A Decomposition Algorithm for Local Access Telecommunication Network Design Problem

Geon Cho

Information Infrastructure Planning Section, ETRI

ABSTRACT

In this paper, we develop detailed algorithms for implementing the so-called Limited Column Generation procedure for Local Access Telecommunication Network (LATN) Design problem. We formulate the problem into a tree-partitioning problem with an exponential number of variables. Its linear programming relaxation has all integral vertices, and can be solved by the Limited Column Generation procedure in just n pivots, where n is the number of nodes in the network. Prior to each pivot, an entering variable is selected by detecting the Locally Most Violated (LMV) reduced cost, which can be obtained by solving a subproblem in pseudo-polynomial time. A critical step in the Limited Column Generation is to find all the LMV reduced costs. As dual variables are updated at each pivot, the reduced costs have to be computed in an on-line fashion. An efficient implementation is developed to execute such a task so that the LATN Design problem can be solved in $O(n^2H)$, where H is the maximum concentrator capacity. Our computational experiments indicate that our algorithm delivers an outstanding performance. For instance, the LATN Design problem with $n = 150$ and $H = 1000$ can be solved in approximately 67 seconds on a SUN SPARC 1000 workstation.

1 Introduction

The telecommunication industry is undergoing rapid and fundamental change. Technological innovation creates new market opportunities and growing customer demand for enhanced products and services such as multimedia applications. The ongoing legislative regulatory reform is reshaping the structure and business practice of major industry segments. The federal government has challenged the private sector to participate in an effort to modernize the nation's entire information infrastructure by building the "information super-highway." As a result, many major telecommunication service providers are in the process of upgrading and expanding their facilities and services. The planning and design of various types of telecommunication networks plays an important role in this process. It is well known that optimization techniques can be used to optimize network design. In this paper, we study a portion of the telecommunication system, the Local Access Telecommunication Network (LATN).

Most existing LATNs have a tree structure. Each customer node has a demand representing the required number of circuits from that node to the switching center. This demand can be satisfied by either connecting the node directly through a cable to the switching center, or routing it first to a concentrator which compresses the incoming traffic into a higher frequency signal 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 the total cost.

Recently, many researchers have been working on the LATN Design and Expansion problems, which are NP-complete problems since they include the Knapsack Problem as a special case. A discussion of the modeling issues can be found in Balakrishnan *et al.* [3], and a Lagrangian relaxation based approach has been developed by Balakrishnan, Magnanti and Wong [4]. Aghezzaf, Magnanti and Wolsey [2] studied the combinatorial structure of the model. A related but different model was formulated by Cook [11], and a solution procedure and related knapsack-type problem have also been studied by Bienstock [6, 7].

A so-called Limited Column Generation procedure was developed by Shaw [18, 19]. The LATN Design and Expansion problem is first formulated as a tree-partitioning problem with an exponential number of variables, each of which corresponds to a subtree (see Shaw [19]). Chvatal [10] and Gavril [12] have shown that the polyhedron of its linear programming relaxation is integral. Hence, the tree-partitioning problem can be simply reduced to a linear program (with an exponential number of variables, though). Then, a dedicated column generation procedure is developed so that the linear program can be solved in just $n+1$ pivots ($n+1$ is the number of nodes in the tree). Prior to each pivot, a so-called Locally Most Violated (LMV) reduced cost has to be computed to decide the entering variable.

Based on the framework of the Limited Column Generation procedure, one of our main results is to design and implement detailed algorithms for the LATN Design and Expansion problems. Our computational experiments indicate that the Limited Column Generation is a very efficient approach for these models. The LATN Design problem with 150 nodes can be solved in approximately 67 seconds on a SUN SPARC 1000 workstation.

The LMV reduced cost can be obtained by solving n subproblems, each of which is the Tree Knapsack Problem (TKP) and has been studied by Cho and Shaw [8], Johnson and Niemi [15], Lukes [17], and Shaw [1]. Johnson and Niemi [15] solved the TKP by the so-called "left-right" dynamic programming algorithm with a running time of $O(nC^*)$, where C^* is the optimal value of the TKP. Recently, Cho and Shaw [9] proposed the so-called "depth-first" dynamic programming algorithm with a running time of $O(nH)$ which can be regarded as a refinement of Johnson and Niemi [15].

The Limited Column Generation procedure requires that the LMV reduced costs are checked in a bottom-up order. The reversed Depth-First-Search (DFS) order and the reversed Breadth-First-Search (BFS) order are two typical examples of the bottom-up order. A bottom-up dynamic programming procedure developed by Shaw [19] for the TKP runs in $O(nH^2)$ time and can be combined with the Limited Column Generation procedure to solve the LATN Design problem in $O(n^2H^2)$, where H is the maximum concentrator capacity.

A straight-forward approach for combining Limited Column Generation with a low complexity TKP solver such as the "depth-first" dynamic programming algorithm developed by Cho and Shaw [9] leads to an $O(n^3H)$ algorithm for the LATN Design problem because $n+1$ pivots are required and prior to each pivot, n CTKPs have to be solved, each of which requires $O(nH)$ time. However, as reduced costs are related to the dual variables, which are updated at each iteration, the LMV reduced costs have to be computed in an on-line fashion. The main result of this paper is to develop an efficient on-line implementation which recursively computes all the LMV reduced costs in the total of $O(n^2H)$ time for the LATN Design problem. As a result, we obtain an algorithm for the problem with the overall complexity of $O(n^2H)$.

This paper is organized as follows. In Section 2 we present the fixed-charge network formulations for the LATN Design and Expansion problems, and then we describe the Limited Column Generation procedure for the LATN Design problem and its implementation in Section 3. In Section 4 we present an on-line recursive procedure for computing all the LMV reduced costs. The implementation detail for the LATN Design problem is presented in Section 5, and we present our computational test results in Section 6.

2 Problem Description

In this section, we describe the LATN Design and Expansion problems in detail. The LATN is a tree rooted at the switch center, and each non-root node is a customer node and is associated with a demand, which represents the number of circuits required from the customer node to the switching center. This demand can be satisfied by either connecting a cable from the customer node to the switching center or routing the circuits to a concentrator which compresses the incoming traffic into a higher frequency signal that requires fewer outgoing cables. A variety of electronic devices can perform traffic compression through frequency division or time division such as concentrators, multiplexers, remote switches and fiber optical terminals. Since these devices perform essentially equivalent functions from our modeling point of view, we collectively refer to them as concentrators. In this paper, we assume that the compressed signal requires a dedicated cable point-to-point (home-run) routing to the switching center. The LATN Expansion problem is to make tradeoff between installing concentrators and expanding cable capacities that already exist along the arcs in order to minimize the total cost. The LATN Design problem is its special case, where all of the existing cable capacities along the arcs are equal to zero.

Let $\hat{T} = (V, E)$ be an undirected tree rooted at the switching center labeled as node 0, representing the layout of the LATN. Then, without loss of generality, we assume that all nodes in $\hat{T}$ are labeled by the Depth-First-Search (DFS) order. Let $P[i,j]$ be a unique path on tree $\hat{T}$ from node i to node j. Let $S(i) = \{j | p_j = i\}$ be the set of successors of node i, where p_j is the predecessor of node j. We denote p_j^i as the predecessor of node j with respect to node i, which is defined as the first node other than node j on the path $P[j,i]$. By labeling arc (p_i, i) and arc (i, p_i) as arc i and arc $-i$ respectively, a directed network $\overrightarrow{T}$ is derived from $\hat{T}$. Let A be the node–arc incidence matrix of the directed graph $\overrightarrow{T}$ which excludes the first row corresponding to node 0.

We denote d_i as the demand at node i for $i = 1, 2, \cdots, n$. The total investment on the LATN involves two categories of costs: the concentrator installation and cable expansion.

Suppose there are m different capacities for concentrator modules. A concentrator with capacity h^t can be installed at node i with the fixed cost $\hat{F}_i^t$, which represents the concentrator purchase cost and installation costs as well as other infrastructure investment, and there is also a variable concentrator cost $\hat{c}_i$ representing the operating expenses, where $i = 0, 1, 2, \cdots, n$, and $t = 1, 2, \cdots, m$. We assume $h^1 < h^2 < \cdots < h^m$. In this paper, we regard the concentrator as an "aggregate concentrator" and assume that at most one concentrator can be installed at each node. More general models are discussed in Shaw [1].

The existing cable capacity on arc i is b_i for all $i = \pm 1, \pm 2, \cdots, \pm n$, and $b_i = b_{-i}$, as all cables are two-way. If the cable requirement on arc i exceeds the existing capacity b_i, a fixed cable expansion cost F_i as well as a marginal variable cable expansion cost of c_i occurs, where $i = \pm 1, \pm 2, \cdots, \pm n$. The fixed cost represents the expenses for digging trenches and laying pipes on arc i, while the variable cost represents the cable purchasing and maintenance cost on arc i.

Now we introduce the decision variables as follows. Let $\hat{x}_i$ be the load of the concentrator at node i, $i = 0, 1, 2, \cdots, n$ and x_i be the cable requirement on arc i, $i = \pm 1, \pm 2, \cdots, \pm n$. Then we set

$$x_i = x_i' + x_i'',$$

where $0 \le x_i' \le b_i$, and x_i'' is the amount of the cable expansion beyond existing capacity, for $i = \pm 1, \pm 2, \cdots, \pm n$.

We also define binary variables as follows:

$$y_i' = \begin{cases} 1 & \text{if } x_i' > 0 \\ 0 & \text{otherwise}, \quad i = \pm 1, \pm 2, \cdots, \pm n \ , \end{cases}$$

$$y_i'' = \begin{cases} 1 & \text{if } x_i'' > 0 \\ 0 & \text{otherwise}, \quad i = \pm 1, \pm 2, \cdots, \pm n \ , \end{cases}$$

and

$$\hat{y}_i^t = \begin{cases} 1 & \text{if } h^t \text{ is used at node } i \\ 0 & \text{otherwise}, \quad i = 0, 1, 2, \cdots, n \text{ and } t = 1, 2, \cdots, m. \end{cases}$$

In practice, to reduce the complexity of network planning, management and maintenance, some restrictions are imposed on the routing patterns by planners. To simplify the problem, we also make additional assumptions. A discussion of these assumptions can be found in Balakrishnan *et al.* [3] and Balakrishnan, Magnanti, and Wong [4]. These restrictions can be summarized as follows:

1. Only one-level traffic compression is allowed (i.e., all demands can be compressed at most once before reaching the switching center).

2. The compressed signals are point-to-point routed to the switching center through a dedicated cable (for example, fiber optics).

3. The *non-bifurcated routing*, that is, all circuits from one customer node must follow the same routing pattern. Clearly, to make such an assumption, we have to assume $h^m \ge d_i$ for $i = 1, 2, \cdots, n$.

4. The ***contiguity restriction***, that is, if node j's traffic is compressed by a concentrator located at node i, then traffic from all nodes on path $P[i,j]$ is compressed by the same concentrator located at node i. In particular, if a concentrator is installed at node i, then the demand at node i should be routed to that concentrator.

Based on the above discussion, the LATN Expansion problem can be formulated as the following special fixed-charge network flow problem (E):

$$
(E)\qquad
\begin{array}{lll}
\min & c^T x'' + F^T y'' + \hat{c}^T \hat{x} + \sum_{t=1}^{m} (\hat{F}^t)^T \hat{y}^t & (2.1)\\
\text{s.t.} & A(x' + x'') + \hat{x} = d & (2.2)\\
 & 0 \le x' \le b * y' & (2.3)\\
 & 0 \le x'' \le M y'' & (2.4)\\
 & 0 \le \hat{x} \le \sum_{t=1}^{m} h^t \hat{y}^t & (2.5)\\
 & y'_i + \sum_{t=1}^{m} \hat{y}^t_i + \sum_{j \in S(i)} y'_{-j} = 1, \quad i = 1, 2, \cdots, n & (2.6)\\
 & y' \ge y'' & (2.7)\\
 & y',\ y'' \in \{0,1\}^{2n},\ \hat{y}^t \in \{0,1\}^n, &
\end{array}
$$

where $M = \sum_{k=1}^{n} d_k$ and "*" denotes the component by component product.

The objective function (2.1) seeks to minimize the sum of the cable and the concentrator installation costs. Constraints (2.2) specify the flow conservation constraints, that is, the amount of traffic coming into node i minus the amount of traffic going out from node i plus the load of concentrator located at node i should be exactly equal to the demand d_i at node i. Constraints (2.3) and (2.4) are simply the upper bounds for the existing cable capacity and for the cable expansion requirement, respectively. Constraints (2.6) imply $\sum_{t=1}^{m} \hat{y}^t_i \le 1$, which means that only one type of concentrator modules can be used at each node. Constraints (2.6) and (2.7) together state that the demand at node i is satisfied from either one of the incoming arcs or a concentrator installed at that node, which reflect the non-bifurcated routing and contiguity assumptions. Obviously, constraints (1.7) simply states that if the existing cable capacity on arc i is not used then cable expansion should not occur on arc i.

In the LATN Design problem, the existing cable capacity b_i on arc i is zero for all $i = \pm 1, \pm 2, \cdots, \pm n$. Therefore, we eliminate constraints (1.3) and (1.7) in (E), and set $x' = 0$, $x = x''$, and $y = y''$. Then we have $y_i + \sum_{t=1}^{m} \hat{y}^t_i + \sum_{j \in S(i)} y_{-j} = 1$. The following is the fixed-charge network flow formulation for the LATN Design problem (D);

$$
(D)\qquad
\begin{array}{ll}
\min & c^T x + F^T y + \hat{c}^T \hat{x} + \sum_{t=1}^{m} (\hat{F}^t)^T \hat{y}^t\\
\text{s.t.} & Ax + \hat{x} = d\\
 & 0 \le x \le My\\
 & 0 \le \hat{x} \le \sum_{t=1}^{m} h^t \hat{y}^t
\end{array}
$$

$$y_i + \sum_{t=1}^{m} \dot{y}_i^t + \sum_{j \in S(i)} y_{-j} = 1, \quad i = 1, 2, \cdots, n$$

$$y \in \{0,1\}^{2n}, \; \dot{y}^t \in \{0,1\}^n, \quad \text{where } M = \sum_{k=1}^{n} d_k.$$

In the next section, we introduce a tree partitioning formulation and the Limited Column Generation procedure for the LATN Design and Expansion problem developed by Shaw [18].

3 Tree Partitioning Formulation and Limited Column Generation Procedure

Because of the contiguity assumption, the set of nodes allocated to the same concentrator forms a subtree of $\dot{T}$. Hence, we can formulate the LATN Design problem as a tree partitioning problem (see Cho [8]) for the LATN Expansion problem).

We define the *routing cost* c_{ij} from node j to a concentrator location i for the LATN Design problem as follows : for $i = 0, 1, 2, \cdots, n$ and $j = 1, 2, \cdots, n$,

$$c_{ij} = \begin{cases} d_j(\dot{c}_i + \sum_{k \in K_{ij}} c_k) + F_j & \text{if } j \notin P[i,0] \\ d_j(\dot{c}_i + \sum_{k \in K_{ij}} c_k) + F_{-p_j^i} & \text{if } j \in P[i,0] \setminus \{i\} \\ d_j \dot{c}_j & \text{otherwise,} \end{cases} \tag{3.1}$$

where K_{ij} is the set of arcs on the path $P[i,j]$ from node i to node j.

Let T be a subtree of $\dot{T}$ rooted at node k. Then, we assume that a node is in T if and only if it is served by a concentrator located at node i inside T, which is called the *center* of T. We also assume that if $0 \in T$ then the center of T is node 0. We now define the total cost c_T^i of T with the center i as

$$c_T^i = \sum_{j \in T} c_{ij} + \dot{F}_i^{t^*}. \tag{3.2}$$

where $t^* = \min \{t \mid \sum_{j \in T} d_j \leq h^t, \; t = 1, 2, \cdots, m\}$.

Let $c_T = \min_{i \in T} c_T^i$ be the cost of assigning all nodes in T to a common concentrator.

Then, the LATN Design problem can be reformulated as follows:

$$(P) \quad \begin{array}{ll} \min & c^T \xi \\ \text{s.t.} & G\xi = 1 \\ & \xi_T \in \{0,1\} \quad \text{for all subtree } T. \end{array}$$

where $c = (c_T)$, $\xi = (\xi_T)$ and G is a node subtree incidence matrix.

As the intersection graph of G is a chordal graph and, of course, a perfect graph (for detail, see Chvatal [10], Gavril [12], Golumbic [14], and Lovasz [16]), the linear programming relaxation of (P) has all integral vertices.

Shaw [18] presents an elementary proof that the linear programming relaxation of (P) always has an integer optimal solution. Therefore, (P) can be written as

$$(P) \quad \begin{array}{ll} \min & c^T\xi \\ \text{s.t.} & G\xi = 1 \\ & \xi_T \in [0,1] \quad \text{for all subtree } T. \end{array}$$

Since the number of columns in G can be an exponential number in terms of n, a natural way to solve this problem is to use a column generation technique, which was first proposed by Gilmore and Gomory [13]. However, because of the special structure of our problem, much better results can be achieved. Shaw [18] has shown that there exists a pivot rule which selects the so-called Locally Most Violated (LMV) reduced cost and solves (P) in just $n+1$ pivots.

Let B be the $(n+1) \times (n+1)$ basis for the system of constraints in (P) and π_i be the dual variable of node i (or i-th constraint) in (P) for $i = 0, 1, 2, \cdots, n$. Then the reduced cost of a variable ξ_T is

$$\gamma_T = \sum_{i \in T} \pi_i - c_T.$$

The Locally Most Violated (LMV) reduced cost γ_k^* is defined as

$$\gamma_k^* = \max_{r(T)=k} \gamma_T = \gamma_{T_k^*},$$

where $T_k^* = \arg \max_{r(T)=k} \gamma_T$ and $r(T) = \min\{k | k \in T\}$ is the root of T (recall that the node in $\hat{T}$ is labeled by DFS order).

Initially, we partition $\hat{T}$ into singleton sets $\{k\}$ for $k = 0, 1, 2, \cdots, n$ (i.e., $\hat{T} = \cup_{k=0}^n \{k\}$). This can be interpreted as the case that every node has its own "concentrator." In such a case, the basis B is an $(n+1) \times (n+1)$ identity matrix I and the dual variable π_k is

$$\pi_k = c_{\{k\}} = d_k \hat{c}_k + \hat{F}_k^{t_k^*},$$

where $t_k^* = \min\{t | d_k \leq h^t,\ t = 1, 2, \cdots, m\}$ and $k = 0, 1, \cdots, n$.

Let $1_{T_k^*} = (\nu_k) \in R^{n+1}$, where $\nu_k = \begin{cases} 1 & \text{if } k \in T_k^* \\ 0 & \text{otherwise.} \end{cases}$

Now, the Limited Column Generation procedure developed by Shaw [18] can be formally described as follows:

Algorithm 1. Limited Column Generation;

```
begin
{comment: Initialization}
  $B := I$;
  for $k := 0$ up to $n$ do    $\pi_k := c_{\{k\}}$;
{comment: Main Loop}
  for $k := n$ down to $0$ do
  begin
    Find_$\gamma_k^*$;    {comment: $\gamma_k^* = \max_{r(T)=k} \gamma_T = \gamma_{T_k^*}$ }
    if ($\gamma_k^* > 0$) then
      replace $k$-th column of $B$ by $1_{T_k^*}$;
```

```
          $\pi_k := \pi_k - \gamma_k^*$;
          **Update_$\gamma_k^*$**;
        **end if**
      **end**
      *opt_value* $:= \sum_{k=0}^{n} \pi_k$;
      **Opt_Solution**;
    **end**
```

Theorem 1. *If the LMV reduced cost is given at each pivot, then the LATN Design problem can be solved by Algorithm 1 which essentially takes $n+1$ simplex pivots.*

The proof of Theorem 1 can be found in Shaw [18]. Because of Theorem 1, a critical step for solving the LATN Design problem is to find the LMV reduced costs by the procedure **Find_γ_k^*** at each pivot. As γ_k^* depends on $\{\pi_i | i \in T(k)\}$, which is updated at each iteration, we show in the next section how the set $\{\gamma_k^* | k = n, n-1, \cdots, 0\}$ can be computed recursively in an on-line fashion. The procedures **Find_γ_k^*** and **Update_γ_k^*** are given in the next section.

Lemma 3.1. *If we exclude the computational time for computing* **Find_γ_k^*** *and* **Update_γ_k^***, *then Algorithm 1 can be terminated in $O(n^2)$ time.*
Proof: As replacing a column of B takes $O(n)$ time and there are $n+1$ iterations in Algorithm 1, the overall complexity is $O(n^2)$. □

Finally, we present the procedure **Opt_Solution** which is a tree search algorithm and can be solved in linear time. We define an array $SOLUTION(j)$ as follows: $SOLUTION(j) = k$ if and only if node j is covered by a subtree rooted at k.

Procedure 1.3 **Opt_Solution**;

```
begin
  $k := 0$;  $STACK := \emptyset$;
  for $j := 0$ up to $n$  do
  begin
    if ($j > LAST(k)$)  then
      pick $k$ from $STACK$ such that $LAST(k) \geq j$;
      if ($B_{jk} = 0$)  then     {comment: $B = (B_{ij})$ is the optimal basis}
        put $k$ to $STACK$;
        $k := j$;
      end if
    end if
    if ($B_{jk} = 0$)  then
      put $k$ to $STACK$;
      $k := j$;
    end if
    $SOLUTION(j) := k$;
  end
end
```

One remark we would like to make here is that the center of the subtree at k can be specified by defining an array $LOCATION(k)$ in the procedure **Find_γ_k^*** as follows: $LOCATION(k) = i$ if and only if the subtree rooted at k is served by a concentrator located at i. Then $LOCATION(SOLUTION(j))$ specifies the location of concentrator that serves node j.

4 Computing the LMV Reduced Cost

Let π_i be the dual variable of node i, $i = 0, 1, 2, \cdots, n$, in (P). Then we have

$$\begin{aligned}
\gamma_k^* &= \max_{r(T)=k} (\sum_{j \in T} \pi_j - c_T) \\
&= \max_{r(T)=k} (\sum_{j \in T} \pi_j - \min_{i \in T} c_T^i) \\
&= \max_{r(T)=k} \max_{i \in T} (\sum_{j \in T} \pi_j - c_T^i) \\
&= \max_{i \in T(k)} \max_{\substack{T \ni i \\ r(T)=k}} (\sum_{j \in T} \pi_j - c_T^i) \\
&= \max_{i \in T(k)} \gamma_k^i,
\end{aligned}$$

where

$$\gamma_k^i = \max_{\substack{T \ni i \\ r(T)=k}} (\sum_{j \in T} \pi_j - c_T^i). \tag{4.1}$$

Given node k, let $i \in T(k)$. For the LATN Design problem, we define

$$(S_k^i(h)) \quad \begin{aligned}
\max \quad & \sum_{j \in T(k)} \bar{c}_{ij} x_j \\
\text{s.t.} \quad & x_{p_j^i} \geq x_j, \quad j \in T(k) \setminus \{i\} \\
& \sum_{j \in T(k)} d_j x_j \leq h \\
& x_k = 1 \\
& x_j \in \{0, 1\},
\end{aligned}$$

where $\bar{c}_{ij} = \pi_j - c_{ij}$ and c_{ij} is defined in (3.1).

We use $P_{T(k)}(i, k, h)$ to denote the optimal value of $(S_k^i(h))$ (see $(CTKP)$ in Section 5 for detailed discussion about notation). Then, it follows from (3.1)—(3.3) and (4.1) that γ_k^i for the LATN Design problem can be obtained as follows:

$$\gamma_k^i = \max_{1 \leq t \leq m} \{P_{T(k)}(i, k, h^t) - \hat{F}_i^t\}. \tag{4.2}$$

Hence, a critical step in computing γ_k^i for the LATN Design problem is to solve a subproblem $(S_k^i(h))$. When $i = k$, $(S_i^i(h))$ is the TKP discussed by Cho and Shaw [9], Johnson and Niemi [15], Lukes [17], Shaw [1], and Shaw and Cho [22]. In particular, Cho and Shaw [9] proposed an efficient Depth-First dynamic programming algorithm that runs in $O(|T(k)|h)$ time to solve the TKP. In general, for $i \in T(k)$ (or $k \in P[i, 0]$), we call $(S_k^i(h))$ the Centered Tree Knapsack Problem (CTKP). In the following section, we show that for fixed i, the CTKP $(S_k^i(h))$ can be solved recursively from an optimal solution of $(S_{p_k^i}^i(h))$ in $O(|T(k) \setminus T(p_k^i)|h)$ time for all $k \in P[i, 0] \setminus \{i\}$. Consequently, we are able to compute γ_k^i through (4.2) and finally obtain γ_k^*. However, all $(S_k^i(h))$ are related to $\{\pi_i | i = 0, 1, 2, \cdots, n\}$, which are updated in the reverse order of node label in Algorithm 1. Therefore, all $(S_k^i(h))$ with $i \in T(k)$ have to be solved in an on-line fashion. The following two procedures, **Find_γ_k^*** and **Update_γ_k^*** resolve these difficulties. Technically, we define $\underline{d} = \min\{d_j | j \in V \setminus \{0\}\}$.

```
Procedure 1.1  Find_$\gamma_k^*$;

begin
```

```
    compute $P_{T(k)}(k,k,h)$ for all $h = \underline{d}, \underline{d}+1, \cdots, H$;
    if $(k \neq 0)$ then
      for $(i \in T(k) \setminus \{k\})$ do
      begin
        compute $P_{T(k)}(i,k,h)$ starting from $P_{T(k)}(i,p_k^i,h)$ for all $h = \underline{d}, \underline{d}+1, \cdots, H$;
      end
        $\gamma_k^* := \max_{i \in T(k)} \max_{1 \le t \le m} \{P_{T(k)}(i,k,h^t) - \hat{F}_i^t\}$;
    end if
  end
```

Procedure 1.2 **Update_γ_k^***;

```
begin
  $w := p_0^k$;
  for $(i \in T(w))$ do $\bar{c}_{ik} := \bar{c}_{ik} - \gamma_k^*$;
  for $(i \in T(k))$ do
  begin
    for $h := \underline{d}$ up to $H$ do $P_{T(k)}(i,k,h) := P_{T(k)}(i,k,h) - \gamma_k^*$;
  end
end
```

Consequently, we have the following main results.

Theorem 2. *Algorithm 1 solves the LATN Design problem (D) in $O(n^2H)$, where $H = h^m$.*
Proof: The correctness of Algorithm 1 is given in Theorem 1. Because of Lemma 3.1, we only need to estimate the total complexity for the procedures **Find_γ_k^*** and **Update_γ_k^***. We will show in Theorem 3 that for a given k, $P_{T(k)}(i,k,h)$ can be computed in $O((|T(k)| - |T(p_k^i)|)H)$ time by starting from $P_{T(k)}(i,p_k^i,h)$ for $h = \underline{d}, \underline{d}+1, \cdots, H$. Therefore, each **Find_γ_k^*** takes $O(|T(k)|H) + O(\sum_{i \in T(k)\setminus\{k\}} (|T(k)| - |T(p_k^i)|)H) + O(nm)$ time. Hence, the total complexity of Algorithm 1 for performing the procedure **Find_γ_k^*** is

$$\begin{aligned} & O\{\sum_{k=0}^{n}(|T(k)|H + \sum_{i \in T(k)\setminus\{k\}} (|T(k)| - |T(p_k^i)|)H)\} \\ = \; & O\{\sum_{i=0}^{n}(|T(i)|H + \sum_{k \in P[i,0]\setminus\{i\}} (|T(k)| - |T(p_k^i)|)H)\} \\ = \; & O\{\sum_{i=0}^{n}(|T(i)| + |T(0)| - |T(i)|)H\} \\ = \; & O(n^2H). \end{aligned}$$

Moreover, since each **Update_γ_k^*** takes $O(|T(p_0^k)| + |T(k)|H) = O(nH)$, the total time taken by Algorithm 1 for performing the procedure **Update_γ_k^*** is $O(n^2H)$. Therefore, the overall complexity taken by Algorithm 1 for solving the LATN Design problem is $O(n^2H)$. □

5 Solving the Subproblem for the LATN Design Problem

Let T be a subtree of $T(k)$ rooted at k and $i \in T$ be the center of the tree T. For any $v \in T$, we define

$$P_T(i,v,h) = \max \sum_{j \in T} \bar{c}_{ij}x_j \tag{5.1}$$

$$(CTKP) \qquad \text{s.t.} \quad x_{p_j^i} \geq x_j, \quad j \in T \setminus \{i\} \tag{5.2}$$

$$\sum_{j \in T} d_j x_j \leq h \tag{5.3}$$

$$x_v = 1 \tag{5.4}$$

$$x_j \in \{0,1\}. \tag{5.5}$$

Then our objective is to find the optimal value $P_{T(k)}(i,k,h)$ of the Centered Tree Knapsack Problem $(S_k^i(h))$ using a dynamic programming algorithm.

We can solve $(S_k^i(h))$ by the depth-first dynamic programming algorithm by applying the following recursive rules which are similar to ones for the TKP discussed by Cho and Shaw [9];

1. (Initialization)

$$P_{\{i\}}(i,i,h) = \begin{cases} \bar{c}_{ii} & \text{if} \quad h \geq d_i \\ -\infty & \text{otherwise} \end{cases}$$

2. (Forward move to expand T)
For $v \notin T$ and $p_v^i \in T$,

$$P_{T \cup \{v\}}(i,v,h) = P_T(i,p_v^i,h-d_v) + \bar{c}_{iv} \quad \text{if} \quad h \geq \sum_{j \in P[i,v]} d_j$$

3. (Backward move to visit p_v from v)
For $v \in T \setminus P[i,k]$,

$$P_T(i,p_v,h) = \max\{P_{T \setminus T(v)}(i,p_v,h), P_T(i,v,h)\}.$$

Suppose that we have solved a TKP $(S_i^i(h))$ on $T(i)$. It is important to observe that, for all $k \in P[i,0] \setminus \{i\}$, the problem $(S_k^i(h))$ can be solved from an optimal solution of $(S_{p_k^i}^i(h))$ as follows:

Step 1) Perform a 'forward move' from node p_k^i to node k, by applying

$$P_{T(p_k^i) \cup \{k\}}(i,k,h) = P_{T(p_k^i)}(i,p_k^i,h-d_k) + \bar{c}_{ik} \quad \text{if} \quad h \geq \sum_{j \in P[i,k]} d_j$$

Step 2) Apply the depth-first dynamic programming algorithm by using the above recursive rules on $T(k) \setminus T(p_k^i)$ to find $P_{T(k)}(i,k,h)$.

To find the optimal solution for $(S_k^i(h))$, we need to define the so-called *index* $I_T(i,v,h)$ corresponding to $P_T(i,v,h)$ for all $v \in T(k)$ as follows:

1. (Initialization)

$$I_{\{i\}}(i,i,h) = 1 \quad \text{if} \quad h \geq d_i$$

2. (Forward move to expand T)
For $v \in P[i,k]$ and $p_v \notin T$,

$$I_{T \cup \{p_v\}}(i,p_v,h) = \begin{cases} 1 & \text{if} \quad h \geq \sum_{j \in P[i,p_v]} d_j \\ 0 & \text{otherwise} \end{cases}$$

3. (Backward move to visit p_v from v)
For $v \in T \setminus P[i,k]$,

$$I_T(i,v,h) = \begin{cases} 1 & \text{if} \quad P_{T \setminus T(v)}(i,p_v,h) < P_T(i,v,h) \\ 0 & \text{otherwise.} \end{cases}$$

We now present an algorithm which solves the Centered Tree Knapsack Problem ($S_k^i(h)$) recursively, starting from ($S_{p_k^i}^i(h)$). It is important to notice that as the 'forward move' follows the Depth First Search order and the 'backward move' follows its reverse order in Algorithm 1, the value $P_T(i, v, h)$ can be uniquely determined by (i, v, h), that is, T can be uniquely determined by v (we denote it as T^v). If v has not been visited by a 'backward move', then $T^v = \{k, k+1, \cdots, v\}$. Otherwise, $T^v = T^u$, where u is a successor of v from which a 'backward move' visits v. Therefore, we can omit the T from the notation and simply use $P(i, v, h)$ and $I(i, v, h)$ in implementing the algorithm. This result comes from the nature of the depth-first dynamic programming procedure.

```
Algorithm 2.   CTKP($i, k, pred$);
{comment: $i = center$,  $k = root$,  $pred = p_k^i$}

begin
  if ($k \neq 0$)   then
    $d\_path := \sum_{i \in P[i,pred]} d_i$;
  else
    $d\_path := 0$;
  end if
  if ($i \neq k$)   then
    Forward_Move($i, pred, k$);
  end if
  $j := k + 1$;
  while ($j \leq LAST(k)$)   do
  begin
    if ($j \neq pred$)   then
      Forward_Move($i, p_j, j$);
      if ($j = LAST(j)$)   then        {comment: node $j$ is a leaf node}
        $w := j$;
        do
          Backward_Move($i, w, p_w$);
          $w := p_w$;
        while ($LAST(w) = j$ and $w \neq k$)
        {comment: $w$ has no successor $t$ such that $t > j$ and $w \neq k$}
      end if
      $j := j + 1$;
    else
      $j := LAST(pred) + 1$;
    end if
  end
end
```

```
Procedure 2.1   Forward_Move($i, j, k$);

begin
  $d\_path := d\_path + d_k$;
  for $h := \underline{d}$ up to $d_k - 1$    do        $P(i, k, h) := -\infty$;
  for $h := d_k$ up to $H$    do
  begin
    if ($d\_path \leq h$)   then
      $P(i, k, h) := P(i, j, h - d_k) + \bar{c}_{ik}$;
      if ($p_j = k$)   then
        $I(i, k, h) := 1$;
      end if
    else
      $P(i,k,h) := -\infty$;
    end if
  end
end
```

Procedure 2.2 **Backward_Move**(i, j, k);

```
begin
  $d\_path := d\_path - d_j$;
  for $h := \underline{d}$ up to $H$ do
  begin
    if ($P(i,k,h) \geq P(i,j,h)$) then
      $I(i,j,h) := 0$;
    else
      $P(i,k,h) := P(i,j,h)$;
      $I(i,j,h) := 1$;
    end if
  end
end
```

With the above procedures, we can prove the following theorem.

Theorem 3. *If $i = k$, then $(S_i^i(H))$ can be solved in $O(|T(i)|H)$ time, where $|T(i)|$ is the number of nodes in the tree $T(i)$. If $i \neq k$, then the problem $(S_k^i(H))$ can be solved by Algorithm 2 in $O((|T(k)| - |T(p_k^i)|)H)$, provided that the optimal value $P(i, p_k^i, H)$ of $(S_{p_k^i}^i(H))$ is given.*

Proof: If $i = k$, then $(S_i^i(H))$ is the Tree Knapsack Problem and therefore it can be solved in $O(|T(i)|H)$ time (see Cho and Shaw [9]). When we reroot the tree to node i, the set of nodes in $T(k)$ can be considered as the set of nodes extended from $T(p_k^i)$ by attaching a subtree $T(k) \setminus T(p_k^i)$ as shown in Figure 1. Hence, the correctness of this algorithm is obvious from the correctness of the algorithm for the TKP centered at the root (see Cho and Shaw [9]). As **Forward_Move**($\cdot$) and **Backward_Move**($\cdot$) require $O(H)$ time, respectively, we only need to count the number of 'forward moves' and 'backward moves'. Clearly, every node in $T(k) \setminus T(p_k^i)$ is visited by **Forward_Move**($\cdot$) and if a 'forward move' moves into a node $v(\neq k)$, then there must be a 'backward move' which moves out from the node v. Hence, the total number of 'forward moves' and 'backward moves' is $O(|T(k)| - |T(p_k^i)|)$. Therefore, the overall complexity of Algorithm 2 is $O((|T(k)| - |T(p_k^i)|)H)$. □

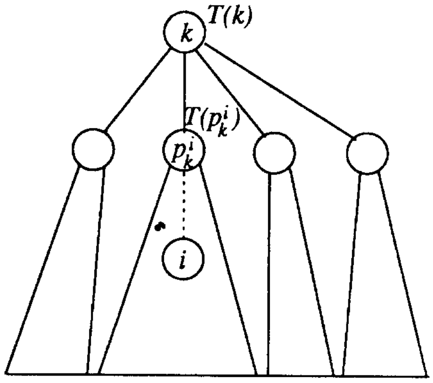

Figure 1: $T(k) = T(p_k^i) \cup (T(k) \setminus T(p_k^i))$

By following Theorem 3, we can see that if $(S_{p_k^i}^i(H))$ has been solved, then γ_k^i is obtained in $O((|T(k)| -$

$|T(p^i_k)|)H)$, where $H = h^m$.

We now rewrite the procedure **Find_γ^*_k** for finding the LMV reduced cost γ^*_k for the LATN Design problem by using the algorithm **CTKP**(i,k,v) to find $P(i,k,h)$ for all $h = \underline{d}, \underline{d}+1, \cdots, H$.

```
Procedure 1.1′  Find_$\gamma^*_k$;

begin
  CTKP($k, k, -1$);      {comment: as $p^k_k$ is undefined, we set $p^k_k = -1$}
  if ($k \neq 0$)  then
    $pred := k + 1$;     {comment: $pred = p^i_k$}
    for $i := k + 1$ up to $LAST(k)$   do
    begin
      if ($i > LAST(pred)$)  then
        $pred := i$;
      end if
      CTKP($i, k, pred$);
    end

        $\gamma^*_k := \max_{i \in T(k)} \max_{1 \leq t \leq m} \{P(i,k,h^t) - \hat{F}^t_i\}$;

  end if
end
```

6 Computational Results and Data Structure

In this section, we report the computational results of the Limited Column Generation procedure for the LATN Design problem which incorporates the depth-first dynamic programming algorithm presented in Section 4. The algorithms were coded in C language and run on a SUN SPARC 1000 workstation. We use two of one-dimensional arrays, the predecessor p_i and the last node $LAST(i)$ in subtree $T(i)$, to represent the topology of the tree. The two-dimensional array B_{ij} is used to store the optimal basis and the three-dimensional arrays $P(i,v,h)$ and $I(i,u,h)$ are used in dynamic programming procedure for the LATN Design problem. We have tested our algorithm on a set of randomly generated problems. To generate a tree randomly, we specified the total number n of nodes in the tree first. Starting from the root node, we randomly generated the number of successors of each node from an interval $[0, \log_2 n]$ in BFS order until the total number of nodes was met.

Two types of concentrator capacities $h^1 < h^2 < \cdots < h^m = H$ were generated randomly in the interval $[H/2, H]$. For each node i, we generated $\hat{c}_i \in [1,50], \hat{F}^1_i < \hat{F}^2_i \cdots < \hat{F}^m_i \in [1,1000]$, and $d_i \in [1,50]$ if $H = 500$, $d_i \in [1,100]$ if $H = 1000$ randomly. For each arc i, we also randomly generated $c_i, F_i \in [1,50]$.

We fixed the number of concentrator types $m = 3$ and tested eight problems generated from the ranges defined above. We averaged the CPU time over those eight randomly generated test problems and also reported the worst and the best CPU time obtained in each case. The CPU time reported here is the sum of the user time and the system time and is measured in seconds. The results are shown in Table 1.

As shown in the table, the CPU time increases approximately at a quadratic rate with respect to n and at a linear rate with respect to H. Because of the nature of dynamic programming, the complexity of our algorithm is actually $\Omega(n^2H)$ for the LATN Design problem, and we can see from Table 1 that the running time for the worst case in each category is less than twice that of the average case, whereas the running time for the best case is about one half of that of the average case.

Table 1. Computational results for the LATN Design problems

n	H	LATND		
		worst	average	best
20	500	1.06	0.72	0.43
	1000	1.94	1.31	0.77
30	500	2.28	1.41	1.00
	1000	4.25	2.41	1.61
50	500	6.94	3.95	2.68
	1000	12.28	9.75	5.87
80	500	17.48	9.09	6.14
	1000	11.29	9.20	6.38
100	500	26.61	13.79	8.18
	1000	50.44	33.94	23.36
150	500	63.72	32.16	14.77
	1000	112.29	66.92	22.20

7 Conclusions

In this paper we have successfully developed an efficient implementation of the Limited Column Generation approach for the Local Access Telecommunication Network (LATN) Design problem which was originally proposed by Shaw [18]. Our computational results indicate that the Limited Column Generation is a very effective and efficient method for this kind of problems. The LATN Design model has a very special combinatorial structure. For example, an appropriate Lagrangian relaxation approach can yield a bound exactly equal to the optimal value of the integer program (see Shaw [20]). In general, this model is the special case of the general tree–partitioning problems (see Barany, Edmonds and Wolsey [5]), or the general facility location problems (see Shaw [21]); the Limited Column Generation method is a special-purpose algorithm for this class of problems, which essentially decomposes the problem into a series of subproblems.

Because of the NP–complete nature of the original problem, the subproblem itself is a NP–complete discrete optimization problem. Usually, the convex hull of the integer feasible solutions can be described either by its facets through valid inequalities or by its vertices directly. In this paper, we adopt the latter approach through dynamic programming. Such an approach is natural and straightforward, but proven to be very effective. As a Knapsack Problem is a special case of the Tree Knapsack Problem (TKP), the complexity of $O(nH)$ for the TKP (and CTKP) might be the best we can expect.

References

[1] E.H. Aghezzaf, T.L. Magnanti, and L.A. Wolsey. Optimizing constrained subtrees of trees. Technical report, Center for Operations Research & Econometrics, Universite Catholique De Louvain,Louvain-La-Neuve, Belgium, 1992.

[2] A. Balakrishnan, T.L. Magnanti, A. Shulman, and R.T. Wong. Models for planning capacity expansion in local access telecommunication networks. *Annals of Operations Research*, 33:239–284, 1991.

[3] A. Balakrishnan, T.L. Magnanti, and R.T. Wong. A decomposition algorithm for expanding local access telecommunications networks. Technical report, Sloan School of Management, M.I.T., Cambridge, MA, 1991.

[4] I. Barany, J. Edmonds, and L.A. Wolsey. Packing and covering a tree by subtrees. *Combinatorica*, 6:221–233, 1986.

[5] D. Bienstock. A computational experience with an effective heuristic for some capacity expansion problems in local access networks. *Telecommunication Systems*, 1:379–400, 1993.

[6] D. Bienstock. A lot-sizing problem on trees,related to network design. *Mathematics of Operations Research*, 18:402–422, 1993.

[7] G. Cho. *Limited Column Generation and Related Methods for Local Access Telecommunication Network Design and Expansion - Formulation, Algorithm, and Implementation.* PhD thesis, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1994.

[8] G. Cho and D.X. Shaw. A depth-first dynamic programming procedure for tree knapsack problem. Technical report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1994.

[9] V. Chvatal. On certain polytopes associated with graphs. *J. Combin. Theory B*, 18:138–154, 1975.

[10] W. Cook. Integer programming solutions for capacity expansion of the local access network. Technical Report Manuscript, Bell Communication Research, 1990.

[11] F. Gavril. The intersection graphs of subtrees in tree are exactly the chordal graphs. *J. Combin. Theory B*, 16:47–56, 1974.

[12] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[13] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[14] D.S. Johnson and K.A. Niemi. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research*, 8:1–14, 1983.

[15] L. Lovasz. A characterization of perfect graphs. *J. Combin. Theory B*, 13:95–98, 1972.

[16] J.A. Lukes. Efficient algorithm for the partitioning of trees. *IBM Journal of Research and Development*, 18:214–224, 1974.

[17] D.X. Shaw. Limited column generation technique for several telecommunications network design problems. Technical report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1993.

[18] D.X. Shaw. A pseudo-polynomial algorithm for single-item capacitated economic lot sizing with general cost structures. Technical report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1994.

[19] D.X. Shaw. Reformulation and column generation for several telecommunications network design problems. In *Proceedings of the 2nd International Telecommunication Conference*, Nashville, Tennessee, 1994.

[20] D.X. Shaw. Reformulation, column generation and lagrangian relaxation for local access network design problems. Technical report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1994.

[21] D.X. Shaw. A unified approach for a class of general facility location problems: Limited column generation. Technical report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1994.

[22] D.X. Shaw and G. Cho. A branch-and-bound procedure for the tree knapsack problem. Technical report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1994.