Sociedad de Estadística e Investigación Operativa
Top (1998) *Vol.* 6, *No.* 2, pp. 205–221

Efficient Solution Methods for Covering Tree Problems

T. B. Boffey
University of Liverpool
Liverpool L69 3BX, Great Britain

Abstract

In recent years, interest has been shown in the optimal location of 'extensive' facilities in a network. Two such problems – the Maximal Direct and Indirect Covering Tree problems – were introduced by Hutson and ReVelle. Previous solution techniques are extended to provide an efficient algorithm for the Indirect Covering Tree problem and the generalization in which demand covered is attenuated by distance. It is also shown that the corresponding problem is NP-hard when the underlying network is not a tree.

Key Words: location, covering, tree covering.

AMS subject classification: 90B80, 90C27, 90C35.

1 Introduction

A considerable literature has built up on the location of point facilities in a network including, in particular, covering models. Recently, interest has also been shown in the location of more extensive facilities such as paths and trees (Mesa and Boffey, 1996). Hutson and ReVelle (1989) and Church and Current (1993) noted the relevance of tree structures with regard to placing dirt roads in a logging region: trees are felled and logs are hauled to a nearest road for onward transport provided there is a road within some prescribed distance S. Other relevant situations mentioned include planning of subway, electricity transmission, road and pipeline networks. Hutson and ReVelle (1989, 1993) introduced two tree covering problems and solved example problem instances by integer programming. Related tree covering problems have been studied by Kincaid et al., (1988) and Kim et al., (1989, 1990).

In 1989, Hutson and ReVelle introduced two versions of the *Maximal Direct Covering Tree Problem* (or 'MDCT' for short) which we will refer to

Received: August 1997; Accepted: November 1998

as the *anchored* and *floating* forms and denote, respectively, by 'MDCTa' and 'MDCTf'. For both, an underlying tree $T = (V, E)$ is given with $V = V(T)$ being the set of nodes and $E = E(T)$ the set of arcs. Each node k has an associated weight $c_k^o \geq 0$ (its *population*) and each arc ij an associated length $d_{ij}^o \geq 0$ (its *cost*). The general aim is to determine a subtree T^* of T which maximizes the *cover* $(\Sigma_{k \in V(T^*)} c_k^o)$ provided by T^* and minimizes its *cost* $(\Sigma_{ij \in E(T^*)} d_{ij}^o)$. The anchored variant imposes the constraint that a specified node, $\mathcal{A}$, called the *anchor node*, must belong to T^*, whereas for the floating variant there is no such constraint. (Actually, in the original paper, a subtree was specified rather than an anchor *node* but it may be seen that shrinking this subtree to a single node does not materially affect the problem and no generality has been lost.) Four years later the same authors introduced the *Maximal Indirect Covering Tree Problem* (or 'MICT' for short) which differs from the direct case in that a node can be covered from a node of T^* if it is within a service distance $S \geq 0$ of that node irrespective of whether the covered node is, or is not, in T^* itself. Again anchored and floating versions MICTa and MICTf can be defined in an analogous and obvious way.

Consider first MDCTa and suppose that the arcs of T have been assigned orientations so that they all 'point towards' the anchor node. Then, to each node $i \in V \setminus \{\mathcal{A}\}$ there is a unique outgoing arc ij which we refer to as the *out arc from* i. The MDCTa problem may now be formulated as a bicriterion integer linear program in which $x_{ij} = 1$ if and only if arc ij is in the solution subtree T^*, and $y_k = 1$ if and only if node k is covered. ($y_{\mathcal{A}}$ is fixed at 1.)

$$\text{MDCT}a\text{: maximize} \sum_{k \in V} c_k^o y_k$$

$$\text{minimize} \sum_{ij \in E} d_{ij}^o x_{ij}$$

subject to

$$y_j = x_{jk}, \quad \forall j \neq \mathcal{A},\ jk \in E \tag{1}$$

$$x_{ij} \leq x_{jk}, \quad \forall j \neq \mathcal{A},\ jk \in E,\ \forall ij \in E \tag{2}$$

$$y_k, x_{ij} \in \{0,1\}, \quad \forall k \in V, \forall ij \in E \tag{3}$$

Constraints in set (2) reflect the requirement that if $j \neq \mathcal{A}$ and ij is in

T^* then so must be the (unique) next arc jk towards the anchor node. (3) are the usual integer constraints. Constraints in set (1) state that $j \neq \mathcal{A}$ is in T^* if and only if jk (the out arc from j) is. It may be noted that this formulation differs from that of Hutson and ReVelle (1989) only in the constraints defining cover. They regard a node as being covered if *any* incident arc is in the solution subtree. Thus, using the present notation the Hutson–ReVelle model may be written as above except that constraint set (1) is replaced by

$$y_j \leq x_{jk} + \Sigma_{ij \in E} x_{ij}, \forall j \neq \mathcal{A}, jk \in E \qquad (1')$$

While the two formulations are equivalent this is not, in general, true for the linear programming relaxations. This is important since the 'stronger' constraints (1) can potentially help to eliminate fractional optimal solutions of the linear programming relaxation.

Of course, such a bicriterion problem will not, in general, possess a feasible solution that optimizes each objective in isolation and it is natural to look for *efficient solutions* (Cohon, 1978). Hutson and Revelle (1989) generated the *efficient frontier*, that is, the set of *extreme* efficient solutions (in objective function space). This is achieved by solving the single objective problem MDCT$a(w)$:

$$\text{MDCT}a(w)\text{: maximize } w\Sigma_{k \in V} c_k^o y_k - (1-w)\Sigma_{ij \in E} d_{ij}^o x_{ij}$$
$$= \Sigma_{k \in V} c_k y_k - \Sigma_{ij \in E} d_{ij} x_{ij}$$
$$\text{subject to (1), (2) and (3)}$$

for a carefully selected set of values of the parameter w lying in the range $0 < w < 1$. Note that, for simplicity of exposition, we replace wc_k^o by c_k and $(1-w)d_{ij}^o$ by d_{ij} from now on, the weight w being accounted for implicitly rather than explicitly. Similarly, single objective problems MDCT$f(w)$, MICT$a(w)$ and MICT$f(w)$ may be defined by replacing the separate cover and distance objectives by the weighted objective used in MDCT$a(w)$.

MDCT$a(w)$ may possess alternative optima – this will happen when there is an optimal solution subtree T^* to which an arc (or arcs) can be added which will increase the cover (relative to coefficients $\{c_k\}$) by exactly the same amount that the cost (relative to coefficients $\{d_{ij}\}$) is increased. To facilitate our discussion it will be assumed from now on that T^* is the unique maximal solution subtree of MDCT$a(w)$, that is, no more arcs can be added – this will be called the *maximality assumption.*

Temporarily denoting the optimal solution subtree of MDCT$a(w)$ by $T^*(w)$, it is clear that every node belonging to $T^*(w_1)$ also belongs to $T^*(w_2)$ if $w_1 < w_2$ – that is, increasing the relative importance of 'cover' cannot lead to nodes becoming 'uncovered'. This nesting of subtrees implies there can be at most n (the number of nodes in the underlying tree network) distinct solution subtrees. Moreover, it is not difficult to construct an example in which there are indeed n distinct subtrees. The complexity of determining the efficient frontier is thus n times the complexity of solving the corresponding single objective problem. Consequently, we shall from now on be concerned with solving the single objective problems.

Hutson and ReVelle solved MDCT$a(w)$ by using a commercial integer programming package. They acknowledged that more efficient methods might be developed but added that 'These solution methods may yield significant improvements in solution times for direct covering tree problems, but are not expected to apply readily to the indirect covering tree problem'. Church and Current (1993) introduced an alternative formulation, devised an algorithm that performed operations carried out directly on the underlying tree T and pointed out that MICT$f(w)$ could be solved by solving MDCT$f(w)$ on a modified network though the the modified network might possess $O(n^2)$ nodes. The same strategy could be adopted with the Hutson–ReVelle approach. It will be shown here that there are $O(n^2)$ time algorithms for both MDCT$f(w)$ and MICT$f(w)$.

In the next section algorithms for MDCT$a(w)$ and MDCT$f(w)$ are outlined. Section 3 then looks at the indirect covering problems: an algorithm and proof of validity are given. In section 4, a generalization in which cover is attenuated by distance to the facility is outlined. It is also shown that MDCT$a(w)$ becomes NP-hard for general underlying networks.

2 Algorithms For MDCT$a(w)$ and MDCT$f(w)$

In order to establish ideas, at this point we only outline network algorithms for MDCT$a(w)$ and MDCT$f(w)$. More precise descriptions and proof of validity follow, by setting $S = 0$, from the treatment of the next section.

Since constraints (1) are equalities, we may add $\Sigma_{i \neq A} \lambda_i (x_{ij} - y_i)$ to the objective of MDCT$a(w)$ without changing the identity of optimal solutions or their values. The new objective is thus $Z = \Sigma_{k \in V} \hat{c}_k y_k - \Sigma_{ij \in E} \hat{d}_{ij} x_{ij}$ where, for $k \neq \mathcal{A}$, $\hat{c}_k = c_k - \lambda_k$ is the *reduced population* at node k ($\hat{c}_{\mathcal{A}} = c_{\mathcal{A}}$) and, for all $ij \in E, \hat{d}_{ij} = d_{ij} - \lambda_i$ is the *reduced length* of ij. This will be referred to as *(population) trading*, the reduction in length of the (unique) arc from a node being exactly balanced by the reduction in the population of that node.

Consider now a particular node $i \neq \mathcal{A}$ and set $\lambda_i = \min\ (c_i, d_{ij})$ resulting in $\hat{c}_i = 0$ or $\hat{d}_{ij} = 0$. It is clear that solving MDCT$a(w)$ with the 'reduced' data leads to the same optimal solution subtree and solution value – if $ij \notin T^*$ then nothing is affected whereas if $ij \in T^*$ then cover and cost have been reduced by the same amount leading to no net change in objective. Suppose $\hat{d}_{ij} = 0$ and $j \in T^*$, then arc ij can be added at zero extra cost and thus must also belong to T^* (by the maximality assumption). That is, we now have $x_{ij} = y_j$. Provided $j \neq \mathcal{A}$, this together with $y_i = x_{ij}$ and $y_j = x_{jk}$ (see formulation of MDCTa) yields a new constraint $y_i = x_{jk}$. Thus, in a sense, cover is being provided indirectly by jk. Adding $\lambda_{ik}(x_{jk} - y_i)$ to the objective with $\lambda_{ik} = \min\ (\hat{c}_i, \hat{d}_{jk})$ will result in $\hat{c}_i = 0$ ($\hat{c}_i$ here denoting population after *two* reductions) or $\hat{d}_{jk} = 0$, in addition to $\hat{d}_{ij} = 0$. In this way population can be traded from a node until the reduced population is zeroed or $\mathcal{A}$ is reached in which case i must be in T^* – this will be termed *cascading*. (An alternative interpretation is to consider the zero length arc as being shrunk to a point thus *coalescing* its end nodes the new 'composite node' having the total of the populations of its 'constituent nodes'. That is, the process of trading could be regarded as producing an equivalent problem on fewer nodes.) Figure 1 illustrates the effect of population trading for the data given in figure 1(a). First trading population from nodes 1, 2, 3, 4 and 5 in turn leads to $\hat{c}_1 = \hat{c}_2 = \hat{c}_3 = \hat{c}_4 = \hat{c}_5 = 0$ and $\hat{d}_{1\mathcal{A}} = 2$, $\hat{d}_{21} = 0$, $\hat{d}_{31} = 1$, $\hat{d}_{41} = 1$ and $\hat{d}_{54} = 0$ (see figure 1(b)). So far no cascading has taken place. Now consider node 6: trading three units of population gives $\hat{c}_6 = 4$ and $\hat{d}_{63} = 0$; trading a further one unit gives $\hat{c}_6 = 3$ and $\hat{d}_{31} = 0$; and finally, trading a further two units yields $\hat{c}_6 = 1$ and $\hat{d}_{1\mathcal{A}} = 0$ – the trading is now complete since $\mathcal{A}$ has been encountered (see figure 1(c)). The solution to the problem is clearly as shown in figure 1(d).

It is noted that the above process is applicable since each node of the underlying network other than the anchor node has a unique outgoing arc. The nodes can be considered in any order but a simple and effective order is to work outwards from node $\mathcal{A}$.

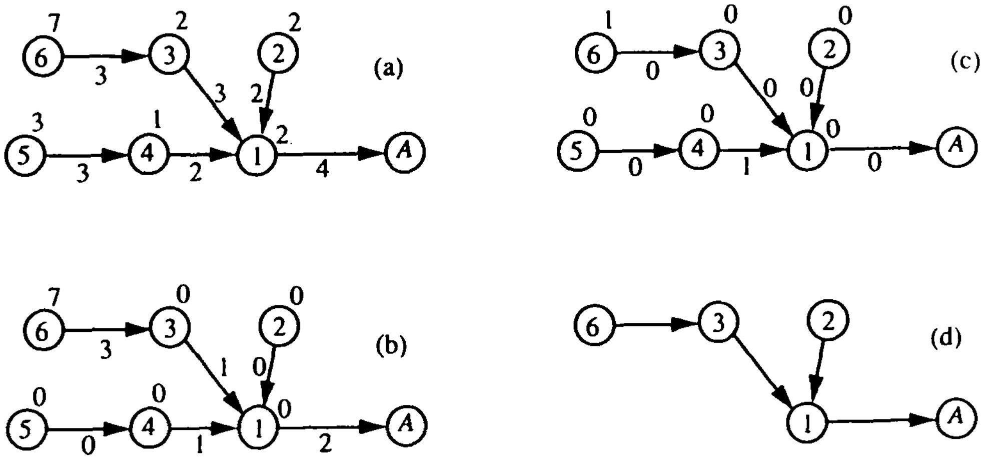

Figure 1 (a) the original tree T with populations shown beside nodes and arc lengths beside arcs. (b) After trading population from nodes 1, 2, 3, 4 and 5. (c) After trading from node 6. (d) The optimal solution subtree T^*.

MDCT$f(w)$ can be solved by taking each node in turn as the anchor node and comparing the n solution subtrees obtained. This gives an $O(n^2)$ algorithm, where $n = |V|$, though the following result might lead to some reduction in computational effort.

Theorem 2.1. *If b is in the optimal subtree T_e^* relative to e as anchor node, then the optimal subtree T_b^* relative to b as anchor node is contained in T_e^*.*

Proof. The arcs in T_b^* but not in T_e^* together with their end nodes form a set of disjoint trees τ_i where τ_i has the node i in common with both T_b^* and T_e^*. Then the objective value (cover minus cost) for each τ_i must be nonnegative otherwise they would not be part of T_b^*. However, this means that all nodes in each τ_i could be added to T_e^* without worsening the objective. This is a contradiction because of the optimality of T_e^* and the maximality assumption.

It may be noted that the above result (and proof) remains valid for the case of indirect cover ($S > 0$).

3 Indirect Covering

MICT$a(w)$ can be solved in much the same way as that described for MDCT$a(w)$. First, note that if the length of arc ij is less than S then i can be covered even though ij is not in the solution subtree. This leads us to modify the concept of 'out arc'.

Definition 3.1. Let $\pi(i) \equiv i_1 i_2 \ldots i_{t-1} i_t$ be the unique path from $i = i_1$ to $\mathcal{A} = i_t$. Suppose $i_r i_{r+1}$ is an arc of $\pi(i)$ satisying $d(i, i_r) \leq S$, $d(i, i_{r+1}) > S$, where $d(x, y)$ denotes the distance from x to y relative to the unreduced arc lengths. Then $i_r i_{r+1}$ is called the *S-out arc* from i and i_r is the *S-out node* of i. On the other hand if no arc satisfies this condition then there is no S-out arc from i and the *S-out node* of i is taken to be $\mathcal{A}$.

Population at node i is not traded for the length of any arc between i and its S-out arc $i_r i_{r+1}$ (if such exists) since inclusion of $i_r i_{r+1}$ would ensure that i is covered albeit indirectly. (Of course the distance between i and i_r may later be reduced when trading population from a node other than i.)

We are now in a position to formulate the algorithm. $\mathcal{L}$ will denote the set of nodes 'ready for trading' and corresponds to the set of temporarily labelled nodes in the context of label setting shortest path algorithms. The algorithm terminates when $\mathcal{L} = \emptyset$

```
Algorithm {Input: a tree $T$ with arcs directed towards anchor node $\mathcal{A}$.}
Step 1 (Setup)
    Set $\mathcal{L} = \{m \mid m\mathcal{A} \in E(T)\}$.
    Set $\hat{c}_k = c_k, \forall k \in V(T)$; $\hat{d}_{ij} = d_{ij}, \forall ij \in E(T)$.
Step 2 (Termination/node selection)
    If $\mathcal{L} = \emptyset$
    then Set $\mathcal{R} = \{i \mid i$ is at zero reduced distance from $\mathcal{A}\}$.
            Set $T^*$ to be the subtree on node set $\mathcal{R}$. Stop.
    else Select $i \in \mathcal{L}$.
```

```
Step 3 (Out arc check)
    If $i$ has an $S$-out arc $jk$
    then Set $ab \leftarrow jk$.
    else Set $\mathcal{L} \leftarrow (\mathcal{L} \setminus \{i\} \cup \{m \mid mi \in E(T)\}$. Go to step 2.
Step 4 (Population trading)
    Set $\lambda = \min\ (\hat{c}_i, \hat{d}_{ab})$.
    Set $\hat{c}_i \leftarrow \hat{c}_i - \lambda$ and $\hat{d}_{ab} \leftarrow \hat{d}_{ab} - \lambda$.
Step 5 (Cascade check)
    If $b = \mathcal{A}$ or $\hat{c}_i = 0$
    then Set $\mathcal{L} \leftarrow (\mathcal{L} \setminus \{i\} \cup \{m \mid mi \in E(T)\}$. Go to step 2.
    else Replace $ab$ by next nonzero reduced length arc in $T$ towards $\mathcal{A}$.
        Return to step 4.
```

Theorem 3.1. *The algorithm terminates after $O(n)$ population trades at most where $n = |V(T)|$, and the optimal solution subtree is the subtree of T on node set $\mathcal{R}$.*

Proof. The algorithm terminates after $O(n)$ population trades since for a tree $|E(T)| = n - 1$ and at each trade either a reduced population becomes zero or a reduced arc length becomes zero.

Next we note that trading as described does not affect the identity of the optimal solution subtree. This is so since:

1. MICT$a(w)$ can be formulated as an integer program with among its constraints ones of the form $y_i = x_{jk}$ where jk is the S-out arc from i;
2. adding $\lambda = \min(\hat{c}_i, \hat{d}_{jk})$ times $(x_{jk} - y_i)$ to the objective does not alter the value of any solution and so the modified problem has the same optimal solution subtree;
3. trading from i can take place using arcs jk, $k\ell$, ... successively (ignoring zero length arcs) until trading is complete or $\mathcal{A}$ is encountered (see the discussion in section 2).

Consequently we may find the optimal solution subtree by solving the *final problem*, that is, relative to the reduced populations and reduced arc lengths at termination of the algorithm.

Let $\mathcal{R}^+$ be the set of nodes not in $\mathcal{R}$ but at a distance not greater than S (relative to *unreduced* arc lengths) from a node in $\mathcal{R}$. Now T^* is the subtree on node set $\mathcal{R}$ so all nodes in $\mathcal{R}^+$ are covered by T^*. Consider $i \notin \mathcal{R}^+$. By the definition of $\mathcal{R}^+$, i must have an S-out arc $j'k'$ say, with $j' \notin \mathcal{R}$. Consequently, either $j'k'$ or some arc on the path from j' to $\mathcal{A}$ has nonzero length. This means that $\hat{c}_i = 0$ since otherwise further population could have been traded from node i. Hence, in the final problem, adding to T^* any node not in $\mathcal{R}^+$ provides no extra cover but increases the cost (since every arc $j''k''$ with $j'' \notin \mathcal{R}$ and $k'' \in \mathcal{R}$ has nonzero reduced length). It follows that the optimal subtree is a subtree of T^* and the proof is complete by the maximality assumption.

While only $O(n)$ population trades are required, this does not automatically mean that the algorithm can obtain an optimal subtree T^* in $O(n)$ time. However, by implementing the algorithm appropriately the following result holds.

Corollary 3.1. *The algorithm can be implemented such that:*

(1) T^* can be determined in $O(n)$ and $O(n^2)$ time for MDCTa(w) and MDCTf(w) respectively. The efficient frontier can be found in $O(n^2)$ time for MDCTa.

(2) T^* can be determined for both MICTa(w) and MICTf(w) in $O(n^2)$ time. The efficient frontier can be found in $O(n^2)$ time for MICTa.

In order to show how the result of theorem 2 translates into the time complexities quoted we briefly outline an implementation of the algorithm. In order to facilitate the following discussion we shall assume, for convenience, that all arc lengths and node populations are strictly positive; however, the complexity results are still valid if this assumption is not made.

First, it is essential that the network be represented in an appropriate way. One possibility is to use a *Forward Star* representation, that is, predecessor nodes are stored in blocks in an array, *pred* say, with predecessors of node 1 first, followed by predecessors of node 2, etc. The beginning of the block of predecessors of node i is stored in array element *start(i)*. (Another possibility is to keep the successors of each node as a linear linked list – this would have advantages but some disadvantages also and was not tried.)

Further arrays that can be used are *succnode*, *succarc*, *outnode* and *first* where

succnode(i) stores the successor node of $i, i \neq \mathcal{A}$

succarc(i) stores the successor arc of $i, i \neq \mathcal{A}$

outnode(i) stores the S-out node of i

first(i) is used to determine the first node between i and $\mathcal{A}$ with a nonzero length outgoing arc (or, $\mathcal{A}$ if no such arc exists).

Consider first MDCT$a(w)$. *succnode*, *succarc* and *outnode* can be set up in $O(n)$ time (*outnode(i)* being just i). Initially $first(i)$ is set to i for all i. When trading in step 4 results in the length of arc ab being reduced to zero the operation *first(j)* $\leftarrow$ *first(b)* is performed. To see the effect of this, consider a linear chain and trade from nodes working away from $\mathcal{A}$. As execution of the algorithm proceeds, so 'strings' of zero reduced length arcs, which we term *zero sequences*, form and grow. More precisely

Definition 3.2. A *zero sequence*, ZS, is a nonempty set of arcs satisfying:

(1) all arcs in ZS have zero reduced length;

(2) arcs in ZS form a path $\pi \equiv i_1 i_2, \ldots, i_s$ in T; the node i_1 is called the *start node* of ZS.

(3) ZS is maximal in that no set of arcs $ZS' \neq ZS$, $ZS' \supset ZS$, satisfies (1) and (2).

It may be seen that the update *first(i)* $\leftarrow$ *first(b)* maintains the condition that *first(i)* points to the start of the first arc between i and $\mathcal{A}$ with nonzero reduced length *when i is the start node of a zero sequence*. For the linear chain this is sufficient to enable jumping over zero reduced length arcs when required in step 5.

For a more general tree network, however, it may be necessary to use *first(i)* for nodes 'within' a zero sequence. Assuming nodes are considered for trading in a depth-first manner, this can be taken account of straightforwardly by updating *first(i)* as i is encountered when backtracking along a branch. This latter modification permits arcs to be jumped over in step 5 after their lengths have been reduced to zero. Again such operations can be accomplished in $O(1)$ time for each node visited and hence in $O(n)$ time overall.

Since initialisation of *succnode*, *succarc*, *outnode* and *first*, and updates of *first* (as indicated above) can all be carried out in $O(n)$ time it follows the algorithm can solve MDCT$a(w)$ in $O(n)$ time. This immediately implies an $O(n^2)$ algorithm for MDCTa.

The situation is not so simple for indirect cover. It is not difficult to see that all S-out nodes (S-out arcs) can be calculated initially in $O(n^2)$ time. While this may seem very conservative the author was unable to devise a way of *guaranteeing* that the algorithm will run in $O(n)$ time. (The difficulty is that, though rather unlikely, $O(n)$ nodes may be separated from their S-out nodes by $O(n)$ arcs.) This yields only an $O(n^2)$ time algorithm for MICT$a(w)$. However, for MICTa the S-out nodes need only be calculated once so the efficient frontier can be found in $O(n^2)$ time.

For the floating variants of the tree covering problem, there is no essential difference between the direct and indirect cover cases and we shall discuss only the indirect case. For MICT$f(w)$ it is true that MICT$a(w)$ may need to be solved n times and the set of S-out nodes will be different at each application. However, we can get round this by traversing the network moving from one node to another with only incremental changes being required. Using a 'depth-first search' this may be accomplished in $2n - 2$ steps (once along each arc in either direction). Each step requires updating the Forward Star representation to reflect the new anchor node – this is 'messy' but requires only $O(n)$ time. A full calculation of *first* must be performed for the initial application of MICT$a(w)$ but thereafter *outnode* and *dist* can be updated in a single pass over the nodes at each subsequent application of MICT$a(w)$. For this it is only necessary to change the values of *outnode(i)* for a few i from the old anchor node to the new one, and for a few i from the new anchor node to the old one – again this can be accomplished in $O(n)$ time. With this strategy the initial $O(n^2)$ calculation of *outnode* values is ameliorated over the $O(n)$ MICT$a(w)$ problems that are solved in order to solve MICT$f(w)$. Consequently, the latter problem can be solved in $O(n^2)$ time.

4 Nnmerical Experiments

The algorithm was tested on tree networks randomly generated as follows. The number of nodes, n, is specified then, for each node i in order of generation, p predecessor nodes (and hence arcs incoming at i) are generated. p is sampled from a uniform distribution on $\{0, 1, ..., 4\}$ but is capped as soon as n nodes have been generated. Costs c_i are sampled from a uniform distribution on the set $\{5, 6, ..., C\}$ and lengths d_{ij} from a uniform distribution on the set $\{5, 6, ..., L\}$. Clearly, if C/L is very small then the solution subtree will be very small (perhaps just the anchor node) and if L/C is very small the solution subtree will be large (perhaps the whole of the underlying tree). We used $C = 20$ and $L = 25$ leading, *very roughly*, to 40% of the nodes being covered for the case of direct cover and 80% for indirect cover when $S = 10$. In the indirect case, the cover was nearly always provided by smaller solution subtrees than for direct cover (for the same weight w) – in our experiments, with parameters as specified above, the solution subtree contained, *very roughly*, 30% of all nodes.

The algorithm was coded in Ada and run on a Sun 3. Results for MDCT$a(w)$ and MICT$a(w)$ are given in Table 1 with each row corresponding to a set Gxxx of five separate problems each with xxx nodes. T denotes the *total* of the separate times (in seconds) of all five problems in a set. (The last digit is always even since the timing mechanism rounds to the nearest 0.02 seconds.) It is seen that the results are consistent with a linear time complexity for MDCT$a(w)$. *For the data tested*, MICT$a(w)$ appeared to require linear time, that is, better than the $O(n^2)$ time suggested by the Corollary to Theorem 2.

The floating variants naturally took much longer as each problem potentially required the solution of n corresponding anchored problems. Consequently, the implementation outlined in section 3 for MDCT/MICT$f(w)$ was tested for the smaller data sets G20, G40, G60 and G80 on 20, ... , 80 nodes respectively and each containing five separate problems. The result of Theorem 1 was incorporated by omitting to perform trading for an anchor node which had been found present in the solution subtree for an earlier anchor node – of course, the overheads of updating the network and data structures still needed to be carried out. The values of $100T/n^2$ (where again T is the total time for five separate problems) were 10.0, 9.6, 9.4 and 9.1 for G20, ..., G80 respectively. This is consistent with the $O(n^2)$ complexity quoted in the Corollary.

Problem set	n	S	T (secs)	$100T/n$
G100	100	0	0.08	0.08
G200	200	0	0.10	0.05
G300	300	0	0.18	0.06
G400	400	0	0.22	0.055
G500	500	0	0.24	0.05
G100	100	10	0.10	0.10
G200	200	10	0.14	0.07
G300	300	10	0.28	0.09
G400	400	10	0.34	0.085
G500	500	10	0.40	0.08

Table 1 Anchored covering tree problems

5 Extensions

At this point we note that Hutson and ReVelle required the solution subtree to have at least one arc. If desired, this may be accommodated as follows. Let $T(i)$ and $T(j)$ be the two subtrees that would result from T if arc ij were removed where $i \in T(i)$ and $j \in T(j)$. Now take each arc ij in turn and apply our algorithm with the following modification: i is regarded as the anchor node for the subtree $T(i)$ and j is regarded as the anchor node for the subtree $T(j)$.

Next, consider the situation in which demand at node i is satisfied by traveling to a nearest node j of the facility. (An example is provided by passengers traveling to a station to access a subway network.) It may be expected that willingness to travel decreases as the distance $d(i,j)$ between i and j increases. In other words, the *level of cover* y_i at i is $h(d(i,j))$ where the *attenuation function* h may reasonably be assumed to satisfy:

$$h(0) = 1; \;\; h(x) \geq 0, \;\; x > 0; \;\; h \text{ is monotonic decreasing,}$$

and y_i is no longer restricted to being an integer. We now outline the way in which population trading can be modified to take account of attenuation.

Let $\pi(i) \equiv i_1 i_2 \ldots i_{t-1} i_t$ be the unique path from $i = i_1$ to $\mathcal{A} = i_t$ and suppose that $i_r i_{r+1}$ is not in the solution subtree but all arcs from i_{r+1} to $\mathcal{A}$ are. Node i is a distance $d(i, i_{r+1})$ from the solution subtree so the level of cover at i is $h(d(i, i_{r+1}))$. Upon adding arc $i_r i_{r+1}$ to the subtree, the level of cover at i increases to $h(d(i, i_r))$. That is, the *extra* level of cover provided by adding arc $i_r i_{r+1}$ is $\epsilon_{i_r i_{r+1}} = h(d(i, i_r)) - h(d(i, i_{r+1})) \geq 0$. Based on this observation we obtain the constraint

$$y_i - \epsilon_{i_1 i_2} x_{i_1 i_2} - \cdots - \epsilon_{i_{t-1} i_t} x_{i_{t-1} i_t} - \delta_i = 0 \tag{5}$$

where $\delta_i = h(d(i, \mathcal{A}))$. To see that this is correct, note that if $i_1 i_2, i_1 i_2, \ldots, i_{s-1} i_s$ are **not** in the solution subtree, but $i_s i_{s+1}, \ldots, i_{t-1} i_t$ (where $i_t = \mathcal{A}$) are, then

$$\begin{aligned} y_i &= \sum_{j=s}^{t-1} \epsilon_{i_j i_{j+1}} + \delta_i \\ &= \sum_{j=s}^{t-1} [h(d(i, i_j)) - h(d(i, i_{j+1}))] + h(d(i, i_t)) \\ &= [h(d(i, i_s)) - h(d(i, i_t))] + h(d(i, i_t)) \\ &= h(d(i, i_s)) \end{aligned}$$

as it should. Adding λ times the left hand side of (5) to the objective gives an equivalent problem and the update in step 4 of the algorithm above becomes:

Set $\lambda = \min(\hat{c}_i, \min\{\hat{d}_{jk} \mid jk \in Q\})$
Set $\hat{c}_i = \hat{c}_i - \lambda_i$; $\hat{d}_{jk} = \hat{d}_{jk} - \lambda$, $\forall jk \in Q$.

where $Q = \{jk \mid jk \in \pi(i) \text{ and } \epsilon_{jk} \neq 0\}$). If $Q = \emptyset$ then min $\{jk \mid jk \in Q\}$ is interpreted as zero. The idea of an S-out arc is no longer applicable and step 3 is removed from the algorithm. The same effect is obtained for the MICT$a(w)$ by only trading population for length of those arcs jk with *nonzero* ϵ_{jk}.

Trading as just specified either results in $\hat{c}_i = 0$, or $\hat{c}_i \neq 0$ but $\hat{d}_{jk} = 0$ for some $jk \in \pi(i)$. In the latter case arc jk could automatically be added to the subtree if k is in the subtree. That is, if $k \neq \mathcal{A}$ and $\hat{d}_{k\ell} \neq 0$ we have the constraint $x_{jk} = x_{k\ell}$ which may be accounted for by setting

$\epsilon_{k\ell} = \epsilon_{k\ell} + \epsilon_{jk}$, $k \neq \mathcal{A}$ and $\delta_i = \delta_i + \epsilon_{jk}$, $k \neq \mathcal{A}$

then resetting ϵ_{jk} to zero – arc jk now takes no part in further population trading. This corresponds to the first cascading step – further cascading towards $\mathcal{A}$ can take place as other reduced lengths become zero.

When all trading has taken place $\hat{c}_i = 0$ or $\hat{d}_{jk} = 0$ for all $jk \in \pi(i)$ since if this were not so then further trading could take place. The optimal subtree is, as before, the subtree on the set of nodes a zero reduced distance from $\mathcal{A}$. The proof of validity of the algorithm is now essentially that of theorem 2.

The reader may note that the above prescription for population trading reduces to that of the algorithm for MICT$a(w)$ by setting $h(x) = 1$ if $x \leq S$; and $h(x) = 0$ if $x > S$.

While there are clearly applications in which the solution T^* should be a subtree, it is less clear why the underlying network should be constrained to be a tree. The present author when studying a generalization of the Hierarchical Network Design Problem (Current et al., 1986) in which cover is not mandatory, developed a Lagrangean relaxation in which one subproblem is just MDCT$a(w)$ over a general network. Since Lagrangean techniques typically involve solving many relaxations (corresponding to different multiplier sets) the solution of these relaxations should be as efficient as possible. For these reasons it is appropriate to look briefly at the more general version, MDCTGa, of MDCTa where the underlying network need not be a tree.

MDCTGa is the problem of finding a subtree, T^*, of a symmetric network $N = (V, E)$ containing a given node $\mathcal{A}$ so as to maximize cover provided and to minimize cost involved. (By a 'symmetric network' we mean that ji is in the network whenever ij is, and has the same length.) As before, $y_i = 1$ if and only if $i \in T^*$ and $x_{ij} = 1$ if and only if ij is an arc of T^*. Then we may assert that $y_i = \Sigma_{j|ij \in E} x_{ij}$ which means that population trading from node i can be performed by subtracting the same amount λ_i from *each* of the arcs *outgoing* from i as is subtracted from the population. However, we cannot validly cascade trading towards $\mathcal{A}$ as before since there is no longer a unique path from i to $\mathcal{A}$. (It is not difficult to construct an example in which cascading would lead to an upper bound to the optimal value of MDCTG$a(w)$ for which the bound is not exact.)

Theorem 5.1. *MDCTGa(w) is NP-hard.*

Proof. Let $\mathcal{A}$ be the anchor node with U a subset of $V \setminus \{\mathcal{A}\}$. Set the populations of all nodes in U to be M (a very large number) and the populations of all other nodes to be zero. Then it is clear that if M is large enough every node in $U \cup \{\mathcal{A}\}$ must be in T^* but otherwise all that is required is to minimize cost. This is clearly a Steiner Tree problem in which the nodes in $U \cup \{\mathcal{A}\}$ are to be connected. Since the Steiner tree problem is NP-hard then so must be MDCT$a(w)$ over a general network (Garey and Johnson, 1979).

6 Conclusion

An algorithm, similar to that of Church and Current (1993), has been given for solving MDCT$a(w)$ in $O(n)$ time and MICT$a(w)$ in $O(n^2)$ time. Numerical results were consistent with these complexities though, for the data tested, the time required for MICT$a(w)$ grew more nearly linearly.

We feel that MDCT$f(w)$ is probably a less practically oriented tree covering problem. However, a relatively efficient solution of this is still possible.

The way in which the algorithm may be generalized to the case in which cover is attenuated by distance was also outlined. Finally, we have proved that MDCT$a(w)$ becomes NP-hard when the underlying network is not restricted to being a tree, and consequently so are the other covering problems mentioned. It is unlikely, therefore, that any polynomial time algorithm exists for their solution.

References

Church, R. L. and J. R. Current (1993). Maximal covering tree problems. *Naval Research Logistics* **40**, 129–142.

Cohon, J. L. (1978). *Multiobjective Programming and Planning*. Academic Press, New York.

Current, J. R., C. S. ReVelle and J. L. Cohon (1986). The hierarchical network design problem. *European Journal of Operational Research* **21**, 188-197.

Garey, M. J. and D. S. Johnson (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

Hutson, V. A. and C. S. ReVelle (1989). Maximal direct covering tree problems. *Transportation Science* **23**, 288-299.

Hutson, V. A. and C. S. ReVelle (1993). Indirect covering tree problems on spanning tree networks. *European Journal of Operational Research* **65**, 20-32.

Kim, T. U., T. J. Lowe, J. E. Ward and R. L. Francis (1989). A minimum length covering subgraph of a network. *Annals of Operations Research* **18**, 245-260.

Kim, T. U., T. J. Lowe, J. E. Ward and R. L. Francis (1990). A minimum length covering subtree of a tree. *Naval Research Logistics Quarterly* **37**, 309-326.

Kincaid, R. K., T. J. Lowe and T. L. Morin (1988). The location of central structures in trees. *Computers & Operations Research* **15**, 103-113.

Mesa, J. A. and T. B. Boffey (1996). Location of extensive facilities in networks. *European Journal of Operational Research* **95**, 592-603.