The feasibility of LATN design using Tree Knapsack and Extended Tree Knapsack models[1]

D.J. van der Merwe[2] , J.M. Hattingh. North-west University, Potchefstroom Campus

Abstract— Local Access Telecommunication Networks (LATN) is a cost intensive aspect of service provisioning by telecommunication operators.

Research papers by Shaw *et al.* [] present certain models called Tree Knapsack – and Extended Tree Knapsack problems and discuss algorithms for their solution.

In this paper we present empirical results that indicate the feasibility of solving large problems of this type. Algorithms are presented that can be used in conjunction with professional optimization software like CPLEX from ILOG. This work demonstrates that many large network planning problems can be solved in reasonable computational times.

Index Terms—Network Planning – Planning Issues

I. INTRODUCTION

A. Local Access Telecommunication Network Planning

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."

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). Van der Merwe and Hattingh [8] investigated this problem and solution strategies for.

The extended tree knapsack (ETKP) is a (more realistic) generalized version of the TKP, where traffic flow cost is also considered. 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 $V = \{0,1,2, \ldots,n\}$. 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

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

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:

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

$$\text{s.t. } x_{p_j} \ge x_j, \quad j = 1,2,\ldots,n$$

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

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

We may assume that

$d_j \le H$ 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 nodes 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

[1] This work forms part of the research done at the North-west university, Potchefstroom Campus within the TELKOM CoE research programme, funded by TELKOM, GRINTEK TELECOM and THRIP

[2] Corresponding author: rkwdjvdm@puk.ac.za

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. Solution of TKP instances

1) Dynamic Programming

Cho and Shaw [7] have proposed a depth-first dynamic programming approach to the TKP, and stated that the complexity of the algorithm is $O(nH)$, where n is the number of nodes in the TKP and H is the capacity. Initial work done by Johnson and Niemi [3] presents a so called "left-to-right" dynamic programming approach having complexity $O(nC^*)$, where n is the number of nodes in the TKP en C^* is the optimal objective function value for the problem

2) Branch and Bound Methods

Shaw and Cho [6] presented a branch and bound algorithm for the TKP problem. The algorithm presented by them proved to outperform their dynamic programming algorithm. Their conclusion was that their Branch and Bound algorithm was not only superior to their dynamic programming version, but also superior to the approach by Johnson and Niemi. This result at first glance seems to be surprising but it correlates with the experience with the knapsack problem according to Martello *et al.* [4] where a branch and bound scheme outperformed a specialized dynamic programming approach of Pisinger [5], known to have pseudo-polynomial time complexity, quite handsomely

3) Proposed solution strategy

In this paper we investigated a branch and bound approach in conjunction with a partitioning strategy. The algorithm is described below.

The full details of this algorithm is presented in van der Merwe and Hattingh [8]. The algorithm consists out of a two part partitioning scheme. In order to describe the algorithm it is firstly necessary to introduce some notation and definitions. Firstly a first order partition is introduced and after that a second order partition extending the first order partition is presented.

a) First order partition

The first order partition relies on the concept of cardinality. A cardinality of a problem is the sum of the variables in the optimal solution and may be defined as $\sum_{j=0}^{n} x_j$ where $x = (x_0, x_1, x_2, \ldots, x_n)$ is the solution vector to a TKP. In order to use the concept of cardinality it is necessary to estimate cardinality 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 $0 \le x_i \le 1$, we use the notation ILPR(TKP) to denote the LP relaxation of ILP(TKP). Denote the 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.

b) Second order partition

Assume that an ILP(TKP,p) is a first order partition 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. If all such exchanges are considered, the optimal solution for ILP(TKP,p) must 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 $j = 0,1,2,\ldots,n$.
- 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 $V = \{0,1,2,\ldots,n\}$ define $S_l = \{ j \mid x_j^* = 1, j = 0,1,2,\ldots,n \}$ and $S_{n-l} = V \setminus S_l$.
- Choose $q \in \{l-1, l-2, l-3, \ldots, 0\}$
- Define the second order partitioned model ILP(TKP,p,q) as follows:

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

$$\text{s.t. } x_{p_j} \ge x_j, \quad j = 1,\ldots,n,$$

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

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

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

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

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

c) Partitioning TKP Algorithm

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

procedure PART_TKP
begin

$\text{CLB} = -\infty$

Set Z_{TKPR}= Solution to ILPR(TKP) and identify the solution values x_j^* for $j = 0,1,2,\text{K}\ ,n$

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

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 $Z_{\text{TKP}(p')}$ identify the solution values $x_j^{'}$ for $j = 0,1,2,\text{K}\ ,n$

Define $S_l = \{ j \mid x_j^{'} = 1\ , j = 0,1,2,\text{K}\ ,n\ \}$ and set $S_{n-l} = V \setminus S_l$

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

Set $Q = \{l-1, l-2, l-3, \text{K}\ ,0\}$

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

begin

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

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

begin

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

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

end

end

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

end.

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

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

end (while)

Optimal solution = CLB

end.

D. *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

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

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 seen figure 1.

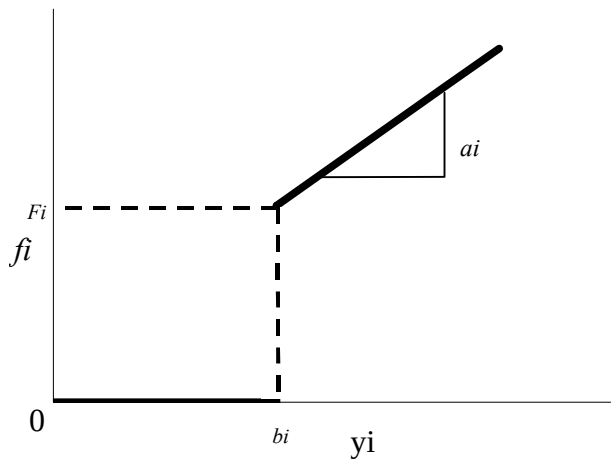

Figure 1 Graphical representation of function *fi*

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 existing capacity.

E. *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 ($y_{i1} \le b_i$) and y_{i2} the part of the flow greater than the current capacity of the link.

Secondly, introduce new 0-1 variables, δ_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: $y_{i2} \le H\delta_i$.

Define $\mathbf{y_1} = (y_{11}, y_{21,\ldots,} y_{n1})$ and $\mathbf{y_2} = (y_{12}, y_{22,\ldots,} y_{n2})$.

The ETKP is now be formulated as following:

$$\max \sum_{j=0}^{n} c_j x_j - \sum_{j=0}^{n} \left(F_i \delta_i + \alpha_i y_{i2}\right)$$
$$\text{s.t. } x_j - x_{p_j} \le 0 \quad j = 1,2,$$
$$x_0 = 1$$
$$D\mathbf{x} - B\left(\mathbf{y}_1 + \mathbf{y}_2\right) = \mathbf{0},$$
$$\boldsymbol{d}^{\mathrm{T}} \mathbf{x} \le H,,$$
$$\mathrm{y}_{i2} \le H\delta_i$$
$$0 \le y_{j1} \le b_j \qquad j = 0,1,\mathrm{K}\ n,$$
$$y_{j1}, y_{j2} \ge 0 \qquad j = 0,1,\mathrm{K}\ n,$$
$$x_j \in \{0,1\} \qquad j = 0,1,\mathrm{K}\ n,$$
$$\delta_j \in \{0,1\} \qquad j = 0,1,\mathrm{K}\ n.$$

Note the flow balance constraints that have been added to incorporate the traffic flow. B is the usual node-arc incidence matrix and D is the diagonal demand matrix. 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 $0 \le x_i \le 1$ and that $0 \le \delta_i \le 1$.

F. Solution methods for the ETKP

At a previous SATNAC conference solution strategies similar to the strategy proposed for the TKP were presented [9]. Additional work to improve the performance of these strategies was conducted and will now be presented.

In the course of the research done it was observed that a large integrality gap results with the formulation of the ETKP as stated previously. In order to counteract this, valid inequalities were added that greatly enhanced the performance of the algorithm. Two procedures that produce these inequalities are given below.

It was noticed that an inequality could be added that forces δ variables to one if they are to be added from logical considerations. The rationale of these valid inequalities was to force certain δ-variables to the value of 1 if the planned flow requirement at that node requires it.

The procedures can be summarized as follows:

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

procedure add_cut2
begin
  for $i = 0,1,2,\ldots,n$ do
  begin
    Define $C_i = \{ j \mid j \text{ child node of } i \}$ and
    $|C_i| =$ number of child nodes of i
    if $C_i \ne \varnothing$ then
    begin
      dem_sum = $\sum_{k \in C_i} d_k$
I if dem_sum $\ge b_i$ then do
      begin
        add_constraint $x_i + \sum_{j \in Ci} x_j - d_i \le |C_i|$
      end
    end
  end
End.

II. Results of Empirical work

A. Tree Knapsack results

Extensive work has been on the TKP and a summary of these results is now presented. In order to validate the performance of our partitioning algorithm it was decided to compare the partitioning algorithm with an algorithm presented by Shaw and Cho [3]. Attempts to obtain an implementation of such an algorithm from the original authors were not successful. Consequently it was decided to implement the algorithm from the literature. The algorithm implemented was the Branch and Bound algorithm presented in [3]. A dynamic programming algorithm was also presented in [1] and [7] by the same authors but the branch and bound algorithm were found to be a more efficient by their own admission. It was thus decided only to implement the Branch and Bound algorithm and to make comparisons with it.

As the partitioning algorithm utilizes standard software, in our case CPLEX from ILOG, and the TKP can in principle be solved using standard software, benchmarks also include the time taken solve a specific problem instance by using CPLEX.

The empirical work was done on TKP data instances and the number of nodes ranged between 500 and 50,000 nodes. 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 Partitioning TKP algorithm. For each instance with a fixed number of nodes, 4 different tree configurations were generated and each solved with 9 different capacity constraints on the knapsack

A graphical representation of the results can be seen in figure 2. The legend for the graphical representation is as follows:

- SHAW: The algorithm developed by Shaw and Cho.
- ALG: The partitioning TKP algorithm using CPLEX as solver for all linear programming and integer linear programming problems..
- CPLEX: The standard CPLEX without enhanced modelling.

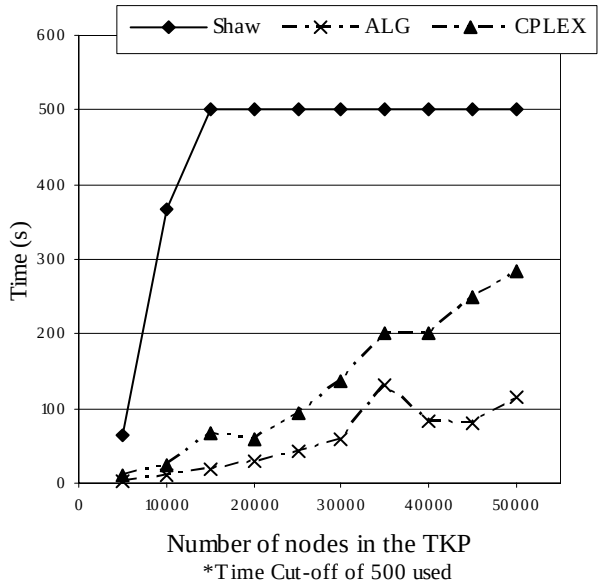

Figure 2 Average times taken to solve TKP instances

The average times taken to solve the problem instances are also presented in table 1. This table clearly shows that the partitioning algorithm performed better than the algorithm presented by Shaw and Cho and also CPLEX.

Nodes	Shaw & Cho	Partitioning Algorithm	CPLEX
500	0.13	0.39	0.53
1000	0.62	0.77	0.66
2000	3.93	2.38	2.73
3000	12.41	2.95	6.83
4000	23.44	3.90	5.41
5000	63.44	3.81	11.20
10000	366.68	10.85	23.45
15000	892.72	17.77	65.76
20000	2717.28	29.91	59.88
25000	6608.14	41.93	93.65
30000	9197.58	58.25	135.81
35000	30031.72	130.92	200.21
40000	38599.19	82.66	201.05
45000	54182.57	79.47	248.24
50000	76162.94	116.04	284.09

Table 1 Average time taken to solve TKP instances

From these results it seems as if the proposed algorithm was at least twice as fast as CPLEX and many times faster than the algorithm developed by Shaw and Cho. This generally enables one to solve more complex problems.

B. Extended Tree Knapsack Problem

Research is still being conducted on the ETKP. Preliminary results will now be presented. The first results show the effect of the adding the valid inequalities to the formulation of the ETKP.

As example we will present the results for a relatively small ETKP instance. The problem instance has 100 nodes, the capacity for the ETKP was set at 50% of the sum of all the demands of the individual nodes. The results are summarized in table 2. it is evident that by adding simple inequalities the objective function value is decreased and hence the integrality gap is decreased by approximately 32%. This in turn leads to fewer second order partitions that needs to be examined. This leads to a significant improvement in the performance of the partitioning algorithm.

	Objective function value
Standard LP	7890.024
LP with added inequalities	7859.233
Optimal solution	7795.2

Table 2 Objective function values of the LP solutions

Another avenue of research was to implement using the standard modeling capabilities of CLPEX to handle the non-linear portion of the formulation made in section I D. The non-linear functions were modeled using a CPLEX data structures developed specifically for piece-wise linear functions. Results indicate that the partitioning algorithm performs a lot better with the added valid inequalities. This is evident in the execution time presented in table 3. for the same example as given above.

	Time in seconds
CPLEX	2.407
CPLEX using piece-wise linear	0.764
Partitioning algorithm	0.772
Partitioning algorithm with valid inequalities added	0.222

Table 3 Comparison of execution times of different algorithms for the ETKP.

III. Current Research

Current research is concerned with different flow cost functions. An alternative to the flow-cost function used in this paper is a so-called step function This would imply that adding capacity is done in increments where there is not a linear addition for additional flow after expansion.

IV. Conclusion

The research conducted up to this point indicates that the Tree Knapsack - and the Extended Tree Knapsack problem can be solved efficiently using standard software with enhanced modeling. This can be seen when comparing the performance of the partitioning strategies with standard software. In the case of the TKP comparisons were made with the best known algorithms in literature and the partitioning algorithm gave satisfactory performance enabling one to solve realistic large problems more easily.

REFERENCES

[1] G. Cho and D.X. Shaw, "A depth-first dynamic programming algorithm for the tree knapsack problem." *INFORMS journal on computing*, 9 (1997) 431 - 438.

[2] G. Cho, D.X. Shaw and S.L. Kim, "An efficient algorithm for a capacitated subtree of a tree in local access telecommunication networks." *Computer Operations Research*, 24 (1997) 737 – 748.

[3] D.S. Johnson and K.A. Niemi. "On Knapsacks, partitions and a new dynamic programming technique for trees." *Mathematics of Operations Research*, 8 (1983) 1-14.

[4] S. Martello, D. Pisinger and Toth P. "New trends in exact algorithms for the 0-1 knapsack problem." *European journal of operational research*, 123 (2000) 325 - 332.

[5] D. Pisinger, "A minimal algorithm for the 0-1 knapsack problem." *Operations research,* 46 (1997) 758-767.

[6] D.X. Shaw and G. Cho, "The critical-item, upper bounds, and a branch-and-bound algorithm for the tree knapsack problem." *Networks*, 31 (1996) 204-216.

[7] D.X. Shaw, C. Cho and H. Chang. "A depth-first dynamic programming procedure for the extended tree knapsack problem in local access network design." *Telecommunication systems*, 7 (1997) 29 - 43.

[8] D.J. van der Merwe D.J. and J.M. Hattingh, "Tree Knapsack Approaches For Local Access Network Design". Accepted for publication in *European Journal of Operational Research.*

[9] D.J. van der Merwe D.J. and J.M. Hattingh, "An adapted exact algorithm for solving extended tree knapsack problems for Local Access Telecommunication Network design issues" SATNAC September 2003.

David J van der Merwe

David is a PhD student in the School of Computer-, Statistical- and Mathematical Sciences of the North-west University, Potchefstroom campus, specializing in Computer Science and Informatics. He holds a BSc with majors Mathematics and Computer Science, an Honns BSc in Computer Science as well as an MSc in Computer Science.

Fields of interest include the Application of Mathematical Programming techniques to real problems and Parallel Processing