Volume 26 (1), pp. 1–16
http://www.orssa.org.za

ORiON
ISSN 0529-191-X
©2010

# Solving the Extended Tree Knapsack Problem with fixed cost flow expansion functions

David J van der Merwe[*] J Michiel Hattingh[†]

*Received: 25 November 2008; Revised: 13 April 2010; Accepted: 26 April 2010*

**Abstract**
Parts of the Local Access Telecommunication Network planning problem may be modelled as an Extended Tree Knapsack Problem. The Local Access Telecommunication Network can contribute up to 60% of the total network costs. This paper presents partitioning algorithms that use standard off-the-shelf software coupled with enhanced modelling. Enhancements to the algorithms and empirical results for both the Tree Knapsack Problem and Extended Tree Knapsack Problem are presented.

**Key words:** Tree algorithms, telecommunications, location problems, capacity expansion.

## 1 Introduction

The *Local Access Telecommunication Network* (LATN) connects individual subscribers to telecommunications networks through switching centres [1]. Generally telecommunication networks are hierarchically divided into two [3] or three parts [2], with the local access network contributing a large portion of the total network development cost — according to [1] as much as 60% in the case of fixed line situations.

Part of the LATN expansion problem may be modelled as an *Extended Tree Knapsack Problem* (ETKP) [16]. Expansion may be required when subscriber numbers increase or if the demand for service from existing subscribers increases beyond the capacity of the existing network. This problem is treated extensively in [1, 3, 7], where models and solution strategies for the problem are given. In many cases LATN design and expansion problems are implemented sequentially, see as example [6, 8]. In some of these cases the *Tree Knapsack Problem* (TKP) and ETKP may be used as model. To solve the overall design problem, it may be necessary to solve many TKP and ETKP models. It is therefore important to be able to solve these problems efficiently.

[*]Corresponding author: Department of Computer Science and Information Systems, North-West University, Potchefstroom Campus, Private bag X6010, Potchefstroom, 2530, Republic of South Africa, email: davidvdmerwe@gmail.com

[†](**Fellow of the Operations Research Society of South Africa**), Department of Computer Science and Information Systems, North-West University, Potchefstroom Campus, Private bag X6010, Potchefstroom, 2530, Republic of South Africa.

A solution strategy for the ETKP with a fixed charge cost function is proposed in this paper. The research presented in [18] proposed partitioning algorithms for the TKP and gives some experimental performance results. In this paper we report on extended experimental work for TKP algorithms, by testing the algorithms on classes of problem instances with varying complexity. The algorithmic approaches have been extended to the ETKP case, where flows and capacities along the links are considered and flow expansion decisions are part of the LATN design process. Models with non-linear cost functions form the basis of the empirical work for ETKP presented here. Solving the ETKP is recognised to be computationally challenging. Algorithmic enhancements are introduced using valid inequalities and heuristics to obtain tighter lower bounds. Comparative empirical results are presented.

In the next section a general background on knapsack problems is presented, followed by a section on the ETKP model. General solution strategies are briefly discussed, followed by the proposed algorithm for the ETKP which is a major contribution of this paper. Valid inequalities and a heuristic procedure for the ETKP are also presented. These are followed by empirical results, conclusions and a brief note on possible future work.

## 2 Background

Before describing the ETKP model, a brief note on the general knapsack problem, or 0-1 *Knapsack Problem* (KP), is presented. The KP is a widely studied problem in operations research and combinatorial optimization. The motivation for this is that it has many variants and is often a subproblem of other optimization problems. Generally, in the KP a set of items are available with associated values. A subset of items must be chosen to maximise the total value, satisfying some capacity constraint. This type of problem occurs when a hiker or soldier has to fill his knapsack with items from a given set, while satisfying a suitable capacity constraint. This constraint may involve a maximum allowable weight or volume, or some other quantity [12].

In the design of LATN networks a special type of KP is encountered where a precedence order is imposed on the items to be chosen. This means that in order to include a specific item, it is required that another item is included first. For example, batteries must be included before a flash light may be included by a hiker. If batteries are included, it may be possible to pack a radio. TKP and ETKP models are discussed in [4, 5, 15, 16].

## 3 The ETKP model

The ETKP model considered here may be seen as an extension to the TKP model, as reported in [18, 19]. In the following discussion the nodes in the ETKP may be seen as customers who can possibly be serviced. The ETKP may be seen as choosing a subtree from a potential tree that maximises the profit for nodes included in the subtree while satisfying feasible flow constraints.

Given a directed tree $T = (V, E)$ rooted at node 0 where $V = \{0, 1, \ldots, n-1\}$ is the set of nodes and $E$ is the set of potential arcs or links between the nodes. The direction of each arc is assumed to point towards the root node. Also, for each node $j$ we designate the parent node as the first node encountered along the path to the root node. For each node $j \in V$ it is assumed that a positive integer $c_j$ exists representing the profit obtained by servicing node $j$, as well as a positive integer $d_j$ representing the capacity consumed by serving node $j$. The vector $\boldsymbol{x}$ has binary elements $x_j \in \{0, 1\}$ denoting the exclusion or inclusion of a node $j$ in the subtree. Two vital assumptions for the ETKP are:

1. *Indivisible demand*: Each node is either serviced completely, *i.e.* all its demand is serviced or none at all.
2. *Contiguity constraint*: All nodes on the path between a specific node and the root node, must also be included if a node is to be included in the subtree.

The root node has limited capacity for servicing the demand of the customers and this is denoted by a non-negative integer $H$. Each node also generates flow $y_j$ (to the parent node denoted by $p_j$), with an associated cost $f_j\left(y_j\right)$. This cost function may take on several forms. In this paper a specific form of the cable expansion cost function $f_j\left(y_j\right)$, as presented in [16], is used. The expansion cost function is defined as

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

where $b_j$ is the current capacity of the link between nodes $j$ and $p_j$, $a_j$ is the variable cost for flow exceeding $b_j$ and $F_j$ is the fixed cost for sending flow in excess of $b_j$ from node $j$ to $p_j$.

This cost function represents the case where no cost is incurred for sending flow less than or equal to $b_j$, from node $j$ to $p_j$. Sending more than $b_j$ units of flow incurs a fixed expansion cost of $F_j$ and a variable cost of $a_j$ per extra unit of flow. This may be seen as an expansion cost incurred for additional flow. In order to formulate the problem as a *Mixed Integer Linear Program* (MILP), certain modelling changes are implemented to the ETKP model presented in [16].

These modelling changes involve splitting the variables $y_j$ into two parts, such that $y_j = y_{j1} + y_{j2}$, where $y_{j1}$ is the part of the flow less than or equal to the current capacity ($y_{j1} \leq b_j$) and $y_{j2}$ the part of the flow exceeding the current capacity of the link. Secondly, a new set of 0-1 variables $\delta_j$ is introduced, corresponding to the expansion decision for each $j$, with logical constraints of the form $y_{j2} \leq H\delta_j$ in order to force $y_{j2}$ to zero if $\delta_j = 0$. For each $j$, $\delta_j$ corresponds to whether expansion is necessary, *i.e.* $\delta_j = 1$ if expansion costs are incurred at node $j$. Expansion in this case refers to sending flow in excess of the current cable capacity $b_j$.

Define the vectors of flows as $\boldsymbol{y}_1^{\mathrm{T}} = \left[y_{11}, y_{21}, \ldots, y_{(n-1)1}\right]$ and $\boldsymbol{y}_2^{\mathrm{T}} = \left[y_{12,}y_{22,}\ldots, y_{(n-1)2}\right]$ such that $\boldsymbol{y} = \boldsymbol{y}_1 + \boldsymbol{y}_2$. Define $\boldsymbol{D}$ as a diagonal matrix with diagonal elements $d_1, d_2, \ldots,$ $d_{n-1}$ and let $\boldsymbol{B}$ be an $(n-1) \times (n-1)$ incidence matrix of $T$ with the root node row being excluded. By definition, each row in $\boldsymbol{B}$ corresponds to a node and each column corresponds to an arc in the directed tree. This means that the $i$-th column of $\boldsymbol{B}$ has zero entries, except in row $j$ and row $p_j$, which have values of 1 and $-1$ respectively.

The ETKP may now be formulated as a MILP in which the objective is to

$$\text{maximise} \sum_{j=0}^{n-1} c_j x_j - \sum_{j=1}^{n-1} (F_j \delta_j + a_j y_{j2}) \tag{1}$$

subject to the constraints

$$x_j - x_{p_j} \leq 0, \qquad j = 1, 2, \ldots, n-1, \tag{2}$$

$$x_0 = 1, \tag{3}$$

$$\boldsymbol{D}\left[x_1, x_2, \ldots, x_{n-1}\right]^T - \boldsymbol{B}\left(\boldsymbol{y}_1 + \boldsymbol{y}_2\right) = \boldsymbol{0}, \tag{4}$$

$$\sum_{j=0}^{n-1} d_j x_j \leq H, \tag{5}$$

$$0 \leq y_{j2} \leq H\delta_j, \qquad j = 1, 2, \ldots n-1, \tag{6}$$

$$0 \leq y_{j1} \leq b_j, \qquad j = 1, 2, \ldots n-1, \tag{7}$$

$$x_j \in \{0, 1\}, \qquad j = 0, 1, \ldots n-1, \tag{8}$$

$$\delta_j \in \{0, 1\}, \qquad j = 1, 2, \ldots, n-1. \tag{9}$$

The set of constraints (4) ensures conservation of generated flow, where $\boldsymbol{B}$ represents the node-arc incidence matrix. The profit obtained from choosing a subtree may be seen as the profit resulting from the inclusion of nodes, less the cost of flow generated by the nodes chosen.

## 4 General solution strategies

Various solution strategies exist for solving optimization problems, such as standard off-the-shelf software, heuristics, dynamic programming, metaheuristics, *etc.* For the TKP and ETKP models various dynamic programming and branch-and-bound methods are available; see, for example, [4, 5, 9, 15, 16]. Further approaches to the general planning problem is presented in [1].

A depth-first dynamic programming algorithm designed specifically for the ETKP is presented in [16]. Note that the ETKP is represented in a depth-first manner here in order to facilitate implementation of this algorithm. The depth-first dynamic programming algorithm for the ETKP uses recursive rules to build solutions for subtrees and to generate an optimal solution value for the problem in a recursive fashion. A negative aspect is that once an optimal objective function value has been obtained, the optimal solution sub-tree and flow dimensioning is not known. An additional procedure is presented in [16] which may be used to obtain the corresponding optimal solution values for the variables.

## 5 Propossed algorithm

The solution approach proposed here for the ETKP is a method that uses standard off-the-shelf software combined with enhanced modelling and partitioning. The problems are modelled as MILPs. This approach has the following advantages:

- Custom developed algorithms (such as dynamic programming) are seldom readily available for use by other practitioners. Using standard off-the-shelf software will cut down on development times. Here standard software refers to third party software, *i.e.* proprietary or even open source software.
- Many dynamic programming algorithms give no guarantee regarding the quality of solutions obtained in cases where time or storage constraints prevent it from running to completion. The solution process proposed here will create interim feasible solutions with known bounds. This is preferred to methods that only produce solutions upon completion.
- Advances in the standard off-the-shelf software as vendors improve their products, will automatically improve the efficiency of the solution process.

The proposed algorithms produce optimal solutions with the added advantage of intermediate solutions generated during the solution process. These solutions produce bounds on the optimal objective function value and improve the performance of the solution process.

A two-phase partitioning of the search space is performed. The first level of partitioning aims at exploiting the notion of cardinality. The idea of using cardinality is to estimate the number of variables with value one in an optimal solution. This estimate may be used to reduce the search space. Also helpful is the knowledge that the cardinality can only be an integer value. This is called *first-order partitioning* in this paper. This notion of cardinality was advocated in [13, 14].

For a specific first-order partition, the *linear programming* (LP) relaxation is used to identify a set of promising variables. This set is used to implement a *second-order partition.* The overall goal of the partitioning is to solve a number of easier problems, rather than one difficult problem, where the solution times of the easier problems are relatively short.

An algorithm based on these ideas was developed for the TKP and was published in [19, 20]. An analogous algorithm was developed for the ETKP and the details and performance in empirical experiments are reported below.

Let ILP(ETKP) denote a MILP model for the ETKP and let ILPR(ETKP) denote the LP relaxation of the problem. A first-order MILP partitioning constrained to $p$ nodes is denoted by ILP(ETKP, $p$) and the corresponding LP relaxation is denoted by ILPR(ETKP, $p$). A second-order partitioning (defined below) is denoted by ILP(ETKP, $p, q$) for the MILP and by ILPR(ETKP, $p, q$) for the LP relaxation. The second-order partition is based on the observation that if $p$ is a cardinality of a correct first-order partition, and the set of indices $\{0, 1, \ldots, n-1\}$ is partitioned into a "promising" set (called $S_\ell$ in the algorithm below) and a remainder set (referred to as $S_{n-\ell}$), a systematic evaluation assigning $p - q$ variables $x_j$ (with $j \in S_\ell$,) the value of 1 and $q$ variables $x_i$ ($i \in S_{n-\ell}$) the value of 1 for all non-negative integer values of $q$ will solve the problem if $q$ is systematically enumerated. Solving the resulting problems ILP(ETKP, $p, q$), the optimal solution to ILP(ETKP, $p$), and by assumption ILP(ETKP), must be obtained as the best value of ILP(ETKP, $p, q$) over the range of $q$. The algorithm is given in pseudo-code form in Algorithm 1.

**Algorithm 1** Partitioning Algorithm

```
1: procedure PART_ETKP
2:     $CLB \leftarrow 0$
3:     $x_j^{CLB} \leftarrow 0,\ j \in \{0,1,\ldots,n-1\}$
4:     $\delta_i^{CLB} \leftarrow 0,\ i \in \{1,2,\ldots,n-1\}$
5:     $P \leftarrow \{1,2,\ldots,n-1\}$
6:     Solve parametrically ILPR(ETKP, $p$) $\forall p \in P$, obtain objective function values $Z_{ETKPR(p)}$. If
       ILPR(ETKP$p$) is infeasible, $Z_{ETKPR(p)} \leftarrow -\infty$.
7:     while $P \neq \emptyset$ do
8:         $p' \leftarrow t$ where $t$ is defined by $Z_{ETKPR(t)} = \max_k \{Z_{ETKPR(k)} | k \in P\}$
9:         For $Z_{ETKP(p')}$ identify the solution values $x'_j$ for $j \in V$
10:        $S_l \leftarrow \{j | x'_j = 1, j = 0,1,2,\ldots,n-1\}, S_{n-\ell} \leftarrow V \backslash S_\ell$
11:        $Q \leftarrow \{|S_\ell|, |S_\ell|-1, |S_\ell|-2, \ldots, \max\{0, p-|S_{n-\ell}|\}\}$
12:        while $Q \neq \emptyset$ do
13:            Solve parametrically ILPR($ETKP, p', q$) $\forall\ q \in Q$, obtain objective function values of
               $Z_{ETKPR(p',q)}$
14:            If ILPR(ETKP,$p', q$) is infeasible, $Z_{ETKPR(p',q)} \leftarrow -\infty$
15:            $q' \leftarrow s$ where $s$ is defined by $Z_{ETKPR(p',s)} = \max_j \{Z_{ETKPR(p',j)} | j \in Q\}$
16:            if $Z_{ETKPR(p',q')} > CLB$ then
17:                Solve ILP(ETKP, $p', q'$), optimal objective function value $Z_{ETKP(p',q')}$ with solution
                   values $x'_j, j = 0,1,\ldots,n-1$ and $\delta'_i, i = 1,2,\ldots,n-1$
18:                if $Z_{ETKP(p',q')} > CLB$ then
19:                    $CLB \leftarrow Z_{ETKP(p',q')}$
20:                    $x_j^{CLB} \leftarrow x'_j,\ j = 0,1,\ldots,n-1$
21:                    $\delta_i^{CLB} \leftarrow \delta'_i,\ i = 1,2,\ldots,n-1$
22:                    $Q \leftarrow \{r | r \in Q$ and $Z_{ETKPR(p',r)} > CLB\}$
23:                end if                                         ▷ $Z_{ETKP(p',q')} > CLB$
24:            end if                                             ▷ $Z_{ETKPR(p',q')} > CLB$
25:            if $q' \in Q$ then
26:                $Q \leftarrow Q \backslash \{q'\}$
27:            end if
28:            $P \leftarrow P \backslash \{p'\}$
29:            $P \leftarrow \{i | i \in P$ and $z_{ETKPR(i)} > CLB\}$
30:        end while                                              ▷ $Q \neq \emptyset$
31:    end while                                                  ▷ $P \neq \emptyset$
32:    return $CLB, x_j^{CLB}$ and $\delta_i^{CLB}, j = 0,1,\ldots,n-1$ and $i = 1,2,\ldots,n-1$
33: end procedure
```

The ETKP partitioning algorithm initially produced poor computational results. This was due to large integrality gaps and subsequently a large number of second-order partitions to investigate. Here the term integrality gap refers to the difference between the linear programming relaxation and integer valued solutions. To enhance the computational efficiency, several additional ideas were implemented.

The first such idea was to add valid inequalities to the formulation. The notion of valid inequalities is discussed in more detail in [10, 11]. Adding valid inequalities reduces the integrality gap and limits the number of second-order partitions to investigate, improving solution times dramatically. The valid inequalities investigated in this paper are shown below. The procedures presented below add constraints to the formulation during run time, depending on the data instance, and is presented in pseudo-code form as Algorithms 2–4.

In Algorithm 2 the first set of valid inequalities aims to exploit the fact that capacity is used up on a path between a node and the root node. This, in combination with contiguity constraints, forces capacity expansion higher up in the network if a node is added.

**Algorithm 2** Add first set of valid inequalities to ETKP

```
1: procedure AddInequalities 1
2:     for $i = 1, 2, \ldots, n-1$ do
3:         $j \leftarrow i$
4:         $path\_capacity \leftarrow 0$
5:         while $j \neq 0$ do
6:             $path\_capacity = path\_capacity + d_j$
7:             if $path\_capacity \geq b_j$ then
8:                 Add constraint $x_i \leq \delta_j$
9:             end if
10:            $j \leftarrow p_j$
11:        end while
12:    end for
13: end procedure
```

The second type of valid inequality (embodied in Algorithm 3) aims to exploit the capacity required when adding all the child nodes of a specific node. If the sum of the capacities of the child nodes of node $i$ exceeds the capacity $b_i$ available at node $i$, expansion would be required before all the child nodes can be added.

**Algorithm 3** Add second set of valid inequalities to ETKP

```
1: procedure AddInequalities 2
2:     for $i = 1, 2, \ldots, n-1$ do
3:         $C_i \leftarrow \{j | p_j = i\}$
4:         if $C_i \neq \emptyset$ then
5:             $sum\_demands \leftarrow d_i + \sum_{k \in C_i} d_k$
6:             if $sum\_demands \geq b_i$ then
7:                 Add constraint $x_i + \sum_{j \in C_i} x_j - \delta_i \leq |C_i|$
8:             end if
9:         end if
10:    end for
11: end procedure
```

Before the next type of valid inequality is discussed it is necessary to introduce additional notation. Let $P(i, j)$ refer to the set of nodes on the path between node $i$ and $j$, excluding nodes $i$ and $j$ in the set.

The third type of valid inequality (contained in Algorithm 4) involves the capacity used up on the path between the root node and the node for which the inequality is added, and limits the additional flow $y_{i2}$.

**Algorithm 4** Add third set of valid inequalities to ETKP

```
1: procedure AddInequalities 3
2:     for $i = 1, 2, \ldots, n-1$ do
3:         Add constraint $y_{i2} \leq \left(H - \sum_{j \in P(0,i)} d_j - b_i\right) \delta_i$
4:     end for
5: end procedure
```

A fourth enhancement is due to the fact that the capacity $H$ used in the inequality $y_{i2} \leq H\delta_i$ results in a LP relaxation solution that is too optimistic and contributes to a relativity large integrality gap. It was found that adding the valid inequality

$$y_{i2} \leq \left(\sum\nolimits_{j\in T(i)} d_j\right)\delta_i, \tag{10}$$

where $T(j) = \{i|i \text{ is a descendant of } j\} \cup \{j\}$, decreases the integrality gap. This inequality implies that $y_{i2} = 0$ if $\delta_i = 0$. If $\delta_i = 1$ an upper bound on $y_{i1} + y_{i2}$ is $\sum_{j\in T(i)} d_j$. This means that the inequality in (10) is valid.

## Additional heuristic method

In the algorithms developed, a basic step is to choose a first-order partition with a certain cardinality to be investigated further. This is done on the basis of the objective function value obtained from a relaxed problem for that cardinality. In certain instances more than one partition (cardinality) gives the same objective function value. A heuristic is given below that differentiates between different first-order partitions. The heuristic also provides good starting solutions that are valid for the overall problem. It produces good bounds and in some cases solutions are obtained that are proved to be optimal by the bounds generated.

The heuristic is based on the observation that the variables with solution values of $x_j = 1$ in the relaxed problem define a subtree. Denote this subtree by $T_{B_\ell}$. This is due to the contiguity constraints present in the formulations of both the TKP and ETKP models. The idea of the heuristic is to focus on child nodes (not in the subtree) of nodes in the subtree $T_{B_\ell}$. The subtree $T_{B_\ell}$ uses up a certain capacity, but also gives a certain profit. In the case of the TKP, a simple 0-1 KP may be constructed using as variables the direct child nodes (that are not in the subtree) of the subtree $T_{B_\ell}$. Suppose that $T_{B_\ell}$ depletes a capacity of $H_{B_\ell}$, then the new 0-1 KP created has a residual capacity of $H - H_{B_\ell}$. The nodes identified by the solution to the 0-1 KP are then added to $T_{B_\ell}$ to form another extended subtree that may be used as a feasible solution to the TKP. The optimal objective function value for the 0-1 KP must be added to the profit obtained for $T_{B_\ell}$ in order to obtain an objective function value for the original TKP problem. The empirical experiments with this heuristic indicated that the 0-1 Knapsack constructed has a relatively small capacity and is relatively easy to solve. More details on this heuristic for the TKP case are given in [18]. In this paper the focus is more on the ETKP.

The same type of heuristic cannot be applied directly to ETKP. This is due to the flow constraints that are present in the problem formulation, which means that the profit obtained from using a 0-1 KP created by the child nodes of a valid subtree cannot simply be added to the profit of the subtree. Feasible flows and flow constraints have to be accounted for. However, a similar type of heuristic is applicable for the ETKP case that utilises the same idea as discussed previously, but overcomes the problems with flow conservation and subsequent costs. To implement this heuristic, the objective function of the original problem must be used. New knapsack type constraints are introduced using the child nodes of a subtree with limited capacity, but keeping the formulation of the original problem.

This heuristic method was implemented after solving an LP relaxation ILPR(ETKP) to obtain solution values $x_j^*$ for $j = 0, 1, \ldots, n-1$. The set $B_\ell$ is formally defined as the set of nodes included (with $x_j^*$ values of 1) in the solution of ILPR(ETKP). Define the set $B_{Adl} = \{j \mid p_j \in B_\ell, j \notin B_\ell\}$. The capacity depleted by the subtree $T_{B_\ell}$ (corresponding to $B_\ell$) is $H_{B_\ell} = \sum_{j \in B_\ell} d_j$. The capacity available for the Knapsack type heuristic is $H - H_{B_\ell}$. The ILP heuristic ILP(ETKP,$H_{B_\ell}$) is then the problem of

$$\text{maximising} \sum_{j=0}^{n-1} c_j x_j - \sum_{j=1}^{n-1} (F_j \delta_j + a_j y_{j2}) \tag{11}$$

subject to the constraints

$$\begin{aligned}
x_j - x_{p_j} &\le 0 & j &= 1, 2, \ldots, n-1, & (12)\\
x_0 &= 1, & & & (13)\\
\boldsymbol{D}\left[x_1, x_2, \ldots, x_{n-1}\right]^T - \boldsymbol{B}\left(\boldsymbol{y}_1 + \boldsymbol{y}_2\right) &= \boldsymbol{0}, & & & (14)\\
0 \le y_{j2} &\le H\delta_j, & j &= 1, 2, \ldots, n-1, & (15)\\
0 \le y_{j1} &\le b_j, & j &= 1, 2, \ldots n-1, & (16)\\
x_j &= 1, & j &\in B_\ell, & (17)\\
x_i &= 0, & i &\in V \backslash \left(B_\ell \cup B_{Adl}\right), & (18)\\
\sum_{j \in B_{Adl}} x_j d_j &\le H - H_{B_\ell}, & & & (19)\\
x_j &\in \{0, 1\}, & j &= 0, 1, \ldots n-1, & (20)\\
\delta_j &\in \{0, 1\}, & j &= 1, 2, \ldots, n-1. & (21)
\end{aligned}$$

This heuristic was integrated into Algorithm 1 given above and the results are presented in the following section.

## 6 Results

The results presented in this paper are divided into two parts. The first part constitutes additional results to the empirical results presented in [19]. Complexity classes are considered similar to those of the 0-1 KP presented in [14], where degrees of correlation between the profit and demand coefficients are considered. The experience reported in [14] indicate that the computational difficulty of solving 0-1 KPs often depends on the interrelationship between the profit and demand coefficients. The performances of the proposed algorithms are reported for these cases. Data instances were generated using a pseudo-random number generator.

As both TKP and ETKP aim to choose a subtree from a potential tree, data instances for these trees are generated in a systematic manner. After a valid candidate tree was created, profit values ($c_j$) and demand values ($d_j$) for each node were generated. For the ETKP the existing node capacity ($b_j$), fixed cost ($F_j$) and variable cost for expansion ($a_j$) were also generated. These values were generated to fall within specified ranges. This allowed various classes of data instances to be implemented for each tree setup.

Four data classes were investigated, given in increasing order of perceived complexity:

- *Uncorrelated problems.* In this situation there is no direct correlation between the profit and demand of a specific node. This means that $c_j \in \{0, 1, \ldots, \mathit{UpperLevel}\}$ and $d_j \in \{0, 1, \ldots, \mathit{UpperLevel}\}$.
- *Weakly correlated problems.* Here a weak correlation exists between the profit and demand of a specific node. The implementation implies that a demand is chosen, say $d_j \in \{0, 1, \ldots, \mathit{UpperLevel}\}$. The profit for the node is then generated to have some correlation to the demand. This is achieved by generating a random integer value, say $r \in [-\mathit{Interval}, +\mathit{Interval}]$ and adding this to the value $d_j$, *i.e.* $c_j = d_j + r$. If a negative value is generated for $c_j$, the $c_j$ value is set to one.
- *Strongly correlated problems.* Here a strong correlation exists between the profit and demand of a specific node. In practice this was implemented in the following way. A demand $d_j$ was randomly chosen from the set $\{0, 1, \ldots, \mathit{UpperLevel}\}$. The profit $c_j$ was then calculated as $c_j = d_j + r$, where $r$ is a non-negative integer value chosen from a narrow band.
- *Subset sum problems.* Here the profit and demand is the same for each node. This means that $c_j = d_j, j = 0, 1, \ldots, n-1$. In practice this means that $d_j$ is randomly generated from the set $\{0, 1, \ldots, \mathit{UpperLevel}\}$. Afterwards, the value of $c_j$ is set to the same value as $d_j$ for all the nodes in the problem instance. This type of data instance is sometimes referred to as the *value-independent case* [3].

The values of *UpperLevel*, *Interval*, $r$, the branching factor and an initialisation (seed) value for the pseudo random number generator have been stored. This gives all the necessary information required to recreate a specific data instance. It was only necessary to start the tree generation process with the correct seed value.

## 6.1 TKP results

The results obtained for the TKP case are presented in this section. For each progressively larger tree size, four tree configurations were used. For each tree configuration, nine different total capacities ($H$) were used. This means that for each data point displayed in the Figures 1–4 below, the average corresponding to 36 experiments is given. The partitioning algorithm presented above was benchmarked against standard software (CPLEX in this case) and against an implementation of a branch-and-bound algorithm presented in [15], referred to as *Shaw and Cho [15]* in the figures below. The branch-and-bound algorithm developed was implemented from pseudo-code. A dynamic programming algorithm for the TKP is presented in [4], but according to the authors of [4] the branch-and-bound version is superior. This is the motivation for comparisons with the branch-and-bound version presented here.

For the TKP empirical tests a Pentium IV 3.2 GHz with 1Gb of memory running SuSE 10.1 was used. CPLEX 10.1 from ILOG with the GNU C++ 4.3 compiler was used to solve all MILP and LP problems.

Graphical representations of the relative performance of the different algorithms for the different classes of data instances are given in Figures 1–4.

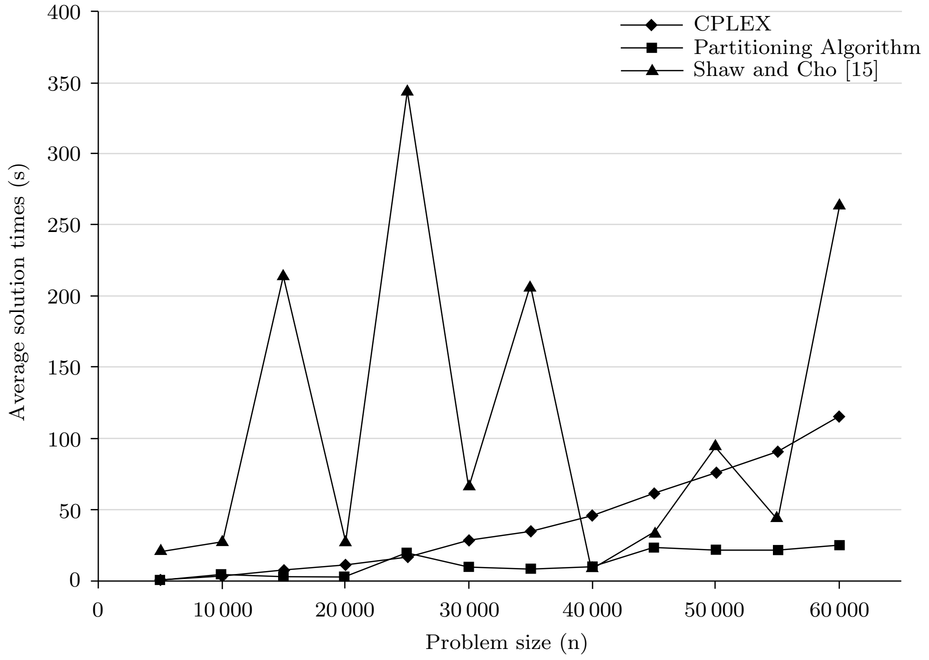

**Figure 1:** *Average solution times for uncorrelated TKP data class.*

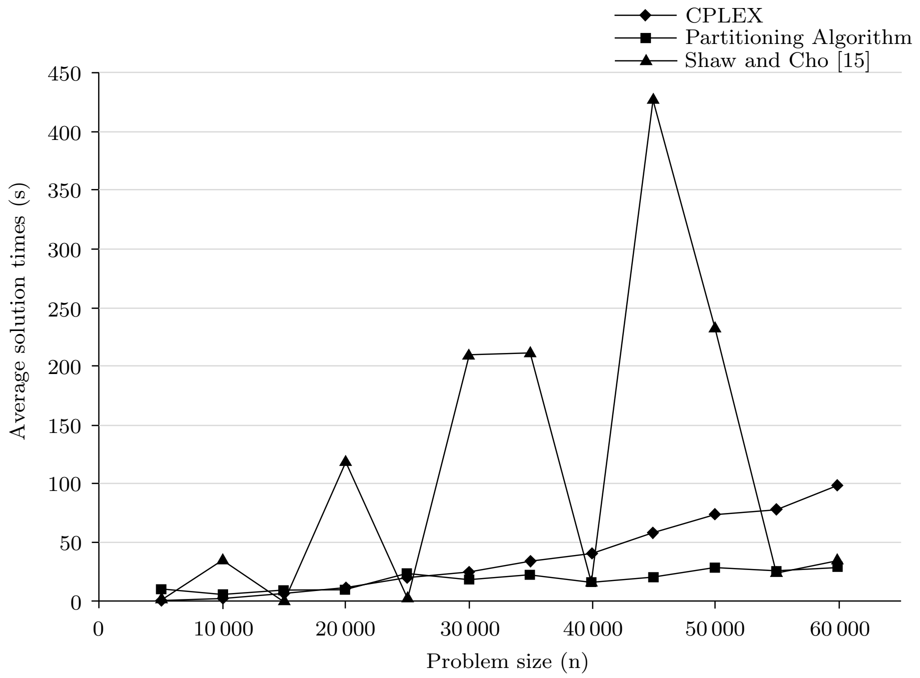

**Figure 2:** *Average solution times for the weakly correlated TKP data class.*

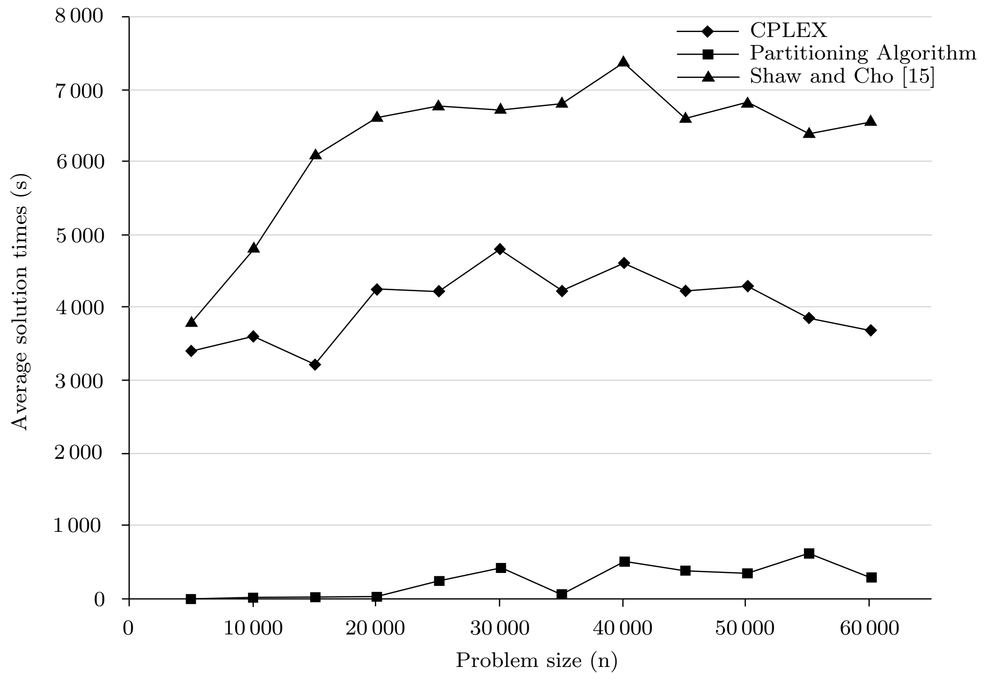

**Figure 3:** *Average solution times for the strongly correlated TKP data class.*

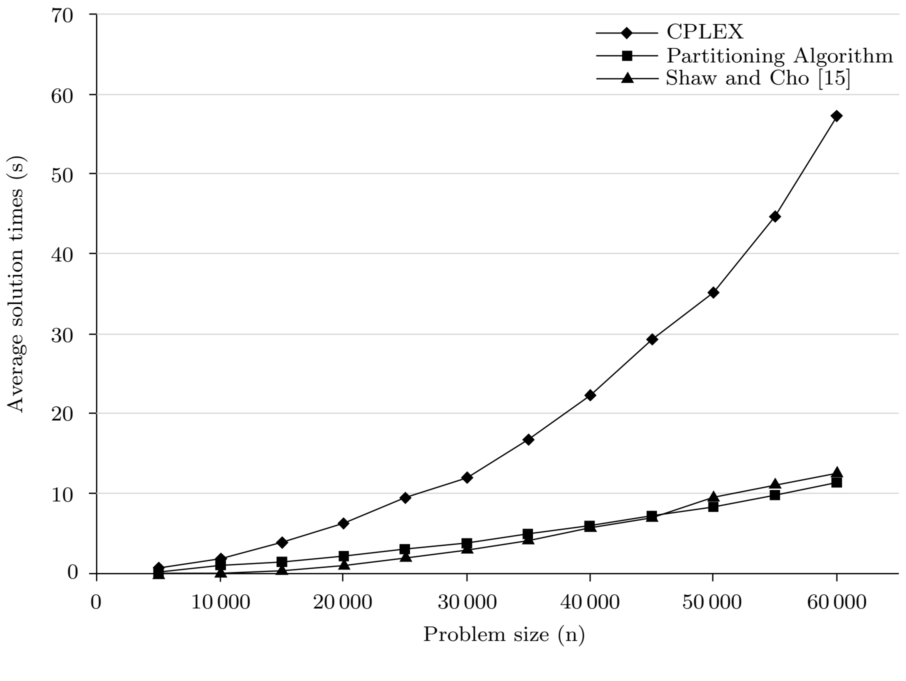

**Figure 4:** *Average solution times for the subset sum data TKP class.*

## 6.2 Conclusions for the TKP results

The empirical results demonstrate that the partitioning TKP algorithm is capable of performing quite well on all the different types of data described above. In general the partitioning algorithm solved problem instances in less time, on average, than the standard software alone or the algorithm in [15]. The algorithm published in [15] 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 of 7 200 seconds. This is partly true for the standard software as well.

In general it seems that 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. For the strongly correlated data class and larger problem sizes (more than 60 000 nodes) CPLEX and the algorithm in [15] often failed to solve the problem instances. 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 the types of data classes considered. This means that if no information is available on the nature of the data of the problems to be investigated, using the partitioning algorithm should be a good strategy. The algorithm in [15] was often outperformed by the standard CPLEX software alone and some problem instances were not solved due the memory or time constraints, meaning that it was intractable to solve the problem on the machine used.

The empirical experience gained here with the complexity of the data instances largely corresponds to that reported for the complexity classes of the 0-1 KP, with the exception of the subset sum case. Both the proposed algorithm and the algorithm in [15] solved this class surprisingly fast.

## 6.3 ETKP results

Computational experiments were performed using CPLEX from ILOG as the standard off-the-shelf optimization software. This software was used to solve all linear-, mixed integer linear- and integer programming problems. This also presented a benchmark for the enhanced modelling and partitioning algorithm. Since numerous errors were found in the algorithm presented in [16] this algorithm was not implemented and could not be tested for computational efficiency.

Problems were generated using a pseudo-random number generator similar to the process discussed previously. Problem sizes measured in number of nodes ranged between 500 and 10 000. For the empirical results time cut-off values of 7 200 seconds were imposed for the solution times of each instance. For each problem size four different tree configurations were again generated and within each configuration nine different capacity sets were used. Progressively larger capacities were used for larger problems. This is in contrast to work presented in [16] where the capacities of the problem instances were kept fixed even when the problem size was increased. It is felt that this practice may have led to a misrepresentation of the actual performance of the algorithm.

Without the added valid inequalities the partitioning algorithm did not produce viable solution times to problem sizes with more than 300 nodes. In the empirical work reported here the partitioning method with valid inequalities and the heuristic was employed. Much larger sized problems could be accommodated with these enhancements.

For the empirical work an AMD DL145G2 OPTERON 64 bit running RedHat Linux was used. The machine had 4Gb of RAM and CPLEX 10.1 was used.

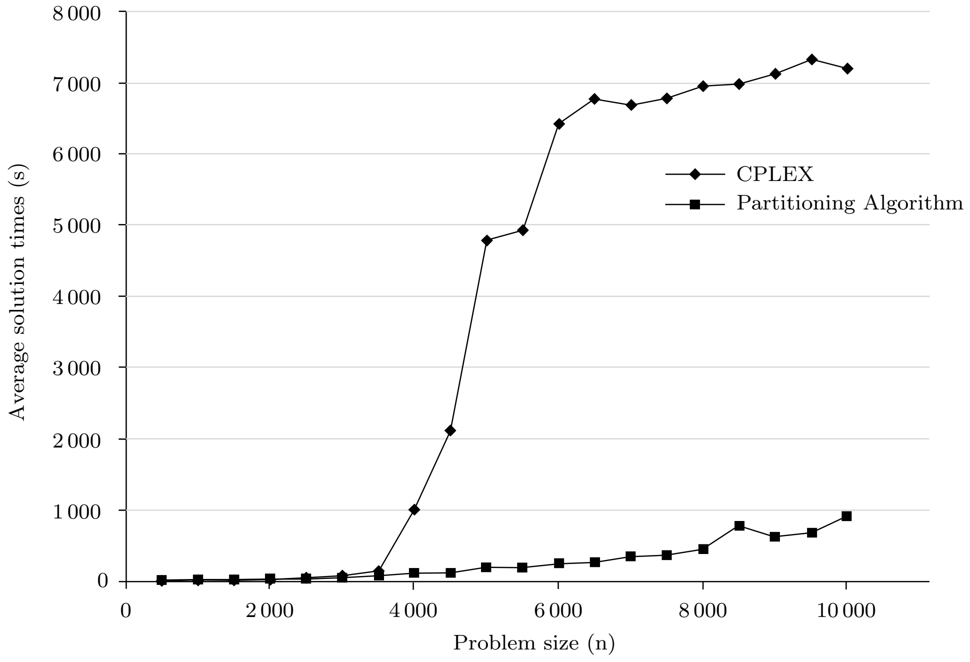

**Figure 5:** *Average solution times for the patitioning algorithm and CPLEX.*

The standard deviations of the solution times for the various problem sizes are shown in Table 1. It may be seen from the graph in Figure 5 that, on average, the partitioning algorithm solved ETKP data instances in less time than did standard CPLEX. All solutions obtained were optimal solutions. It is furthermore evident from Figure 5 that the average CPLEX solution times were close to the cut-off value of 7 200 seconds for larger problems, implying that CPLEX could not reliably solve all the data instances.

Generally the standard deviations obtained for the partitioning algorithm were also less than the deviations of CPLEX, shown in Table 1. This means that the solution times were less variable for the partitioning algorithm than for CPLEX. The exception was for some of the larger problem sizes. This is due to fact that in many cases CPLEX reached the cut-off value. This led to small variations in solution times, but reaching the cut-off time is not a desirable outcome, as the procedure is stopped before a proven optimal solution is obtained.

## 7 Conclusion

The partitioning TKP algorithm performed very well when tested against the other solution approaches investigated in this paper. Generally, complexity classes identified for the 0-1 KP also hold for the TKP in the experience gained here. The partitioning algorithm

| Problem size | CPLEX | Partitioning algorithm |
|---:|---:|---:|
| 500 | 1.1 | 0.9 |
| 1 000 | 5.2 | 2.3 |
| 1 500 | 7.1 | 6.8 |
| 2 000 | 10.2 | 10.6 |
| 2 500 | 21.6 | 17.2 |
| 3 000 | 72.1 | 23.9 |
| 3 500 | 140.1 | 27.3 |
| 4 000 | 1 773.9 | 45.6 |
| 4 500 | 2 713.8 | 59.6 |
| 5 000 | 2 845.6 | 135.7 |
| 5 500 | 3 038.5 | 95.2 |
| 6 000 | 2 307.9 | 119.2 |
| 6 500 | 1 861.8 | 129.6 |
| 7 000 | 1 987.1 | 244.6 |
| 7 500 | 1 849.9 | 238.3 |
| 8 000 | 1 643.2 | 318.4 |
| 8 500 | 1 342.2 | 1 174.5 |
| 9 000 | 896.3 | 352.9 |
| 9 500 | 23.5 | 374.2 |
| 10 000 | 796.8 | 752.1 |

**Table 1:** *Standard deviations of the solution times for the partitioning algorithm and CPLEX.*

appears to be the best strategy to use if nothing is known about the complexity class of a TKP problem investigated.

Solving ETKP problems with a partitioning algorithm that uses standard off-the-shelf software employing the enhanced modelling techniques described in this paper, appears to be a viable solution strategy. This may be very helpful as the ETKP is a basic subproblem in LATN design problems. A major conclusion is that the empirical experience with the new algorithms indicates that large problems with more than 6 000 nodes can be reliably solved, whereas the other methods tested failed to do so in reasonable times.

## 8 Future work

It is suggested that further future work is done to investigate different cost functions and to test whether the approach of using standard software with this type of enhanced modelling and partitioning strategies can be used in other models for LATN design. Parallel computing was not utilised during the computational tests, but may be considered in future work as the algorithms developed have inherent parallelisation possibilities.

## References

[1] Balakrishnan A, Magnanti TL, Shulman A & Wong RT, 1991, *Models for planning capacity expansion in local access telecommunication networks*, Annals of Operations Research, **33**, pp. 239–284.

[2] Balakrishnan A, Magnanti TL & Wong RT, 1995, *A decomposition algorithm for local access telecommunications network expansion planning*, Operations Research, **43**, pp. 58–76.

[3] Balas E & Zemel E, 1980, *An algorithm for large 0-1 knapsack problems*, Operations Research, **28**, pp. 1130–1154.

[4] Cho G & Shaw DX, 1997, *A depth-first dynamic programming algorithm for the tree knapsack problem*, INFORMS Journal on Computing, **9**, pp. 431–438.

[5] Cho G, Shaw DX & Kim S, 1997, *An efficient algorithm for a capacitated subtree of a tree problem in local access telecommunications networks*, Computers and Operations Research, **8**, pp. 737–748.

[6] Corte-Real M & Gouveia L, 2007, *Network flow models for the local access network expansion problem*, Computers and Operations Research, **34**, pp. 1141–1157.

[7] Flippo OE, Kolen AWJ, Koster AMCA & Van de Leensel RLMJ, 2000, *A dynamic programming algorithm for the local access telecommunication network expansion problem*, European Journal of Operational Research, **127**, pp. 189–202.

[8] Godor I & Magyor G, 2005, *Cost-optimal topology planning of hierarchical access networks*, Computers and Operations Research, **32,** pp. 59–86.

[9] Johnson DS & Niemi KA, 1983, *On knapsacks, partitions and a new dynamic programming technique for trees*, Mathematics of Operations Research, **8**, pp. 1–14.

[10] Johnson EL & Nemhauser GL, 1992, *Recent developments and future directions in mathematical programming*, IBM Systems Journal, **33(1)**, pp. 79–93.

[11] Linderoth JT & Savelsbergh MW, 1999, *A computational study of search strategies for mixed integer programming*, INFORMS Journal on Computing, **11(2)**, pp. 173–187.

[12] Martello S & Toth P, 1987, *Algorithms for knapsack problems*, Annals of Discrete Mathematics, **31**, pp. 213–258.

[13] Martello S & Toth P, 1997, *Upper bounds and algorithms for hard 0-1 knapsack problems*, Operations Research, **45**, pp. 768–778.

[14] Pisinger D, 1995, *Algorithms for knapsack problems*, PhD Dissertation, University of Copenhagen, Copenhagen.

[15] Shaw DX & Cho G, 1996, *The critical-item, upper bounds, and a branch-and bound algorithm for the tree knapsack problem*, Networks, **31**, pp. 205–216.

[16] Shaw DX, Cho G & Chang H, 1997, *A depth-first dynamic programming procedure for the extended tree knapsack problem in local access network design*, Telecommunication Systems, **7**, pp. 29–43.

[17] Sherali HD, Lee Y & Park T, 2000, *New modeling approaches for the design of local access transport networks*, European Journal of Operational Research, **127**, pp 94–108.

[18] Van der Merwe DJ, 2007, *The use of partitioning strategies in local access telecommunication network problems and other applications*, PhD Dissertation, Department of Computer Science and Information Systems, North-west University, Potchefstoom.

[19] Van der Merwe DJ & Hattingh JM, 2005, *The feasibility of LATN design using tree knapsack and extended tree knapsack models*, Proceedings of Southern African Telecommunication Network and Application Conference (SATNAC) 2005, Telkom, pp. 189–194.

[20] Van der Merwe DJ & Hattingh JM, 2006, *Tree knapsack approaches for local access network design*, European Journal of Operational Research, **174**, pp 1968–1978.