Abstract - In the process of solving many forms of the Local Access Network Design problem, the basic model of the Extended Tree Knapsack Problem (ETKP) is used as a building block for the search engine of the solution strategy. The ETKP is a natural extension to the Tree Knapsack Problem (TKP). The ETKP can be solved using standard computing facilities and off the shelf optimisation software like OSL from IBM, CPLEX from ILOG, MINOS from Stanford Optimization Laboratory and others. For large design problems this approach is not satisfactory in terms of solution time. Various authors have suggested tailor made algorithms and code that often produce results faster. Access to these codes is often problematic. We consider a modelling approach to the problem that mainly utilizes standard software. The idea is to add value to standard optimisation software by utilising enhanced modelling, which is directly portable to newer versions of the software, while still providing exact solutions.

I. BACKGROUND

A. Local Access Telecommunication Network

As an introduction to this class of problems we quote Balakrishnan et al [1]: “Because modernizing and expanding switching and transmission facilities requires enormous investments, telephone companies emphasize cost effectiveness in implementing selected expansion problems with high demand growth potential. For each project, network planners face complex choices concerning where and when to expand capacity or replace current technology in order to meet the increasing demand for different types of services. The emergence of new communication technologies has created additional decision alternatives and tradeoffs and, hence, new modelling challenges that did not arise in the traditional analog and copper environment. For instance deploying concentrators and multiplexes in the local access network now provides an alternative means (instead of cable expansion) to increase network capacity. Consequently, network planners require new decision support models to identify cost effective expansion and modernization strategies.”

Shaw [4] indicate the structure of the Local Access Telecommunication Network (LATN) design problem. We quote them below: “Most existing LATN’s have a tree structure. Each customer node associates with a demand representing the required number of circuits from that node to the switching centre. This demand can be satisfied by either connecting the node through a cable along the unique path to the switching centre, or routing it first to a concentrator, which compresses the incoming traffic into a higher frequency that requires fewer outgoing lines. The objective of the LATN Design and Expansion problem is to make a trade-off between cable expansion and concentrator installation to minimize total cost.”

A very important subproblem in the LATN design problem can be modelled as a so-called Tree Knapsack Problem (TKP), see Shaw (1994). Previous work done, van der Merwe [6] investigated this problem and solution strategies for it.

The extended tree knapsack (ETKP) is a generalized version of the TKP, where traffic cost is also imposed. The traffic flow cost function can be used as indicative of the “cable expansion” cost. This model will be discussed in this paper, as an extension to the TKP.

B. Tree Knapsack Problems

Suppose there is an undirected tree \( T = (V,E) \) rooted at node 0 where \( V = \{0,1,2, ..., n\} \). The tree can be labelled in either depth or breadth first fashion. We assume it is labelled in breadth first fashion, (from left to right). For each node \( i \in V \) there exists an integer \( c_i \) representing its profit, and a nonnegative integer \( d_i \) representing the demand at node \( i \). Let \( p_i \) be the predecessor of node \( i \). Let \( H \) be the given capacity, that is the capacity of the knapsack. The Tree Knapsack
Problem (TKP) is to find the subtree $T' = (V', E')$ of $T$ rooted at node $0$ such that

$$\sum_{i \in V'} d_i \leq H$$

and

$$\sum_{i \in V'} c_i$$

is maximized.

Let $x_i = \begin{cases} 1 & \text{if } i \text{ is chosen} \\ 0 & \text{otherwise.} \end{cases}$

The Tree Knapsack problem (TKP) can be formulated as the following integer linear programming problem:

$$\max \sum_{j=0}^{n} c_j x_j$$

s.t.

$$x_{p_i} \geq x_j, \quad j = 1, 2, \ldots, n$$

$$\sum_{j=0}^{n} d_j x_j \leq H$$

for $x_j \in \{0,1\}$ where $j = 0, 1, \ldots, n$.

We may assume that $d_j \leq H$ for all $j = 0, 1, 2, \ldots, n$ and that

$$\sum_{j=0}^{n} d_j > H.$$

We will denote this problem by ILP(TKP).

For the TKP an indivisible demand assumption is enforced, meaning that a node is selected and its demand is fully served, or alternatively it is rejected. A so-called contiguity assumption is also enforced, stating that if a certain node is served, all the node on the path between the node and the root node must also be served.

The problem can be viewed as choosing a subtree of the given tree such that the profit is maximized while the capacity constraint is not violated.

We assume that $c_j$, $d_j$ and $H$ are positive integers (this assumption can be relaxed resulting in slight changes in the algorithms we suggest below).

### C. Extended Tree Knapsack

In this model, which can be seen as a natural extension to the TKP, a cost is incurred when flow $y_i$ is transmitted from a node $i$ to its predecessor $p_i$.

In this study the cost incurred can be seen as the cable expansion cost. To represent the cable expansion cost, a function $f_i$ where $f_i(0) = 0$ is used. Although many general classes of functions $f_i$ could be considered, we have a special interest in functions $f_i$ in the class below. Let

$$f_i(y_i) = \begin{cases} 0, & \text{if } y_i \leq b_i \\ F_i + a_i (y_i - b_i), & \text{otherwise,} \end{cases}$$

where $b_i$ is the current capacity of the link between node $i$ and its predecessor, node $p_i$.

$F_i$ is a fixed cost incurred if the capacity of a link is expanded and $a_i$ is the marginal cost incurred when expanding capacity.

A graphic representation of function $f_i$ can be seen in Figure 1.

![Graphical representation of function $f_i$](image)

Note that $f(0) = 0$ and that if $b_i = 0$, $i = 0, 1, \ldots, n$ that the problem becomes a network design and not an expansion problem, since no link has any capacity.

Mathematically the ETKP can be formulated with basically the same notation as the TKP. Let $T = (V, E)$ be an undirected tree, rooted at node $0$, where the nodes are labelled in a depth first manner. Define $T' = (V, A)$, where $A = \{p_i, i \mid i \in V\}$, alternatively $A$ is the set of directed arcs from nodes to their immediate child nodes.

Define $B$ as an $n \times n$ node-arc incidence matrix for $T'$, excluding the row corresponding to the root node. The matrix $B$ is constructed in the following way: the $i$-th column of the matrix consist of zeros except in row $i$, which contains a $1$ and a $-1$ in row $p_i$ (except where $p_i = 0$). The necessity for this representation will be explained later.

The indivisible demand and contiguity assumptions are also enforced for the ETKP.

Assuming the ETKP has capacity $H$, it can now be mathematically formulated as follows:

$$\max \sum_{j=1}^{n} c_j x_j - \sum_{j=1}^{n} f_j(y_j)$$
\[ \text{s.t. } x_j - x_{p_j} \leq 0 \quad \forall j \neq 0 \]
\[ x_0 = 1 \]
\[ Dx - By = 0, \]
\[ d^T x \leq H, \]
\[ y_j \geq 0, \quad j = 0, 1, \ldots, n \]
\[ x_j \in \{0,1\}. \]

where \( c_j \) is the profit realised by including node \( j \), \( f_j \) is the cost function described above, \( d_j \) is the demand of node \( j \), \( D = \text{diag}(d_j) \) \( j=1,2,\ldots,n \), \( x = (x_1,x_2,\ldots,x_n) \) and \( d=(d_1,d_2,\ldots,d_n) \).

The constraint \( Dx - By = 0 \) can be explained by noting that the constraint set ensures that the flow into a node minus the flow out of a node is equal to the demand absorbed by the node. For example assume that the successor nodes of node \( i \) belongs to set \( S \), then \( \sum_{k \in S} d_{ij} - \sum_{k \in S} d_{kj} = 0 \).

The model investigated varies slightly from the model used by Shaw et al. (1997), in that the constraint \( d^T x = H \) is preferred to the constraint used by Shaw et al. which was \( dx = 0 \). Their constraint may lead to strange behaviour of the model for certain data instances and possible sub optimal solutions.

D. Modelling the ETKP

Looking at the ETKP, it is evident that the model contains a function \( f_i \) that is non-linear. Such models may be hard to solve with standard software. We thus decided to look at a class of functions \( f_i \) of the form displayed in Figure 1. This class leads to an optimisation model that is an Integer Linear program as shown below.

Firstly the \( y_i \) variables is divided into two parts, such that \( y_i = y_{i1} + y_{i2} \), where \( y_{i1} \) is the part of flow less than the current capacity ( \( y_{i2} \leq b_i \) ) and \( y_{i2} \) the part of the flow greater than the current capacity of the link.

Secondly, introduce new 0-1 variables, \( \delta_i \) \( i=0,1,\ldots,n \) where
\[ \delta_i = \begin{cases} 0 & \text{if } y_{i2} = 0 \\ 1 & \text{if } y_{i2} > 0 \end{cases} \]

These variables are used to ensure that the cost function is correctly computed by introducing the following constraints: \( y_{i2} \leq H\delta_i \).

Define \( y_1=(y_{11}, y_{12}, \ldots, y_{1n}) \) and \( y_2=(y_{11}, y_{12}, \ldots, y_{2n}) \).

The ETKP is now be formulated as following:

\[
\max \sum_{j=0}^n c_j x_j - \sum_{j=0}^n (F_j \delta_i + \alpha_{ij}) \\
\text{s.t. } x_j - x_{p_j} \leq 0 \quad j = 1, 2, \\
x_0 = 1 \\
Dx - B(y_1 + y_2) = 0, \\
d^T x \leq H, \\
y_{i2} \leq H\delta_i \\
0 \leq y_{j1} \leq b_j \quad j = 0, 1, \ldots, n, \\
y_{j1}, y_{j2} \geq 0 \quad j = 0, 1, \ldots, n, \\
x_j \in \{0,1\} \quad j = 0, 1, \ldots, n, \\
\delta_i \in \{0,1\} \quad j = 0, 1, \ldots, n.
\]

This problem will be denoted by ILP(ETKP) in subsequent sections. The Linear Programming relaxation denoted by ILPR(ETKP) will be defined by relaxing the integer constraints \( x_i \in \{0,1\} \) and \( \delta_i \in \{0,1\} \) to \( 0 \leq x_i \leq 1 \) and that \( 0 \leq \delta_i \leq 1 \).

II. Solution Approaches

A. Standard Mathematical Programming Software

The ETKP can be solved using standard OSL from IBM or software from other vendors. The solution method used is usually an implementation of the branch and bound algorithm, which appears to be a solution option of choice used in the industry today. This approach guarantees that optimal solutions will be found. It also gives intermediary results, this means that feasible solutions are generated while searching for the optimal value. For very large ETKP instances, OSL and other optimization software codes may fail and indeed does so in some instances we have experimented with.

B. Other Approaches

1) Heuristic

Feasible solutions to the problem can in principle be found by using heuristic approaches. These approaches try to find “good” solutions using innovative heuristic methods. These solutions are often not optimal.

2) Dynamic Programming

Shaw et al. (1997) have proposed a dynamic programming approach to the ETKP. Their approach used a depth first dynamic programming procedure to solve the problem.

One problem associated with the classic dynamic programming approach is that it only produces an optimal solution right at the end of the process. This state may not be reached in very large problems. In such an instance it would
mean that no solution, not even a sub optimal solution would be found.

C. Hybrid Approaches

These approaches could include the use of heuristics combined with exact methods. The heuristics could give a starting solution to the problem that might be able to enhance the effectiveness of the exact methods.

Another approach would be to combine dynamic programming principles with other exact methods or heuristics.

III. PROPOSED SOLUTION METHOD

A. Cardinality

One advance that has been made in the solution of the ordinary single item zero-one knapsack problems (which can be considered a special case of the ETKP) has been the use of cardinality constraints. See Caprara (2000). This approach generally tries to exploit knowledge of the cardinality (number of nodes in subtree) of a candidate solution to find feasible solutions faster.

In this paper, the use of cardinality is also investigated to trim the branch and bound tree that is a result of the tree search that it basically employs. In most very large problems, the growth in the search tree of Branch and Bound is a basic problem as it often exceeds the primary computer memory capacity in later stages of the search.

If applied to the ETKP, we could e.g. try to find the cardinality of the optimal subtree by solving the Linear Programming (LP) relaxations of the ETKP. We denote the ILP equivalent of ILP(ETKP) with cardinality constraint \( p \) as ILP(ETKP,\( p \)) and use ILPR(ETKP,\( p \)) to denote the LP relaxation with cardinality \( p \).

To find a reasonable indication of the range of values that give attractive cardinalities \( p \) we first solve the ILPR(ETKP) without any cardinality constraint to find a solution \( x^* \). Calculating \( \sum_{j=0}^n x_j^* \) yields a value \( p^* \) that may not be integer. Solving ILPR(TKP,\( p \)) for integer values close to \( p^* \), gives a good indication of a probable cardinality for optimal solution to the problem. These solutions to ILPR(TKP,\( p \)) can be used to find upper bounds that may be used to guide the search process.

B. Generating Lower Bounds

Any heuristic procedure can be used to generate a feasible solution. If such a solution is found, it provides a lower bound that can be used to trim the Branch and Bound search tree.

IV. ADAPTED ALGORITHM

An algorithm is now presented utilising the concept of cardinality and partitioning.

- Choose a cardinality \( p \) to be investigated,
- In the linear programming relaxation with a cardinality constraint \( p \) (ILPR(ETKP,\( p \))), count the nodes that are included with value 1, i.e. \( x_j = 1 \).
- Assume they are \( \ell \) in number. Rearrange the nodes so that these nodes are the first \( \ell \).
- Solve the following Integer Linear Program (using some optimisation software, say OSL of IBM)

\[
\begin{align*}
\max & \sum_{j=0}^n c_j x_j - \sum_{j=0}^n \left( f_j \delta_j + \alpha_j y_{j2} \right) \\
\text{s.t.} & x_j - x_{j-p_j} \leq 0 \quad j = 1,2,\ldots,n \\
& x_0 = 1 \\
& d^T x \leq H, \\
& \sum_{j=0}^{\ell-1} x_j = q, \\
& \sum_{j=q}^n x_j = p-q, \\
& y_{j2} \leq H \delta_j \quad j = 0,1,\ldots,n \\
& y_{j1} \leq h_j \quad j = 0,1,\ldots,n \\
& y_{j1}, y_{j2} \geq 0 \quad j = 0,1,\ldots,n \\
& x_j \in \{0,1\}, \quad y_j \in \{0,1\} \\
& \delta_j \in \{0,1\} \\
\end{align*}
\]

where \( q = \ell \) in the first iteration.

- Denote this model analogously to the notation introduced before by ILP(ETKP,\( p,q \)). If the solution process is successful we now have a feasible solution that may be suboptimal and thus a lower bound, if not, go to the next step.
- Repeat the previous step with \( q = q-1 \)

Compare the current best solution and continue in this way until an optimal solution is found (for the specific cardinality).

- Return to the first step with another feasible cardinality (with the potential of a better solution as indicated by ELPR(ETKP,\( p \)) if any feasible cardinalities remain.
- Repeat until feasible cardinalities are exhausted.

The best feasible solution generated in this way is optimal.

V. REFINEMENTS
Incorporating LP upper bounds into the algorithm may further enhance the algorithms outlined in section IV. Let \( z_{p,q} \) be the objective function value obtained by solving \( \text{ILPR}(\text{TKP}, p, q) \).

Let the current integer lower bound found either by a heuristic procedure or discovered in a previous step of the algorithm be \( B \). The only ILP instances that have to be examined at this point are those where the LP relaxation of the ILP has an optimal value where \( z_{p,q} \geq B \).

The algorithm proceeds as follows:

- Solve \( \text{ILPR}(\text{ETKP}, p, q) \) for all the promising cardinalities \( p \) and values for \( q \).
- Set the current lower bound \( B \) to 0 (or to a value found by applying a heuristic method).
- Solve the ILP(\( \text{TKP}, p, q \)) for an instances where the solution \( z_{p,q} \) to ILP(\( \text{ETKP}, p, q \)) gives \( z_{p,q} \geq B \) (It is customary to choose the one with the largest \( z_{p,q} \)).
- Update \( B \). If no such \( p \) and \( q \) exist, stop.
- If it is possible, choose the best and update \( B \). Otherwise stop.
- Repeat the process until no more promising ILP models remain.
- The best value achieved in this way will be the optimal value.

VI. EMPIRICAL WORK

Empirical work was undertaken to test the effectivity of the adapted algorithm presented. In order to test the algorithm data had to be generated. This was due to the fact that no suitable data could be obtained in order to test the proposed algorithm.

The data was generated in a systematic manner using a pseudo random number generator to generate integer numbers to represent the profit and demand for each node in a TKP. In this way problems of increasing complexity could be generated and used to test the proposed algorithm. The proposed algorithm was implemented in C++ and used IBM OSL as solver to solve all linear- and integer programming parts. It was decided to use third party software to solve linear- and integer programming problems.

The standard software with and without presolving the mathematical programming problems, were investigated. Presolving applies certain methods to make a problem “easier” to solve. These might include rewriting the problem with less constraints and probing some variables to determine good starting solutions. For the specific methods that OSL use, more information can be found in IBM (1995:81). In general presolving allows much larger problem instances to be solved, and is worthwhile considering when solving optimisation problems.

Solving larger problem instances than could be previously solved was an important goal in this research. One of the aims of the adapted algorithm is to assist the third party solver in using primary memory more efficiently and to thus allow larger problem instances to be solved successfully.

Different ways of measuring performance are available. The total time taken to solve a problem instance, the number of simplex iterations performed and also the number of branch and bound nodes developed during the Branch and Bound search process.

A. Solution classes

In the data represented the following key will be used:

- ALG. This is the time needed to solve the problem instance using the adapted algorithm. None of the integer programs are being presolved;
- PR ALG. Here all the integer programs of the TKP instances are presolved while the adapted algorithm is used;
- NO PR. This is the standard software without the use of the adapted algorithm. This corresponds to the way in which a “normal” user would solve the problem;
- PR. This is the standard software using the presolver. As in the previous case, the adapted algorithm is not used.

The hardware used was an IBM RS/6000 44P Model 270 machine with the AIX 4.3.3. operating system with 1024MB RAM.

B. Empirical results

The first results presented in Figure 2 is the time needed to solve ETKP instances. It is evident that initially no real gains is obtained by using the adapted algorithm. This is due to the fact that the algorithm does additional work which takes extra time. However for larger problem instances the adapted algorithm outperforms the standard software, as the additional work starts paying off. An upper bound of 5000 seconds was imposed, this value only came into play for NO PR and PR which could not be solved at all.

The next results presented in Figure 3 are the number of Branch and Bound nodes expanded during the solution process. For the adapted algorithm the number of Branch and Bounds expanded for the subproblems are summed together and this value is used in the graph. An upper bound of 500 000 Branch and Bound nodes were imposed. This upper bound was only reached at 250 nodes for the standard software with and without presolving.

The number of Branch and Bound nodes expanded gives a fair indication of the memory requirements needed during the solution process. The less nodes expanded, the less memory
was needed to solve the problem instance. It may also be worthwhile to look at the implementation of a parallel version of the algorithm, in such a case the relevant number of nodes to compare with other methods would be the maximum number of nodes expanded by any sub partition of the problem.

Figure 2 Time needed to solve ETKP instances

The last graph presented in Figure 4 shows the number of simplex iterations done by the solver during the solution process. For the adapted algorithm the number of iterations needed for the subproblems were again summed up and compared to the total number of nodes expanded by the standard software with and without presolving.

Looking at the number of simplex iterations, there were not such big gains made by the adapted algorithm, mostly because many subproblems were solved.

Figure 3 Number of Branch and Bound nodes expanded

Figure 4 Number of simplex iterations done

C. Conclusions from empirical work

It appears that the adapted algorithm often allows much larger problem instances to be solved successfully and in less time. In Table 1 the sum totals taken by each solution for all the test instances is presented. It is evident that the adapted exact algorithm took less time in total than the standard software alone. This makes it worthwhile to use the adapted algorithm. The adapted algorithm also produced results more consistently than the standard software with presolving. It is however necessary to point out that this is work in progress and that more empirical work is being done.

Table 1 Summary of results

<table>
<thead>
<tr>
<th>ALG</th>
<th>PR ALG</th>
<th>NO PR</th>
<th>PR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>308</td>
<td>295</td>
<td>4579</td>
</tr>
<tr>
<td>B&amp;B Nodes</td>
<td>60565</td>
<td>59841</td>
<td>659244</td>
</tr>
<tr>
<td>Iterations</td>
<td>407945</td>
<td>397773</td>
<td>1458868</td>
</tr>
</tbody>
</table>

VII. FUTURE WORK

Future work might be to investigate the portability of the enhanced modelling and partitioning to other integer programming problems. It may be worthwhile to investigate other classes of function $f_i$. More realistic models for local access telecommunication network design may also be investigated and the algorithmic experience gained here may be applied to such a model. More empirical work will have to be undertaken.

VIII. REFERENCES