Journal of the Korean Institute
of Industrial Engineers
Vol.22, No.3, September, 1996

A Pseudopolynomial-time Algorithm for Solving a Capacitated Subtree of a Tree Problem in a Telecommunication System

Geon Cho*

Abstract

For a tree T rooted at a concentrator location in a telecommunication system, we assume that the capacity H for the concentrator is given and a profit c_v and a demand d_v on each node v of T are also given. Then, the capacitated subtree of a tree problem (CSTP) is to find a subtree of T rooted at the concentrator location so as to maximize the total profit, the sum of profits over the subtree, under the constraint satisfying that the sum of demands over the subtree does not exceed H. In this paper, we develop a pseudopolynomial-time algorithm for CSTP, the depth-first dynamic programming algorithm. We show that a CSTP can be solved by our algorithm in $\theta(nH)$ time, where n is the number of nodes in T. Our algorithm has its own advantage and outstanding computational performance incomparable with other approaches such as CPLEX, a general integer programming solver, when it is incorporated to solve a Local Access Telecommunication Network design problem. We report the computational results for the depth-first dynamic programming algorithm and also compare them with those for CPLEX. The comparison shows that our algorithm is competitive with CPLEX for most cases.

1. Introduction

Given an undirected tree $T = (V,E)$ rooted at node 0 with a node set V and edge set E, we assume that a profit c_v and a demand d_v on each node v of T are given. Let H be a given capacity for a concentrator located at root 0. Then the capacitated subtree of a tree problem (CSTP) is to find a subtree T' of T rooted at node 0 so that the sum of profit over

* Department of Business Administration, Chonnam National University, Kwangju 500-757, Korea

T' is maximized and the sum of demand over T' does not exceed H.

CSTP plays an important role in a telecommunication system such as the Local Access Telecommunication Network (LATN) design. Most existing LATNs have a tree structure. The LATN connects individual customer nodes to the corresponding switching center (central office) that communicates with other switching centers via the backbone network. Each customer node has a demand which represents the required number of circuits from that node to the switching center. This demand can be satisfied in two ways: either by providing a dedicated cable from the customer node to the switching center for each required circuit, or by routing the circuits to a concentrator. The concentrator combines incoming signals on multiple lines into a single composite higher frequency signal that requires one outgoing line (see Balakrishnan *et al.* [3,4]). The objective of the LATN design problem is to select concentrator locations and to assign each customer node to one of the selected concentrators so as to minimize the total cost, subject to the concentrator capacity constraint. The optimal solution of this problem can be obtained by solving a series of CSTPs (see Cho and Shaw [5] and Shaw [13]). Another application of CSTP arises in a single machine scheduling problem (see Ibarra and Kim [8]). However, CSTP is an NP-complete problem (Garey and Johnson [7]), since the 0-1 knapsack problem is a special case of CSTP where the depth of the tree is one. Johnson and Niemi [9] introduced certain "left-right" dynamic programming algorithm for CSTP which could find the optimal value C^* in $\theta(nC^*)$ where n is the number of nodes in T. Shaw [13] solved it through a dynamic programming algorithm in $O(nH^2)$.

Many important tree-optimization problems, including the CSTP, p-median problems on tree (Kariv and Hakimi [10]), and uncapacitated facility location problems on tree (Cornuejols, Nemhauser, and Wolsey [6]), can be solved through dynamic programming algorithms. One popular approach used in these dynamic programming algorithms is the bottom-up approach, which was proposed first by Lukes [11]. Recently, Magnanti and Wolsey [12] have used such an approach to solve CSTP and related problems in telecommunication systems.

In this paper, we present a new dynamic programming technique on tree, the depth-first approach, which exploits the tree in the depth-first search (DFS) order (see Ahuja *et al.* [2]) and solves a CSTP in $\theta(nH)$ time. Such an approach is motivated from the "left-right" dynamic programming algorithm developed by Johnson and Niemi [9]. However, our algorithm has much simpler procedure and easier implementation than Johnson and Niemi's. By combining those two algorithms, CSTP can be solved by a dynamic programming algorithm in $\theta(n \min(C^*,H))$.

The depth-first dynamic programming algorithm has some advantages for performing

sensitivity analysis to solve a series of CSTPs. As we mentioned before, since the LATN design problem can be solved by solving the series of CSTPs, our algorithm plays a central role in solving LATN design problem (see Cho and Shaw [5]). In this paper, we briefly introduce the main idea discussed in Cho and Shaw [5] how to solve the LATN design problem by incorporating our depth-first dynamic programming algorithm for CSTP. It turns out that our depth-first dynamic programming code for CSTP not only is very competitive to a general integer programming solver, CPLEX, but also gives outstanding computational performance incomparable with CPLEX when it is incorporated to solve the LATN design problem. For example, CPLEX could not even solve LATN design problems with n=15, H=1000 on SUN SPARC 1000 workstation within 24 hours, whereas the developed algorithm in Cho and Shaw [5] could solve problems with n=150, H=1000 within 2 minutes (for more details, see Cho and Shaw[5]).

This paper is organized as follows. We first formulate the capacitated subtree of a tree problem (CSTP) in Section 2. Section 3 describes the depth-first dynamic programming algorithm for CSTP. In Section 4, we provide a formal proof of the correctness of our algorithm and also show that its complexity is $\theta(nH)$, where n is the total number of nodes in T and H is the given concentrator capacity. Section 5 introduces a generalized version of the CSTP which acts as a subproblem in LATN design problem. One example and the computational results on a set of randomly generated problems are given in Section 6. Finally, Section 7 concludes the paper.

2. Problem Description

Given an undirected tree $T = (V,E)$ rooted at node 0, where $V=\{0, 1, 2, \cdots, n\}$, we assume that nodes in T are labeled in DFS order, starting from the root node. For each node $i \in V$, an integer c_i and a nonnegative integer d_i represent the profit and the demand at node i, respectively. Let p_i be the predecessor of node i and let $\gamma(i, j)$ denote the unique path from node i to node j. Then $T(i)=\{j \mid i \in \gamma(0, j)\}$ is the complete subtree rooted at node i. Define a relation '$\prec$'as follows:

$$V' \prec V \Leftrightarrow T'=(V',E') \text{ is a subtree of } T = (V,E) \text{ rooted at } 0.$$

If H is a given capacity for a concentrator located at the root node, then the capacitated subtree of a tree problem (CSTP) is to find a subtree $T^*=(V^*,E^*)$ of T rooted at node 0, where

$$V^*=\operatorname{argmax}_{V' \prec V}\left\{ \sum_{i \in V'} c_i \mid \sum_{i \in V'} d_i \le H \right\}.$$

Let

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

Then, CSTP can be formulated as the following integer programming problem:

$$\max \sum_{j=0}^{n} c_j x_j \tag{2.1}$$

$$\text{(CSTP) s.t. } x_{p_j} \geq x_j, \quad j=1, 2, \cdots, n \tag{2.2}$$

$$\sum_{j=0}^{n} d_j x_j \leq H \tag{2.3}$$

$$x_j \in \{0, 1\}. \tag{2.4}$$

We assume that

$d_j \leq H$ for all $j=0, 1, 2, \cdots, n$
and

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

Otherwise, either the problem size can be reduced or the constraint (2.3) can be eliminated, and the problem is reduced to the uncapacitated subtree of a tree problem, which can be solved in $O(n)$ time (see Shaw and Cho [14]). We now sketch the main idea of our "depth-first" dynamic programming algorithm for CSTP.

3. Dynamic Programming Algorithm for CSTP

For $k \leq n$, let $L = \{0, 1, 2, \cdots, k\}$ be a subset of V representing labeled (visited) nodes and $T_L=(L,E_L)$ be the induced subtree of T. For a given capacity h and a given node $v \in L$, we now define a value $P_L(v,h)$ as follows:

$$P_L(v,h)=\max \left\{ \sum_{i=0}^{k} c_i x_i \,\middle|\, x_{p_j} \geq x_j,\ 0<j \leq k,\ \sum_{i=0}^{k} d_i x_i \leq h, \text{ and } x_v=1 \right\}. \tag{3.1}$$

Then, $\max\{P_V(0,H),0\}$ is the optimal value of CSTP. More precisely, let $L_k=\{0, 1, 2, \cdots, k\}$. Then we can find the value $P_V(0,H)$ in $\theta(nH)$ time by applying the following recursive rules:

1. (Initialization)

$$P_{L_0}(0,h)=\begin{cases} c_0 & \text{if } d_0 \leq h \leq H \\ -\infty & \text{otherwise} \end{cases}$$

2. (Forward move to expand the set of labeled nodes)
For $k \neq 0$ and for each $h = 0, 1, 2, \cdots, H$,

$$P_{L_k}(k,h)=\begin{cases} P_{L_{k-1}}(p_k,h-d_k)+c_k & \text{if } \sum_{j \in \gamma(0,k)} d_j \leq h \\ -\infty & \text{otherwise} \end{cases}$$

3. (Backward move to revisit labeled nodes)
Let $v \neq 0$ and $L=L_{v-1} \cup T(v)$.
Then, for each $h=0, 1, 2, \cdots, H$,
$P_L(p_v,h)=\max\{P_{L_{v-1}}(p_v,h), P_L(v,h)\}$.

Note that recursive rule 3 shows that, whenever we revisit a labeled node p_v, the predecessor of node v, we can find the $p_L(p_v,h)$ by comparing the current values associated with two subtrees, one which includes node v, and one which does not.

We now summarize the main idea of our DFS-approach as follows: we begin with the root node 0. We first initialize $P_{L_0}(0,h)$ for all $h = 0, 1, 2, \cdots, H$ by using recursive rule 1. Whenever we visit a new node v in DFS-order,

we include it as a member of L, a set of labeled nodes, and evaluate $P_L(p_v,h)$ for all h = 0, 1, 2,$\cdots$, H by using recursive rule 2. If we visit a leaf node v or a node v such that all of its successors have been labeled (visited), we revisit p_v and evaluate $P_L(p_v,h)$ for all h = 0, 1, 2,$\cdots$, H by using recursive rule 3. Otherwise, there are some unlabeled successors of node p_v; then we perform the forward move by visiting the first unlabeled successor of the node p_v. We continue the above procedure until the root node can be revisited from its last successor, and finally find the optimal value, $\{P_V(0,H),0\}$.

To find an optimal solution, we define the *index* $I_L(v,h)$ of node v which corresponds to $P_L(v,h)$ as follows:

1. $$I_{L_0}(0,h)=\begin{cases}1 & \text{if } d_0 \le h \le H\\ 0 & \text{otherwise}\end{cases}$$

2. Let $v \neq 0$ *and* $L=L_{v-1}\cup T(v)$. Then

$$I_L(v,h)=\begin{cases}1 & \text{if } P_{L_{v-1}}(p_v,h) < P_L(v,h) \text{ and } \sum_{j\in\gamma(0,v)} d_j \le h\\ 0 & \text{otherwise.}\end{cases}$$

Here '1' and '0' stand for 'adding' and 'deleting' a node during the DFS-approach, respectively. We don't need to keep track of the indices $I_{L_k}(k,h)$ which correspond to $P_{L_k}(k,h)$ (i.e., forward move) because we will update the indices $I_{L_k}(k,h)$ whenever we perform the backward move. We will discuss how to find the optimal solution in detail right after Algorithm 1 is presented.

Now we present our algorithm which finds the optimal value for CSTP. We will use $P(v,h)$ and $I(v,h)$ in the algorithm instead of using $P_{L_k}(k,h)$ and $I_{L_k}(v,h)$, respectively. It is important to observe that we don't have to keep track of L_k in the algorithm because $P(v,h)$ and $I(v,h)$ are overwritten by the newly evaluated values. In fact, the values $P_{L_k}(v,h)$ and $I_{L_k}(v,h)$ are not updated until a 'backward move' revisits the node v from one of its successors. Note that the values $P_V(v,h)$ and $I_V(v,h)$ are determined by a 'backward move' from the last successor of v to the node v. Consequently, we just need $\theta(nH)$ storage space to find the optimal value and optimal solution. Let $last(i)$ = $\max\{j \mid j\in T(i)\}$ be the last node in $T(i)$.

Algorithm 1. Optimal_Value_CSTP;

```
begin
{comment: Initialization}
  d:=min{d_j | j∈V};
  for h:=d up to d_0 − 1 do
    P(0,h):= −∞; I(0,h):=0;
  for h:=d_0 up to H do
    P(0,h):=c_0; I(0,h):=1;
{comment: Main Loop}
  d_path:=d_0;
  for k := 1 up to n do
  begin
   Forward_Move(k);
   if (k = last(k)) then
   {comment: k is a leaf node}
     v : = K ;
     do
```

```
      Backward_Move(v);
      v:=p_v;
    while (last(v) = k and v ≠ 0)
    {comment: v has no successor i such
    that i > k and v ≠ 0}
   end if
 end
end

Procedure Forward_Move(u);

begin
 d_path:=d_path+d_u;
 for h:=d_u up to d_u−1 do
    P(u,h):= −∞;
 for h:=d_u up to H do
 begin
  if (d_path≤h) then
    P(u,h):=P(p_u,h−d_u)+c_u;
  else
    P(u,h):= −∞;
  end if
 end
end

Procedure Backward_Move(u);

begin
 d_path:=d_path−d_u;
 for h:=d_u up to H do
 begin
  if (P(p_u,h)≥P(u,h)) then
    I(u,h):= 0;
  else
    P(p_u,h):=P(u,h);
    I(u,h):= 1;
  end if
 end
end
```

The values of $P(v,h)$ and $I(v,h)$ at the end of Algorithm 1 are, in fact, $P_L(v,h)$ and $I_L(v,h)$ with $L=L_{v-1}\cup T(v)$, respectively.

To find the optimal solution for CSTP, we need to understand how our recursive rules are applied to find the optimal value $P_V(0,H)$. For this purpose, let $S(i)$ be the set of successors of node i and $|S(i)|$ be the cardinality of $S(i)$. Let $succ(i,k)$ be the k-th successor of node i. Then we begin with $w = succ(0,|S(0)|)$. Since

$$P_V(0,H)=\max\left\{P_{L_{w-1}}(0,H),P_V(w,H)\right\},$$

we need to consider the following two cases:

Case 1. $P_{L_{w-1}}(0,H)\geq P_V(w,H)$

Case 2. $P_{L_{w-1}}(0,H)< P_V(w,H)$.

For Case 1, the returned index $I(w,H)$ (in fact, $I_V(w,H)$) from Algorithm 1 must be 0; therefore we set $x_k=0$ for all $k\in T(w)$, update w by $succ(0,|S(0)|-1)$, and continue to check the indices for all $v\in T(w)$ with the capacity H. For Case 2, the returned index $I(w,H)$ (in fact, $I_V(w,H)$) must be 1 and thus we set $x_w=1$. In this case, if $|S(w)|=0$ (i.e., if w is a leaf) then we update w by $succ(0,|S(0)|-1)$ and continue to check the indices for all $v\in T(w)$ with the capacity $H-d_w$. If $|S(w)|>0$ then,

since

$$P_V(w,H)=\max\{P_{L_{u-1}}(w,H), P_V(u,H)\}$$

with $u = succ(w, |S(w)|)$, we, again, have two cases to consider as before. By repeating the above procedure, we are able to find a set

$$K=\{k\in T(w)\,|\,I_{L_{i-1}\cup T(k)}(k,H)=1\}.$$

We then set $x_k=1$ for all $k\in K$ and replace w by $succ(0, |S(0)|-1)$ and continue to check the indices for all $v\in T(w)$ with the capacity $H-\sum_{k\in K} d_k$.

By continuing the above procedure until we can check the indices for all $v\in T(w)$ with $w = succ(0,1)$, we are able to find the optimal solution for CSTP. Our algorithm for finding the optimal solution for CSTP is described as follows.

Algorithm 2. Optimal_Solution_CSTP;

begin
  $num := |S(0)|$;
  while $(num \neq 0)$ **do**
  begin
    $w := succ(0, num)$;
    $K := \{i\in T(w)\,|\,I(i,H)=1\}$;
    $x_i:=1$ for all $i\in K$;
    $H:=H-\sum_{i\in K} d_i$;
    $num := num - 1$;
  end
end

Note that we also need $\theta(nH)$ storage space to find the optimal solution for CSTP.

4. Correctness and Complexity of Algorithm 1

We use induction to prove rigorously the correctness of Algorithm 1 and also estimate its complexity.

Theorem 1. *Algorithm 1 is correct.*

Proof: To show the correctness of the algorithm we only need to prove that the value $P(0,H)$ produced at the end of Algorithm 1 is indeed the optimal value of CSTP, subject to $x_0=1$. To prove this, it suffices to establish the following claim: for every tree T rooted at node 0 with $n+1$ nodes, the value $P_{L_n}(v,h)$ computed by Algorithm 1 is exactly equal to $\overline{P}_{L_n}(v,h)$ for all $v\in\gamma(0,n)$ and for all $h = 0, 1, 2,\cdots, H$, where

$$\overline{P}_{L_n}(v,h)\equiv\max\{\sum_{i=0}^{n} c_i x_i\,|\,x_{p_j}\geq x_j,\ 0<j\leq n,$$
$$\sum_{i=0}^{n} d_i x_i\leq h,\ \text{and}\ x_v=1\}. \qquad (4.1)$$

For $n=0$, it is trivial. Assume that the claim is true for all trees with k nodes, $k\leq n$. Now we consider a tree T rooted at node 0 with $n+1$ nodes. If $v=n$ then we have

$$P_{L_n}(n,h)=P_{L_{n-1}}(p_n,h-d_n)+c_n, \text{ by recursive rule 2.}$$

Since $p_n\in\gamma(0,n)$, we have $P_{L_{n-1}}(p_n,h-d_n)=\overline{P}_{L_n}-1(p_n,h-d_n)$,

by the induction hypothesis. Therefore,

$$P_{L_n}(n,h)=\overline{P}_{L_{n-1}}(p_n,h-d_n)+c_n \quad (4.2)$$

$$=\overline{P}_{L_n}(n,h), \quad (4.3)$$

where the last equality follows from the definition in (4.1).

Now, let $\gamma(0,n)=\{0=v_0, v_1, \cdots, v_{q-1}, v_q=n\}$. We want to prove that the claim is true for all v_i, $i=0, 1, 2,\cdots, q$. Clearly, we have already proved the case for $i = q$ in (4.3). For $i = q-1$, we have, by recursive rule 3,

$$P_{L_n}(v_{q-1},h)=\max\left\{P_{L_{v_{q-1}}}(v_{q-1},h), P_{L_n}(v_q,h)\right\}. \quad (4.4)$$

Since $v_q-1=n-1<n$ and $v_{q-1}\in\gamma(0,n)$, we have, by the induction hypothesis,

$$P_{L_{v_{q-1}}}(v_{q-1},h)=\overline{P}_{L_{v_{q-1}}}(v_{q-1},h).$$

Therefore, it follows immediately from (4.3) and (4.4) that

$$P_{L_n}(v_{q-1},h)=\max\left\{\overline{P}_{L_{v_{q-1}}}(v_{q-1},h), \overline{P}_{L_n}(v_q,h)\right\}$$

$$=\overline{P}_{L_n}(v_{q-1},h),$$

where the last equality follows from the definition in (4.1).

Consequently, by going up along the path $\gamma(0,n)$ we have $P_{L_n}(v_i,h)=\overline{P}_{L_n}(v_i,h)$ for all $i = 0, 1, 2,\cdots, q-1$. Therefore, we have proved that, for every tree T with $n + 1$ nodes, the claim is true for all $v\in\gamma(0,n)$ and for all $h = 0, 1, 2,\cdots, H$. □

The following theorem proves that the complexity of Algorithm 1 is $\theta(nH)$

Theorem 2. *A CSTP can be solved by Algorithm 1 in the time-complexity and the space-complexity of* $\theta(nH)$

Proof: Algorithm 1 performs essentially two basic operations: the forward move and the backward move for every $v\in V\backslash\{0\}$. In fact, if a node has an unvisited successor then it performs the forward move. Otherwise, it performs the backward move. At the root node, the algorithm performs only the forward move. Thus, Algorithm 1 has exactly $n+1$ forward moves and n backward moves. Since each move requires $O(H)$ time, the overall time complexity for Algorithm 1 is $\theta(nH)$. Since the algorithm requires storing $P(v,h)$ and $I(v, h)$ for all $v\in V$ and for all $h = 0, 1, 2,\cdots, H$, respectively, the space complexity for the algorithm is also $\theta(nH)$. □

5. Generalized Version of Capacitated Subtree of Tree Problem

In this section, we briefly introduce a generalized version of CSTP (GCSTP) discussed in Cho and Shaw [5]. In fact, to solve a LATN design problem, we have to solve problems for which the concentrator can be located anywhere on the tree. Therefore, we need to consider a GCSTP defined in the following. Since this problem acts as a

subproblem in solving a LATN design problem, we also want to restrict the problem on each subtree $T(k)$ for k = 0, 1, 2,⋯, n. Let node i be the concentrator location and p_j^i be the predecessor of node j with respect to node i, which is defined as the first node other than node j on the path $P[j,i]$. Then, GCSTP restricted on $T(k)$ with the conccentrator location i is defined as follows:

$$(GCSTP_k^i)\quad \max \sum_{j\in T(k)} c_j x_j$$
$$\text{s.t. } x_{p_j^i} \geq x_j,\ j\in T(k)\backslash\{k\}$$
$$\sum_{j\in T(k)} d_j x_j \leq H$$
$$x_j\in\{0,1\},\ j\in T(k)\backslash\{k\}.$$

Note that $(GCSTP_k^i)$ becomes a CSTP when the concentrator location i is equal to the root node k of $T(k)$. To solve a LATN design problem, we have to solve a class of $(GCSTP_k^i)$ for $k\in P[i,0]$ and i = 0, 1, 2,⋯, n. From our previous analysis, each $(GCSTP_k^i)$ can be solved by the depth-first dynamic programming algorithm in $\theta(|T(k)|H)$ time. One significant advantage of the depth-first dynamic programming algorithm in contrast to other approaches is to be able to solve $(GCSTP_{p_k^i}^i)$ in $\theta(|T(p_k^i)\backslash T(k)|H)$ time, once the optimal value of $(GCSTP_k^i)$ is given. Hence, a series of $(GCSTP_k^i)$ for all $k\in P[i,0]$ and fixed i can be solved in just $\theta(nH)$ time. Consequently, we can solve the class of $(GCSTP_k^i)$ for all $k\in P[i,0]$ and i = 0, 1, 2,⋯, n (and thus, the whole LATN design problem) in $\theta(n^2H)$ time, whereas one of the best algorithm which incorporates a bottom-up approach developed by Shaw [13] could solve it in $\theta(n^2H^2)$ time. This is the main reason why the depth-first dynamic programming algorithm was incorporated to solve the GCSTPs in developing algorithms for the LATN design problem in Cho and Shaw [5].

6. Example and Computational Results

Consider the tree in Figure 1 with H= 18. In this example, all nodes are labeled in DFS order. Numbers in the square box stand for demands and numbers above the box stand for profits. First, we begin with node 0. Since d_0=2, we have

$$P_{\{0\}}(0,h)=\begin{cases}30 & \text{if } 2\leq h\leq 18\\ -\infty & \text{otherwise}\end{cases}$$

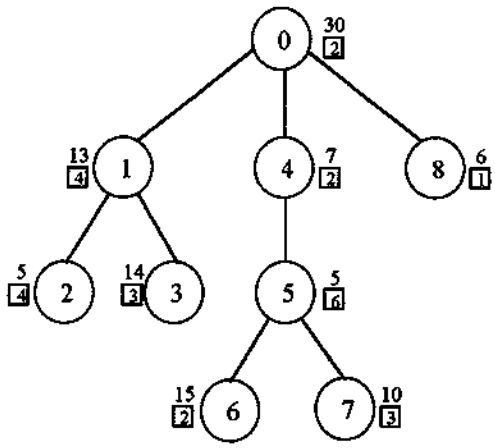

Figure 1. Example

and

$$I_{\{0\}}(0,h)=\begin{cases}1 & \text{if } 2\le h\le 18\\ 0 & \text{otherwise.}\end{cases}$$

Then we visit node 1 and evaluate $P_{\{0,1\}}(1,h)$ by using recursive rule 2 as follows:

$$P_{\{0,1\}}(1,h)=P_{\{0\}}(0,h-d_1)+c_1$$
$$=\begin{cases}43 & \text{if } 6\le h\le 18\\ -\infty & \text{otherwise.}\end{cases}$$

Now we visit node 2 and can get $P_{\{0,1,2\}}(2,h)$ in a similar way:

$$P_{\{0,1,2\}}(2,h)=\begin{cases}48 & \text{if } 6\le h\le 18\\ -\infty & \text{otherwise.}\end{cases}$$

Since node 2 is a leaf node (*i.e.*, $|S(2)|=0$), we revisit its predecessor, node 1, (backward move) and evaluate $P_{\{0,1,2\}}(1,h)$ and $I_{\{0,1,2\}}(1,h)$ as follows:

$$P_{\{0,1,2\}}(1,h)=\max\{P_{\{0,1\}}(1,h),P_{\{0,1,2\}}(2,h)\}$$
$$=\begin{cases}48 & \text{if } 10\le h\le 18\\ 43 & \text{if } 6\le h<10\\ -\infty & \text{otherwise}\end{cases}$$

and

$$I_{\{0,1,2\}}(2,h)=\begin{cases}1 & \text{if } 10\le h\le 18\\ 0 & \text{otherwise.}\end{cases}$$

Since node 1 has an unlabeled successor node 3, we visit node 3 and evaluate $P_{\{0,1,2,3\}}(3,h)$ as before. Since node 3 is a leaf node, we revisit node 1 and also evaluate $P_{\{0,1,2,3\}}(1,h)$ and $I_{\{0,1,2,3\}}(3,h)$. Now, since node 1 has no unlabeled successors, we revisit node 0 (backward move) and evaluate

$$P_{\{0,1,2,3\}}(0,h)=\max\{P_{\{0\}}(0,h),P_{\{0,1,2,3\}}(1,h)\}$$

and

$$I_{\{0,1,2,3\}}(1,h)=\begin{cases}1 & \text{if } 6\le h\le 18\\ 0 & \text{otherwise.}\end{cases}$$

Continuing this way, we can find $P_{L_7}(7,h)$ with $L_7=\{i\,|\,0\le i\le 7\}$ as follows:

$$P_{L_7}(7,h)=P_{L_6}(5,h-d_7)+c_7$$
$$=\begin{cases}67 & \text{if } 15\le h\le 18\\ 52 & \text{if } 13\le h<15\\ -\infty & \text{otherwise.}\end{cases}$$

Since node 7 is a leaf node, we revisit node 5 and find

$$P_{L_7}(5,h)=\max\{P_{L_6}(5,h),P_{L_7}(7,h)\}$$
$$=\begin{cases}70 & \text{if } 16\le h\le 18\\ 67 & \text{if } h=15\\ 57 & \text{if } 12\le h<15\\ 42 & \text{if } 10\le h<12\\ -\infty & \text{otherwise}\end{cases}$$

and

$$I_{L_7}(7,h)=\begin{cases}1 & \text{if } h=15\\ 0 & \text{otherwise.}\end{cases}$$

Since node 5 now doesn't have an unlabeled successor node, we revisit its predecessor, node 4, and evaluate

$$P_{L_7}(4,h)=\max\{P_{L_4}(4,h),P_{L_7}(5,h)\}$$

and

$$I_{L_7}(5,h)=\begin{cases}1 & \text{if } 16\le h\le 18\\ 0 & \text{otherwise.}\end{cases}$$

By continuing the above procedure we can finally obtain the optimal value $P_V(0,18)=76$ from the following result:

$$P_V(0,h)=\max\{P_{L_7}(0,h),P_V(8,h)\}$$

$$=\begin{cases}76 & \text{if } 17\le h\le 18\\ 75 & \text{if } h=16\\ 70 & \text{if } 12\le h<16\\ 64 & \text{if } h=11\\ 63 & \text{if } h=10\\ 57 & \text{if } h=9\\ 50 & \text{if } h=8\\ 49 & \text{if } h=7\\ 43 & \text{if } 5\le h<7\\ 37 & \text{if } h=4\\ 36 & \text{if } h=3\\ 30 & \text{if } h=2\\ -\infty & \text{otherwise}\end{cases}$$

and

$$I_V(8,h)=\begin{cases}1 & \text{if } h=3,5,6,7,10, \text{ or } 12\le h\le 18\\ 0 & \text{otherwise.}\end{cases}$$

Figure 2 shows how our depth-first procedure has been performed to find the optimal value $P_V(0,18)$ in this example.

Now, to find the optimal solution, we first check $I_V(8,18)$, which is 1. Therefore, node 8 is included in the optimal solution (i.e., $x_8=1$). Since $d_8=1$ we check $I_{L_7}(k,17)$ for all $k\in T(4)$. We then have a set $K=\{4, 5, 6,\}$ of nodes in

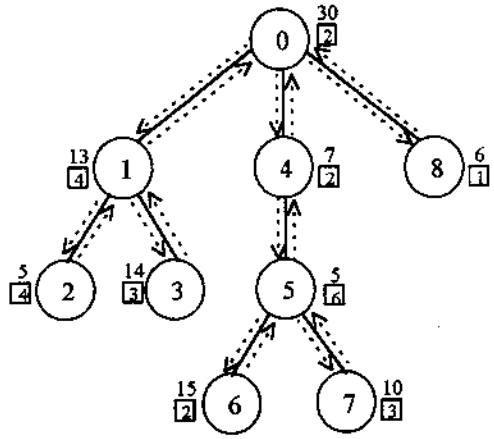

Figure 2. Depth-first procedure for the example

$T(4)$ whose indices are 1. We now replace 17 by $17-\sum_{k\in K} d_k=7$ and continue to check $I_{L_1}(k,7)$ for all $k\in T(1)$. Finally, we see that only $I_{L_1}(1,7)$ is 1, and therefore we obtain the following optimal solution shown in Figure 3:

$$x_j=\begin{cases}1 & \text{if } j=0, 1, 4, 5, 6, 8\\ 0 & \text{otherwise.}\end{cases}$$

We now report the computational results for our depth-first dynamic programming algorithm for the CSTP. To the best of our knowledge, there are no computational results as well as the benchmark of test problems for CSTP in the literature. We then tested these algorithms on a set of randomly generated test problems. To observe the performance of our algorithm roughly, we compared our code with a general integer programming solver, CPLEX, on a set

of randomly generated problems. Table 1 presents the worst, the average, and the best CPU time (measured in seconds) out of eight randomly generated test problems in each case. It also includes the average of each of the following values obtained from CPLEX: the number of nodes generated in the decision tree, the optimal value of LP relaxation of CSTP, and the optimal value of CSTP. The test results show that our algorithm is very competitive to CPLEX for most cases. Note that, however, this comparison may not be significant, since the CPU time for our algorithm is proportional to the capacity H, whereas one for CPLEX seems not to be affected much by H. Instead, as we discussed in the previous section, the key advantage of using the depth-first dynamic programming approach in application areas such as a LATN design problem is to be able to solve the series of GCSTPs, $\{GCSTP_k^i \mid k \in P[i,0]\}$ in $\theta(nH)$ time which is equal to the complexity of a single CSTP.

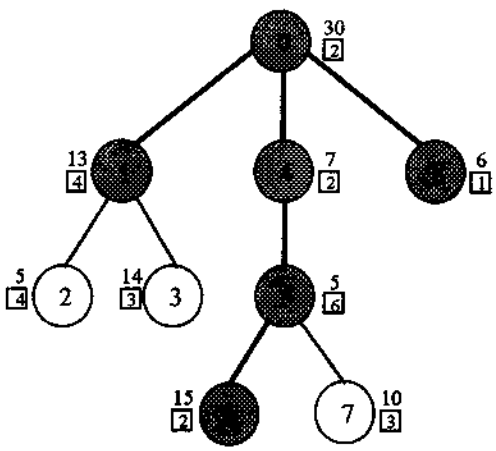

Figure 3. Optimal tree for the example

To generate a problem randomly, we specified the total number of nodes n in the tree first. Then, starting from the root node, we randomly generated the number of successors of each node from an interval $[0,1og_2n]$ in the Breadth First Search(BFS) order until the total number of nodes was met. In our test problem set, the number of node n and the capacity H were in the range [50,500] and [5000,10000], respectively. The demand d_i was randomly generated from the range [1,1000]. We believe that those values generated in this paper is quite big enough to be considered as the real data. Algorithms were coded in C language and run on a SUN SPARC 1000 workstation.

7. Conclusions

In this paper, we have developed a pseudo-polynomial-time algorithm, the so-called depth-first dynamic programming algorithm which solves CSTP in $\theta(nH)$ time, where n is the total number of nodes and H is the given capacity. Our research was motivated from studying LATN design problem, an important application problem in telecommunication systems. Since LATN design problem can be solved by solving a series of CSTPs and our depth-first dynamic programming algorithm has we are able to develop a pseudopolynomial-time algorithm for LATN design problem by incorporating the solution procedure of our depth-first approach for CSTP.

Table 1. Computational results for comparing DP and CPLEX

n	H	DP			CPLEX					
		worst	average	best	worst	average	best	Nodes	$Z_{\overline{CSTP}}$	Z_{CSTP}
50	5,000	2.23	2.10	1.88	1.08	0.82	0.61	42	2139.85	2101
	10,000	4.60	4.49	4.38	1.50	1.15	0.81	43	3364.72	3343
100	5,000	4.55	4.27	4.05	4.81	2.78	1.50	116	10522.20	10519
	10,000	9.20	8.95	8.78	13.42	5.93	3.45	83	11238.43	11220
200	5,000	8.95	8.45	7.83	59.86	15.35	2.28	134	15837.52	15831
	10,000	17.78	17.62	17.18	24.73	9.69	1.70	127	19592.34	19590
300	5,000	13.03	12.76	12.42	58.24	33.46	1.17	143	18982.75	18977
	10,000	27.38	26.55	26.00	136.61	43.21	5.63	688	26196.04	26183
500	5,000	22.57	21.83	21.45	106.53	36.11	4.68	131	21547.52	21539
	10,000	46.42	45.19	44.02	318.52	65.23	18.34	693	33332.82	33330

Nodes : the average number of nodes of decision tree generated from CPLEX

$Z_{\overline{CSTP}}$: the average optimal value of LP relaxation of CSTP

Z_{CSTP} : the average optimal value of CSTP

References

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

[2] Ahuja, R.K., Magnanti, T.L., and Orlin, J. B., *Network Flows*, Prentice Hall, Inc. A Simon & Schuster Co., Englewood Cliffs, NJ 07632, 1993.

[3] Balakrishnan, A., Magnanti, T.L., Shulman, A., and Wong, R.T.,"Models for Planning Capacity Expansion in Local Access Telecommunications Networks,"*Annals of Operations Research*, Vol.33, pp.239-284, 1991.

[4] Balakrishnan, A., Magnanti, T.L., and Wong, R.T.,"A Decomposition Algorithm for Local Access Telecommunications Networks Expansion Planning,"*Operations Research*, Vol.43, No.1, pp.58-76, 1995.

[5] Cho, G. and Shaw, D.X.,"Limited Column Generation for Local Access Telecommunication Network Design," Technical Report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1994.

[6] Cornuejols, G., Nemhauser, G.L., and Wolsey, L.A.,"The Uncapacitated Facility Location Problem," in: *Discrete Location Theory*, P.B. Mirchandani & R.L. Fran-

ciseds., John Wiley and Sons., 1990.

[7] Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., San Francisco, 1979.

[8] Ibarra, O.H. and Kim, C.E.,"Approximation Algorithms for Certain Scheduling Problems," *Mathematics of Operations Research*, Vol.3, pp.197-204, 1978.

[9] Johnson, D.S. and Niemi, K.A.,"On Knapsacks, Partitions, and A New Dynamic Programming Technique for Trees," *Mathematics of Operations Research*, Vol. 8, pp1-14, 1983.

[10] Kariv, O. and Hakimi, S.L.,"An Algorighmic Approach to Network Location Problems II: The *p*-medians,"*SIAM J. Appl. Math.*, vol.37, pp.539-555, 1979.

[11] Lukes, J.A.,"Efficient Algorithm for the Partitioning of Trees," *IBM Journal of Research and Development*, vol.18, pp. 214-224, 1974.

[12] Magnanti, T.L. and Wolsey, L.A.,"Optimal Trees," CORE Discussion Paper, Center for Operations Research & Economics, Universite Catholique De Louvain, Louvain-La-Neuve, Belgium, 1992.

[13] Shaw, D.X.,"Reformulation and Column Generation for Several Telecommunications Network Design Problems," Technical Report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1993.

[14] Shaw, D.X. and Cho, G.,"A Branch-and-Bound Procedure for the Tree Knapsack Problem," Technical Report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1994.

95년 10월 최초 접수, 96년 7월 최종 수정