Journal of the Korean
OR/MS Society
Vol. 22, No. 3, September 1997

A Branch and Bound Algorithm for Solving a Capacitated Subtree of a Tree Problem in Local Access Telecommunication Networks

Geon Cho * · Seong-Lyun Kim**

Abstract

Given a rooted tree T with node profits and node demands, the capacitated subtree of a tree problem(CSTP) consists of finding a rooted subtree of maximum profit, subject to having total demand no larger than the given capacity H. We first define the so-called critical item for CSTP and find an upper bound on the optimal value of CSTP in $O(n^2)$ time, where n is the number of nodes in T. We then present our branch and bound algorithm for solving CSTP and illustrate the algorithm by using an example. Finally, we implement our branch-and-bound algorithm and compare the computational results with those for both CPLEX and a dynamic programming algorithm. The comparison shows that our branch-and-bound algorithm performs much better than both CPLEX and the dynamic programming algorithm, where n and H are in the range of [50, 500] and [5000, 10000], respectively.

1. Introduction

Given an undirected tree $T=(V,E)$ rooted at $0 \in V$, let c_i and d_i be the given profit and demand at node $i \in V$, respectively. Then, for a given capacity H, the capacitated subtree of a tree problem(CSTP) is to find a subtree T' of T rooted at node 0 so as to maximize the sum of profits over the subtree T' under the constraint of which the total demand over T' does not exceed H.

* School of Business Administration, Chonnam National University, Kwangju, Korea.
** Electronics and Telecommunications Research Institute, Taejeon, Korea

CSTP can be served as a subproblem in the local access telecommunication network(LATN) design problem(see Aghezzaf *et al*[1], Balakrishnan *et al*[2] and Shaw[15]). Let the root of the tree be the location of a central office and the other nodes represent the potential subscribers in LATN. Then, the central office communicates with other central offices through the backbone network and the LATN contains a dedicated communication channel connecting each subscriber node to the central office. Each subscriber node has a demand which represents the required number of circuits from that node to the central office. This demand can be satisfied in either routing the circuits to the central office directly by using the dedicated cable or routing those to a concentrator, an electronic device that compresses incoming signals on multiple lines into a single higher frequency signal that requires one outgoing line(see Balakrishnan *et al*[2]). Here, we assume the *indivisible demand requirement*, that is, all circuits from one subscriber node must have the same routing pattern. We also assume the *contiguity restriction*, that is, if one subscriber node is served by a concentrator, then all subscribers on the path from that node to the concentrator must be served by the same concentrator. Let each concentrator have the given capacity. Then, the objective of the LATN design problem is to select concentrator locations and to assign each subscriber to one of the selected concentrators so as to minimize the total cost, subject to the concentrator capacity constraint. This problem can be solved by solving a sequence of CSTPs(see Aghezzaf *et al*[1] and Shaw [15]). However, CSTP is an NP-complete problem, since it is reduced to the 0-1 knapsack problem when the depth of the tree is one.

Other applications of CSTP arise in modeling a single machine scheduling problem(see Ibarra and Kim [6]), a p-median problem on an undirected tree network(see Kariv and Hakimi [8]), and a capacitated facility location problem on an undirected tree network(see Mirchandani and Francis [13]).

Cho and Shaw [4] and Johnson and Niemi [7] proposed dynamic programming(DP) algorithms for solving CSTP in $O(nH)$ and $O(nC^*)$, respectively, where n is the number of nodes in the tree and C^* is the optimal value of CSTP. Consequently, CSTP can be solved by a DP algorithm in $O(n\min(C^*, H))$.

In this paper, we develop a branch-and-bound(B&B) algorithm for solving CSTP. The so-called *critical-item* plays a central role in determining a bound during the process of our B&B algorithm. For the 0-1 knapsack problem which is a special case of CSTP, the critical--item can be easily obtained through sorting in $O(n\ln n)$ time(see Horowitz and Sahni [5] and Martello and Toth [11,12]). Moreover, by using the median-finding procedure, the critical-item for the 0-1 knapsack problem can be found in $O(n)$ time as in Balas and Zemel [3] and Lawler [9]. It is well known that the critical item for the 0-1 knapsack problem not only determines the optimal solution of the linear programming(LP) relaxation of the problem, but also plays an important role in defining several upper bounds on the optimal value of the problem(see Martello and Toth [12]). However, defining the *critical-item* for CSTP is not a trivial problem because of the contiguity assumption.

One of main contributions of this paper is to be able to successfully define the *critical-item* for CSTP and to find it in $O(n^2)$ time. Based on the procedure of finding the critical-item, we develop a B&B algorithm for CSTP. Computational results indicate that our B&B algorithm is much superior to CPLEX, a general integer programming solver, and the depth-first DP algorithm developed by Cho and Shaw [4], where n and H are in the range of [50,500] and [5000,10000], respectively.

This paper is organized as follows. We first formulate the capacitated subtree of a tree problem(CSTP) in Section 2. Then, in Section 3, we develop a polynomial time algorithm for the uncapacitated subtree of a tree problem(USTP), which will be used for solving Lagrangian relaxation of CSTP with respect to the knapsack constraint. Section 4 defines the critical item for CSTP and finds an upper bound on the optimal value of CSTP in $O(n^2)$ time. We then present our branch-and-bound algorithm for CSTP in Section 5. In Section 6, an example and computational results are provided. Finally, Section 7 concludes the paper.

2. Problem Formulation

Let $T=(V,E)$ be a given undirected tree rooted at node 0, where $V=\{0,1,2,\cdots,n\}$. We assume that all nodes in T are labeled in the Breadth First Search(BFS) order. For each node $i \in V$, c_i, is an integer representing the potential profit at node i and d_i is a non-negative integer representing the demand at node i. Let p_i denote the predecessor of node i and $P[i,j]$ denote the unique path from node i to node j. We define $P(i,j]$ as $P[i,j] \setminus \{i\}$. Define a relation '<' as follows:

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

Let H be the given capacity for the concentrator located at the root node. Then the capacitated subtree of a tree problem(CSTP) is to find a subtree $\hat{T} = (\hat{V}, \hat{E})$ of T rooted at node 0, where.

$$\hat{V} = \underset{V' < V}{\arg\max} \{ \sum_{i \in V'} c_i \mid \sum_{i \in V'} d_i \le H \}$$

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:

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

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

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

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

Without loss of generality, we assume that

$$d_j \leq H, \quad j=0,1,2,\ldots,n$$

and

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

Otherwise, either the problem size can be reduced or the knapsack constraint (2.3) can be eliminated, and the problem is reduced to the uncapacitated subtree of a tree problem(USTP). In the next section, we will present an algorithm for solving an USTP in $O(n)$ time.

3. Uncapacitated Subtree of A Tree Problem

The uncapacitated subtree of a tree problem(USTP) can be formulated as follows:

$$\begin{aligned} &\max \quad \sum_{j=0}^{n} c_j x_j \\ \text{(USTP)}\quad &\text{s.t.} \quad x_{p_j} \geq x_j,\ j=1,2,\cdots,n \\ &\qquad\quad x_j \in \{0,1\}. \end{aligned}$$

It can be easily seen that (*USTP*) can be solved by the following algorithm in $O(n)$ time. Let

$x_{T'} = (x_j)_{i \in T'}$, where T' is a subtree of T. Define

$$\begin{aligned} T(i) &= \{j \mid j \text{ is a descendant of } i\} \\ &= \{j \mid i \in P[0, j]\} \end{aligned}$$

Then $T(i)$ is a complete subtree rooted at node i.

Algorithm 1.

```
begin
  for all $i \in V$ do
  begin
    set $\overline{C_i} := c_i$;  $x_i := 1$;
  end
  for $i := n$ down to 0 do
  begin
    if ($\overline{C_i} \le 0$) then
       $x_{T(i)} := 0$;
    else
       $\overline{C}_{p_i} := \overline{C}_{p_i} + \overline{C_i}$;
    end if
  end
end
```

Algorithm 1 is a bottom-up process, which follows the reverse of BFS order. Clearly, max $(0, \overline{C_0})$ is the optimal value of (*USTP*). It is also obvious that $x_i = 1$ implies $\overline{C_i} \ge 0$, but the reverse is not true. Now, we prove that the LP relaxation of (*USTP*) satisfies the integrality property(A different proof can be found in [10]).

Theorem 1. *The linear programming relaxation of (USTP) has an integral optimal solution.*

Proof: Given undirected tree $\hat{T} = (V, E)$, we define a directed out-tree $\vec{T} = (V, A)\, T$ where $A = \{(p_i, i) \mid i \ne 0, i \in V\}$. Let B be the arc-node incidence matrix of $\vec{T}$. Then the constraint $x_{p_i} \ge x_i$ can be written as $B^T x \le 0$. Let (LP) denote the LP relaxation of (USTP) and Z_{LP} be the optimum value of (LP). Then, the dual of (LP) is a minimum cost network flow problem and $Z_{LP} \leq \Sigma_{j=0}^{n} | c_j |$. Therefore, the system of linear inequalities in (LP), $B^T x \leq 0$, is Totally Dual Integral(see Nemhauser and Wolsey [14] for details). Since the right-hand side of $B^T x \leq 0$ is integral (= 0), the polyhedron of the feasible solutions of (LP) has only integral extreme points. □□

4. Critical Item and Upper Bound

In this section, we first define the *critical item* for CSTP. As we mentioned earlier, the critical item for the 0-1 knapsack problem can be found by consecutively inserting sorted items into the knapsack until the first item which results in exceeding the knapsack capacity is found. The critical item for the 0-1 knapsack problem plays a key role in determining the optimal solution of the LP relaxation of the problem(see Martello and Toth [11,12]). One of the main reasons that we call the root of the last deleted subtree the *critical item* for CSTP is that it also plays a central role in determining the optimal solution of the LP relaxation of CSTP as we will see in Theorem 2.

Without loss of generality, we assume that all c_i's are non-negative integers, since (*CSTP*) can be reduced to the problem with non-negative c_i's by applying Algorithm 1. Let $C_i = \Sigma_{j \in T(i)} c_j$, $D_i = \Sigma_{j \in T(i)} d_j$, and $r_i = C_i / D_i$. Let $r_{i_1} = \min\{ r_i \mid i \in T\}$. If $D_0 - D_{i_1} \leq H$, then i_1 is called the *critical item* for CSTP. Otherwise, after updating C_i, D_i, and r_i by $\overline{C_i} = \Sigma_{j \in T(i) \setminus T(i_1)} c_j$, $\overline{D_i} = \Sigma_{j \in T(i) \setminus T(i_1)} d_j$, and $\overline{r_i} = \overline{C_i} / \overline{D_i}$ for all $i \in P(i_1, 0]$, we find the smallest ratio

$$r_{i_2} = \min\{ \overline{r_i} \mid i \in T \setminus T(i_1)\}. \tag{4.1}$$

Note that we used in $\overline{r_i} = r_i$ in (4.1) for all $i \in T \setminus (T(i_1) \bigcup P(i_1, 0])$. If $D_0 - (D_{i_1} + \overline{D_{i_2}}) \leq H$, then i_2 is called the critical item for CSTP. Otherwise, we continue the above procedure. In general, the critical item for CSTP can be defined by the following.

Definition 1. Let i_k be a node such that $\overline{r_{i_k}} = \min\left\{ \overline{r_i} \mid i \in T \setminus \bigcup_{t=1}^{k-1} T(i_t)\right\}$. If $s = \min\left\{ k \mid D_0 - \Sigma_{t=1}^{k} \overline{D_{i_t}} \leq H\right\}$, then i_s is called the critical item for CSTP.

Note that $\overline{C}_{i_t} = \sum_{j \in T(i_t) \setminus \bigcup_{q=1}^{t-1} T(i_q)} c_j$, $\overline{D}_{i_t} = \sum_{j \in T(i_t) \setminus \bigcup_{q=1}^{t-1} T(i_q)} d_j$, and $\overline{r_{i_{t+1}}} \geq \overline{r_{i_t}} \geq 0$, for all

$t=1,2,\cdots,s$. We now utilize a Lagrangian relaxation method to obtain an upper-bound on the optimal value of CSTP. For any $\lambda \in R_+ = \{\text{non-negative real numbers}\}$, the Lagrangian relaxation of CSTP with respect to the knapsack constraint (2.3) is as follows:

$$
\begin{array}{lll}
 & f(\lambda)=\max & \sum_{j=0}^{n}(c_j-\lambda d_j)x_j+\lambda H \\
LR(\lambda) & \quad \text{s.t.} & x_{p_j}\geq x_j, \quad j=1,2,\cdots n \\
 & & x_j\in\{0,1\}.
\end{array}
$$

Then we have the following Lagrangian dual of CSTP

$$
(LD) \qquad f(\lambda^*)=\min_{\lambda\geq 0} f(\lambda)
$$

For a given $\lambda, LR(\lambda)$ is an USTP and can be solved in $O(n)$ time by Algorithm 1. Therefore, we can prove the following lemma.

Lemma 1. *Let i_s be the critical item for CSTP. Then, we have*

$$
f(\bar{r}_{i_s})=(C_0-\sum_{t=1}^{s}\bar{C}_{i_t})+\bar{r}_{i_s}(H-(D_0-\sum_{t=1}^{s}\bar{D}_{i_t})).
$$

Proof : Let $a_j=c_j-\bar{r}_{i_s}d_j$ for all $j=0,1,2,\cdots,n$. Then, we have $f(\bar{r}_{i_s})=\max(0, \bar{A}_0+\bar{r}_{i_s}H)$ by Algorithm1. Since $\bar{r}_{i_{k+1}} \geq \bar{r}_{i_k} \geq 0$ for all $k=1,2,\cdots,s$, we know that

$$
\bar{A}_{i_k} \equiv \sum_{j\in T(i_k)\setminus\bigcup_{q=1}^{k-1}T(i_q)} a_j=\bar{C}_{i_k}-\bar{r}_{i_s}\bar{D}_{i_k} \leq 0 \ \text{ for all } k=1,2,\cdots,s.
$$

Therefore, by Algorithm1, we can obtain an optimal solution of $LR(\bar{r}_{i_s})$

$$
x_j=\begin{cases} 1 & \text{if } j\notin\bigcup_{t=1}^{s}T(i_t) \\ 0 & \text{if } j\in\bigcup_{t=1}^{s}T(i_t) \end{cases}
$$

and we have

$$
f(\bar{r}_{i_s})=(C_0-\sum_{t=1}^{s}\bar{C}_{i_t})+\bar{r}_{i_s}(H-(D_0-\sum_{t=1}^{s}\bar{D}_{i_t})). \ \square\square
$$

Now, let $(\overline{CSTP})$ be the LP relaxation of CSTP. Then, we have the following theorem.

Theorem 2. *Let i_s be the critical item for CSTP. Then $f(\lambda^*)=f(\bar{r}_{i_s})$ and it is also the optimal value of $(\overline{CSTP})$. Moreover, $x^*=(x_j^*)$ defined by*

$$x_j^*=\begin{cases} 1 & \text{if } j\notin \bigcup_{t=1}^{s} T(i_t) \\ 0 & \text{if } j\in \bigcup_{t=1}^{s-1} T(i_t) \\ \mu & \text{if } j\in T(i_s) \end{cases}$$

is the optimal solution of $(\overline{CSTP})$, where $\mu=H-(D_0-\sum_{t=1}^{s}\bar{D}_{i_t})/\bar{D}_{i_s}$.

Proof : Since i_s is the critical item for CSTP, we have $H-(D_0-\sum_{t=1}^{s}\bar{D}_{i_t})\geq 0$ and $H-(D_0-\sum_{t=1}^{s-1}\bar{D}_{i_t})<0$. Therefore,

$$0\leq\mu= H-(D_0-\sum_{t=1}^{s}\bar{D}_{i_t})/\bar{D}_{i_s}=(H-(D_0-\sum_{t=1}^{s-1}\bar{D}_{i_t})+\bar{D}_{i_s})/\bar{D}_{i_s}<1,$$

and thus, $0\leq x_j^*\leq 1$ for all j. Since $x_{p_j}^*\geq x_j^*$ clearly for all $j=1,2,\cdots,n$, $x^*=(x_j^*)$ defined above is a feasible solution of $(\overline{CSTP})$. We now prove that $f(\bar{r}_{i_s})=Z^*_{(\overline{CSTP})}$, where $Z^*_{(\overline{CSTP})}$ denotes an objective value of $(\overline{CSTP})$ corresponding to x^*.

$$\begin{aligned} Z^*_{(\overline{CSTP})} &= \sum_{j\notin\bigcup_{t=1}^{s}T(i_t)} c_j+\Big(\sum_{j\in T(i_s)} c_j\cdot(H-(D_0-\sum_{t=1}^{s}\bar{D}_{i_t})/\bar{D}_{i_s}\Big) \\ &=(C_0-\sum_{t=1}^{s}\bar{C}_{i_t})+\bar{C}_{i_s}\cdot(H-(D_0-\sum_{t=1}^{s}\bar{D}_{i_t})/\bar{D}_{i_s}) \\ &=(C_0-\sum_{t=1}^{s}\bar{C}_{i_t})+\bar{r}_{i_s}(H-(D_0-\sum_{t=1}^{s}\bar{D}_{i_t})) \\ &=f(\bar{r}_{i_s}). \end{aligned}$$

But, we know that $Z_{\overline{CSTP}}\leq\tilde{f}(\lambda^*)$, where $Z_{\overline{CSTP}}$ and $\tilde{f}(\lambda^*)$ are the optimal values of $(\overline{CSTP})$ and Lagrangian dual of $(\overline{CSTP})$, respectively. Moreover, by the integrality property of USTP proven in Theorem 1, we have $\tilde{f}(\lambda^*)=f(\lambda^*)$. Therefore, $Z_{\overline{CSTP}}=f(\lambda^*)=f(\bar{r}_{i_s})$. Consequently, $x^*=(x_j^*)$ defined above is the optimal solution of $(\overline{CSTP})$. □□

Corollary 1. $U_1=\lfloor f(\lambda^*)\rfloor$ *is an upper bound on the optimal value of CSTP, where $\lfloor a\rfloor$ denotes the largest integer not greater than a.*

The following algorithm finds the critical item (and thus finds U_1) in a strongly polynomial time, $O(n^2)$ time.

Algorithm 2.

```
begin
   Apply Algorithm 1 ;
   Compute $r_i$ for all $i$ with $x_i=1$;
   $\overline{D_0}: =\Sigma_{\{j \mid x_j=1\}} d_j$ ;
    while $(\overline{D_0} > H)$ do
    begin
       $i^*: =\arg\min\{r_i \mid x_i=1\}$;
       $x_{T(i^*)}: =0$;
       for $(i \in P(i^*, 0])$ do
       begin
          $\overline{C_i} : =\overline{C_i} - \overline{C}_{i^*}$;
          $\overline{D_i} : =\overline{D_i} - \overline{D}_{i^*}$;
          $r_i : =\overline{C_i} / \overline{D_i}$ ;
        end
     end
     $\lambda^* := r_{i^*}$;   $U_1 := \lfloor f(\lambda^*) \rfloor$ ;
   end
```

Note that the output i^* of Algorithm 2 is the critical--item for CSTP. Algorithm 2 finds the critical item i^* in $O(n^2)$ time, since the while-loop in the algorithm can be repeated at most n + 1 times and also each while-loop can delete at least one node and at most n + 1 nodes. Algorithm 2 can be also interpreted as a procedure of deleting subtrees rooted at a node having the smallest ratio until the remaining subtree of T has the total demand that does not exceed H.

We call U_1 *Dantzig upper bound* and incorporate it to develop a B&B algorithm for CSTP in the next section.

5. A Branch-and-Bound Algorithm for CSTP

In this section, we develop a B&B algorithm for CSTP that utilizes the procedure of finding the critical item. *STACK* is used to store all variables which have been fixed during the algorithm. We put " i " into the *STACK* if $x_i = 1$ and put " $-i$ " otherwise. A *forward move* consists of deleting a subtree rooted at a node having the smallest ratio, which is done by the procedure **delete(·)**. The procedure **find_critical_item** which performs a sequence of *forward move* determines:

i) the critical-item
ii) a feasible solution which is used to update the incumbent solution.

After we have found the critical-item s, the root of the last deleted subtree, we branch on the node s by setting $x_s = 1$. Then, all nodes on path $P[s,0]$ must be included (i.e., set to 1). This is equivalent to compress nodes in $P[s,0]$ into a super root, which is done by the procedure **compress**(s). After the compression, we again use the procedure **find_critical_item** to obtain a new upper-bound for the compressed tree. We continue the above process until the current upper bound is less than or equal to the incumbent solution value (i.e., a fathoming condition is satisfied). Then, a *backtracking move* is performed. The *backtracking move* consists of adding subtrees deleted in the process of finding the latest critical-item by applying the procedure **add(·)**. Precisely speaking, suppose that we have just found a critical-item s and an upper-bound U_1 which is less than or equal to the incumbent solution value. Let t be the last node in *STACK* being set to 1. Then we continue adding subtrees deleted in the process of finding the critical-item s until we meet " t " in the *STACK*. Then we decompress node t by taking out t from the compressed super root. It is done by applying the procedure **decompress**(t). We continue applying **decompress(·)** until we hit the last node k in the *STACK* being set to 0. Then we branch on the node k by setting $x_k = 1$ (i.e., by applying **compress(k)**). If such a node k does not exist, the algorithm is terminated.

As computing an upper-bound is relatively expensive (it requires $O(n^2)$ time), we store upper-bounds into a stack, *STACK_UB*. Whenever we perform a **compress(·)**, we put an upper-bound into the *STACK_UB*. Whenever we perform a **compress(·)**, we take out an upper-bound from the stack.

Algorithm 3.

```
begin
  Apply Algorithm 1;
  $incumbent\_value$ := 0;
  $upper\_bound$ := $+\infty$;
  s := find_critical_item;
  $stop$ := 0;
  while ($stop$ = 0) do
  begin
    while ($incumbent\_value < upper\_bound$) do
    begin
      compress( $s$ );
      $s$ := find_criticalitem;
    end
    $back\_tracking$ := 1;
    $next\_back$ := 1;
    while ($back\_tracking$ = 1) and ($STACK \neq \emptyset$) do
    begin
      pick $s$ from $STACK$;
      if ($s > 0$) then
        decompress(s);
        $next\_back$ := 0;
        if ($STACK = \emptyset$) then
          $stop$ := 1;
        end if
      else
        $s$ := $-s$;
        add($s$);
        $back\_tracking$ := $next\_back$;
      end if
    end
  end
end
```

```
Procedure find__critical__item

begin
  $s := \arg\min \{r_i \mid x_i = 1\}$;
  $C\_check := C_0 - C_s$;
  $D\_check := C_0 - C_s$;
  while $(D\_check > H)$ do
  begin
     delete($s$);
       $s := \arg\min \{r_i \mid x_i = 1\}$;
       $C\_check := C_0 - C_s$;
       $D\_check := C_0 - C_s$;
     end
     $upper\_bound := \lfloor C\_check + r_s(H - D\_check) \rfloor$;
     if ($C\_check > incumbent\_value$) then
        $incumbent\_value := C\_check$;
     end if
     return $s$;
   end

Procedure add($s$)

begin
  $x_{T(s)} := 1$;
  $STACK := STACK \setminus \{-s\}$;
  $i := s$;
   while $(i \neq 0)$ do
   begin
      $i := p_i$;
      $C_i := C_i + C_s$;
      $D_i := D_i + D_s$;
      $r_i := C_i / D_i$;
```

```
        end
    end

Procedure delete($s$)

begin
    $x_{T(s)} := 0$;
    $STACK := STACK \cup \{-s\}$;
    $i := s$;
    While ($i \neq 0$) do
    begin
        $i := p_i$;
        $C_i := C_i - C_s$;
        $D_i := D_i - D_s$;
        $r_i := C_i / D_i$;
    end
end

Procedure compress($s$)

begin
    $STACK := STACK \cup \{ s \}$:
    $STACK\_UB := STACK\_UB \cup \{upper\_bound\}$
    modify the data structure to compress node s to the root 0;
end

Procedure decompress($s$)

begin
    $STACK := STACK \setminus \{ s \}$;
    $STACK\_UB := STACK\_UB \setminus \{upper\_bound\}$;
    modify the data structure to decompress node $s$ from the root 0;
end
```

Before we close this section, we discuss some implementation details to improve the efficiency of the computation. After a compression occurs, the total demands in the super root may exceed the capacity H. If such a case happens, the following **find_critical_item** procedure produces a trivial critical-item, i.e., $i^* = 0$ and a trivial incumbent solution value of 0. Therefore, it is better to check whether the compressed super node would cause such a trivial case or not beforehand. Now, let

$$r_{\min}(i) = \min\{r_j \mid j \in T(i) \text{ and } x_j = 1\}$$

Then, it can be computed recursively by

$$r_{\min}(i) = \min\{r_i, \min_{j \in S(i)} r_{\min(j)}\}$$

where $S(i)$ represents the set of successors of node i.

Then, the smallest ratio over the tree can be easily obtained as $r_{\min}(0)$. Whenever a subtree $T(i)$ is deleted, we only need to update $r_{\min}(\cdot)$ along the path $P[i, 0]$ and all other nodes are unchanged.

6. Example and Computational Results

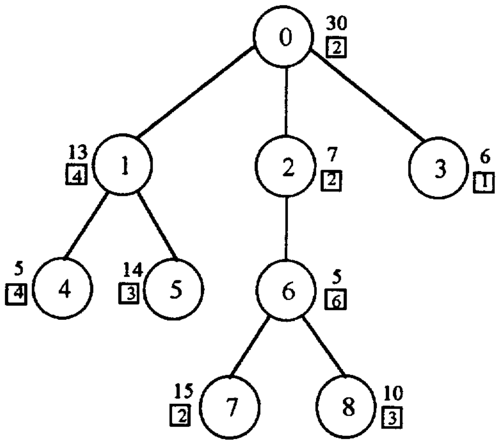

Figure 1 : Example

To illustrate our algorithm, we consider an example in Figure 1. We assume that all nodes in the tree are labeled in BFS order and $H = 18$. Numbers in the square box and above the box in Figure 1 stand for demands and profits, respectively. Initially, we set x_i to 1 for all i and assume the initial incumbent solution value is 0. First, we find a node having the minimum ratio, which is node 4, and delete the subtree rooted at node 4(in this case, node 4 itself). We then check whether the current $\overline{D_0}$(the remaining total demand) is greater than 18 or not. Since it is 23(< 18), we first store '-4' in *STACK* and update the ratio r_1 by $\frac{13+14}{4+3}=27/7$ (this procedure is done by performing **delete(4)**). We find another node having the minimum ratio from the remaining tree, which is node 6, and again delete the subtree rooted at node 6. Since the current $\overline{D_0}$ is now 12(< 18), we have found a feasible solution 70 with the objective value of 70 which gives the new incumbent solution value of max {0,70} = 70.

Note that node 6 is the critical-item. Once we find the critical-item 6, we evaluate an upper bound U_1, which is 86. Since the upper bound of 86 is greater than the incumbent solution value of 70, we may have a chance to improve the incumbent solution value by fixing more variables by either 0 or 1. Therefore, we branch on node 6 by setting x_6 to 1 and store '6' in *STACK* (this is done by **compress**(6)). Note that, after node 6 is compressed, all nodes on the path $P[6,0]$ become a super root node. Since the total demand of this compressed super root is 10(< 18), we find a node having the minimum ratio in the compressed tree, which is node 8. We then delete node 8 and have the current of $\overline{D_0}$ of 20(> 18). Then we store '-8' in *STACK* and find another node having the minimum ratio in the compressed tree. It is node 1 which is the critical-item, and thus we have found a feasible solution with the objective value of 63, which is less than the incumbent solution value of 70 (thus, the incumbent solution value is still 70). We also evaluate an upper bound of 82, which is greater than the incumbent solution value of 70, and then perform the compression by setting x_1 to 1. We store '1' in *STACK* and again try to find the critical-item and continue the above procedure.

After we repeat the above procedure, we can find an incumbent solution value and an upper bound which are equal to 76 with *STACK*$=\{-4,6,-8,1,5,-3\}$. Since it satisfies the fathoming condition, we pick the last node -3 in *STACK* and perform add(3), which adds node 3 to the current tree, which is equivalent to setting x_3 to 1. Then we continue to perform the **decompress**(·) on nodes having positive values in *STACK* until we meet a negative value in *STACK*, which is -8 here. We perform **add(8)** and follow by **compress(8)** immediately. At this point, *STACK* consists of $\{-4,6,8\}$. We again try to find the critical-item for the newly changed subtree by applying the same procedure as before. We stop when *STACK* is empty. We could find the optimal solution with the optimal value of 76 as shown in Figure 2. The decision-tree of our branch-and-bound algorithm is shown in figure 3.

We now report the computational results for our B&B algorithm for CSTP. The algorithm was coded in C language and run on a SUN SPARC 1000 workstation. All the test problems are

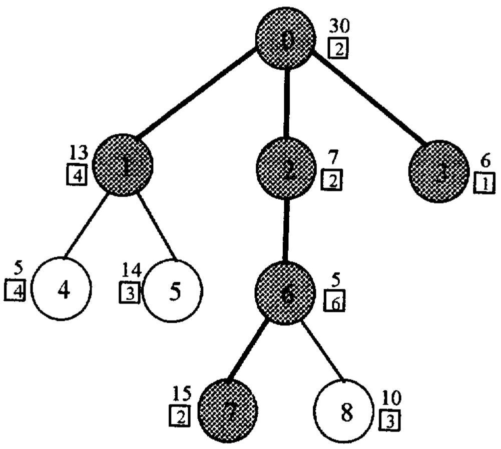

Figure 2 : Optimal tree of the example

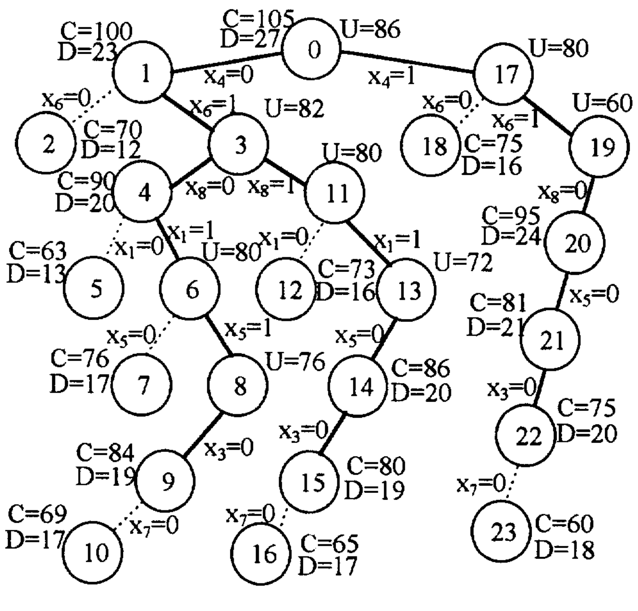

Figure 3 : Decision-tree of B&B procedure for the example

randomly generated. To generate a tree randomly, we first specified n, the total number of nodes in the tree. 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. In our test problem set, the number of nodes n was in the range [50,500] and two types of the capacity H, 5000 and 10000, were used. The demand d_i was randomly generated in the range of [1,100]. We compared our B&B code with both CPLEX and the depth-first DP code developed by Cho and Shaw [4]. 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 shows that our code is much superior to the depth-first DP code and CPLEX for most cases.

Table 1. Computational results for B&B, DP, and CPLEX

n	H	B&B			DP			CPLEX		
		worst	average	best	worst	average	best	worst	average	best
50	5,000	0.03	0.01	0.01	1.33	1.30	1.26	0.05	0.03	0.02
	10,000	0.03	0.01	0.01	2.68	2.58	2.50	0.05	0.04	0.03
100	5,000	0.03	0.02	0.01	2.66	2.60	2.51	2.10	1.23	0.22
	10,000	0.03	0.02	0.01	5.30	5.15	5.01	0.13	0.09	0.08
200	5,000	0.03	0.47	0.33	12.10	7.64	4.93	72.25	17.22	3.48
	10,000	0.80	0.03	0.03	10.45	10.31	10.15	16.35	4.01	0.18
300	5,000	0.05	1.33	1.01	7.76	7.53	7.36	69.43	28.97	8.93
	10,000	2.00	0.47	0.38	15.61	15.10	14.60	122.25	47.39	6.00
500	5,000	0.65	5.27	4.63	12.78	12.37	11.96	61.22	29.46	3.07
	10,000	5.10	3.65	3.13	38.98	27.07	24.10	207.65	53.17	6.88

7. Conclusions

In this paper, we have successfully defined the critical-item for the capacitated subtree of a tree problem(CSTP) and have shown that an upper bound on the optimal value of CSTP can be obtained by incorporating the Lagrangian dual of CSTP in $O(n^2)$ time. Based on this result, we have presented a B&B procedure for CSTP and have also discussed some implementation details which are useful for speeding up the computational time. The computational results indicate that our algorithm performs much better than the depth-first DP algorithm and CPLEX for most cases. Despite of this success, there is still plenty of room for improving upper bounds on the optimal value of CSTP, so that our B&B algorithm will have more efficient fathomming rules to speed up the computational time. Our future research will be addressed to this interesting area.

References

[1] Aghezzaf, E.H., 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] Balakrishnan, A., T.L. Magnanti, and R.T. Wong, "A Decomposition Algorithm for Expanding Local Access Telecommunications Networks," ***Operations Research***, vol. 43 (1995), pp. 43-57.

[3] Balas, E. and E. Zemel, "An Algorithm for Large Zero-One Knapsack Problem," ***Operations Research***, vol. 28 (1980), pp. 1130-1154.

[4] Cho, G. and D.X. Shaw, "A Depth-First Dynamic Programming Algorithm for The Tree Knapsack Problem," Technical Report, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, 1994.

[5] Horowitz, E. and S. Sahni, "Computing Partitions with Applications to The Knapsack Problem," ***J. of the ACM*** (1974), vol. 21, pp. 277-292.

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

[7] Johnson, D.S. and K.A. Niemi, "On Knapsacks, Partitions, and A New Dynamic Programming Technique for Trees," ***Mathematics of Operations Research*** (1983), vol. 8, pp. 1-14.

[8] Kariv, O. and S.L. Hakimi, "An Algorithmic Approach to Network Location Problems II : The p-medians," ***SIAM J. on Applied Mathematics*** (1979), vol. 37, pp. 539-555.

[9] Lawler, E.L., "Fast Approximation Algorithms for Knapsack Problems," ***Mathematics of Operations Research*** (1979), vol. 4 , pp. 339-356.

[10] Magnanti, T.L. and L.A. Wolsey, "Chapter 9. Optimal Trees," ***Handbooks in OR and MS***, vol. 7, North-Holland, 1995.

[11] Martello, S. and P. Toth, "Algorithms for Knapsack Problems," ***Annals of Discrete Mathematics*** (1987), vol. 31, pp. 213-258.

[12] Martello, S. and P. Toth, ***Knapsack Problems***, John Wiley and Sons., New York, 1990.

[13] Mirchandani, P.B. and R.L. Francis, ***Discrete Location Theory***, John Wiley and Sons., New York, 1990.

[14] Nemhauser, G.L. and L.A. Wolsey, ***Integer and Combinatorial Optimization***, John Wiley and Sons., New York, 1988.

[15] 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.