Journal of the Korean
OR/MS Society
Vol. 23, No 2, June 1998

A Decomposition Algorithm for a Local Access Telecommunication Network Design Problem

Geon Cho*

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

Over the past two decades telecommunication industry has been undergoing rapid and fundamental change. Technological innovation has created new market opportunities and high customer demand for enhanced products and services such as multimedia applications. The government has challenged private sectors to participate in modernizing the nation's entire information infrastructure by building the "information super-highway". As a result, many telecommunication service providers are in

* School of Business Administration, Chonnam National University, Kwangju, Korea

the process of upgrading and expanding their facilities and services. Since the huge investment is required in this process, finding a cost-effective network design technique should be the critical issue to service providers. In this paper, we propose an efficient algorithm for solving such a network design problem, in particular, in 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 located at the root node. 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 is installed in a customer node and compresses the incoming traffic into a higher frequency signal that requires fewer outgoing lines. It is assumed that the compressed signal requires a dedicated cable point-to-point routing to the switching center. Then the objective of the LATN design problem is to find the best concentrator installation places so that all of the demands can be fully satisfied by the installed concentrators with the minimum total cost.

Recently, many researchers have been working on the LATN design problem, which is an NP-complete problem since it includes the knapsack problem as a special case. A discussion of the modeling issues can be found in Balakrishnan *et al.* [2], and a Lagrangian relaxation based approach has been developed by Balakrishnan, Magnate and Wong [3]. Aghezzaf, Magnanti and Wolsey [1] have studied the combinatorial structure of the model. Related but different model has been formulated by Cook [9], and a solution procedure and related knapsack-type problem has been studied by Bienstock [4,5].

In this paper, the LATN design problem is first formulated as a tree-partitioning problem with an exponential number of variables, each of which corresponds to a subtree(see Shaw [17]). Chvatal [8] and Gavril [10] have shown that the polyhedron of its linear programming relaxation is integral. Hence, the tree-partiti -oning problem can be simply reduced to linear program(with an exponential number of varia-bles, though). Then, we incorporate a dedicated column generation procedure developed by Shaw [16,17], the so-called Limited Column Gene-ration procedure, to solve the linear program 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 problem. Our computational experiments indicate that the Limited Column Generation seems to be a very promising approach for this model. For example, 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 [6], Johnson and Niemi [13], Lukes [15], and Shaw [18]. Johnson and Niemi [13] have 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 [7] have 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's algorithm [13], where H is the maximum concentrator capacity.

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 [17] for the TKP runs in $O(nH^2)$ time and can be combined with the Limited Column Generation procedure to slove the LATN design problem in $O(n^2H^2)$.

On the other hand, 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 [7] results in an $O(n^3H)$ algorithm for the LATN design problem because $n+1$ pivots are required and prior to each pivot, n number of the so-called Centered Tree Knapsack Problem(CTKP) whose complexity is $O(nH)$ time have to be solved. However, we obtain an algorithm for the problem with the overall complexity of $O(n^2H)$. The idea is that, 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 contribution of this paper is to develope 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.

This paper is organized as follows. We describe the LATN design problem in Section 2 and present the tree-partitioning formulation for the LATN design problem in Section 3. We then introduce the Limited Column Generation procedure for the LATN design problem in Section 4. In Section 5 we present an on-line reclusive procedure for computing all the LMV reduced costs. The implementation detail for the LATN design problem is presented in Section 6, and our computational results are given in Section 7.

2. Problem Description

In this section, we describe the LATN design problem in detail. Let $\hat{T}=(V, E)$ be an undirected tree rooted at the switching center labeled as node 0, representing the layout of the LATN, where $V = \{0, 1, \ldots, n\}$ is the set of nodes and $E=\{(p_i, i) \mid i \in V \setminus \{0\}\}$ is the set of arcs in $\hat{T}$. Here, p_i denotes the predecessor of node i. Then, without loss of generality, we assume that all nodes in $\hat{T}$ are labeled by the Depth-First-Search(DFS) order. 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. Each non-root node is also considered as a candidate installation place of a concentrator.

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 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.* [2] and Balakrishnan, Magnanti, and Wong [3]. Let $P[i, j]$ be a unique path on tree $\hat{T}$ from node i to node j. Then these assumptions 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. *Non-bifurcated routing assumption*: all circuits(demands) from one customer node must use the same cable sections, i.e., all demands on a node must be fully satisfied by only one concentrator.

3. *Contiguity assumption*: if all demands on node j are satisfied by a concentrator located at node i, then all demands on nodes over the path $P[i, j]$ are also satisfied 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 satisfied by that concentrator.

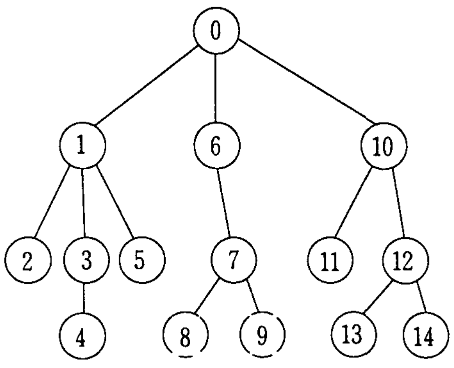
It is assumed that at most one

concentrator can be installed at each node and each concentrator has m kinds of capacities and there are the fixed concentrator cost corresponding to each capacity and the variable concentrator cost. The fixed concentrator cost represents the concentrator purchase cost and installation cost as well as other infrastructure investment and the variable concentrator cost represents the operating expenses. In this paper, we assume that the variable concentrator costs are identical for all kinds of concentrator capacities. Moreover, it is assumed that there are the fixed arc cost and the variable arc cost on each arc. The fixed arc cost represents the expenses for digging trenches and laying pipes on each arc and the variable arc cost represents the cable purchasing and maintenance cost.

Based on the above information, the LATN design problem determines where to locate concentrators and how many concentrators should be installed, so that all of the demands can be fully satisfied by the installed concentrators with the minimum total cost.

For a given solution for the LATN design problem, it can be easily seen that both the set of all nodes whose demands are satisfied from one concentrator and the set of related arcs form a subtree of the given tree network because of the contiguity assumption. For example, let us consider the following tree network given in Figure 1.

One of the solution for the LATN design

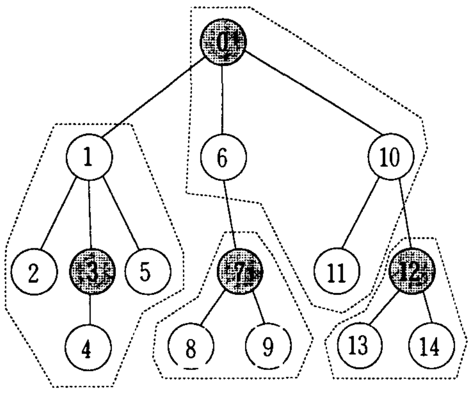


[Figure 1] An Example of Local Access Telecommunication Network

problem in Figure 1 is to locate concentrators on nodes 3, 7, and 12 so that all demands are satisfied as follows:

1. all demands at node 1, 2, 3, 4, and 5 are satisfied by the concentrator at node 3.
2. all demands at node 7, 8, and 9 are satisfied by the concentrator at node 7.
3. all demands at node 12, 13, and 14 are satisfied by the concentrator at node 12.
4. all demands at node 0, 6, 10, and 11 are satisfied by the switching center at node 0.

The above solution divides the tree network into four components based on three concentrators and the switching center. Moreover, each component forms a subrtee as shown in Figure 2. Consequently, the LATN design problem can be formulated as a tree-partitioning problem. We present the tree-partitioning formulation for the LATN design problem in the next section.



[Figure 2] A Solution for the LATN Design Problem in FIgure 1

3. Tree Partitioning Formulation

Let p^i_j be the predecessor of node j with respect to node i, which is defined as the first node followed by node j on the path $P[j,i]$. Since there is one-to-one correspondence between E and $V\backslash\{0\}$, we are able to rename arc (p_i, i) and arc (i, p_i) as arc i and arc $-i$, respectively. We denote d_i as the demand at node i for $i=1,2,\cdots,n$. Let h^t be the concentrator capacity for $t=1,2,\cdots,m$ and $h^1 < h^2 < \cdots < h^m$. Let $\hat{F}^t_i$ be the fixed concentrator cost corresponding to the concentrator capacity h^t at node i and $\hat{c}_i$ be the variable concentrator cost, where $i=0,1,2,\cdots,n$ and $t=1,2,\cdots,m$. Here, we assume that $\hat{c}_0=0$, since no concentrator can be installed at the root node 0. Let $\hat{F}_i$ be the fixed arc cost and c_i be the variable arc cost, where $i=\pm1,\pm2,\ldots,\pm n$. We define the *routing cost* c_{ij} from node j to a concentrator location i (or the switching center 0) as follows: for $i=0,1,2,\ldots,n$ and $j=1,2,\ldots,n$,

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

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 $\hat{T}$ 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^i_T of T with the center i as

$$c^i_T=\sum_{j\in T}c_{ij}+\hat{F}^{t*}_i, \tag{3.2}$$

where

$$t^*=\min\{t \mid \sum_{j\in T}d_j\le h^t,\ \ t=1,2,\ldots,m\}.$$

Let $c_T=\min_{i\in T}c^i_T$ be the minimum cost of routing all the demands at nodes in T to a common concentrator i. Let $|T|$ be the number of nodes in T. Then, for a fixed center $i \in T$, we define a vector of decision variable $\xi_T = (\xi_j) \in \{0,1\}^{|T|}$ as follows:

$$\xi_j = \begin{cases} 1 & \text{if node } j \text{ is served by the center } i \\ 0 & \text{otherwise.} \end{cases}$$

Then, the LATN design problem can be formulated as the following tree-partitioning problem:

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

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

It is well known that the intersection graph of G is a chordal graph and, of course, a perfect graph(for detail, see Chavatal [8], Gavril [10], Golumbic [12], and Lovasz [14]), and the linear programming relaxation of (P) has all integral vertices. Shaw [16] presents an elementary proof that the linear programming relaxation of (P) always has an integer optimal solution. Therefore, (P) can be written as

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

Note that there are exponential number of decision variables in (P), since there are exponentially many subtrees T of $\hat{T}$. Therefore, a natural way to solve this problem is to use a column generation technique, which was first proposed by Gilmore and Gomory [11]. However, because of the special structure of our problem, much better solution technique can be achieved. In the next section, we introduce the so-called Limited Column Generation procedure developed by Shaw [16].

4. Limited Column Generation Procedure

Shaw [16] 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. Therefore, before we describe the Limited Column Generation procedure for solving (P), we first need to define the LMV reduced cost which plays central role in the procedure.

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 $i = 0,1,2,\dots,n$. Then the reduced cost of a variable ξ_T is obtained by $\gamma_T = \sum_{i \in T} \pi_i - c_T$. The 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 $r(T) = \min\{k \mid k \in T\}$ is the root of T (recall that the node in $\hat{T}$ is labeled by DFS order).

Initially, we partition $\widehat{T}$ into singleton sets $\{k\}$ for $k = 0,1,2,\ldots,n$ (i.e., $\widehat{T} = \cup_{k=0}^{n}\{k\}$). 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

$$\begin{aligned}\pi_k &= c_{\{k\}} \\ &= d_k \widehat{c}_k + \widehat{F}^{t_k^*}_k,\end{aligned}$$

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

Let $1_{T_k^*} = (\nu_i) \in R^{n+1}$,

where $\nu_i = \begin{cases} 1 & \text{if } i \in T_k^* \\ 0 & \text{otherwise.}\end{cases}$

Then, the Limited Column Generation procedure developed by Shaw [16] 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 o 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 $n+1$ simlex pivots.*

The proot of Theorem 1 can be found in Shaw [16]. 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 $\{\pi_i \mid i \in T(k)\}$, which is updated at each iteration, we show in the next section how the set $\{\gamma_k^* \mid k = n, n-1, \ldots, 0\}$ can be computed recursively in an on-line fashion. The procedure **Find**_γ_k^* and **Update**_γ_k^* are also given in the next section.

Lemma 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. Let $LAST(k) = \max\{j \mid j \in T(k)\}$ be the last node in $T(k)$, where $T(k) = \{j \mid k \in P[0, j]\}$ is the complete subtree of $\hat{T}$ rooted at node k. 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 1.1 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
```

Notice that the center of the subtree rooted 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.

5. Computing the LMV Reduced Cost

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

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

where $\gamma_k^i = \max_{\substack{T \ni i \\ r(T)=k}} \Big(\sum_{j \in T} \pi_j - c_T^i \Big)$. (5.1)

Given a node k and center $i \in T(k)$, let x_j be the decision variable defined by

$$x_j = \begin{cases} 1 & \text{if node } j \text{ is served by the center } i \\ 0 & \text{otherwise.} \end{cases}$$

Then we define a problem as follows:

$$
(S_k^i(h))\quad
\begin{aligned}
\max \quad & \sum_{j\in T(k)} \overline{c_{ij}}\, x_j \\
\text{s.t.} \quad & x_{p_j^i} \ge x_j,\; j\in T(k)\setminus\{i\} \\
& \sum_{j\in T(k)} d_j x_j \le h \\
& x_k = 1 \\
& x_k \in \{0,1\},
\end{aligned}
$$

where $\overline{c_{ij}} = \pi_j - c_{ij}$ 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 6 for detailed discussion about notation). Then, it follows from (3.1)-(3.2) and (5.1) that γ_k^i for the LATN design problem can be obtained as follows:

$$\gamma_k^i = \max_{1\le t\le m} \{P_{T(k)}(i,k,h^t) - \hat{F}_i^t\}. \qquad (5.2)$$

Hence, a critical step in computing γ_k^i for the LATN design problem is to solve a subproblems $(S_k^i(h))$. When $i=k$, $(S_i^i(h))$ is the TKP discussed by Cho and Shaw [7], Johnson and Niemi [13], Lukes [15], Shaw [18], and Shaw and Cho [19]. In particular, Cho and Shaw [7] 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 $(S^i_{p_k^i}(h))$ 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 (5.2) and finally obtain γ_k^*. However, all $(S_k^i(h))$ are related to $\{\pi_i | i=0,1,2,\dots n\}$, 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 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 1.2 **Find_** γ_k^*;

```
begin
  compute $P_{T(k)}(k,k,h)$ for all $h=\overline{d}, \overline{d}+1,\dots,H$;
  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 $h=\overline{d}, \overline{d}+1,\dots,H$;
    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 1.3 **Update_** γ_k^*;

```
begin
  $w := p_o^k$;
  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 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 $h = \underline{d}, \underline{d}+1, \ldots, H$. 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

$$O\{\sum_{k=0}^{n}(|T(k)|H + \sum_{i \in T(k) \setminus \{k\}}(|T(k)| - |T(p_k^i)|)H)\}$$

$$= O\{\sum_{i=0}^{n}(|T(i)|H + \sum_{k \in P[i,0] \setminus \{i\}}(|T(k)| - |T(p_k^i)|)H)\}$$

$$= O\{\sum_{i=0}^{n}(|T(i)| + |T(0)| - |T(i)|)H\}$$

$$= O(n^2H).$$

Moreover, since each **Update_** γ_k^* takes $O(|T(p_o^k)| + |T(k)|H) = O(nH)$, 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 $O(n^2H)$.

6. 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

$$P_T(i, v, h) = \max \sum_{j \in T} \bar{c}_{ij} x_j \qquad (6.1)$$

$$\text{s.t. } x_{p_j^i} \geq x_j, \quad j \in T \setminus \{i\} \qquad (6.2)$$

$$(CTKP) \qquad \sum_{j \in T} d_j x_j \leq h \qquad (6.3)$$

$$x_v = 1 \qquad (6.4)$$

$$x_j \in \{0, 1\}. \qquad (6.5)$$

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 [7]:

1. (Initialization)

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

2. (Forward move to expand T)

For $v\notin T$ and $p_v^i\in T$,

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

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

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

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)

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

2. (Forward move to expand T)

For $v\in P[i,k]$ and $p_v\notin T$,

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

3. (Backward move to visit p_v from v)

For $v\in T\setminus P[i,k]$,

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

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 $T^v=\{k,k+1,\ldots,v\}$. 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
    $d\_path := \sum_{j \in P[i, pred]} d_j$;
  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
```

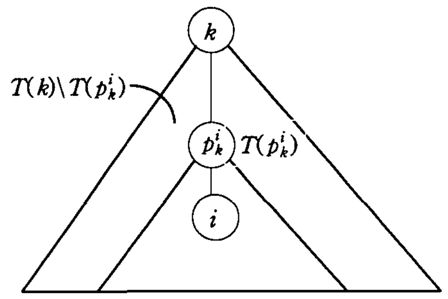
With the above procedures, we can prove the following theorem.

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 [7]). 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 3. 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 [7]). As **Forward_Move**(·) and **Backward_Move**(·) 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**(·) 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 $O((|T(k)| - |T(p_k^i)|)H)$.

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 $O((|T(k)| - |T(p_k^i)|)H)$, where $H = h^m$.

We now rewrite the procedure **Find**_γ_k^* for finding the LMV reduced cost γ_k^* for the



[Figure 3] $T(k) = T(p_k^i) \cup (T(k) \setminus T(p_k^i))$

LATN design problem by using the algorithm **CTKP** (i, k, v) to find $P(i, k, h)$

for all $h=\underline{d},\ \underline{d}+1,\ldots,H$.

```
Procedure 1.2' 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: $pred = p_k^i$}
     for $i := k+1$ up to $LAST(k)$ do
     begin
       if $(i > LAST(pred))$ then
         $pred := i$;
       end if
       CTKP $(i, k, pred)$;
     end
     $\gamma_k^* := \max_{i \in T(k)} \max_{1 \le t \le m} \{P(i, k, h^t) - \widehat{F}_i^t\}$;
   end if
 end
```

7. 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 6. 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 $h^1 < h^2 < \ldots < h^m = H$ were generated randomly in the interval $[H/2, H]$. For each node i, we generated $\widehat{c}_i \in [1,50]$, $\widehat{F}_i^1 < \widehat{F}_i^2 < \ldots < \widehat{F}_i^m \in [1,1000]$, 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

〈Table 1〉 Computational results for the LATN design problems

n	H	CPU time			number of concentrators
		worst	average	best	
20	500	1.06	0.72	0.43	4
	1000	1.94	1.31	0.77	7
30	500	2.28	1.41	1.00	8
	1000	4.25	2.41	1.61	10
50	500	6.94	3.95	2.68	12
	1000	12.28	9.75	5.87	18
80	500	17.48	9.09	6.14	17
	1000	11.29	9.20	6.38	28
100	500	26.61	13.79	8.18	23
	1000	50.44	33.94	23.36	38
150	500	63.72	32.16	14.77	35
	1000	112.29	66.92	22.20	52

Note: $d_i \in [1,50]$ if $H=500$, $d_i \in [1,1000]$ if $H=1000$

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. We also reported the average number of concentrators installed for each class of problems. 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.

8. Conclusions

In this paper we have formulated the Local Access Telecommunication Network (LATN) design problem into a tree-partitioning problem with an exponential number of variables and solved the problem by the so-called Limited Column Generation procedure in $O(n^2H)$ time, where n is the number of nodes in the network and H is the maximum concentrator capacity. We have also developed an efficient way of recursively computing all the LMV reduced costs which play an important role in the Limited Column Generation procedure. The LMV reduced cost can be obtained by solving the so-called Centered Tree Knapsack Problem (CTKP), a subproblem of the LATN design problem. Our computational results indicate that the Limited Column Generation is a very promising method for the LATN design problem.

REFERENCES

[1] E.H. Aghezzaf, T.L. Magnanti, and L.A. Wolsey, "Optimizing Constrained Subtrees of Trees", *Mathematical Programming*, Vol. 71(1995), pp.113-126.

[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*, Vol.33(1991), pp.239-284.

[3] A. Balakrishnan, T.L. Magnanti, and R.T. Wong, "A decomposition algorithm for expanding local access telecommunications networks", *Operations Research*, Vol.43(1995), pp.43-57.

[4] D. Bienstock, "A computational experience with an effective heuristic for some capacity expansion problems in local access networks", *Telecommunication Systems*, Vol.1 (1993), pp.379-400.

[5] D. Bienstock, "A lot-sizing problem on trees, related to network design", *Mathematics of Operations Research*, Vol.18(1993), pp.402-422.

[6] 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.

[7] G. Cho and D. X. Shaw, "A depth-first dynamic programming algorithm for the tree knapsack problem", *INFORMS Journal on Computing*, Vol.9, No.4(1997), pp.431-438.

[8] V. Chvatal, "On certain polytopes associated with graphs", *Journal of Combinatorial Theory B*, Vol.18(1975), pp.138-154.

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

[10] F. Gavril, "The intersection graphs of subtrees in tree are exactly the chordal graphs", *Journal of Combinatorial Theory B*, Vol.16(1974), pp.47-56.

[11] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting-stock problem", *Operations Research*, Vol. 9(1961), pp.849-859.

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

[13] D. S. Johnson and K. A. Niemi, "On knapsacks, partitions, and a new dynamic programming technique for trees", *Mathematics of Operations Research*, Vol.8, No.1 (1983), pp.1-14.

[14] L. Lovasz, "A characterization of perfect graphs", *Journal of Combinatorial Theory B*, Vol.13(1972), pp.95-98.

[15] J. A. Lukes, "Efficient algorithm for the partitioning of trees", *IBM Jorunal of Research and Development*, Vol.18(1974), pp.214-224.

[16] 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.

[17] D. X. Shaw, "A pseudo-polynomial Algorithms for Single-item Capacitated Economic Lot Sizing with General Cost Structures",

Technical Report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1994.

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

[19] 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.