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

Geon Cho

Information Infrastructure Planning Section, ETRI

ABSTRACT

Given a tree T with a root node 0 having the capacity H and a profit c_v and a demand d_v on each node v of T, the capacitated subtree of a tree problem(CSTP) is to find a subtree of T containing the root that has the maximum total profit, the sum of profits over the subtree, and also satisfies the constraint of which the sum of demands over the subtree must be less than or equal to H. We first define the so-called critical item for CSTP and find an upper bound on the linear programming relaxation of CSTP. We then present our branch and bound algorithm for solving CSTP and finally report the computational results on a set of randomly generated test problems.

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.

Clearly, CSTP is an NP-complete problem, since the 0-1 knapsack problem is a special case of CSTP where the depth of the tree is one. CSTP can be served as a subproblem in the local access telecommunication network(LATN) design problem (see Aghezzaf, Magnanti and Wolsey [1], Balakrishnan *et al* [2] and Shaw [11]). 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 mutiple lines into a single higher frequency sinal 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 [11]).

Cho and Shaw [4] and Johnson and Niemi [6] proposed dynamic programming 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. Consequently, CSTP can be solved by a dynamic programming algorithm in $O(n\min(C^*, H))$.

In this paper, we develop a branch-and-bound algorithm for solving CSTP. The so-called *critical-item* plays a central role in determining a bound during the process of our branch-and-bound algorithm. For the 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 [8, 9]). Moreover, by using a median-finding procedure, the critical-item for the knapsack problem can be found in $O(n)$ time as in Balas and Zemel [3] and Lawler [7]. However, defining the *critical item* for CSTP is not a trivial problem because of the contiguity assumption.

One of the main contribution of this paper is to be able to successfully define the *critical item* for the CSTP and to find it in $O(n^2)$ time. Based on the procedure of finding the critical item, we develop a branch-and-bound algorithm for CSTP. Our computational results indicate that our branch-and-bound algorithm performs much faster than CPLEX, the general integer programming solver.

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 discuss the uncapacitated subtree of a tree problem(USTP). Section 4 defines the critical item for CSTP and also finds an upper bound on the optimal value of CSTP. We present our branch-and-bound algorithm for CSTP in Section 5. In Section 6, computational results are provided and 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. Define

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

Then $T(i)$ is a complete subtree rooted at node i. Define a relation '$\prec$' as follows:

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

Let H be the 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

$$\hat{V} = \arg \max_{V' \prec V} \{ \sum_{i \in V'} c_i | \sum_{i \in V'} d_i \leq H \}.$$

Let

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

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

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

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

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

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

Without loss of generality, we assume that

$$d_j \leq H, \ j = 1, 2, \cdots, n$$

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 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:

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

We assumed that nodes in T are labeled in BFS order. Then, it can be easily seen that $(USTP)$ can be solved by the following algorithm in $O(n)$ time. Let $x_{T'} = (x_i)_{i \in T'}$, where T' is a subtree of T.

Algorithm 1.

```
begin
  for all i ∈ V  do
  begin
    set C̄_i := c_i;   x_i := 1;
  end
  for i := n down to 0   do
  begin
    if (C̄_i ≤ 0)  then
      x_{T(i)} := 0;
    else
      C̄_{p_i} := C̄_{p_i} + C̄_i;
    end if
  end
end
```

Algorithm 1 is a bottom-up process, which follows the reverse of BFS order. Clearly, $\max(0, \bar{C}_0)$ is the optimal value of $(USTP)$. It is also obvious that $x_i = 1$ implies $C_i \geq 0$, but the reverse is not true. Now, we prove that the linear programming(LP) relaxation of $(USTP)$ satisfies the integrality property.

Theorem 1. *The linear programming relaxation of (USTP) has an integer optimal solution.*
Proof: Given undirected tree $T = (V, E)$, we define a directed out-tree $T = (V, A)$, where $A = \{(p_i, i) | i \neq 0, i \in V\}$. Let B be the arc-node incidence matrix of T. Then the constraint $x_{p_i} \geq x_j$ can be written as $B^T x \leq 0$. 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 $Z_{LP} \leq \sum_{j=0}^{n} |c_j|$. Therefore, the system of linear inequalities in (LP), $B^T x \leq 0$, is Totally Dual Integral(TDI) (see Nemhauser and Wolsey [10] 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 integral extreme points. □

4 Critical Item and Upper Bound

In this section, we first define the *critical item* for CSTP. Let $C_i = \sum_{j \in T(i)} c_j$, $D_i = \sum_{j \in T(i)} d_j$, and $r_i = C_i / D_i$ (r_i is called the *critical ratio*).

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

Now, we use the Lagrangian relaxation method to obtain an upper-bound on the optimal value of CSTP. The Lagrangian relaxation of CSTP is as follows:

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

Then, we have the following Lagrangian dual of CSTP.

$$(LD) \qquad \begin{aligned} f(\lambda^*) = \quad & \min f(\lambda) \\ & \lambda \geq 0. \end{aligned}$$

For a given λ, $LR(\lambda)$ is an USTP. Therefore, the LP relaxation of $LR(\lambda)$ has an integer optimal solution by Theorem 1. Moreover, we have the following theorem.

Theorem 2. *Let i_s be the critical item for CSTP. Then*

$$f(\lambda^*) \equiv f(r_{i_s}) = (C_0 - \sum_{t=1}^{s} C_{i_t}) + r_{i_s}(H - (D_0 - \sum_{t=1}^{s} D_{i_t}))$$

is the optimal value of both (LD) and LP relaxation of CSTP. Moreover, $x^* = (x_j^*)$ defined by

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

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

Proof : Let i_s be the critical item for CSTP. Then, we have $H - (D_0 - \sum_{t=1}^{s} D_{i_t}) \geq 0$ and $H - (D_0 - \sum_{t=1}^{s-1} D_{i_t}) < 0$. Since $x_{p_j}^* \geq x_j^*$ for all $j = 1, 2, \cdots, n$, $x^* = (x_j^*)$ defined above is a feasible solution of LP relaxation of CSTP. We now prove that $f(\lambda^*) = \bar{Z}_{CSTP}$, where $\bar{Z}_{CSTP}$ denotes the objective value of LP relaxation of CSTP corresponding to x^*.

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

But, by the integrality property of USTP, we have $f(\lambda^*) = \tilde{f}(\lambda^*)$, where $\tilde{f}(\lambda^*)$ denotes the optimal value of Lagrangian dual of LP relaxation of CSTP. Therefore, $f(\lambda^*)$ is the optimal value of both (LD) and LP relaxation of CSTP. Clearly, $x^* = (x_j^*)$ defined above is the optimal solution of LP relaxation of 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.

Algorithm 2.

begin
  Apply **Algorithm 1**;
  Compute r_i for all i with $x_i = 1$;
  $\bar{D}_0 := \sum_{\{j|x_j=1\}} d_j$;
  $k := 0$;
  while $(\bar{D}_0 > H)$ **do**
  begin
    $k := k + 1$;
    $i^* := \arg\min\{r_i | x_i = 1\}$;
    $x_{T(i^*)} := 0$;
    for $(i \in P[i^*, 0] \setminus \{i^*\})$ **do**
    begin
      $\bar{C}_i := \bar{C}_i - \bar{C}_{i^*}$;
      $\bar{D}_i := \bar{D}_i - \bar{D}_{i^*}$;
      $r_i := \bar{C}_i / \bar{D}_i$;
    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. Clearly, Algorithm 2 can be interpreted as a procedure of deleting subtrees rooted at nodes having the smallest critical 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 branch-and-bound algorithm for CSTP in the next section.

5 A Branch-and-Bound Algorithm for CSTP

In this section, we develop a branch-and-bound 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 critical ratio, which is done by the procedure **delete**$(\cdot)$. 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**$(\cdot)$. 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**$(\cdot)$ 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**($\cdot$), we put an upper bound into the $STACK\_UB$. Whenever we perform a **compress**($\cdot$), we take out an upper-bound from the stack.

Algorithm 3.

```
begin
  apply Algorithm 1;
  incumbent_value := 0;
  upper_bound := +∞;
  s := find_critical_item;
  stop := 0;
  while (stop = 0)  do
  begin
    while (incumbent_value < upper_bound)  do
    begin
      compress(s);
      s := find_critical_item;
    end
    back_tracking := 1;
    next_back := 1;
    while (back_tracking = 1) and (STACK ≠ ∅)  do
    begin
      pick s from STACK;
      if(s > 0)  then
        decompress(s);
        next_back := 0;
        if (STACK = ∅)  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 | x_i = 1};
  C_check := C_0 − C_s;
  D_check := D_0 − D_s;
  while (D_check > H)  do
  begin
    delete (s);
    s := arg min{r_i | x_i = 1};
    C_check := C_0 − C_s;
    D_check := D_0 − D_s;
  end
  upper_bound := ⌊C_check + r_s(H − D_check)⌋;
  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 \ {-s};
  i := s;
  while (i ≠ 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 ⋃{-s};
  i := s;
  while (i ≠ 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 ⋃{s};
  STACK_UB := STACK_UB ⋃{upper_bound};
  modify the data structure to compress node s to the root 0;
end

Procedure decompress(s)

begin
  STACK := STACK \ {s};
  STACK_UB := STACK_UB \ {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

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

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 $r_{\min}()$ along the path $P[i,0]$ and all other nodes are unchanged.

6 Computational Results

In this section, we report the computational results for our branch-and bound algorithm for CSTP. The algorithm was coded in C language and run on a SUN SPARC 1000 workstation. All the test problems are 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. Depending on the capacity H, the demand d_i was randomly generated from the intervals of [1,100] and [1,1000]. Since no other computational results for algorithms for solving CSTP had ever been reported in the literature, we just compared our B&B code with CPLEX. 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 algorithm is much superior to CPLEX.

n	H	B&B			CPLEX		
		worst	average	best	worst	average	best
50	5,000	0.03	0.01	0.01	0.05	0.03	0.02
	10,000	0.03	0.01	0.01	0.05	0.04	0.03
100	5,000	0.03	0.02	0.01	2.10	1.23	0.22
	10,000	0.03	0.02	0.01	0.13	0.09	0.08
200	5,000	0.80	0.47	0.33	72.25	17.22	3.48
	10,000	0.05	0.03	0.03	16.35	4.01	0.18
300	5,000	2.00	1.33	1.01	69.43	28.97	8.93
	10,000	0.65	0.47	0.38	122.25	47.39	6.00
500	5,000	6.35	5.27	4.63	61.22	29.46	3.07
	10,000	5.10	3.65	3.13	207.65	53.17	6.88

Table 1. Computational results for Comparing B&B and CPLEX

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 branch-and-bound 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 CPLEX. However, there is still plenty of room for improving upper bounds on the optimal value of CSTP, so that our branch-and-bound algorithm will have more efficient fathomming rules. Our future research will be addressed to this interesting area.

References

[1] Aghezzaf, E.H., Magnanti, T.L. and Wolsey, L.A., "Optimizing Constrained Subtrees of Trees," Technical Report, Center for Operations Research & Econometrics, Universite Catholique De Louvain, Louvain-La-Neuve, Belgium, 1992.

[2] Balakrishnan, A., Magnanti, T.L. and Wong, R.T., "A Decomposition Algorithm for Expanding Local Access Telecommunications Networks," Technical Report, Sloan School of Management, M.I.T., Cambridge, MA, 1991.

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

[4] Cho, G. and Shaw, D.X., "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 Sahni, S., "Computing Partitions with Applications to The Knapsack Problem," *J. of the ACM* 21 (1974) pp. 277-292.

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

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

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

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

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

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