible solutions are obtained and is highly undesirable. Another problem is also that in some cases, even though the algorithm finds the optimal objective function value, the optimal solution may not be available if extensive bookkeeping is not done. This implies that further work often has to be done to obtain the solution corresponding to the objective function value.
- **Code availability:** A major obstacle with most dynamic programming algorithms is that they are only presented in pseudo-code form or by description. Researchers are often reluctant to distribute the implementation of their algorithms in computer programming languages. Consequently, anyone wanting to use the algorithm has to implement the algorithm himself/herself. This raises the issue of computer programming skill. Certain programmers may code the algorithm more efficiently, or more correctly for that matter, than other programmers. This may lead to inaccurate and possibly unreliable comparisons between implementations.
- **Data dependency:** The performance of some dynamic programming and heuristic methods rely on certain characteristic of the data to which it is applied. This causes certain problem instances to be solved very quickly while other problem instances are not solved efficiently or solved at all. The characteristics of the data may not be easily obtainable by real world problem solvers, leading to difficulty in using certain solution methods.

- **Quality of solutions:** Many dynamic programming and heuristic methods lack a guarantee on the quality of the solution produced in solutions where optimality can not be proven. In contrast the schemes based on typical branch and bound implementations usually produce an integrality gap that can be used to evaluate solutions obtained.

The idea of the proposed solution method is to enhance the usability of standard software through the use of advanced modeling and partitioning of the search space. For this approach, various levels of partitioning are proposed. These partitioning ideas and enhanced modeling will be discussed in the following paragraphs. It is important to point out that the solution methods proposed are exact methods, unless stated otherwise. It is an implicit goal of the research to produce exact solution methods.

7.2 First order partitioning

Advances made in the solution of the ordinary zero-one knapsack problem suggest the use of cardinality constraints in optimisation problems. This approach generally tries to exploit knowledge of the cardinality of a candidate solution to find feasible solutions more quickly. This approach has been proposed by Martello and Toth (1997). The idea of cardinality is to estimate the number of variables that will be included in an optimal solution of a problem at the upper bound. In this thesis, the use of cardinality is investigated to try to limit the growth of the branch and bound tree that is a result of the search process. In most large problems, the growth in the search tree of the branch and bound algorithm is a basic problem, as it may exceed the primary computer memory capacity in the later stages of the search.

In order to use the concept of cardinality it is necessary to estimate cardinality (or the sum of the variables included at value one) of a problem instance. Applying the concept to the TKP, we could for example, try to pinpoint the cardinality of the optimal sub tree by solving the Linear Programming (LP) relaxation of the TKP. The LP relaxation is obtained by relaxing the integrality constraints $x_i \in \{0,1\}$ to $0 \leq x_i \leq 1$. We use the notation ILPR(TKP) to denote the LP relaxation of ILP(TKP). Denote the Integer Linear Programming (ILP) equivalent of ILP(TKP) with cardinality constraint p as ILP(TKP,p) and use ILPR(TKP,p) to denote the LP relaxation for ILP(TKP,p). Constraining a TKP instance to a cardinality p implies that the sub-tree will contain exactly p nodes in its solution. This strategy will be considered as a first order partitioning scheme. Note that for the relaxation, the sum of the variables will be

constrained to p. Adding a cardinality constraint forces the solution of the problem instance to have an integer valued sum of the variables, which is a valid constraint, as it is not possible to have a solution that has a non-integer sum.

An exact first order partition is described in pseudo code as:

Procedure FirstOrderPart

begin

CLB = 0

Define $P = \{1,2,3,\ldots,n-1\}$

Solve parametrically ILPR(TKP,p) for $p \in P$ to obtain associated objective function values of ILPR(TKP,p) denoted by $Z_{TKPR(p)}$. If ILPR(TKP,p) is infeasible, set $Z_{TKPR(p)} = -\infty$.

while $P \neq \varnothing$ **do**

begin

Set $p' = t$ where t is defined by $Z_{TKPR(t)} = \max_k \{Z_{TKPR(k)} \mid k \in P\}$

Solve ILP(TKP, p') to obtain $Z_{TKPR(p')}$

Set $P = P \setminus \{p'\}$

if $Z_{\text{TKPR}(p')} > \text{CLB}$ **then**

begin

$CLB = Z_{\text{TKPR}(p')}$

Set $P = \{i \mid i \in P \text{ and } z_{TKPR(i)} > \text{CLB}\}$

end. // **end if**

end. // **end while** $P \neq \varnothing$

end. // **end** Procedure FirstOrderPart

An integer variable CLB is used to store the objective function value of the current best integer solution. The set P is searched and updated as solutions become available. Note that in the pseudo code the string // will indicate comments to improve readability.

7.3 Second order partitioning

During the empirical study it was found that the solutions generated by the first order partition produced solutions with a large number of variables included with a value of 1. This is similar to results obtained for the 0-1 Knapsack. To exploit this fact, a further level of

partitioning of the search space was introduced. This will be referred to as the second order partition, which will be discussed below.

Assume that an ILP(TKP,p) is a first order partitioning problem. In the solution of a specific ILPR(TKP,p), do the following:

- Count the number of nodes (or variables) that have been included with value 1 ($x_j = 1$).
- Assume that they are l in number. Define a set S_l to contain indices of the variables included with value 1 ($x_j = 1$) by ILPR(TKP,p) and form a set S_{n-l} containing the indices of the remaining nodes of the problem.

The aim of the second order partition is to investigate the exchange of nodes from the set S_l for nodes in the set S_{n-l} iteratively. The premise is that the LP relaxation may give a good indication of an environment where a "good" solution may be obtained, but it may not identify the optimal set in its entirety. If all exchanges are considered, the optimal solution for ILP(TKP,p) will be found. A mathematical representation of the second order partition is now presented.

- Assume that a TKP problem instance is constrained to a cardinality p.
- Solve ILPR(TKP,p) to produce a solution with x_j^* for $j = 0,1,2,\ldots,n-1$. If an infeasible solution is obtained, no second order partitions can be investigated.
- Count the number of nodes included in the optimal solution with value 1 ($x_j^* = 1$). Suppose that they are l in number.
- Recalling that $V = \{0,1,2,\ldots,n-1\}$ define $S_l = \{ j \mid x_j^* = 1,\ j = 0,1,2,\ldots,n-1 \}$ and $S_{n-l} = V \setminus S_l$.
- Choose $q \in \{l, l-1, l-2, \ldots, \max\{0, p-|S_{n-l}|\}\}$ [1]
- Define the second order partitioned model ILP(TKP,p,q) as follows:

$$\max \sum_{j=0}^{n-1} c_j x_j \tag{7.1}$$

$$\text{s.t. } x_{p_j} \geq x_j \quad j = 1,2,3,\ldots,n-1 \tag{7.2}$$

---

[1] The value $\max\{0, p-|S_{n-l}|\}$ is used as it is possible that $p-|S_{n-l}|$ may be negative. The minimum number of variables that can be selected from the set S_l is $p-|S_{n-l}|$

$$\sum_{j=0}^{n-1} d_j x_j \le H, \tag{7.3}$$

$$\sum_{j \in S_l} x_j = q, \tag{7.4}$$

$$\sum_{j \in S_{n-l}} x_j = p - q, \tag{7.5}$$

$$x_j \in \{0,1\}, \quad j = 0,1,2,\ldots,n-1 \tag{7.6}$$

Define ILPR(TKP,p,q) as the linear programming relaxation to ILP(TKP,p,q).

By solving ILP(TKP,p,q) with $q \in \{l, l-1, l-2, \ldots, \max\{0, p-|S_{n-l}|\}\}$ the optimal solution to ILP(TKP,p) will be found if all the values of q are enumerated explicitly or implicitly.

In practice it is not necessary to enumerate all the values of q as bounds are generated that limit the number of choices of q. In paragraph 7.4 below we give a complete algorithm that includes both the first and second order partitioning concepts.

7.4 The partitioning TKP algorithm

In this chapter the partitioning algorithm PART_TKP for the TKP problem is presented. The algorithm combines enhanced modeling and the partitioning schemes discussed above to form an efficient solution process to optimally solve the TKP. This algorithm has been published by Van der Merwe and Hattingh (2006).

Introduce a variable CLB to store the current best objective function value obtained. The current lower bound is continuously updated and gives the optimal solution value when the algorithm terminates.

procedure PART_TKP
begin
CLB = -∞
Set Z_{TKPR} = Solution to ILPR(TKP) and identify the solution values x_j^* for $j = 0,1,2,\ldots,n-1$
Define $P = \{1,2,3,\ldots,n-1\}$
Solve parametrically ILPR(TKP,p) for $p \in P$ to obtain associated objective function values of ILPR(TKP,p) denoted by $Z_{TKPR(p)}$. If ILPR(TKP,p) is infeasible, set $Z_{TKPR(p)} = -\infty$.
while $P \neq \varnothing$ **do**
begin
Set $p' = t$ where t is defined by $Z_{TKPR(t)} = \max_k \{Z_{TKPR(k)} \mid k \in P\}$
For $Z_{\mathrm{TKP}(p')}$ identify the solution values x_j' for $j = 0,1,2,\ldots,n-1$
Define $S_l = \{j \mid x_j' = 1, j = 0,1,2,\ldots,n-1\}$ and set $S_{n-l} = V \setminus S_l$

Set $l = \sum_{j \in S_l} x_j'$

Set $Q = \{l, l-1, l-2, \ldots, \max\{0, p' - |S_{n-l}|\}\}$

Solve parametrically ILPR(TKP,p',q) for $q \in Q$ to obtain associated objective function values of ILPR(TKP,p',q) denoted by $Z_{TKPR(p',q)}$, if ILPR(TKP,p',q) is infeasible set $Z_{TKPR(p',q)} = -\infty$

while $Q \neq \varnothing$ **do**

begin

Set $q' = s$ where s is defined by $Z_{TKPR(p',s)} = \max_j \{Z_{TKPR(p',j)} \mid j \in Q\}$

if $Z_{TKPR(p',q')} > CLB$ **then**

begin

Set $Z_{TKP(p',q')}$ = Solution to ILP(TKP, p',q')

if $Z_{TKP(p',q')} > CLB$ **then**

begin

Set CLB = $Z_{TKP(p',q')}$

Set $Q = \{r \mid r \in Q$ and $Z_{TKPR(p',r)} > CLB\}$

end // **end** if

end // **end** if

Set $Q = Q \setminus \{q'\}$

end. // **end** while

Set $P = P \setminus \{p'\}$

Set $P = \{i \mid i \in P$ and $z_{TKPR(i)} > CLB\}$

end // **(while)**

Optimal solution = CLB

end.

Using the notation as defined above, figure 7-1 contains a flow diagram of the partitioning TKP algorithm represented by **procedure** PART_TKP. It gives an indication of the flow of control in the algorithm.

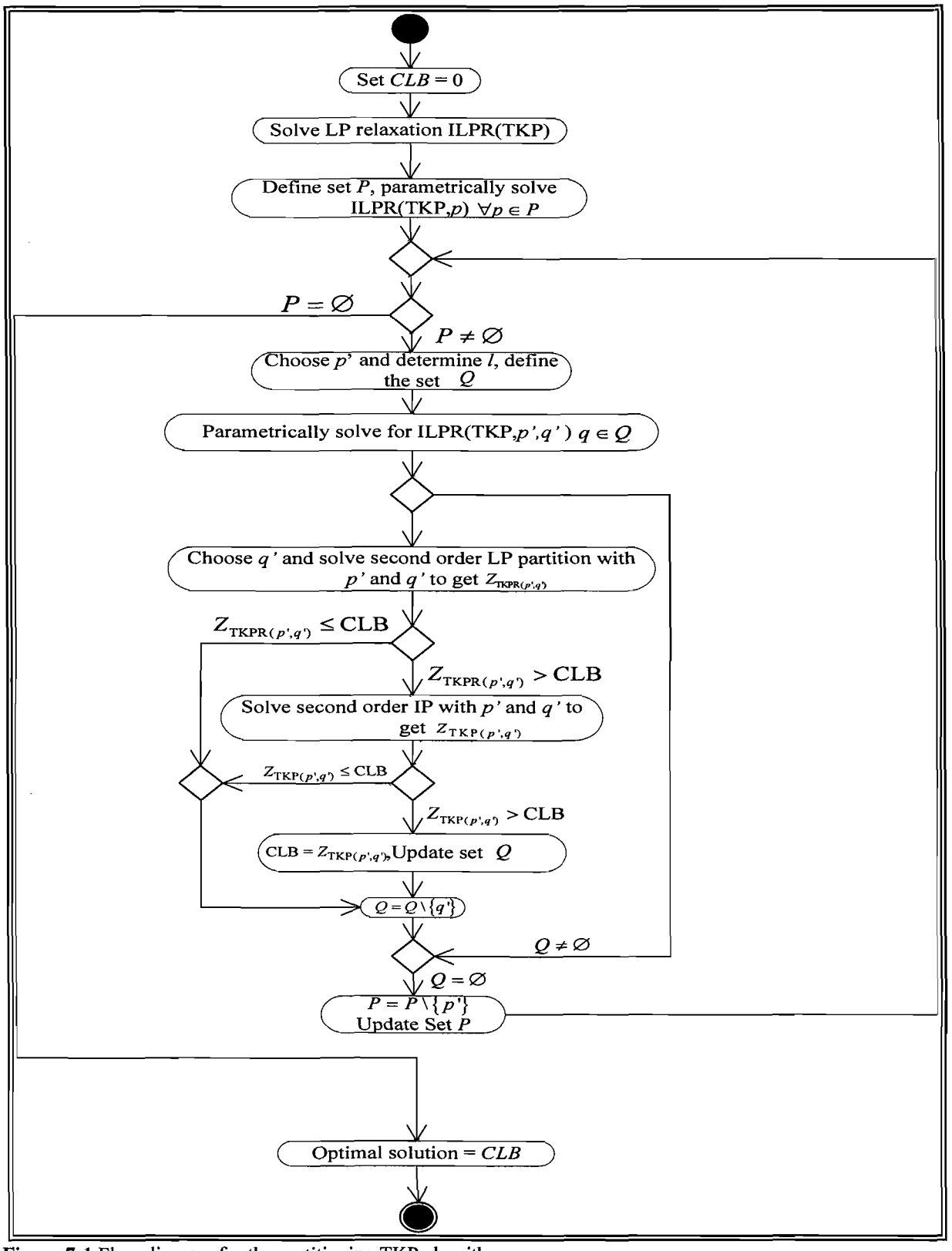

Figure 7-1 Flow diagram for the partitioning TKP algorithm

During the execution of the algorithm, every time the current lower bound is updated, the values of the variables corresponding to the solution is stored. This provides a vector that contains the optimal variable values at the end of the solution process.

7.4.1 Discussion of the partitioning algorithm

The aim of the algorithm is to exploit the information available in the LP relaxation solutions, in order to speed up the solution process. The second order partition uses the information given by the solution of the LP relaxation, ILPR(TKP,p). These values provide an indication of the variables that may be included at their upper bound in the optimal solution. A graphical representation of the group identified for a specific data instance is presented as an example, see figure 7-2. The set S_l is the indices of the nodes connected in the sub tree coloured with solid lines. The nodes connected with the dashed lines determine the set S_{n-l} for a cardinality of $p=18$.

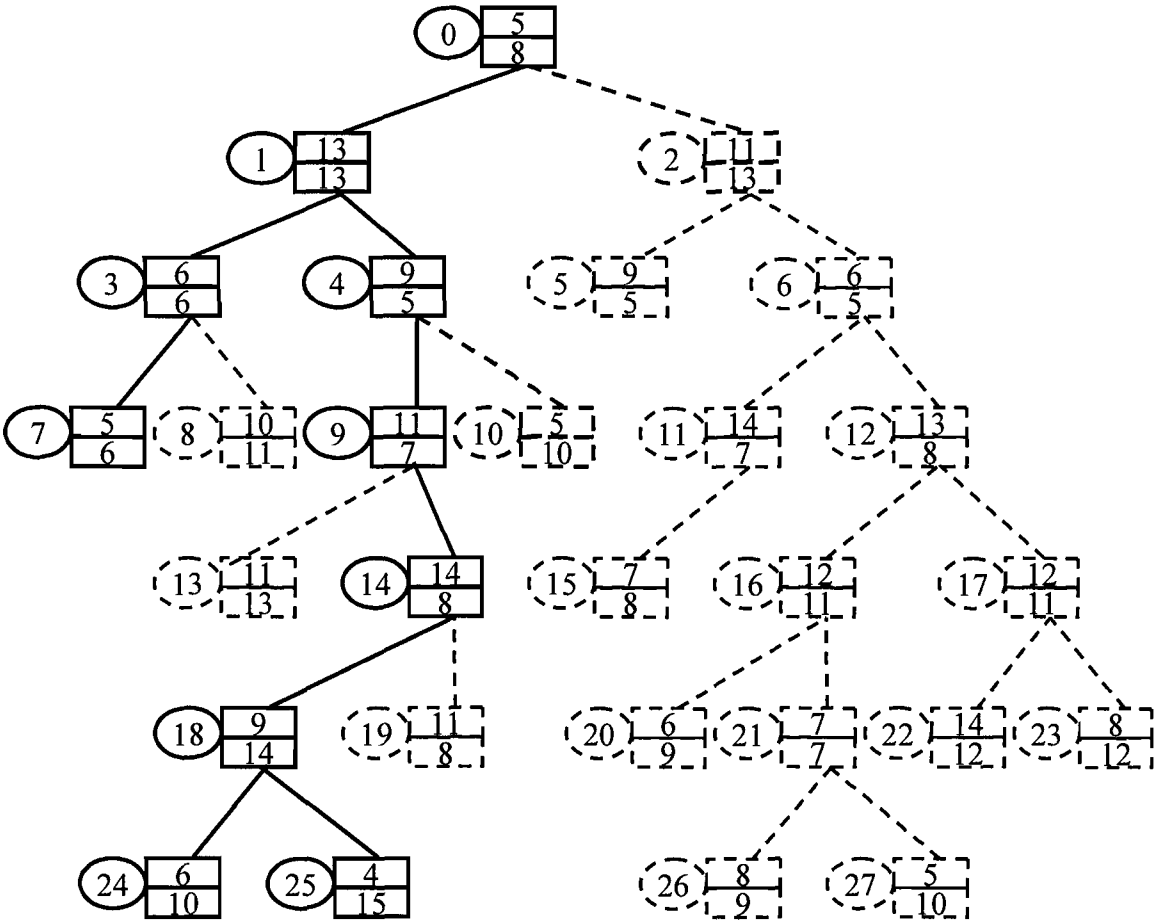

Figure 7-2 Sample tree indicating the sub tree identified by the TKP algorithm

The first order partitioning is done on the cardinality estimate of the problem instance. There may be different methods for estimating the cardinality. The one presented in the algorithm tries to exploit information presented by solving the LP relaxation, ILPR(TKP).

The aim of the second order partition is to exchange nodes from the set S_l for nodes in the set S_{n-l}. By doing this, the algorithm ensures that all possible combinations are fathomed. This implies an implicit enumeration which is facilitated by the use of bounds. The bounds will be discussed in the next section.

Note that the algorithm presented here represents a sequential implementation of the systematically partitioned search space. Parallel implementations could be considered but were not investigated in this thesis.

7.4.2 Bounds used in the partitioning TKP algorithm

The use of bounds is very important for the performance of the partitioning algorithm presented above. Using bounds allows the algorithm to investigate only some partitions and not to search through all available partitions. The fewer first and second order partitions investigated, the shorter the solution time will be required to solve the TKP optimally.

Solving ILPR(TKP) provides an absolute upper bound, Z_{TKPR}, for the problem. Constraining the problem to cardinalities $p \in P$ in the set $P = \{1,2,\ldots,n-1\}$ produces objective function values $Z_{TKPR}(p) \leq Z_{TKPR} \forall p \in P$. In figure 7-2 a graphical representation of the objective function values can be seen for the sample TKP. The graphical representation of the objective function values are concave. The objective function value $Z_{TKPR} = 172.0130$ serves as an upper bound for the whole problem as well as for the problem constrained to a specific cardinality.

The second order partitioning is done once a problem instance has been constrained to a specific cardinality, say p. In this case $Z_{TKPR}(p)$ is an upper bound for all $Z_{TKPR}(p,q)$ and $Z_{TKP}(p,q)$ where $q \in Q = \{l, l-1, l-2, \ldots, \max\{0, p-|S_{n-l}|\}\}$. Also solving ILPR(TKP,$p,q$), produces an upper bound for ILP(TKP,p,q), meaning that $Z_{TKPR}(p,q) \geq Z_{TKP}(p,q)$.

During the solution process the first ILP(TKP,p,q) solved successfully produce a lower bound for the whole problem, ILP(TKP). Every time a different ILP(TKP,p,q) is solved, $Z_{TKP}(p,q)$ is compared to the current lower bound and if it is larger, the current lower bound,

is updated. If a new cardinality p is investigated and $Z_{TKPR}(p)$ is smaller than the current lower bound, it implies that the cardinality cannot produce a better lower bound. Once the algorithm terminates, the current lower bound is the optimal objective function value.

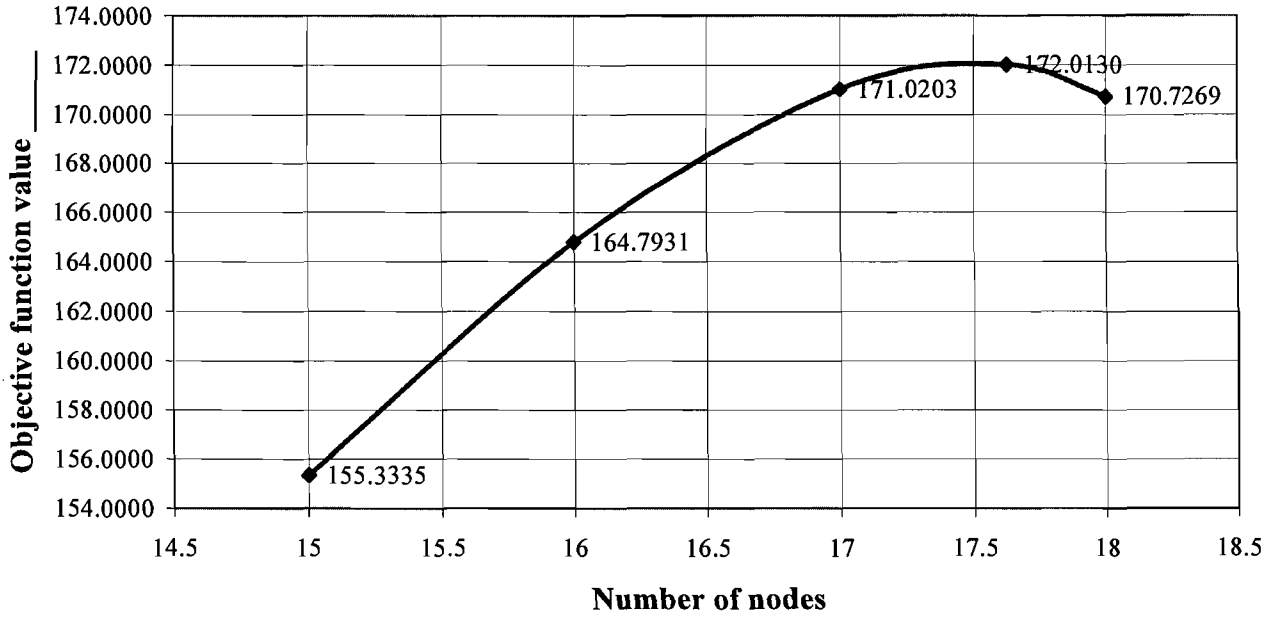

Figure 7-3 Graphical representation of objective function values versus number of nodes in the sample TKP

7.4.3 Enhancements to the solution process

7.4.3.1 Heuristics

Empirical work with certain problem instances resulted in the same objective function values for more than one cardinality to which the problem was constrained ($Z_{TKPR(m)} = Z_{TKPR(n)}, m \neq n$ and $m, n \in P$). A heuristic method was devised with the following goals:

- Differentiate between various first order partitions with the same objective function value for the relaxation. The goal was to generate different values to evaluate, from which a decision could be made which partition to investigate..
- Use heuristics to generate a lower bounds for the partitions, which may also be used for the original TKP problem.

The heuristic aims to exploit the solution provided by the LP relaxation further. The set S_l is the set of node indexes with corresponding variables included with value 1 in the solution of the LP relaxation. Formally, if in the solution of ILPR(TKP,p) produce $x_j, j = 0,1,2,\ldots,n-1$

then $S_l = \{ j \mid x_j' = 1, j = 0,1,2,\ldots,n-1 \}$. The set S_l forms a sub tree which does not violate the capacity, contiguity and indivisible demand constraints of the original problem. However, creating the sub tree uses up a part of the original capacity. The heuristic we investigated creates a 0-1 knapsack of the child nodes directly adjacent to the leaf nodes of the sub tree identified by the set S_l. Define the set K as the set of nodes directly adjacent to the set S_l in the tree T, formally, $K = \{ j \mid p_j \in S_l, j \notin S_l \}$. It is now possible to construct a 0-1 knapsack with a capacity remaining after the nodes in the set S_l have been included. Define the capacity used by creating a sub tree of the nodes in set S_l as H_{S_l}, formally defined as $H_{S_l} = \sum_{j \in S_l} d_j x_j$. Define the profit obtained for creating the sub tree defined by S_l as Z_{S_l}, formally defined as $Z_{S_l} = \sum_{j \in S_l} c_j x_j$.

The heuristic procedure presented below is executed after a ILPR(TKP,p) has been solved, the solution values identified for the ILPR(TKP,p) are x_j' for $j = 0,1,2,\ldots,n-1$. The rationale is that although the LP relaxation gives a good indication of which nodes should be included for a good solution; it could not identify all variables in the solution that should have values of 1. The heuristic aims to add "good" nodes to a valid sub tree created by the LP relaxation. The 0-1 knapsack created has limited capacity and should be relatively easy to solve. The 0-1 knapsack can be solved using standard off the shelf software. This philosophy fits nicely with the goal of the study not to employ custom developed software solvers for optimisation problems. Define the knapsack formulated as TKP_Knap(p) with the notation as defined above as:

$$\max \sum_{j \in K} c_j x_j \tag{7.7}$$

$$\text{s.t.} \sum_{j \in K} d_j x_j \leq H - H_{S_l} \tag{7.8}$$

Assume that a ILPR(TKP,p) has been solved giving solution values x_j' for $j = 0,1,2,\ldots,n-1$ and that the capacity for the TKP problem is H. Note that this formulation is independent of the TKP problem and can be solved separately.

procedure HEUR_TKP(p)
begin

Solve ILPR(TKP,p) to obtain objective function $Z_{\text{TKP}(p')}$ identify the solution values x_j' for $j = 0,1,2,\ldots,n-1$

Identify the set S_l

Identify the set $K = \{j \mid p_j \in S_l, j \notin S_l\}$

Calculate $H_{S_l} = \sum_{j \in S_l} d_j x_j'$

Calculate $Z_{S_l} = \sum_{j \in S_l} x_j' c_j$

Solve TKP_Knap(p) as defined in (7.7) and (7.8) to obtain solution value $Z_{TKP\_Knap(p)}$ with solution values $\hat{x_j}, j \in K$

Compute TKP_Heur(p) = $Z_{TKP\_Knap(p)} + Z_{s_l}$

end

The value TKP_Heur(p) can be used instead of the value $Z_{TKPR(p)}$ to choose which first order partition to investigate. This proved very useful for cases where the value of $Z_{TKPR(p)}$ is the same for different values of p. It differentiates between partitions. During the partitioning algorithm, when a decision was needed regarding the p to choose when $Z_{TKPR(m)} = Z_{TKPR(n)}, m \neq n$ and $m, n \in P$, the heuristic may be used to choose a partition. Another use of the value TKP_Heur(p) is that it can be used to update the current lower bound in the partitioning algorithm, if it is found that TKP_Heur(p) $> CLB$. If the current lower bound is improved, the set S_l and the values $\hat{x_j}, j \in K$ are used to update the vector that will contain the optimal value at the end of the solution process. A cardinality constraint can also be imposed on the Knapsack problem, as the problem ILPR(TKP,p) is already constrained to a certain cardinality. This was not done in the empirical work. It is also possible to implement a similar heuristic on the LP relaxation ILPR(TKP).

During the empirical testing of the algorithm, it was found that in many instances this heuristic produced very good lower bounds, in some cases it even produced the exact optimal solution to the TKP instance. This is another indication that the LP relaxation solution gives a very good indication of what the optimal solution may be. The heuristic may also be applied after the first LP relaxation (ILPR(TKP)) has been solved to directly produce a lower bound for the problem. Another use may be to apply the heuristic each time before an ILP problem (i.e. ILP(TKP,p,q)) is solved. Generally the heuristic improved solution times noticeably.

7.4.3.2 Tweaking the algorithm

Small enhancements to the algorithm also lead to improvement in solution times. One of these enhancements is to note that the profit of the individual nodes is assumed to be a non-negative integer. This implies that the optimal objective function value will also be a non-

negative integer. Looking at the algorithm, it is evident that the LP relaxation ILPR(TKP,p',q') may produce a non-integer optimal value $Z_{\text{TKPR}(p',q')}$. It can be further noted that in order to produce a better current lower bound, it may be tested with $Z_{\text{TKPR}(p',q')} > \text{CLB}+1-\varepsilon$ instead of the usual test $Z_{\text{TKPR}(p',q')} > \text{CLB}$. In this case $0 < \varepsilon < 1, \varepsilon \in \mathbb{R}$, implies that if ε is close to zero, only ILP(TKP,p',q') will be investigated where it may produce $Z_{\text{TKP}(p',q')} \geq \text{CLB}+1$. This is an improved bound that can also be used in the algorithm to consider the residual set P that has to be investigated. The operation to update the set $P = \{i \mid i \in P \text{ and } z_{TKPR(i)} > \text{CLB}\}$, may be rewritten as $P = \{i \mid i \in P \text{ and } z_{TKPR(i)} > \text{CLB}+1-\varepsilon\}$.

7.5 The partitioning ETKP algorithm

The partitioning algorithm developed for the TKP problem can be modified to be applied to the ETKP. The resulting algorithm will be referred to as the partitioning ETKP algorithm and was first presented by Van der Merwe and Hattingh (2003). In the following sections the notation as defined for the TKP will be adapted for use with the partitioning ETKP algorithm. Where ILPR(TKP) was used to indicate an LP relaxation problem for the TKP, ILPR(ETKP) will be used to indicate an LP relaxation for the ETKP. In the same way the other notation can be adapted to reflect the ETKP case, for example Z_{ETKPR} can be defined analogously to Z_{TKPR}.

The partitioning algorithm developed for the TKP ported very well to the ETKP. Certain attributes of the ETKP caused the algorithm to perform less well for the ETKP than it did for the TKP. This necessitated enhancements to the algorithm in order to obtain satisfactory performance by the algorithm.

One of the main problems experienced in experiments with the partitioning ETKP algorithm is an increased integrality gap, i.e. the gap between the LP relaxation and the ILP solution values. A large integrality gap generally gives an indication that a problem instance is more difficult than one with a smaller gap. This leads to a proliferation of first and second order partitions to investigate. An additional problem is that the bounds created in the algorithm were not as tight as those for the TKP. Some of the optimisation opportunities used in the

TKP also did not hold. For example, the cost function $f(y_j) \in \mathbb{R}$ will not necessarily produce an integer value so that the value Z_{ETKP} is not necessarily integer valued. This means that optimisation opportunities like $Z_{\text{TKPR}(p',q')} > \text{CLB} + 1 - \varepsilon$ is no longer valid. This means that the bounds used to prune first and second order partition are no longer as tight and more partitions need to be explored.

7.5.1 Partitioning ETKP algorithm

The partitioning ETKP algorithm is now formally presented. Define CLB as the best solution found so far. Let the vector $\mathbf{x}^{CLB} = \{x_j^{CLB}, j = 0,1,2,\ldots,n-1\}$ be the feasible solution corresponding to the CLB.

procedure PART_ETKP
begin
Set CLB = -∞
Set Z_{ETKPR} = Solution to ILPR(ETKP) and denote the solution values by x_j^* for $j = 0,1,2,\ldots,n-1$
Define $P = \{1,2,3,\ldots,n-1\}$
Solve parametrically ILPR(ETKP,*p*) for $p \in P$ to obtain associated objective function values $Z_{ETKPR(p)}$. If ILPR(ETKP,*p*) is infeasible, set $Z_{ETKPR(p)} = -\infty$.
while $P \neq \varnothing$ **do**
begin
Set $p' = t$ where t is defined by $Z_{ETKPR(t)} = \max_k \{Z_{ETKPR(k)} \mid k \in P\}$
For $Z_{\text{ETKP}(p')}$ identify the solution values x_j' for $j = 0,1,2,\ldots,n-1$
Define $S_l = \{j \mid x_j' = 1, j = 0,1,2,\ldots,n-1\}$ and set $S_{n-l} = V \setminus S_l$
Set $l = \sum_{j \in S_l} x_j'$
Set $Q = \{l, l-1, l-2, \ldots, \max\{0, p - |S_{n-l}|\}\}$
Solve parametrically ILPR(ETKP,p',q) for $q \in Q$ to obtain associated objective function values of ILPR(ETKP,p',q) denoted by $Z_{ETKPR(p',q)}$, if ILPR(ETKP,p',q) is infeasible set $Z_{ETKPR(p',q)} = -\infty$
while $Q \neq \varnothing$ **do**
begin
Set $q' = s$ where s is defined by $Z_{ETKPR(p',s)} = \max_j \{Z_{ETKPR(p',j)} \mid j \in Q\}$
if $Z_{\text{ETKPR}(p',q')} > \text{CLB}$ **then**
begin

Solve ILP(ETKP, p', q') with corresponding solution values $\hat{x_j}, j = 0,1,2,\ldots,n-1$ and variables $\hat{\delta_i}, i = 1,2,\ldots,n-1$

Set $Z_{\text{ETKP}(p',q')}$ = Solution to ILP(ETKP, p', q')

if $Z_{\text{ETKP}(p',q')} > \text{CLB}$ **then**

begin

Set CLB = $Z_{\text{ETKP}(p',q')}$

Update $x_j^{CLB} = \hat{x_j}, j = 0,1,2,\ldots,n-1$

Set $Q = \{r \mid r \in Q \text{ and } Z_{\text{ETKPR}(p',r)} > \text{CLB}\}$

end

end

Set $Q = Q \setminus \{q'\}$

end.

Set $P = P \setminus \{p'\}$

Set $P = \{i \mid i \in P \text{ and } z_{ETKPR(i)} > \text{CLB}\}$

end (while)

Optimal solution = CLB

end.

In the following section optimisation opportunities will be investigated that were implemented to enhance the efficiency of the partitioning ETKP algorithm.

7.5.2 Third order partitioning

One approach investigated to improve the efficiency of the partitioning ETKP algorithm was to introduce further partitions in the algorithm. Analysing the formulation of the ETKP in conjunction with the definition of the algorithm, a possible third order partition was developed. A third order partition can be done on the δ-variables. It was noted in the experimental results that the optimal solution of the ETKP often contained a certain number of δ-variables. This means that expansion costs are incurred for a certain number of nodes. A third order partition can be done after a second order has been solved, i.e. ILPR(ETKP,p', q') and corresponding variables $\hat{x_j}, j = 0,1,2,\ldots,n-1$ and $\hat{\delta_i}, i = 1,2,\ldots,n-1$ obtained. It is possible to estimate the cardinality of the δ-variables in the optimal solution as well. For the first order partition, the cardinality was estimated by calculating the sum of the variables in the solution of ILPR(ETKP). As a first attempt this was done for the δ-variables. This proved unsatisfactory, as the sum of $\hat{\delta}$-variables included in the solution of ILPR(ETKP,p', q') was generally very small. This is one of the contributing factors resulting in an increased integrality gap. This approach gave a very poor indication of the cardinality of the δ-variables.

A different approach to estimate the cardinality of the δ-variables is to count the number of $\hat{\delta}$-variables with non-zero values included by ILPR(ETKP,p',q'). Formally define the set $\hat{D} = \{\hat{\delta_i} \mid \hat{\delta_i} > 0, i = 1,2,3,\ldots,n-1\}$. Mathematically an estimate for the cardinality of the $\hat{\delta}$-variables can be calculated as $\sum_{i \in \hat{D}} 1$.

To formalise a possible third order partition on the $\hat{\delta}$-variables, define the set $D = \{1,2,3,\ldots,n-1\}$ and the problem ILPR(ETKP,p',q',d') as the LP relaxation third order partition for the problem ILPR(ETKP, p',q') where a constraint is added that ensures that d' δ-variables are included in the solution. Formally a third order partition ILP(ETKP,p',q',d') is defined as:

$$\max \sum_{j=0}^{n-1} c_j x_j - \sum_{j=0}^{n-1} f_j(y_j) \tag{7.9}$$

$$\text{s.t. } x_{p_j} \geq x_j \quad j = 1,2,3,\ldots,n-1 \tag{7.10}$$

$$\sum_{j=0}^{n-1} d_j x_j \leq H, \tag{7.11}$$

$$\sum_{j \in S_l} x_j = q, \tag{7.12}$$

$$\sum_{j \in S_{n-1}} x_j = p - q, \tag{7.13}$$

$$\sum_{j=1}^{n-1} \delta_j = d' \tag{7.14}$$

$$x_j \in \{0,1\}, \quad j = 0,1,2,\ldots,n-1 \tag{7.15}$$

$$\delta_j \in \{0,1\}, j = 1,2,3,\ldots,n-1 \tag{7.16}$$

The corresponding LP relaxation where x_j is relaxed to $0 \leq x_j \leq 1$ and δ_j is relaxed to $0 \leq \delta_j \leq 1$ is denoted by ILPR(ETKP,p',q',d').

For a given ILPR(ETKP,p',q') with solution values $\hat{x_j}, j = 0,1,2,\ldots,n-1$ and $\hat{\delta_i}, i = 1,2,\ldots,n-1$, the third order partition is described in the procedure PART3_ETKP given below:

procedure PART3_ETKP
begin

Define the set $D = \{0,1,2,\ldots,n-1\}$

Parametrically solve ILPR(ETKP,p',q',d) to obtain solution values $Z_{ILPR(ETKP,p',q',d)}$ where $d \in D$, if ILPR(ETKP,p',q',d) is infeasible, set $Z_{ILPR(ETKP,p',q',d)} = -\infty$

while $D \neq \varnothing$ **do**

begin

Set $d' = s$ where s is defined by $Z_{ETKPR(p',q',s)} = \max_j \{Z_{ETKPR(p',q',j)} \mid j \in D\}$

Solve ILP(ETKP,p',q',d') to obtain optimal solution value $Z_{ILP(ETKP,p',q',d')}$ with corresponding solution values $x_j, j = 0,1,2,\ldots,n-1$

Set $D = D \setminus \{d'\}$

if ($Z_{ILP(ETKP,p',q',d')} > CLB$) **then**

begin

Set $CLB = Z_{ILP(ETKP,p',q',d')}$

Update $x_j^{CLB} = x_j, j = 0,1,2,\ldots,n-1$

Set $D = \{r \mid r \in D \text{ and } Z_{ETKPR(p',q',r)} > \text{CLB}\}$

end (if)

end (while)

end (procedure PART3_ETKP)

The rest of the partitioning ETKP algorithm continues more or less as described previously. The only difference is that ILP(ETKP,p',q') is no longer solved. This is done implicitly in the third order partition. At the end of the algorithm, the CLB is the exact optimal solution.

7.5.3 Different implementation for a third order partition

A different approach used for a third order partition is to sort the $x_j', j = 0,1,2,\ldots,n-1$ into a set $X_{sort} = \left(x_0^s, x_1^s, x_2^s, \ldots, x_n^s\right)$ obtained from ILPR(ETKP,p) in descending order, i.e. $x_i^s \geq x_j^s \forall i < j$ while maintaining a mapping to the original variables. The third order partition still has the set S_l, but first tries to add the variables that have the largest values, not included in set S_l. This is another way to implement a third order partition.

7.5.4 Valid inequalities

In an effort to reduce the integrality gap and improve the effectiveness of the algorithm, valid inequalities can be added to the formulation. These valid inequalities can be added to the base formulation of the ETKP model, as they have global validity. Other inequalities may be devised that are only valid for certain branches in the search space. This has not been done in this study. The general goal was to decrease the integrality gap and consequently to limit the

number of first and second order partitions to be solved. This is achieved through the use of tighter bounds. Implementing the valid inequalities generally improved the solution times of the algorithms tremendously in the experimental work. The following sections will discuss the valid inequalities, some of which where first presented in Van der Merwe and Hattingh (2005).

The first inequality tries to exploit the fact that capacity is used up on a path between a node and the gateway or root node. This, in conjunction with the contiguity constraints, forces expansion higher up in the network if a node is added on a lower level.

procedure add_cut1
begin
for $i = 1$ to $n-1$ do
begin
    $j = i$
    sub_cap = 0.0
    do
        sub_cap = sub_cap + d_j
        if sub_cap > b_j
        then add_constraint $x_i \leq \delta_j$
    $j = p_j$
    while $j \neq$ root_node
end
End.

The first set of inequalities aims to force the δ-variables included in the solution of LP relaxations to more realistic values. Due to the constraints $y_{i2} \leq H\delta_i$, δ-variables are often found to have very small values in LP relaxations, while the ILP requires them to be either 0 or 1. Adding the inequality forced the δ-variables to a more realistic value and decreased the integrality gap and limited the number of partitions to investigate.

The second type of inequality aims to exploit the capacity required at a node when adding all the child nodes of the node. If the sum of the capacities of the child nodes of node j exceeds

the capacity b_j available at node j, expansion will be required when all the child nodes are included. The aim here is again to force the δ-variables to more realistic values.

procedure add_cut2

begin

for $j = 1,2,3,\ldots,n-1$ do

begin

Define $C_j = \{i \mid p_i = j\}$

if $C_j \neq \varnothing$ then

begin

dem_sum $= \sum_{k \in C_j} d_k$

if dem_sum $> b_j$ then do

begin

add_constraint $x_j + \sum_{k \in C_j} x_k - \delta_j \leq |C_j|$

end

end

end

The inequality is enforced if and only if all the child nodes of a specific node is included and in doing so, the capacity of the node is exceeded and capacity expansion would be required.

Before the next valid inequality is discussed it is necessary to define some notation. Let $P(i,j]$ define the set of nodes on the path between node i and node j, with i not included in the set.

This valid inequality considers the reduced capacity along the path between the root node and the destination node. The idea is that capacity is used up along a path to a node, thus only a part of the total capacity H is available at any particular node.

procedure add_cut3

Begin

for $j = 1$ to $n-1$ do

begin

add_constraint $y_{j2} \leq \left(H - \sum_{k \in P(0,j]} d_k - b_j\right)\delta_j$

end

End.

A fourth set of inequalities is given by: $y_{j2} \leq \left(\sum_{i \in T(j)} d_i\right)\delta_j$ where $T(j)$ is defined as $T(j) = \{i \mid i \text{ is a descendant of} j\} \cup \{j\}$, instead of $y_j \leq H\delta_j$. This inequality set has to do with the capacity required of the sub tree rooted at node j which would have a certain capacity required if all nodes were included. It places a bound on the total capacity required to flow over the link y_j.

Note that the third and fourth sets of inequalities imply that H in the equations $y_{j2} \leq H\delta_j, j = 1,2,3,\ldots,n-1$ can be replaced by $y_{j2} \leq H'\delta_j, j = 1,2,3,\ldots,n-1$ where $H' = \min\left\{H; H - \sum_{k \in P(0,j]} d_k - b_j; \sum_{i \in T(j)} d_i\right\}$.

Adding these valid inequalities to the formulation tightened the bounds used in the algorithm tremendously and improved performance dramatically. Other improvements were also made by using the standard software; CPLEX is the case of this study, efficiently with knowledge about the problem instance being solved. More detail is presented in the section on the empirical results.

7.5.5 Heuristic for the ETKP problem

Similarly to the TKP where a 0-1 knapsack with a relatively small capacity was used to create a heuristic for the TKP problem, a heuristic was also devised for the ETKP. The implementation for the ETKP is a bit more complex. For the TKP it is possible to create a 0-1 knapsack that is independent of the main problem. It is only necessary to have a mapping of the variables of the 0-1 knapsack to the variables in the TKP. The objective function coefficients of the TKP could be used directly, to formulate the Knapsack problem, used in the heuristic function that is used to create "good" feasible solutions.

An analogous heuristic procedure was developed for the ETKP that is a little more complicated, than the one investigated for the TKP above. This is explained below. This procedure aims to produce a relatively good lower bound for the ETKP.

The heuristic is implemented after the LP relaxation of the ETKP, ILPR(ETKP), has been solved to obtain solution values x_j^* for $j = 0,1,2,\ldots,n-1$. The set B_l is defined as the set of nodes included (with x_j values of 1) in the solution of ILPR(ETKP). Define the set B_{Adl} as the set of child nodes of the set B_l. The capacity used up for the sub tree B_l is defined as $H_{B_l} = \sum_{j \in B_l} d_j$ with the capacity available for the heuristic as $H - H_{B_l}$. The ILP heuristic ILP(ETKP, H_{Bl}) is then defined as follows:

$$\max \sum_{j=0}^{n-1} c_j x_j - \sum_{j=1}^{n-1} \left(F_j \delta_j + \alpha_j y_{j2} \right) \tag{7.17}$$

$$\text{s.t. } x_j - x_{p_j} \leq 0 \quad j = 1,2,\ldots,n-1 \tag{7.18}$$

$$x_0 = 1 \tag{7.19}$$

$$D\boldsymbol{x} - B(\mathbf{y}_1 + \mathbf{y}_2) = \mathbf{0}, \tag{7.20}$$

$$\boldsymbol{d}^{\mathrm{T}} \boldsymbol{x} \leq H, \tag{7.21}$$

$$y_{i2} \leq H\delta_i, \tag{7.22}$$

$$0 \leq y_{j1} \leq b_j \qquad j = 1,2,\ldots n-1 \tag{7.23}$$

$$\sum_{j \in B_l} x_j = | B_l | \tag{7.24}$$

$$\sum_{j \in B_{Adj}} x_j d_j \leq H - H_{B_l} \tag{7.25}$$

$$y_{j1}, y_{j2} \geq 0 \qquad j = 1,2,\ldots n-1, \tag{7.26}$$

$$x_j \in \{0,1\} \qquad j = 0,1,\ldots n-1, \tag{7.27}$$

$$\delta_j \in \{0,1\} \qquad j = 1,2,3,\ldots,n-1. \tag{7.28}$$

Equation (7.24) ensures that all the variables in set B_l are included in the solution to ILP(ETKP,HEUR). The set of equations (7.25) corresponds to a simple 0-1 knapsack constraints, the objective function is evaluated differently. The problem ILP(ETKP,HEUR) is relatively easy to solve as most of the variables are fixed and the additional 0-1Knapsack constraints have a relatively low capacity available. This is due to the fact that during the empirical work it was found that the LP relaxation solution to ILPR(ETKP) included many variables at value 1. The complete heuristic step is now presented.

Procedure HEUR_ETKP

begin

Solve ILPR(ETKP) to obtain solution values $x_j = 1, j = 0,1,2,\ldots,n-1$

Define the set $B_l = \{j \mid x_j = 1, j = 0,1,2,\ldots,n-1\}$

Define the set $B_{Adj} = \{j \mid x_{p_j} \in B_l, x_j \notin B_l\}$

Calculate $H_{B_l} = \sum_{j \in B_l} d_j$

Solve ILP(ETKP, H_{Bl}) to obtain objective function value $Z_{ILP(ETKP,H_{Bl})}$, this produces the required lower bound

End (end procedure HEUR_ETKP)

In the empirical results this procedure produced a fairly good lower bound, with accompanying benefits for the algorithm.

7.6 Conclusion

Partitioning algorithms for both the TKP and ETKP cases that use standard software coupled with enhanced modeling have been discussed. Also, enhancements to the algorithms and heuristics that fall within the same class of solution strategies are presented. As the TKP and ETKP models form part of the modeling process of the telecommunications networks and may need to be solved several times, it is necessary to have efficient solution methods available for the models. The empirical work on the efficiency of the algorithms will be presented in the next chapter.

Chapter 8 Empirical work

This chapter is devoted to the results of the empirical work. The planning of the tests that were done, will be presented, followed by the results. The experiments discussed are divided into two parts; first results for the TKP will be presented, followed by results for the ETKP. A section on performance measures, discusses different ways in which solution strategies may be compared. Empirical results for the TKP and ETKP as well as some general conclusions complete the chapter.

8.1 Planning of empirical work

This section discusses the data used for the empirical work. Real world data could not be obtained, hence randomly generated data instances were used to test the partitioning algorithms. It was necessary to plan the generation of test cases carefully, to allow accurate and realistic comparisons to be made. When generating data instances, a pseudo random number generator is used to generate data to represent a tree structure as well as other relevant data. The general methodology used is to create the tree first and then to populate the node and link data values afterwards. This allows a tree to be populated with different demands and profits, as will be seen in subsequent sections. This facilitates comparisons to be made between types of data generated. In order to minimise the data storage requirement for each data instance, trees are generated in a systematic order in which it is only necessary to know the initialisation value of the pseudo random number generator, and ranges wherein the values generated should be. The initialising value for pseudo random number generators gives a starting point for the series of values generated. This means that if the initialising value for the pseudo random number generator is known, the same tree can be generated repeatedly. The other information needed to generate identical trees is the ranges in which values must be generated, for instance the range of the profit for each node. The tree generation methodology is discussed in the next section.

8.1.1 Tree generation

As explained previously, it is only necessary to know the starting value for the pseudo random number generator and certain ranges to define a specific tree. This allowed the data concerning the data instances created to be stored very efficiently. The initialisation value for the pseudo random number generated is sometimes referred to as the seed value. The ranges that needed to be stored is summarised in the following list:

- Branch out factor: Each node in the tree may have child nodes. Generally not all nodes are required to have child nodes. The numbers of child nodes are chosen randomly from an interval between 0 and a positive integer called the maximum branching factor.
- Demand and profit: The demand and profit for each node is also generated from an interval between 0 and a certain upper bound.
- Expansion cost values: For the ETKP, it is necessary to generate values to represent the expansion cost, variable cost and existing capacity of each link. The ranges for these values must also be stored to recreate a tree correctly.

An adjacency matrix representation is used for each tree created. The manner in which a tree is created is now given. The steps to create a tree (given an initialisation value for the pseudo random number generator and appropriate ranges for the values required in the tree) is described in the procedure below. It creates a tree labelled in a breadth-first manner. The number of child nodes, denoted by the variable CurBranch, for each node is an integer number generated from the set $[0, \text{BranchOut}]$, where BranchOut is supplied as a parameter. For the special case where the remaining number of nodes to be created is less than BranchOut, the value for CurBranch is set as the number of remaining nodes. This is done to ensure that the correct number of nodes is generated. It is possible to generate an infeasible number of child nodes in some cases, if this is not done. Assume that the number of nodes to be generated for the tree is represented by the number ProblemSize. The variable NodeCounter is used to keep track of the node currently being worked on. Pseudo code for the procedure is presented below.

```
Procedure CreateTree
begin
  Create node 0 (the root node)
  Update tree representation with root node information.
  Set CurrentNode = 0;
  Set NodeCounter = 1;
  while (nodeCounter <= (ProblemSize -1)) do
  begin
       If (ProblemSize – NodeCounter < BranchOut)
       Then
                    CurBranch = ProblemSize – NodeCounter
       Else
```

```
            CurBranch = GetRandomNumber(1,BranchOut)
        For (counter =1; counter <= BranchOut; Increment Counter)
        Begin
            Add node with parent CurrentNode and with index NodeCounter
            Update tree representation with new node information
            Increment NodeCounter with 1
            end     (end For Loop)
        Increment CurrentNode with 1
    end (while)
end (procedure CreateTree)
```

The procedure GetRandomNumber uses a pseudo random number generator to generate an integer valued number between 0 and BranchOut. Upon completion of the procedure a tree has been created with a valid representation thereof.

8.1.2 Updating tree Values

After a valid tree has been created by the procedure described Section 8.1.1 above, it is necessary to update certain values for the nodes and the links. For both the TKP and ETKP instances the profit (c_j) and demand (d_j) used for each node are updated. For the ETKP the existing node capacity (b_j), fixed cost (F_j) and variable cost for expansion (a_j) must be updated. These variables are updated according to specified ranges. These values are only updated after the tree has been created, to allow various types data instances to be used with the same tree setup.

Some 0-1 Knapsack work done by Pisinger (1995:50) and Kruger(1998:117) indicated that the relationship between the profit and demand of individual nodes leads to different difficulty classes for 0-1 Knapsack problems. One of the contributions of this research was to apply some of this work to the TKP. For the TKP four relationships were tested between the profit and the demand of the individual nodes. This is similar to the classes proposed by Pisinger (1995:50). The classes of problems used are the following:

- Uncorrelated problems: There is no direct correlation between the profit and demand of a specific node. This means that $c_j \in \{0,1,2,\ldots,UpperLevel\}$ and $d_j \in \{0,1,2,\ldots,UpperLevel\}$, are not correlated. When the profits and demands for the individual nodes are plotted, a graph similar to figure 8-1 below is obtained:

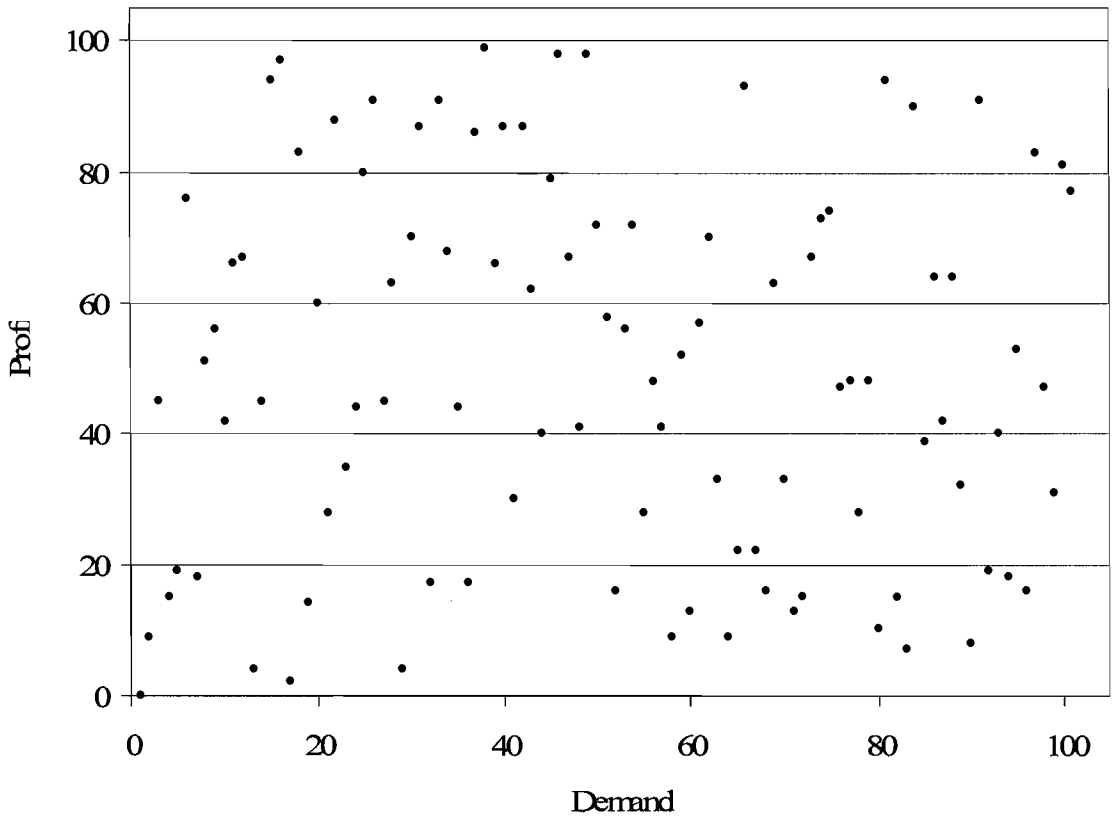

Figure 8-1 Uncorrelated data instance

- Weakly correlated problems: There is a weak correlation between the profit and demand of a specific node. The implementation implied that a demand was chosen, $d_j \in \{0,1,2,\ldots,UpperLevel\}$, and the profit for the node was generated to have some correlation to the demand. This is achieved by generating a random valued integer, say $r_{val} \in [-Interval,+Interval]$ and adding this to the value d_j, i.e. $c_j = d_j + r_{val}$. If a negative c_j is generated,the value for c_j is set to $p_j = 1$. An important remark about the data points for the weakly correlated case is that the points seem to be very well correlated. This is due to the fact that a strong relationship exists between the profit and demand for an individual node. The ratio of the profit versus demand, $\frac{c_j}{d_j}$, for individual nodes may differ for individual nodes as the values differs in magnitude. This ratio plays a significant role in the perceived difficulty of 0-1 Knapsack problems. A typical graphical representation of weakly correlated data instance is given in figure 8-2 below:

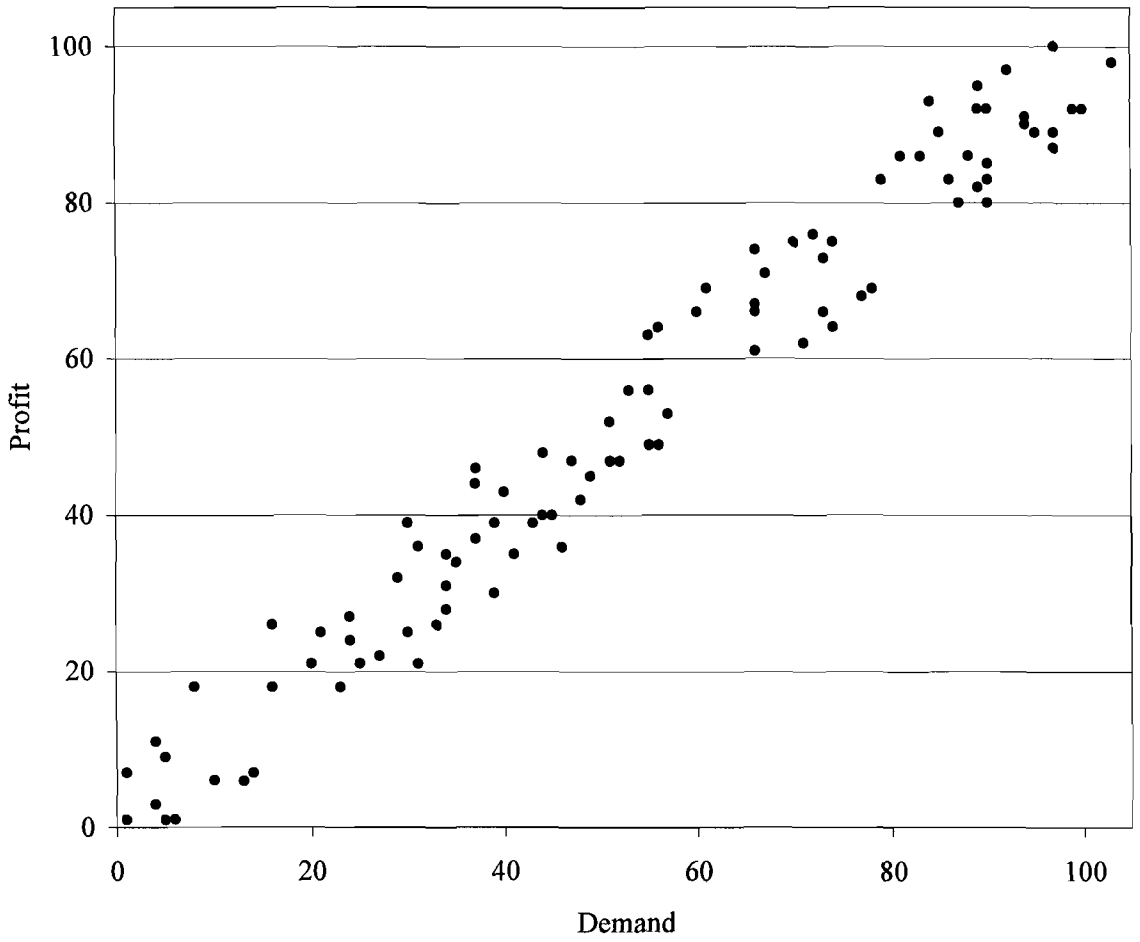

Figure 8-2 Weakly correlated data instance

- Strongly correlated problems: There is a strong correlation between the profit and demand associated with a specific node. In practice this was implemented in the following way. A demand d_j was randomly chosen from the set $d_j \in \{0,1,2,\ldots,UpperLevel\}$. The profit c_j is then calculated as $c_j = d_j + s_{val}$, where r_{val} is a non-negative integer value. A graphical representation of the strongly correlated case is presented below in figure 8-3.

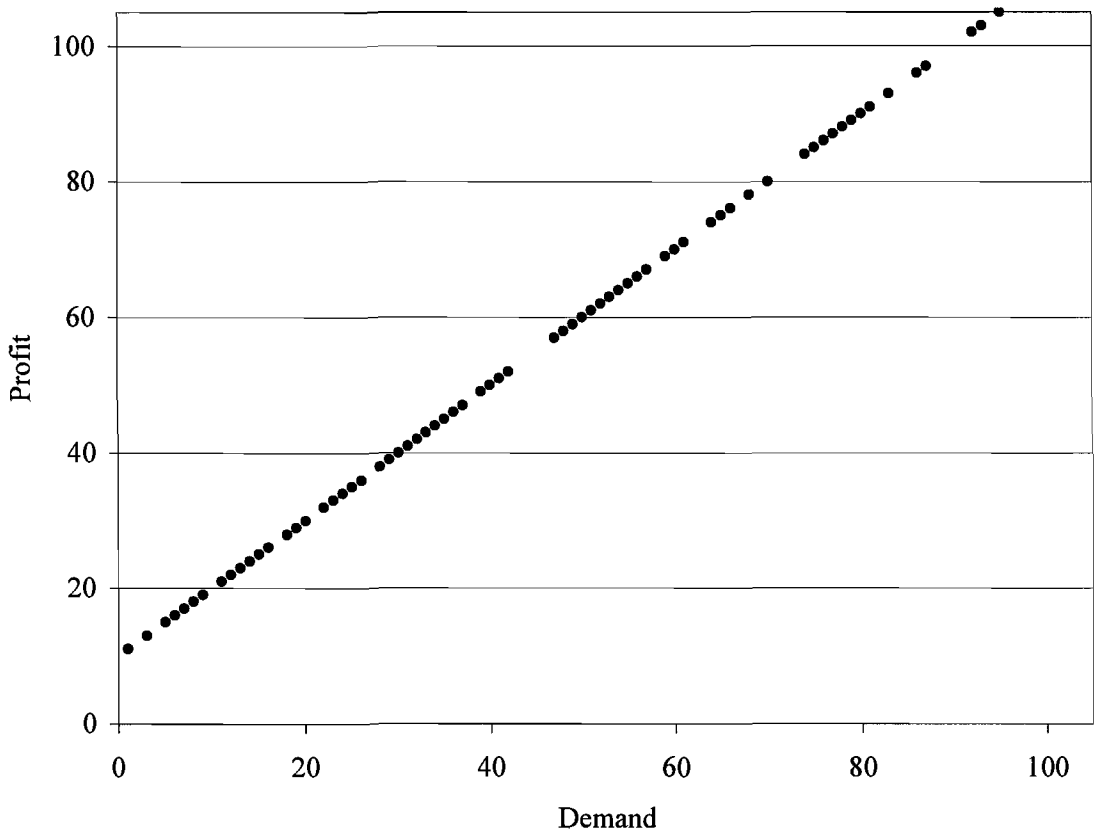

Figure 8-3 Strongly correlated data instance

- Subset sum problems: The profit and demand is the same for a specific node. This means that $c_j = d_j, j = 0,1,2,\ldots,n-1$. In practice Practically this means that d_j is randomly generated from the set $\{0,1,2,\ldots,UpperLevel\}$. Afterwards, the value of c_j is set to the same value as d_j for all the nodes in the problem instance. Graphically this can be seen in figure 8-4 below. This type of data instance is sometimes referred to as the value independent case (Balas and Zemel (1980:1148)).

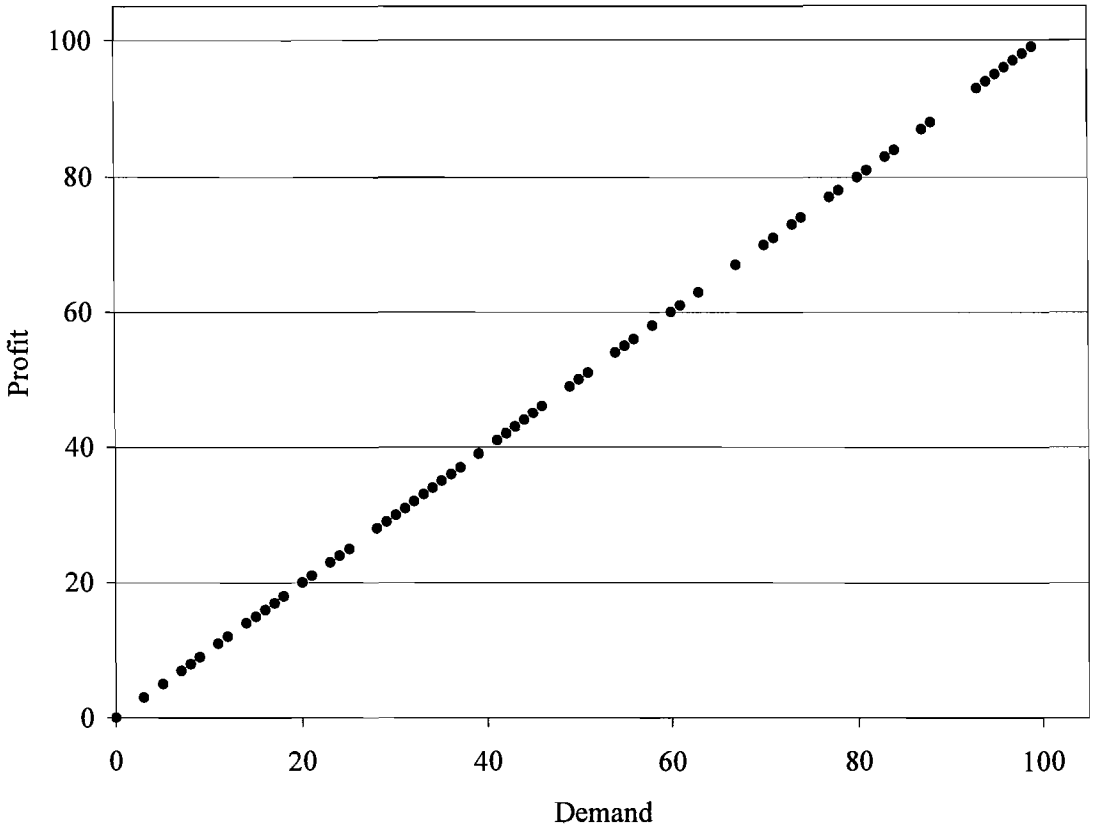

Figure 8-4 Subset sum data instance

In all the figures above (figure 8-1 to figure 8-4), 100 sample data points were randomly generated where $UpperLevel = 100$, $Interval = 10$ and $s_{val} = 10$.

The empirical test system was designed so that the values for $UpperLevel$, $Interval$ and s_{val} can be specified as parameters. If these values, the branching factor and an initialisation value for the pseudo random number generator are stored, all the required information to create a specific data instance is known.

8.2 Performance measures

An important aspect of the empirical work is to show the relative performance of the algorithms developed. Unfortunately, this is not an easy task as there are many ways in which algorithms may be compared. The first and probably the most common method is to compare the total solution times needed to solve problem instances. This gives an indication of how long it takes to solve problem instances on average. It may also be helpful to show the

minimum and maximum time needed to produce solutions for various problems. One problem with this approach is that in some cases good solutions are produced early in the solution process, but proving that these solutions are the best possible solutions may take a long time. Other solution methods, for instance some dynamic programming algorithms, do not produce a usable solution till the end of the solution process. The solution process that produces intermediate solutions may be preferred by some practitioners of optimisation.

In terms of standard MILP and IP software, it may be possible to compare the number of iterations performed and branches expanded during the solution process. This also gives and indication of how much work is done by a solution process. The number of branch and bound nodes expanded during a solution process also points to another performance measure, which is computer memory usage. It may thus also be possible to compare solution strategies in terms of memory requirements. Excessive memory requirements may be a negative factor for a solution strategy.

In this thesis the total time taken to solve problem instances is used. In order to make sure that a valid general trend was identified, numerous data instances were used and an average calculated. The minimum and maximum time needed for data instances of a certain size is also given.

The computing platforms for the empirical work are as follows:

- For the TKP a Pentium IV 3.2 Ghz was used with 1Gb of memory available. CPLEX 10 from ILOG is used to solve all MIP, ILP and LP problems. The operating system used is SuSE 10.1 with C++ compiler g++ version 4.3.
- For the ETKP AMD DL145G2 OPTERON 64 bit machines were used. This is a cluster computing platform with parallel computing capability. Parallel computing was not utilised in this thesis, but may be considered in future work. The operating system is RedHat Linux. The machines have 4Gb of ram and CPLEX 10 was used. This platform had more computing power than the one used for the TKP empirical work.

In CPLEX the so-called concert technologies are used to formulate and pass problems to the CPLEX solver, for more information, refer to Chapter 1, ILOG (2002). This allows the models to be implemented and manipulated relatively easily and efficiently.

8.3 Results

8.3.1 TKP Results

For the TKP, problem instances for up to 60 000 nodes were investigated. The problem size was incremented in 5 000 node steps. For each problem size, 4 tree configurations were used. For each tree configuration, the capacity of the TKP was varied between 10% and 90% of the sum of the capacities of the individual nodes, formally, if $TotCap = \sum_{j=0}^{n-1} d_j$, then H was varied between $0.1*TotCap, 0.2*TotCap, \ldots, 0.9*TotCap$. In this manner 36 data instances were generated for each problem size, with 4 tree configurations with 9 different capacities for each configuration. The same tree configurations were used for the different types of data instances. This was done to compare the difficulty of TKP data instances compared to the 0-1 knapsack instances. According to Balas and Zemel (1980:1151), the difficulty of 0-1 Knapsack instances depends on the ratio between the profits and demands of the individual nodes. This means that if the ratios are the same, or close to the same, for all the nodes, the problem is more difficult, (e.g. subset sum instances) and if there is a lot of variation in the ratios between nodes, the problem is generally easier, e.g. uncorrelated data instances. A second factor is the gap between the LP relaxation of the problem and the ILP solution of the problem. The data instances presented above is presented in order of increasing complexity for the 0-1 Knapsack, i.e. uncorrelated, weakly correlated, strongly correlated and (most difficult), the subset sum instances. As the 0-1 Knapsack is a special case of the TKP, using the same tree configurations, it can be empirically tested whether the same order of complexity holds true for the TKP case.

To compare the performance of the partitioning TKP algorithm, the Branch and Bound algorithm developed by Shaw and Cho (1996) was implemented from pseudo code. According to Shaw and Cho (1996) the Branch and Bound algorithm is a more efficient algorithm than the dynamic programming algorithm they developed (Cho and Shaw (1997)). No implementation could be obtained from the original authors, hence an implementation was done in C++. Another comparison alternative used is standard optimisation software (in the case of this study, CPLEX from ILOG). This gives some indication of how long it would take to solve problem instances without enhanced modeling or partitioning.

The following notation is used in the presentation of the empirical tests:

- Part TKP Alg: The partitioning TKP algorithm presented previously.
- CPLEX: Standard software, CPLEX from ILOG in this study.
- Shaw: An implementation from pseudo code of the Branch and Bound algorithm developed by Shaw and Cho (1996).

A cut-off time was imposed on the solution methods used. If a problem was not solved within 7 200 seconds, it was deemed intractable and the solution process was stopped. This cut-off value was imposed on desktop time and not processor time. The solution processes were not always stopped strictly at 7 200 seconds: in some cases the stop rule was not always encountered at exactly 7 200 seconds.

The partitioning TKP algorithm used, employs the 0-1 Knapsack heuristic as described previously, and uses all the enhancements discussed. The settings for the standard software are the same when solving the TKP problem instances to obtain a benchmark value and when the standard solver, CPLEX in this case, is used in the partitioning algorithm.

The four types of data instances will be covered in separate sections, with a brief discussion at the end of each section. Each section will contain a table summarising the time taken to solve data instances. The average time taken to solve the 36 data instance of a specific size and the minimum and maximum times needed to solve a data instance of the specified size, is also reported. The column headers for the first tables are as follows:

- Avg: The average time needed to solve the 36 problem instances.
- Max: The longest time taken to solve one of the 36 problem instances. This gives an indication of the maximum time needed to solve a problem instance.
- Min: The shortest time needed to solve any of the 36 problem instances.

A graphical representation is given to summarise the results of the tables.

The second table in each section contains a summary of the standard deviation of the solution times of the data instances.

8.3.1.1 Uncorrelated TKP data instances

The first results for the uncorrelated data instances are presented first.

	CPLEX			Part TKP Alg			Shaw		
Nodes	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
5000	1.125	2.800	0.312	0.718	2.374	0.205	22.651	99.550	0.052
10000	3.738	10.843	1.645	4.468	54.485	0.554	28.704	661.703	0.126
15000	7.196	18.054	2.228	2.915	14.934	0.813	215.182	2592.806	0.290
20000	11.385	20.444	5.103	3.105	6.368	1.815	30.196	888.558	0.490
25000	17.584	40.523	7.012	19.395	521.527	1.717	345.001	7324.577	0.814
30000	28.735	94.944	9.841	9.614	47.738	2.207	67.366	2212.705	1.361
35000	35.152	69.102	10.753	8.350	55.685	2.812	207.699	7210.943	1.644
40000	46.382	144.734	11.559	10.703	41.488	3.571	9.786	15.396	2.279
45000	61.340	151.051	16.794	23.432	424.975	4.076	36.624	856.495	2.901
50000	75.986	139.510	25.918	21.880	220.535	5.142	95.932	1441.173	3.555
55000	90.782	199.829	25.803	21.237	73.551	6.641	44.561	968.956	4.025
60000	115.391	240.323	38.363	25.259	150.526	9.283	265.258	7313.795	4.905

Table 8-1 Results summary of uncorrelated TKP data instance solution times

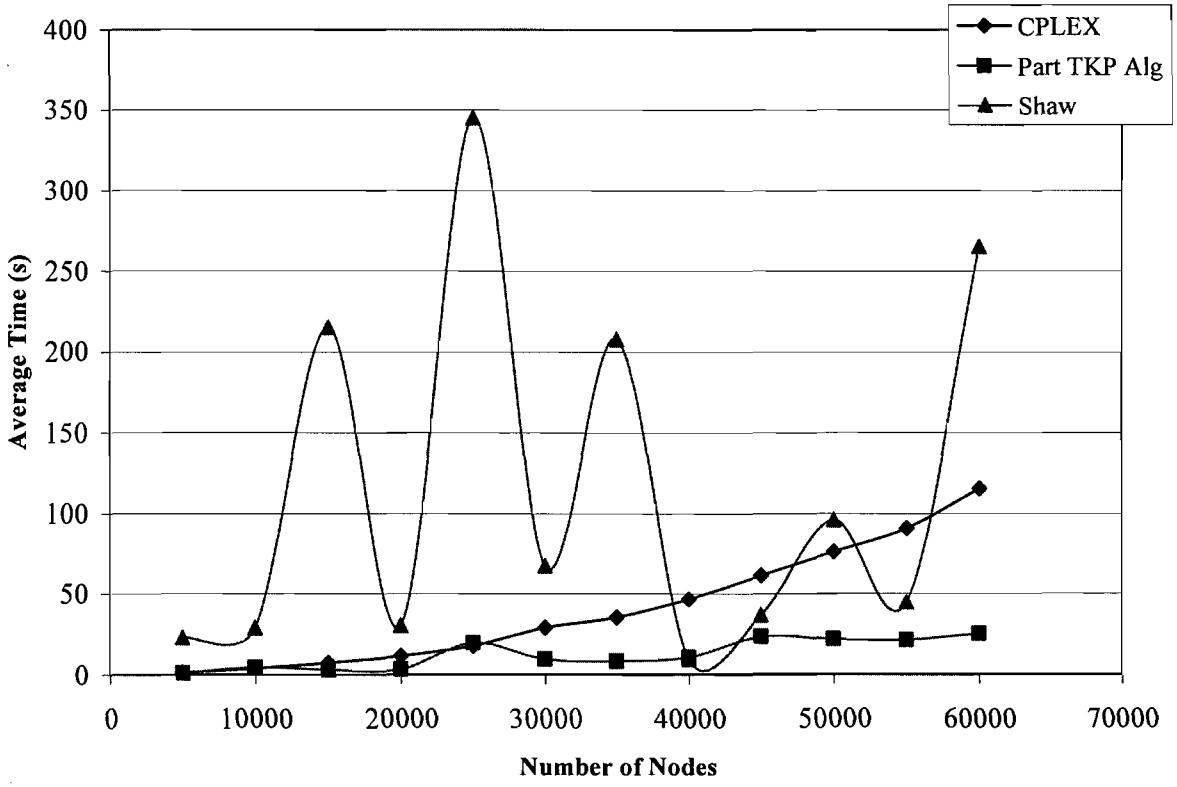

Figure 8-5 Graphical representation of average solution times for uncorrelated TKP data instances

	CPLEX	Part TKP Alg	Shaw
Nodes	Stdev	Stdev	Stdev
5000	0.48	0.50	32.57
10000	1.84	10.61	113.25
15000	3.15	2.75	690.58
20000	3.39	1.21	148.19
25000	7.83	86.16	1305.61
30000	15.17	10.40	367.79
35000	13.72	8.49	1200.56
40000	25.45	7.15	4.32
45000	29.08	69.07	140.77
50000	29.01	35.99	330.44
55000	42.31	15.97	158.81
60000	49.07	29.12	1233.91

Table 8-2 TKP Uncorrelated data instance standard deviation data

From table 8-1 and figure 8-5 it can be seen that the partitioning algorithm generally begins to outperform the other solution methods for problems with more than 30 000 nodes. A possible explanation for this is the overhead generated by the partitioning scheme. For larger problems the extra work invested by the partitioning scheme begins to pay off. The algorithm developed by Shaw and Cho appears to have very variable average solution times. This is evident from figure 8-5 and table 8-4 where the larger standard deviations also point to this fact. The large difference between maximum and minimum times also hints at variable solution times for the algorithm. It is also interesting to note that the maximum time needed to solve any problem instance with the partitioning TKP algorithm is around 521 seconds, whereas the maximum of the algorithm developed by Shaw is already greater than the cut-off value imposed of 7 200 seconds. The standard software generally performed well for the uncorrelated data instances. The partitioning algorithm generally did best on average for most data instances and it also scaled well to large data instances.

8.3.1.2 Weakly Correlated TKP results

Results for the weakly correlated data are now presented. These data instance are considered to be more difficult in the case of 0-1 Knapsack problems.

	CPLEX			Part TKP Alg			Shaw		
Nodes	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
5000	1.903	3.876	0.637	9.685	108.579	0.818	4.787	62.749	0.064
10000	3.588	8.078	1.295	4.950	40.693	1.068	36.686	592.850	0.187
15000	7.375	18.351	3.016	8.440	53.140	1.378	2.060	17.424	0.373
20000	11.815	31.397	4.736	8.980	52.260	1.726	120.217	4208.779	1.076
25000	19.583	56.740	6.386	23.495	327.013	2.004	4.688	15.191	2.855
30000	24.938	57.416	10.105	19.022	134.989	1.912	211.879	4683.640	2.245
35000	34.148	95.949	12.616	22.633	468.816	3.019	213.388	7290.698	5.672
40000	40.960	70.583	17.310	15.419	127.038	3.793	18.090	131.864	3.993
45000	59.047	125.342	22.834	20.320	154.774	3.520	429.095	7323.182	9.999
50000	74.814	236.559	27.697	28.276	130.406	4.837	234.670	7506.470	12.335
55000	78.911	229.962	20.332	24.669	101.173	5.321	26.392	107.251	7.110
60000	99.481	303.356	40.449	30.010	113.442	7.297	35.922	299.572	17.713

Table 8-3 Results summary of weakly correlated TKP data instance solution times

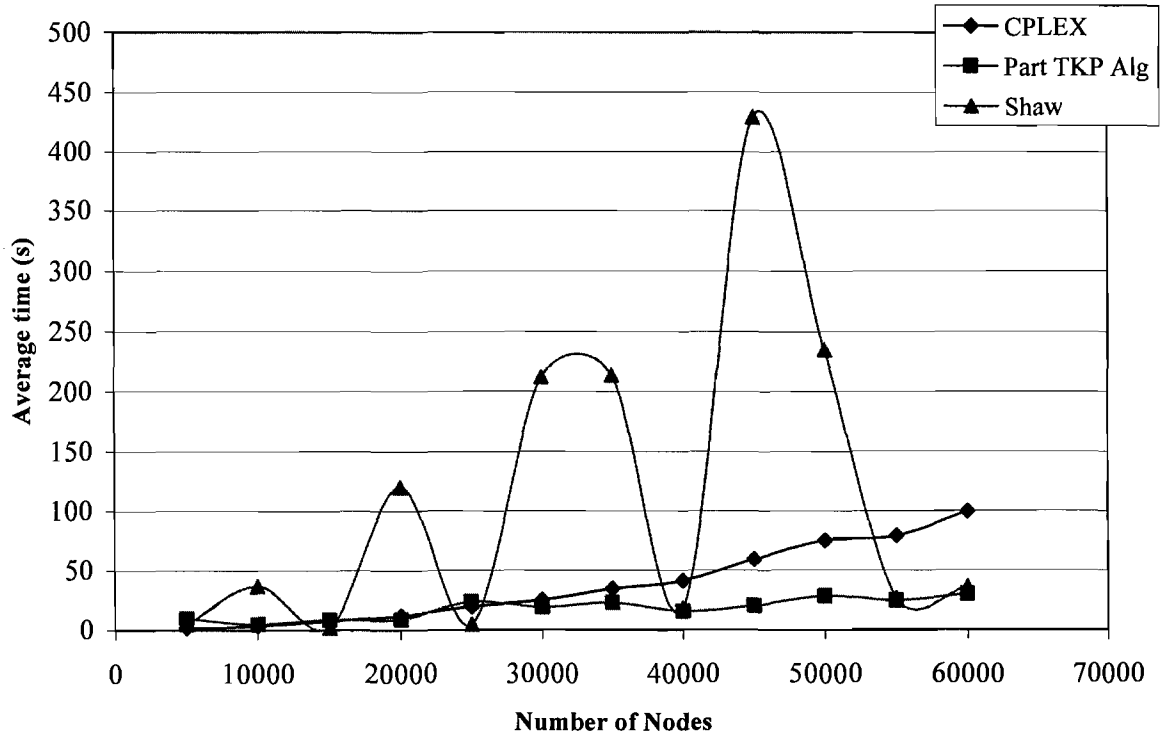

Figure 8-6 Graphical representation of average solution times for weakly correlated TKP data instances

	CPLEX	Part TKP Alg	Shaw
Nodes	Stdev	Stdev	Stdev
5000	0.90	19.39	15.72
10000	1.57	7.48	122.30
15000	3.15	12.12	2.75
20000	5.30	13.15	700.90
25000	11.25	68.91	2.27
30000	10.03	30.60	889.04
35000	18.16	76.64	1213.29
40000	15.95	19.80	22.01
45000	28.30	24.41	1686.24
50000	43.28	28.14	1246.81
55000	43.86	16.21	17.45
60000	54.28	20.69	47.36

Table 8-4 Summary of standard deviation times for weakly correlated TKP data instances

Similarly to the uncorrelated data instances the partitioning TKP algorithm generally outperformed the other approaches for problems with more than 30 000 nodes. The Shaw and Cho algorithm again produced variable solution times, with the largest standard deviation values as well. It also reached the cut-off value several times as can be seen from the maximum times column in table 8-3. In many instances the algorithm of Shaw and Cho could either solve the problem very quickly or a very long time was required. This is reflected in the larger standard deviation table. The partitioning algorithm had less variable solution times, as can be seen in table 8-4. It can be seen that problem instances of similar size needed more or less the same time to solve for the uncorrelated and weakly correlated data instances. This is an indication of the relative complexity of the data instances.

8.3.1.3 Strongly correlated TKP results

The results for the strongly correlated data results are now presented.

Uncorrelated									
	CPLEX			Part TKP Alg			Shaw		
Nodes	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
5000	3410.006	7256.866	0.616	2.281	30.179	0.399	3819.276	7203.323	0.140
10000	3612.863	7258.008	2.067	11.401	169.063	0.701	4841.120	7252.331	0.279
15000	3216.904	7288.550	4.754	23.591	615.769	1.310	6112.044	7332.410	0.785
20000	4259.113	7354.976	4.223	23.458	309.443	2.442	6638.717	7481.478	0.385
25000	4224.778	7315.243	15.429	242.002	7268.730	3.872	6793.937	7663.480	4.353
30000	4824.117	7291.787	6.693	435.205	7327.447	2.998	6728.708	7777.418	7.722
35000	4239.393	7365.350	21.682	39.843	245.304	3.647	6813.317	8168.594	4.289
40000	4636.100	7339.962	24.002	533.177	7456.917	5.082	7408.608	7935.608	7200.346
45000	4234.221	7308.892	30.020	389.520	7477.861	6.394	6575.538	8928.018	3.128
50000	4313.201	7261.320	37.542	336.451	6534.272	10.299	6825.888	9518.135	8.262
55000	3863.082	7301.718	41.659	642.039	7392.421	8.110	6375.769	9209.359	5.375
60000	3691.814	7331.925	38.597	271.411	2889.752	11.139	6565.981	9960.656	9.343

Table 8-5 Results summary of strongly correlated TKP data instance solution times

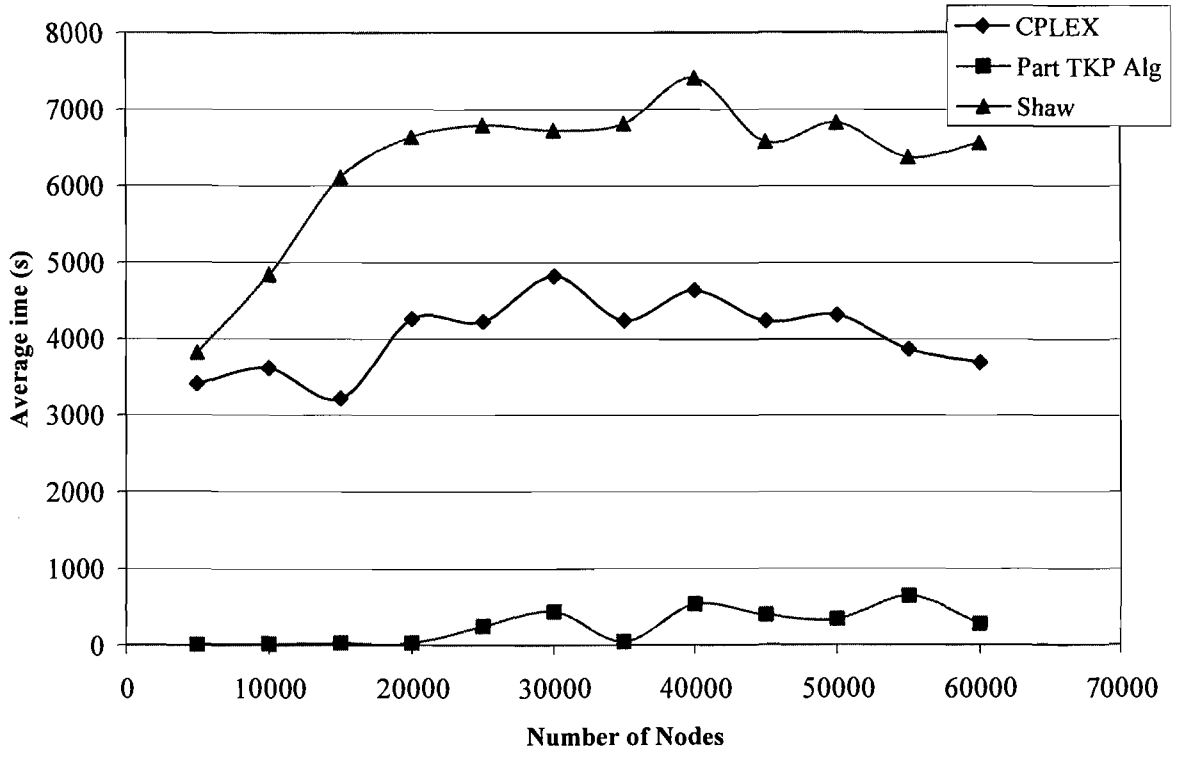

Figure 8-7 Graphical representation of average solution times for strongly correlated TKP data instances

	CPLEX	Part TKP Alg	Shaw
Nodes	Stdev	Stdev	Stdev
5000	3648.12	4.86	3464.05
10000	3660.58	30.06	3178.40
15000	3637.58	101.60	2540.41
20000	3564.35	51.06	2030.20
25000	3598.80	1205.93	1704.17
30000	3439.51	1684.02	2060.73
35000	3583.71	47.63	2097.83
40000	3484.42	1725.00	216.35
45000	3579.06	1307.22	2732.23
50000	3500.31	1130.41	2248.12
55000	3609.79	1792.98	2890.61
60000	3587.11	529.81	2717.62

Table 8-6 TKP Strongly correlated data instance standard deviation results

From the tables and figures above, it is evident that the strongly correlated data instances are more difficult to solve than the previous two types of data instances as the average solution times are greater. This appears to be in line with the results of the 0-1 Knapsack instances reported in literature. The Shaw and Cho algorithm produced very variable solution times, either very short or very long. Using average time as yardstick, the partitioning TKP outperforms the other approaches quite handsomely. For the 0-1 knapsack, it appears that initial work centred on solving the easier problems and only later improvements were made in the hard data cases. Analogously the partitioning seems to work well for the hard case. The standard deviation for the partitioning algorithm is also less than that of the other approaches. However, the algorithm of Shaw and Cho have artificially low standard deviation values for some problem sizes, because for these cases the cut-off time were reached and the average is thus very close to the cut-off value. For the standard software and Shaw and Cho algorithm the cut-off value was reached even for the smallest problem sizes investigated.

In table 8-7 below, the number of problem instances exceeding the cut-off value is presented. It is evident that in many instances the standard software and Shaw and Cho algorithm exceeded the cut-off value imposed. On the other hand, only in 7 instances out of 432 did the partitioning algorithm need more time than the cut-off value, and in these cases interim solutions with quality guarantees were still generated.

Nodes	CPLEX	Part TKP Alg	Shaw
5000	17	0	18
15000	19	0	22
15000	16	0	31
20000	21	0	33
25000	21	1	33
30000	24	2	33
35000	21	0	33
40000	23	2	36
45000	21	1	30
50000	21	0	32
55000	19	1	30
60000	18	0	31
Total	241	7	362

Table 8-7 Number of problem instance solution times exceeding cut-off value

8.3.1.4 Subset sum TKP results

The subset sum results are now presented.

	CPLEX			Part TKP Alg			Shaw		
Nodes	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
5000	0.662	1.236	0.406	0.392	0.872	0.217	0.087	0.202	0.048
10000	1.892	3.544	0.936	0.852	1.061	0.497	0.295	0.501	0.111
15000	3.895	9.860	1.905	1.437	1.868	0.597	0.665	1.222	0.146
20000	6.267	20.146	2.954	2.105	2.698	1.148	1.244	2.693	0.177
25000	9.503	19.666	4.578	2.934	3.838	1.537	2.109	4.421	0.258
30000	12.031	22.898	6.137	3.779	4.972	1.951	3.192	6.890	0.562
35000	16.802	27.002	8.279	4.796	6.303	2.358	4.354	9.559	0.262
40000	22.316	44.415	10.570	5.880	8.035	2.809	5.923	12.438	0.464
45000	29.351	53.945	13.247	7.148	9.761	3.422	7.144	15.787	0.798
50000	35.243	58.080	16.301	8.381	11.574	3.886	9.469	19.344	0.684
55000	44.775	87.142	19.794	9.827	13.941	4.763	11.210	23.341	0.688
60000	57.375	142.660	24.911	11.318	16.321	5.369	12.576	27.570	1.991

Table 8-8 Results summary of subset sum TKP data instance solution times

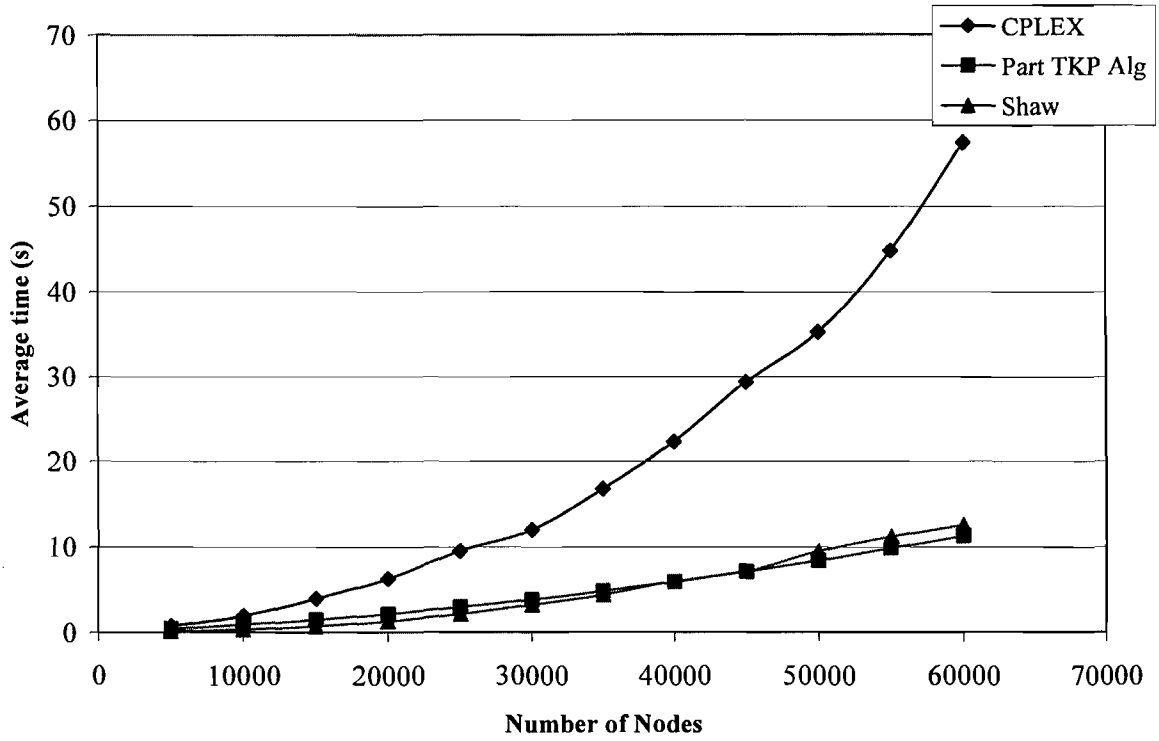

Figure 8-8 Graphical representation of average solution times for subset sum TKP data instances

	CPLEX	Part TKP Alg	Shaw
Nodes	Stdev	Stdev	Stdev
5000	0.19	0.11	0.03
10000	0.71	0.15	0.12
15000	1.64	0.31	0.27
20000	3.28	0.46	0.63
25000	4.04	0.68	1.17
30000	5.03	0.90	1.74
35000	6.82	1.21	2.75
40000	9.60	1.62	3.51
45000	13.43	1.93	4.47
50000	14.89	2.24	5.35
55000	19.46	2.80	6.61
60000	27.57	3.30	7.07

Table 8-9 TKP Subset sum data instance standard deviation results

The results for the subset sum data instance seem to be contradictory when compared to the perceived complexity of the 0-1 Knapsack case. For the subset sum instances, all the nodes have the same ratio between profit and demand as the profit is chosen as the demand. This gives a ratio of 1 for all nodes. Upon closer investigation, it was found that the objective function value of the LP relaxation (ILPR(TKP)) was close to the objective function value of

the IP problem (ILP(TKP)). The use of the heuristic discussed for the TKP in many cases produced optimal solutions to the problem instances. This meant that for many cases the heuristic produced an optimal solution. These results suggest that the subset sum problem is not as difficult for the TKP case, as is the case for the KP problem. All algorithms solved all the problem instances in less than 153 seconds. The partitioning algorithm appeared to have less variable solution times for the subset sum data instances, as can be seen in table 8-9 summarising the standard deviation times.

8.3.1.5 Conclusion on TKP results

The empirical testing shows that the partitioning TKP algorithm performed quite well in all the types of data instances investigated. In general the partitioning algorithm solved problem instances on average in less time that the standard software alone or the algorithm developed by Shaw and Cho. The algorithm developed by Shaw and Cho exhibited extremely variable solution times. In some data instances it solved the problem fairly quickly, while in other cases of the same size it reached the cut-off time. This is partly true for the standard software as well: in some cases the process to obtain the proven optimal value also caused the solution process to exceed the time allowed before the cut-off value.

In general it seems as if the performance of the partitioning algorithm is more robust than the other solution approaches tested. It also allows larger problem instances to be solved with less variable solution times. Only data instances to 60 000 nodes were investigated as very few strongly correlated data instances could be solved for larger problem sizes. This meant that no comparative times could be produced for larger problem sizes. The value of the partitioning algorithm is also that it works quite well for all types of data instances, weakly correlated to subset sum. This means that if no information is available on the nature of the data of the problems investigated, using the partitioning algorithm should be a good strategy. The Shaw and Cho algorithm has very variable solution times and was often outperformed by the standard software alone.

The investigation of the complexity of the data instances largely followed the complexity classes of the 0-1 KP with the exception of the subset sum case. It appears that the initial integrality gaps were quite small resulting in quicker solutions. The heuristic employed was also effective.

8.3.2 ETKP results

The empirical results for the ETKP are presented in a similar manner to the results for the TKP. For the ETKP the partitioning algorithm is compared with standard CPLEX without enhanced modeling or partitioning. An attempt was made to code the depth-first dynamic programming procedure presented by Shaw *et al.* (1997). It does however appear that there are some inaccuracies in the given pseudo code in their paper and this approach was abandoned.

The following notation is used in tables and figures concerning the ETKP empirical work:

- CPLEX: Using standard CPLEX from ILOG to solve the ETKP instances. This represents the benchmark against which the solution strategies developed in this work were tested..
- Part ETKP Alg: The partitioning ETKP algorithm described in this thesis. The heuristics developed in the study is used, along with all the valid inequalities presented for the ETKP.

It is also possible to use another method to solve the ETKP with CPLEX. The ILP solver from CPLEX has the ability to model piecewise linear functions through internal data structures. If this method is used, it is not necessary to formulate the ETKP model with the δ-variables. However, empirical results indicated that this approach is inferior to using the model developed in this study. Only rather small problem instances could be solved using the piecewise linear method and subsequently it was decided not to report on these results. Only results based on the improved modeling developed in the study is presented.

The third order partitioning strategy discussed in Section 7.5.2 was tested. The performance of this strategy was very poor. The results is not reported here. To improve performance, the valid inequalities discussed in Section 7.5.4 were used and the results of the performance benefits of these inequalities are given.

Slight changes in the model were implemented to improve computational speed. One example is that less promising cardinality values below a value p (say) were explored by an inequality requiring a cardinality value less than or equal to $p-1$, rather than models with equality constraints.

In table 8-10 below the average solution times and the minimum and maximum times for a problem instance of a specific size are presented.

	CPLEX			Part ETKP Alg		
Nodes	Avg	Max	Min	Avg	Max	Min
500	0.831	6.785	0.096	1.551	5.282	0.502
1000	4.459	28.864	0.507	5.598	11.666	1.515
1500	6.899	37.737	0.933	12.033	40.380	5.674
2000	12.229	43.108	2.190	20.239	57.791	8.129
2500	25.306	107.293	2.782	31.785	103.892	12.084
3000	63.100	411.608	12.305	47.443	103.658	16.300
3500	127.840	760.944	10.242	69.716	141.256	31.674
4000	1005.495	7332.835	14.335	101.406	199.198	35.573
4500	2118.516	7379.432	32.296	107.038	322.096	40.852
5000	4789.653	7387.948	34.029	179.233	697.230	54.418
5500	4934.597	7404.024	38.957	175.018	543.790	65.575
6000	6432.454	7440.480	48.125	231.518	553.622	77.725
6500	6776.770	7407.253	204.396	253.394	544.708	91.245
7000	6693.602	7402.188	147.163	342.055	1370.477	104.645
7500	6788.085	7419.859	435.039	352.701	1492.155	120.863
8000	6952.149	7429.735	146.509	445.105	1867.962	138.944
8500	6992.721	7393.959	1184.950	777.510	6802.467	156.730
9000	7131.309	7399.760	2773.856	612.337	1428.775	169.312
9500	7332.850	7384.207	7277.157	668.238	1649.501	193.198
10000	7205.479	7398.169	2558.322	902.726	4053.622	181.337

Table 8-10 ETKP data instance solution times summary

Note that a cut-off value of 7 200 seconds is imposed on both solution strategies. From the table it is evident that the standard software already reached this cut-off value for data instances of size 4 000 nodes, while it was not reached at all for the partitioning algorithm for problem sizes less than 10 000 nodes. Looking at the table above and figure 8-9 it is evident from empirical work that the ETKP is more complex than the TKP. This is intuitively evident, as the ETKP model incorporates flow generated by the nodes into the model, which makes it more complex. The ETKP model formulation has double the number of binary variables and an additional $2n$ continuous variables in the formulation. From table 8-10 and figure 8-9 it appears that the partitioning algorithm performs very well when compared to the standard software alone. It also indicates that larger problem instances could be solved more reliably.

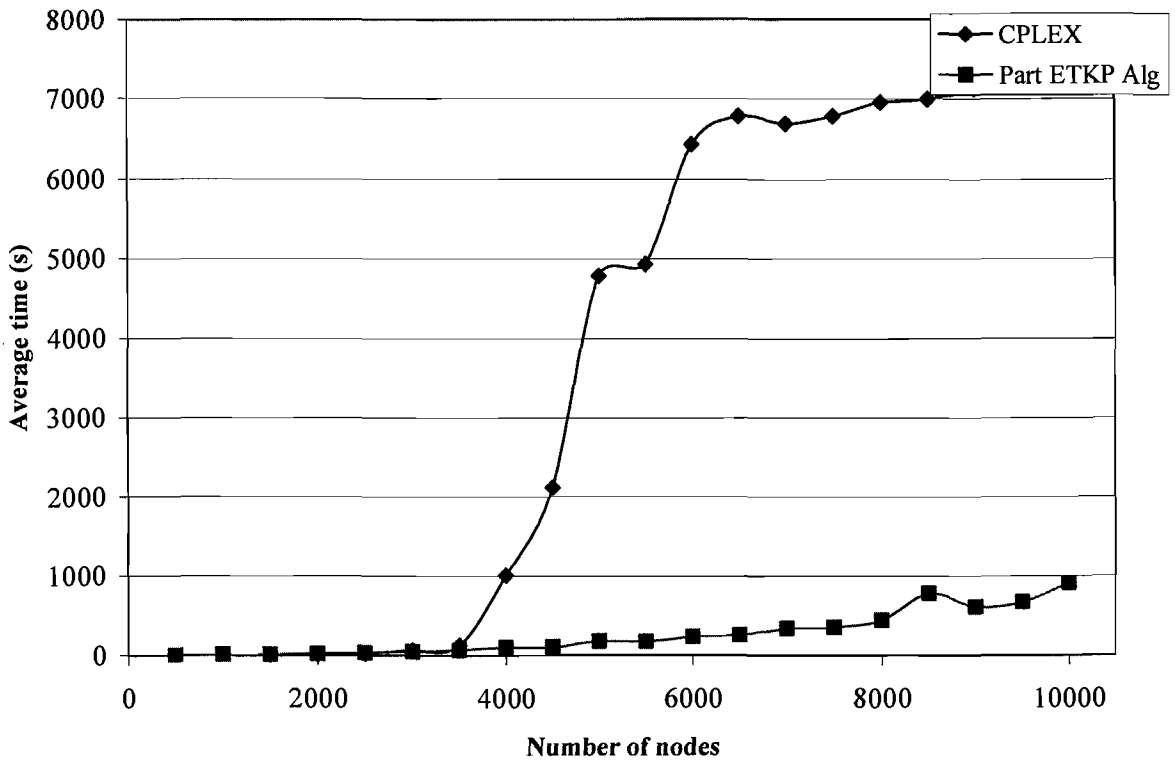

Figure 8-9 Graphical representation of average solution times for ETKP data instances

The figure above indicates that in the empirical study the partitioning algorithm performed better than the standard software, especially for larger problem instances. The partitioning algorithm has similar performance to the standard software for problem instances smaller than 3 500 nodes. This is partly due to the extra work being done in the partitioning algorithm, however, the solution times for problems less than 3 500 node are still small enough for this is not to be a serious problem. For the larger problem instances, the algorithm performed markedly better.

The standard deviations for the partitioning algorithm are generally also less than that of the standard software, as can be seen in table 8-11. This indicates less variable solution times. This in turns indicates that the partitioning algorithm performed more robustly in the experiments. An anomaly appears in the lower part of the table, where the standard deviation values for the standard software appears to decrease. This can again be explained by the fact that the standard software reached the cut-off time in many instances and thus shows an artificially small variability.

	CPLEX	Part ETKP Alg
Nodes	StdDev	StdDev
500	1.104	0.861
1000	5.157	2.344
1500	7.073	6.788
2000	10.210	10.614
2500	21.585	17.217
3000	72.149	23.899
3500	140.059	27.254
4000	1773.873	45.559
4500	2713.800	59.641
5000	2845.631	135.717
5500	3038.496	95.180
6000	2307.930	119.173
6500	1861.777	129.568
7000	1987.087	244.594
7500	1849.912	238.260
8000	1643.248	318.430
8500	1342.197	1174.457
9000	896.310	352.911
9500	23.484	374.184
10000	796.818	752.070

Table 8-11 ETKP data instance standard deviation summary for problem sizes investigated

Generally the partitioning algorithm allowed larger problem instances to be solved more reliably. The partitioning ETKP was tested with problem instances of up to 16 000 nodes with success. As no benchmarks are available for the standard software for problems above 10 000 nodes, its results are not explicitly presented here.

8.4 Conclusion

In general the partitioning algorithms produced solutions more reliably and for larger problem instances with less variable solution times. The algorithm presented by Shaw and Cho produced extremely variable solution times, with the added disadvantage that the algorithm has to be coded from pseudo code.

In general it appears that the partitioning algorithms are robust and no knowledge is required on data distributions and relationships of the problem instance being investigated. As the goal is to develop exact solution methods, no results are presented for cases where the partitioning algorithm was implemented to work to within a specified percentage of the proven optimal solution. This is often a good method to obtain feasible solutions with a pre-specified quality assurance.

The next chapter presents conclusion on the research done as well as an outline of possible future work.

Chapter 9 Conclusions

This chapter presents a summary of the results obtained in this study process and discusses some possible future work. It summarises recommendations on solution strategies to use when solving TKP and ETKP data instances.

9.1 Contributions

The main focus of the research is the partitioning algorithms developed for the TKP and ETKP models and their performance attributes. The algorithms use standard software and is based on enhanced modeling. The partitioning TKP algorithms perform very well when compared to standard software alone and a Branch and Bound procedure published by Shaw *et al* (1997). This algorithm was published by van der Merwe and Hattingh (2005). Additional work done on the TKP case included the investigation of the performance of this type of algorithm on some of the complexity classes for the 0-1 Knapsack problem. The TKP has the 0-1 Knapsack as special case. Empirical results are presented for the relative complexity considering some of the classes presented for the 0-1 Knapsack by Pisinger (1995). Generally the experience with the problem classes is similar to that found for the 0-1 Knapsack, with the exception of the subset sum class. A heuristic was developed for the TKP to differentiate amongst different partitions. An added benefit of this heuristic is that it often produced optimal solutions quickly and in some instances provided a relatively good lower bound. The Branch and Bound algorithm suggested by Shaw and Cho (1996) was implemented and comparisons were made between standard software (CPLEX in this case), the Shaw and Cho algorithm and the partitioning algorithm developed in this study. In general the partitioning algorithm is more robust and allows larger sized problem instances to be solved successfully.

In an analogous way, the partitioning algorithm for the TKP was adapted for the ETKP problem. The ETKP is more complex and needed more computation. Experimental work showed that the number of partitions generated for the ETKP were dramatically larger than those for the TKP. To counter the growth in the number of partitions, valid inequalities were introduced to decrease the integrality gap. This limited the number of partitions to be investigated and improved solution times considerably. The heuristic method created for the TKP was also adapted for the ETKP. In the context of the ETKP it was used to produce good starting solutions. This starting solution served as a lower bound, which on average improved

the overall solution times. When compared to standard software, the partitioning algorithm allowed larger problem instances to be solved. It also had less variable solution times and on average solved the problems faster. The algorithm published by Shaw *et al.* (1997) could not be implemented due to errors in the published article. Efforts to obtain the correct code from the authors were unsuccessful. Comparisons were thus only made between the standard software and the developed partitioning algorithm developed in this work. The partitioning ETKP algorithm appeared more stable in the empirical study and was able to solve larger problems than could be solved with standard software.

9.2 Recommendations

This paragraph aims to give some recommendations to persons interested in solving TKP and ETKP problems in practice.

For the TKP problem three possible solution methods were investigated. The algorithm published by Shaw (1996) solves smaller problems (less than 15 000 nodes) effectively for all classes of problem instances considered. For most of the data classes (except subset sum), it produced variable solution time for larger problem instances. It either solved the problem quickly or not at all for most cases. Standard software gave slightly better results, but also had problems solving some instances. The partitioning algorithm yielded very stable results over the problem sizes investigated and is the recommended solution method. This is especially true if no knowledge is available on the complexity class of the problem instance.

For the ETKP the standard software (CPLEX in this case) could solve problem instances up to 3 500 nodes. Up to this size CPLEX and the partitioning algorithm are both competitive. For larger instances, the partitioning ETKP algorithm is recommended.

9.3 Future work

In the course of the study various opportunities for further research were identified. Some of these possibilities are discussed below.

9.3.1 Parallel implementation of partitioning algorithms

The partitioning algorithms developed during this study was implemented in a sequential manner. The algorithms are especially amenable to parallel implementations where more than

one partition is investigated simultaneously. There are many questions that will need to be answered in this regard, some of which are:

- On which level will the parallelisation take pace? It is possible to solve both first order and second order partitions in parallel.

Updating the best current lower bound. If an improved lower bound is produced, will this value be broadcast to all other processes?

9.3.2 Use of partitioning strategies for other models

A question arising is whether the partitioning schemes used here can be beneficially applied to other mathematical programming problems as well. The partitioning of the search space produced very good results in this study. One specific model that comes to mind is the Steiner tree problem. This is a widely studied model considered to be relevant for network design problems.

9.3.3 Development of a system

Possible future work may include the development of a software application that can solve TKP and ETKP data instances. At present the software developed was for experimental purposes and is not a production system that is in a form for general usage. This development could include a parallel implementation of the algorithms.

9.3.4 Alternative cost functions

The cost function used for the ETKP model is just one possible cost function. This function assumes a fixed cost for expansion and a variable cost for additional traffic flow. Other cost functions, (see for instance Carpenter and Luss (2006)), may also be investigated to test the performance and feasibility of the algorithms developed.

9.3.5 Network expansion problem

Most telecommunications operators have existing networks. Increasing the number of users may cause existing capacity in the network to become insufficient. Additional capacity has to be provisioned to allow services to be delivered. This is in contrast to models in this thesis where mostly considered green field design was considerer, i.e. where no network is present. Possible research may be to investigate models that are more directly applicable to expansion problem.

9.4 General comments

Telecommunication networks are very complex entities. Modeling these networks are generally done in parts. The TKP and ETKP models are part of this process and it may be necessary to solve these problems numerously. As such it is important to be able to obtain solutions for these models efficiently. This thesis contributes to that goal. However, still a lot of interesting research has to be done in this field.

Chapter 10 Bibliography

AGRAWAL, D.P. & ZENG, Q. 2006. Introduction to wireless and mobile systems. 2nd ed. New York : Thompson. 497 p.

ALEVRAS, D. & PADBERG, M.W. 2001. Linear optimization and extensions. Berling : Springer. 449 p.

AMIRI, A. & PIRKUL, H. 1996. Primary and secondary route selection in backbone communication networks, European journal of operational research, 93:98-109.

AMIRI, A. & PIRKUL, H. 1997. Routing and capacity assignment in backbone communications networks, Computers and operations research, 24(3):275-287.

ATAMTURK, A, JOHNSON, E.L., LINDEROTH, J.T. & SAVELSBERGH, M.W.P. 2000. A relational modelling system for linear and integer programming, Operations research, 48(6):846-857.

BALAKRISHNAN, A., MAGNANTI, T.L. & MIRCHANDANI, P. 1994 A dual-based algorithm for multi-level network design, Management Science, 40:567-581.

BALAKRISHNAN, A., MAGNANTI, T.L., SHULMAN, A. & WONG R.T. 1991. Models for planning capacity expansion in local access telecommunication networks. Annals of operations research, 33:239-284.

BALAKRISHNAN, A, MAGNANTI, T.L. & WONG, R.T. 1995 A decomposition algorithm for local access telecommunication network expansion planning, Operations research, 43:58-76.

BALAS, E. & ZEMEL. E. 1980. An algorithm for large 0-1 knapsack problems. Operations research, 28:1130-1154.

BAZARAA, M.S., JARVIS, J.J. & SHERALI, H.D. 1990 Linear programming and network flows. 2nd ed. New York : Wiley. 684 p.

BELLOMO, N. & PREZIOSI, L. 1994. Modelling mathematical methods and scientific computation. Boca Raton : CRC Press. 497 p.

BENDER, E.A. 1978. An introduction to mathematical modelling. New York : Wiley. 256 p.

BIERMAN, H., BONINI, C.P. & HAUSMAN, W.H. 1991. Quantitative analysis for business decisions. 8th ed. Chicago : Irwin. 540 p.

BOURREAU, M. & P. DOGAN. 2005. Unbundling the local loop. European economic review, 49:173-199.

CARLIER, J., LI, Y. & LUTTON, J. 1997. Reliability evaluation of large telecommunication networks, Discrete applied mathematics, 76:61-80.

CARPENTER, T. & LUSS, H. 2006. Telecommunications access network design. (In Resende M.G. & Pardalos P.M., eds. Handbook of optimization in telecommunications. New York : Springer. 313-339 p.)

CHO, G. & SHAW, D.X. 1997. A depth-first dynamic programming algorithm for the tree knapsack problem. INFORMS journal on computing, 9:431-438.

CHO, G., SHAW, D.X. & KIM, S. 1997. An efficient algorithm for a capacitated subtree of a tree problem in local access telecommunications networks. Computers and operations research, 8:737-748.

CORTE-REAL, M. & GOUVEIA, L. 2007. Network flow models for the local access network expansion problem. Computers and operations research, 34:1141 – 1157.

CORTES, P, LARRANETA, J, ONIEVA, L.O., GARCIA, J.M. & CARABALLO, M.S. 2001. Genetic algorithm for planning cable telecommunication networks, Applied soft computing, 1:21-33.

COSTMAGNA, E., FANNI, A. & GIACINTO, G. 1998. A tabu search for the optimisation of telecommunications networks. European journal of operational research, 106:357-372.

CRUZ, F.R.B., MACGREGOR SMITH, J. & MATEUS, G.R. 1999. Algorithms for a multi-level network optimization problem, European journal of operational research, 188:164-180.

DEITEL, H.M. & DEITEL, P.J. 2003. C++ How to program. 4th ed. Upper Saddle River : Prentice-Hall. 1321 p.

FILHO, V.J.M.F. & GALVAO, R.D. 1998. A tabu search heuristic for the concentrator location problem, Location science, 6:189-209.

FISCHETTI, M., LEPSCHY, C., MINERVA, G., ROMANIN-JACUR, G. & TOTO, E. 2000. Frequency assignment in mobile radio systems using branch-and-cut techniques, European journal of operational research, 123:241-255.

FOWKES, N.D. & MAHONY, J.J. 1994. An introduction to mathematical modelling. Chichester : Wiley. 447 p.

GAREY, R.G. & JOHNSON, D.S. 1979. Computers and intractability, a guide to the theory of NP-completeness, New York : Wiley. 340 p.

GAVISH, B. 1991. Topological design of telecommunication networks - local access design methods. Annals of operations research, 33:17-71.

GAVISH, B. 1992. Topological design of computer communication networks – the overall design problem. European journal of operational research, 58:149-172.

GILDER, G. 2002. Telecosm – The world after bandwidth abundance. New York : Simon and Schuster. 355 p.

GIORDANO, F.R. & WEIR., M.D. 1983. A first course in mathematical modeling. Monterey : Brooks/Cole. 382 p.

GONDRAN, M. & MINOUX, M. 1984. Graphs and algorithms. Chichester : John Wiley & Sons. 650 p.

GOTTFRIED, B.S & WEISMAN, J. 1973. Introduction to optimization theory. New Jersey : Prentice Hall. 571 p.

GOUVEIA, L & LOPES, M.J. 1997. Using generalized capacitated trees for designing the topology of local access networks. Telecommunication systems, 7:315-337.

GOUVEIA, L. & PAIXAO, J. 1997. Dynamic programming based heuristic for the topological design of local access networks. Annals of operations research, 33:305-327.

GOUVEIA, L. & PIRES J.M. 2001. Models for a Steiner ring network design problem with revenue. European journal of operational research, 133:21-31

HROMKOVIC, J. 2003. Algorithmics for hard problems. Berlin : Springer. 544 p.

IBARRA, O.H. & KIM, C.E. 1978. Approximation algorithms for certain scheduling problems. Mathematics of operations research, 3:197-204.

ILOG. 2002. ILOG CPLEX 8.0 User's manual. France : ILOG. 422 p.

ILOG. 2006. ILOG CPLEX 10.1 User's manual. France : ILOG 476 p.

JOHNSON, D.S. & NIEMI, K.A. 1983. On knapsacks, partitions and a new dynamic programming technique for trees. Mathematics of operations research, 8:1-14.

JOHNSON, E.L. & NEMHAUSER, G.L. 1992. Recent and future directions in mathematical programming, IBM systems journal, 33(1):79-93.

KERSHENBAUM, A. 1993. Telecommunications network design algorithms. New York: McGraw Hill. 368 p.

KRUGER, M.F. 1998. State-space search models for discrete optimization, knapsack algorithms and related problems. Potcheftroom: Potcheftroom University for CHE (Thesis – Ph.D.) 166 p.

LEE, Y., SHERALI, H.D., HAN, J. & KIM, S. 2000. A branch-and-cut algorithm for solving an intraring synchronous optical network design problem. Networks, 35(3):223-232.

LINDEROTH, J.T. & SAVELSBERGH, M.W.P. 1999. A computational study of search strategies for mixed integer programming. INFORMS journal on computing, 11(2): 173-187.

LUKES, J.A. 1974. Efficient algorithms for the partitioning of trees. IBM journal of research and development, 18:217-224.

MALIK, D.S. 2004. C++ Programming: program design including data structures. 2nd ed. Australia : Thompson. 1555 p.

MARTELLO, S. & TOTH, P. 1987. Algorithms for knapsack problems. Annals of discrete mathematics, 31:213-258.

MARTELLO, S. & TOTH, P. 1997. Upper bounds and algorithms for hard 0-1 knapsack problems. Operations research, 45:768-778.

MATEUS, G.R. & PATROCINIO, Z.K.G. JR. 2006 Optimization issues in distribution network design. (In Resende M.G. & Pardalos P.M., eds. Handbook of optimization in telecommunications. New York : Springer. 341-366 p.)

MULLET, G.J. 2006. Wireless telecommunications systems and networks. New York, Thomson. 644 p.

NEMHAUSER, G.L. & WOLSEY, L.A. 1988. Integer and combinatorial optimization. New York : Wiley. 763 p.

NIELSON, C.A., ARMACOST, A.P., BARNHART, C & KOLITZ, S.E. 2004. Network design formulations for scheduling U.S. air force channel route missions, Mathematical and computer modelling, 39:925-943.

PADBERG, M. & RINALDI, G. 1991. A branch-and-cut algorithm for the resolution of large-scale symmetric travelling salesman problems, SIAM review, 33(1):60-100.

PADBERG, M. & RINALDI, G. 1990. Facet identification for the symmetric travelling salesman prototype, Mathematical programming, 47:219-257.

PAPADIMITRIOU, C.H. 1995. Computational Complexity. Reading, Massachusetts : Addison Wesley. 523 p.

PARK, K., LEE, K., PARK, S. & LEE, H. 2000 Telecommunication node clustering with node compatibility and survivability requirements. Management science, 46:363-374.

PARKER, R.G. & RARDIN, R.L. 1988. Discrete Optimization. Boston : Academic Press. 472 p.

PISINGER, D. 1995. Algorithms for knapsack problems. Copenhagen: University of Copenhagen. (Thesis- Ph.D.) 199 p.

PREISS, B.R. 1999. Data structure and algorithms with object-oriented design patterns in Java. Wiley : New York. 635 p.

PUPILLO, L. & CONTE, A. 1998. The economies of local loop architecture for multimedia services. Information economics and policy, 10:107-126.

RARDIN, R.L. 1998. Optimization in operations research. Prentice Hall: Uppper Saddle River. 919 p.

RESENDE, M.G.C. & PARDALOS, P.M. 2006. Handbook of optimization in telecommunications. New York : Springer. 1134 p.

SALKIN, H.M. & MATHUR, K.M. 1989. Foundations of integer programming. North Holland: New York. 755 p.

SHAW, D.X. & CHO, G. 1996. The critical-item, upper bounds, and a branch-and bound algorithm for the tree knapsack problem. Networks, 31:205-216.

SHAW, D.X., CHO, C. & CHANG, H. 1997. A depth-first dynamic programming procedure for the extended tree knapsack problem in local access network design. Telecommunication systems, 7:29-43.

SHEPARD, S. 2002. Telecom crash course. New York : McGraw-Hill. 484 p.

TANENMBAUM, A.S. 2003. Computer Networks, 4th ed. Upper Saddle River: Prentice hall. 892 p.

VALLETTI, T.M. 1999. The practice of access pricing: telecommunications in the United Kingdom, Utilities policy, 8:83-98.

VAN DER MERWE, D.J. & HATTINGH, J.M. 2003An adapted exact algorithm for solving extended tree knapsack problems for Local Access Telecommunication Network design issues (TELKOM, South African Telecommunications and network applications conference 2003)

VAN DER MERWE, D.J. & HATTINGH, J.M. 2005. The feasibility of LATN design using Tree Knapsack and Extended Tree Knapsack models. (TELKOM, South African Telecommunications and network applications conference 2005)

VAN DER MERWE, D.J. & HATTINGH, J.M. 2006. Tree knapsack approaches for local access network design, European journal of operational research, 274:1968-1978.

VOSS, S. 2006. Steiner tree problems in telecommunications (In Resende M.G. & Pardalos P.M., eds. Handbook of optimization in telecommunications. New York : Springer. 459-492 p.)

WANG, Q. 2006. Pricing and equilibrium in communications networks. (In Resende M.G. & Pardalos P.M., eds. Handbook of optimization in telecommunications. New York : Springer. 545-569 p.)

WILDE, D.J. & BEIGHTLER, C.S. 1967. Foundations of optimization. New Jersey : Prentice Hall. 480 p.

WINSTON, W.L. 1994. Operations research : applications and algorithms. Belmont : Duxbury Press. 1319 p.

WINSTON, W.L. 1995. Introduction to mathematical programming: applications and algorithms. Belmont: Duxbury Press. 818 p.

YOON, S., YOON, G. and LEE, J. 2005. On selecting a technology evolution path for broadband access networks. Technological forecasting and social change, 72:449-470.

ZADEH, N. 1973. A bad network problem for the simplex method and other minimum cost flow algorithms. Mathematical programming, 5:255-266.