Available online at www.sciencedirect.com
SCIENCE DIRECT
Computer Communications 28 (2005) 1371–1379
computer communications
www.elsevier.com/locate/comcom

Least cost heuristic for the delay-constrained capacitated minimum spanning tree problem

Yong-Jin Lee[a], M. Atiquzzaman[b,*]

[a]Department of Computer Science, Woosong University, 17-2 Jayang-Dong, Dong-Ku, Taejon 300-718, South Korea
[b]School of Computer Science, University of Oklahoma, 200 Felgar Street, Norman, OK 73019-6151, USA

Received 2 September 2004; revised 10 January 2005; accepted 10 January 2005
Available online 21 January 2005

Abstract

This study deals with the Delay-Constrained Capacitated Minimum Spanning Tree (DC-CMST) problem arising in topology discovery of local network. While the traditional Capacitated Minimum Spanning Tree (CMST) problem deals with only traffic capacity constraint of a port of a source node, and Delay-Constrained Minimum Spanning Tree (DCMST) considers only the maximum end-to-end delay constraint, the DC-CMST problem deals with mean network delay and traffic capacity constraints simultaneously. The DC-CMST problem consists of finding a set of minimum cost spanning trees to link end-nodes to a source node which satisfy the traffic requirements at end-nodes and the required mean delay of a network. We formulate the DC-CMST problem and present the Least-Cost DC-CMST Heuristic, which consists of node exchange algorithm, node shift algorithm, and mean delay link capacity algorithm to solve the DC-CMST problem. Results from performance analysis show that the proposed heuristic can produce, in a very short computation time, better results than the previous CMST algorithms modified to solve the DC-CMST problem. The proposed Least-Cost DC-CMST Heuristic can be applied to the topological design of large local networks and to the construction of least cost broadcast and multicast trees for efficient routing algorithms.

Keywords: Routing; Topology discovery; DC-CMST problem; Networking; Heuristic algorithm

1. Introduction

Network topology specifies the configuration of connecting computers in a network. Topology affects important factors such as the communication cost and transmission speed of a network. Thus, topology discovery is an important research area in computer networks. Several researchers have proposed algorithms for discovering the topology of the Internet backbone [1–3,9,12–14]. A Local Area Network (LAN) connected to a backbone node can be regarded as an end user node.

In a local network, the total traffic volume of end-user nodes that can be served by a port of the backbone node is limited. So, a *local network* consists of a backbone node (source) and several trees that cover all end user nodes to satisfy the constraints on traffic volume. Topology discovery is required to find all the trees in a local network, which satisfy the traffic constraints. In the literature, issues related to topology discovery for local networks have been classified into two problems: Capacitated Minimum Spanning Tree (CMST) and Delay-Constrained Minimum Spanning Tree (DCMST) problems; they have been solved using heuristic [5,8,20] and exact [4,6,7,11] algorithms.

The CMST problem, which finds a set of minimum spanning trees rooted at the source node satisfying the traffic constraint, is known to be NP-complete [15]. Since optimal solutions for a large problem cannot be obtained in a reasonable amount of computing time, heuristic methods have been developed to obtain approximate solutions for large problems. Among heuristic methods, the algorithm proposed by Esau and William [5] (we call it the EW solution) provides near optimal solution with a memory complexity of $O(n^2)$ and a time complexity of $O(n^2 \log n)$ when the number of nodes, except the source node, is n. EW

* Corresponding author. Tel.: +1 405 325 8077; fax: +1 405 325 4044.
E-mail addresses: yjlee@wsu.ac.kr (Y.-J. Lee), atiq@ou.edu (M. Atiquzzaman).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.01.001

solution is usually used as a benchmark for performance evaluation of solutions to the CMST problem.

The DCMST problem finds the least cost broadcast and multicast trees rooted at the source node which have the minimum total cost among all possible spanning trees having a maximum end-to-end delay less than or equal to a given delay constraint Δ [10,16,17,18,22,23]. The DCMST problem is used for real-time multimedia applications that have rigid end-to-end delay requirements and consume large amount of network resource.

The CMST and the DCMST problems consider traffic capacity and the end-to-end delay constraints, respectively. However, communication delay between end nodes deteriorates quality of service (QoS) of users as the network size grows. To solve the above issue, it is necessary to simultaneously consider the traffic capacity and the mean network delay constraints in a common framework. We, therefore, present the Delay-Constrained Capacitated Minimum Spanning Tree (DC-CMST) problem in this paper.

The DC-CMST problem consists of finding a set of spanning trees from a source node to a number of end-nodes which satisfy the traffic constraints along with the constraint of mean delay of network. The DC-CMST problem has the additional second constraint that the mean delay of a network has to be within a fixed time, as in the CMST problem. It is very important to consider the delay constraint because communication delay between end nodes deteriorates quality of service of users as the network grows.

In this paper, we formulate the problem and present the heuristic (we call it the Least-Cost DC-CMST Heuristic) for the DC-CMST problem which has not been reported in any previous work. The proposed Least-Cost DC-CMST Heuristic is composed of three algorithms (node exchange algorithm, node shift algorithm, and mean delay capacity allocation algorithm) in two phases. Our heuristic uses node sets created by the EW solution for the CMST problem as the initial solution, and continues to improve the solution while changing the topology through application of node exchange and node shift algorithms to nodes between trees based on trade-off criterions. Furthermore, our heuristic assigns link capacities optimally by calling the mean delay link capacity allocation algorithm in order to satisfy the desired mean delay, and finds a set of minimum cost spanning trees.

Several properties of the Least-Cost DC-CMST Heuristic show that our heuristic has small memory and time complexity. Moreover, our performance results demonstrate that the proposed heuristic improves the solutions, which are produced by previous CMST algorithms, modified to satisfy the mean delay constraint in the short execution time. The main contributions of this paper are as follows: (i) presenting and formulating the DC-CMST problem, and (ii) proposing and evaluating the performance of the proposed heuristic solution for the DC-CMST problem.

The rest of this paper is organized as follows. Section 2 presents the mathematical formulation of the DC-CMST problem. Section 3 describes our proposed heuristic algorithm for the DC-CMST problem. Performance of our proposed algorithm is reported in Section 4, followed by concluding remarks in Section 5.

2. Formulation of the DC-CMST problem

We consider the graph $G=(V, E)$, where V and E represent set of nodes and edges, respectively. In this paper, we use edge and link interchangeably. V is composed of $(n+1)$ nodes with indices $\{0, 1, 2,\ldots, n\}$. The source node (node 0) can be viewed as an object, such as the backbone router, with several ports. Now, we want to construct trees connecting to end-user nodes and satisfying the delay constraint. End-user nodes can be a switching hub or a host that generates traffic.

While the CMST and the DCMST problems deal with the traffic capacity and the maximum end-to-end delay constraint, respectively, the DC-CMST problem deals with both the traffic capacity and network mean delay constraint simultaneously, i.e. the mean delay has to be within a given time.

The objective of the DC-CMST problem is to find a collection of least-cost trees rooted at a source node. Since the number of nodes that a port can serve is limited in a real environment, several trees have to be found. In addition, to satisfy the network mean delay constraint as QoS of end-user node, we have to consider the capacity allocation for links included in the trees. In the rest of this section, we formulate the DC-CMST problem.

2.1. Terminology

n	the number of nodes except source node
$lcnt$	the number of trees included in the solution
$kcnt(p)$	the number of nodes included in tree p ($p=1,.,1cnt$)
$cost(p)$	the cost of links included in tree p ($p=1,\ldots,lcnt$)
q_i	traffic requirement at node i ($i=1,.,n$)
$node(p)$	the node set corresponding to tree p
$path(p)$	the link set on the Minimum Spanning Tree (MST) corresponding to $node(p)$
$edge(p)$	the minimum connection cost from tree p to the source node
d_{ij}	distance between node pair (i,j) ($i=0,1,.,n$; $j=0,1,.,n$)
D	distance matrix
d	unit cost of link capacity
d_k	traversal distance of link k in any topology ($k=1,.,n$)
MAX	the maximum traffic to be handled in single spanning tree (or the number of nodes)
$sub1$	index of the tree inclusive of node i

$sub2$ index of the tree inclusive of node j
ks_{ij} trade-off of node pair (i,j) in node exchange algorithm $(i=1,\ldots,n;\ j=1,\ldots,n)$
ms_{ij} trade-off of node pair (i,j) in node shift algorithm $(i=1,\ldots,n;\ j=1,\ldots,n)$
T mean delay time of network (s)
T_k mean delay time of link k in any topology (s)
v total traffic between source-destination pairs
$1/\mu$ average packet length (bits/packet)
C_k the capacity of link k in any topology (bits/s)
λ_k traffic flow on link k in any topology (packets/s)
$Delay$ desired mean delay time of network (s)
D_{cost} total link cost obtained at each phase of DC-CMST algorithm
NEW_{cost} total link cost of any topology

2.2. Assumptions

Our formulation of the DC-CMST problem in the next section is based on the following assumptions.

(1) There is only one source node and its capacity is not limited.
(2) The traffic generated at a node cannot exceed the maximum traffic covered by one tree (maximum$\{q_i\} \le MAX$, $1 \le i \le n$).
(3) The traffic at a node is not splintered.
(4) The total traffic exceeds the maximum traffic served by a tree $\left(\sum_{i=1}^{n} q_i > \mathrm{MAX}\right)$.
(5) Arrival of packets is based on a Poisson distribution.
(6) Service time of packets are exponentially distributed.

2.3. Problem formulation

We formulate the DC-CMST problem in order to satisfy the constraints that the total network mean delay time (T) is less than a given delay time ($Delay$) and also satisfies the traffic capacity limitation imposed by the existing CMST problem. The DC-CMST problem is depicted in Fig. 1.

Generally, the mean delay time (T) of the network is given by Eq. (1) [21]. The reason that the number of links is n in Eq. (1) is because the solution of the DC-CMST problem is a set of spanning trees which has n links when the number of nodes is n.

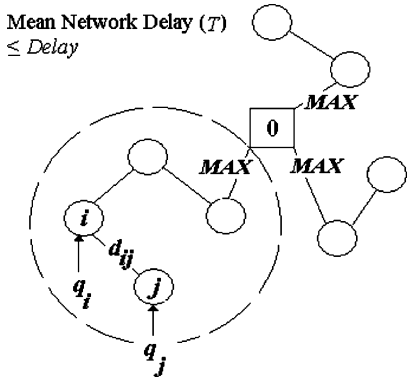

Fig. 1. DC-CMST problem.

$$T = \frac{1}{v}\sum_{k=1}^{n} \lambda_k T_k \tag{1}$$

Since each link, k, can be regarded as an independent M/M/1 queue, mean delay time (T_k) of link k (from queuing theory) is given by Eq. (2).

$$T_k = \frac{1}{(\mu C_k - \lambda_k)} \tag{2}$$

Substituting Eq. (2) into Eq. (1) gives the network mean delay

$$T = \frac{1}{v}\sum_{k=1}^{n} \lambda_k \left[\frac{1}{(\mu C_k - \lambda_k)}\right] \tag{3}$$

The total link cost is assumed to be a linear function as given by Eq. (4).

$$D_{cost} = \sum_{k=1}^{n} dC_k d_k \tag{4}$$

Having defined the network mean delay (Eq. (3)) and the objective function (Eq. (4)) of the DC-CMST problem, we can formulate the DC-CMST problem as in Eqs. (5)–(9).

$$\text{Minimize } D_{cost} = \sum_{k=1}^{n} dC_k d_k \tag{5}$$

S.T.

$$\frac{\lambda_k}{\mu} \le C_k \tag{6}$$

$$T = \frac{1}{v}\sum_{k=1}^{n} \lambda_k \left[\frac{1}{(\mu C_k - \lambda_k)}\right] \le Delay \tag{7}$$

$$\sum_{i \in R_j} q_i x_{ij} \le MAX \tag{8}$$

$$\sum_{i,j} x_{ij} = n \tag{9}$$

$$x_{ij} = 0 \text{ or } 1$$

The objective function of the DC-CMST problem is to find a collection of trees with minimal link cost (Eq. (5)). Constraints of the DC-CMST problem are: (i) average traffic flow on a link should be smaller than capacity of the link (Eq. (6)), (ii) mean delay of the network has to be dropped within allowable mean delay time ($Delay$) (Eq. (7)), (iii) total traffic flow in one tree must be below MAX (Eq. (8)), and (iv) n nodes have to be included in the solution (Eq. (9)). If the number of nodes that a tree takes charge of should be smaller than a specific value, we change q_i into 1, $\forall i$ in Eq. (8).

3. Least-cost heuristic for the DC-CMST problem

EW solution is known as an efficient method providing near optimum solution to the CMST problem. We, therefore, use the EW solution as a first step in our solution of the DC-CMST problem. However, any alternative to the EW solution can also be used as a first step in our solution. The Least-Cost DC-CMST Heuristic computes trade-off criterions between each node pair using sets of nodes included in each tree and least connection costs to the source node. Our solution consists of first applying the EW solution followed by repeatedly applying the Node Exchange Algorithm (NEA) or Node Shift Algorithm (NSA) in order to improve the solution. NEA improves the solution by exchanging nodes between trees based on some trade-off criterions. Node Shift Algorithm (NSA) also improves the solution by shifting nodes to other trees based on some trade-off criterions (which are different from those used in NEA). Both NEA and NSA use the Mean Delay Link Capacity Allocation Algorithm (MDLCAA) that assigns link capacities in order to satisfy the mean delay time constraint, and computes the total cost. Our Least-Cost DC-CMST Heuristic can be represented as follows:

Algorithm 1. Least-Cost DC-CMST Heuristic

1. Find the EW solution
2. Phase 1: Execute **NEA** (call **MDLCAA**)
   If it is impossible to extend for all spanning trees, then goto 4.
   else
   Phase 2: Execute **NSA** (call **MDLCAA**)
3. If node set is changed, goto 2
4. Heuristic is terminated.

3.1. Node exchange algorithm (NEA)

NEA improves any intermediate solution (including the EW solution) by repeatedly exchanging nodes between node sets starting with the EW solution. Let the node sets corresponding to two minimum spanning trees be $node(sub1)$ and $node(sub2)$, and the link costs be $cost(sub1)$ and $cost(sub2)$, respectively. Also, if we let the two new minimum spanning trees obtained by exchanging one element between the two node sets be $sub1'$ and $sub2'$, the corresponding node sets be $node(sub1')$ and $node(sub2')$ and the corresponding costs be $cost(sub1')$ and $cost(sub2')$ respectively, the difference between two costs can be expressed by Eq. (10).

$$sav = cost(sub1) + cost(sub2) - cost(sub1') - cost(sub2') \tag{10}$$

If sav is positive, solution obtained from the previous application of NEA can be improved. In order to maximize sav, we have to apply the Minimum Spanning Tree (MST) algorithm to each tree in Eq. (10). However, when the number of trees is large, the required number of applications of the MST algorithm in each step of Eq. (10) becomes too large. For a large network, we cannot obtain the solution within a reasonable amount of time. We, therefore, define a heuristic trade-off criterion in Eq. (11), where $edge(sub1)$ is the minimum link cost to the source node among nodes included in the spanning tree ($sub1$) to which node i belongs, i.e. $edge(sub1) =$ minimum $\{d_{m0}\}$, $m \in node(sub1)$, and likewise, $edge(sub2) =$ minimum $\{d_{m0}\}$, $m \in node(sub2)$.

$$\begin{aligned}
&\text{If } sub1 \neq sub2 \text{ and } \sum_{m \in node(sub1)} q_m + q_j - q_i \leq MAX \text{ and} \\
&\sum_{m \in node(sub2)} q_m + q_i - q_j \leq MAX \\
&ks_{ij} = edge(sub1) + edge(sub2) - d_{ij} \\
&\text{else} \\
&ks_{ij} = -\infty
\end{aligned} \tag{11}$$

$sub1 = sub2$ implies that nodes i and j belong to the same spanning tree. Even if nodes are exchanged in this case, the cost is the same, and hence the trade-off is set to $-\infty$ to exclude it from the solution. In the case that traffic requirements at nodes are different from each other, the sum of traffic at each tree must be smaller than the maximum traffic (MAX) even if nodes are exchanged for the node pair (i,j).

First, NEA calls MDLCAA on node set of each tree obtained by the EW solution, and then MDLCAA carries out the most suitable link capacity allocation to satisfy the mean delay time ($Delay$) and finds the initial cost (NEW_{cost}). For the node pair (i,j), $i<j$, which belong to the node set of other trees, NEA finds ks_{ij} using Eq. (11). For the node pair (i,j) which belongs to the node set of the same tree, ks_{ij} are set to $-\infty$. Starting from the node pair (i,j) with the maximum positive value of ks_{ij}, we obtain two new tress by exchanging node i for node j. Two new MSTs are found by applying the MST algorithm [19] to these two node sets. The MST algorithm is applied to only two node sets because other node sets already form minimum spanning trees.

Next, to compute the total cost, we compare the cost (D_{cost}) returned from MDLCAA with the existing cost (NEW_{cost}). If $D_{cost} < NEW_{cost}$, NEW_{cost} is substituted for D_{cost} and the corresponding ks_{ij} is set to $-\infty$. Also, we repeat the above process after computing the ks_{ij} matrix again. Otherwise, the corresponding ks_{ij} is set to $-\infty$ and we repeat the above process on node pair (i,j) with the maximum value among the ks_{ij} matrix, which is relevant to former NEW_{cost}. The Node Exchange Algorithm is as follows.

Algorithm 2. Node exchange algorithm (NEA)

step 1: Obtain *node*(*p*) in the EW solution and *path*(*p*) by applying MST algorithm to *node*(*p*).
Call **MDLCAA** on *path*(*p*) and obtain D_{cost}.
Set $NEW_{cost} = D_{cost}$.
step 2: If node *i* and *j* is included in the different *node*(*p*),
Compute the trade-off criterion (ks_{ij}) for node pair (*i*,*j*) by using Eq. (11).
Otherwise, set $ks_{ij} = -\infty$.
If all ks_{ij}'s are negative, stop.
step 3: While there is (*i*,*j*) such that $ks_{ij} > 0$,
Exchange node (*i*,*j*) with the maximum positive ks_{ij}.
Obtain two *path*(*p*)'s by applying the MST algorithm.
Call **MDLCAA** on the modified *path*(*p*)'s and find D_{cost}.
step 4: If $D_{cost} \geq NEW_{cost}$,
Restore node (*i*,*j*) to the previous state.
Set $ks_{ij} = -\infty$ and repeat step 3.
Otherwise, set $NEW_{cost} = D_{cost}$ and set $ks_{ij} = -\infty$ and go to step 2.

3.2. Mean delay link capacity allocation algorithm (MDLCAA)

The MDLCAA is called during the execution of the node exchange and shift algorithm. MDLCAA, called in step 1 of NEA, is carried out on the links belonging to each tree on the EW solution, but is executed only on the links belonging to the newly created two trees if node exchange took place. In order to meet the mean network delay constraint, MDLCAA first finds the mean arrival rate (λ_k) on links of MSTs corresponding to two node sets or all the node sets in step 1 of NEA, and then assigns link capacities. This is formulated in Eq. (12).

$$\text{Minimize } D_{cost} = \sum_{k=1}^{n} dC_k d_k$$

S.T.

$$\frac{1}{\nu}\sum_{k=1}^{n}\frac{\lambda_k}{\mu C_k - \lambda_k} = Delay \tag{12}$$

The optimal solution of Eq. (12) is obtained using the Lagrange multiplier as shown in Eq. (13).

$$C_k = \frac{\lambda_k}{\mu}\left[1 + \frac{\sum_{j=1}^{n}\sqrt{d\lambda_j d_j}}{\nu \cdot \text{Delay}\sqrt{d\lambda_k d_k}}\right] \tag{13}$$

Finally, MDLCAA computes the total cost (D_{cost}) using Eq. (4), and returns the result to NEA or NSA. The details of MDLCAA is given below.

Algorithm 3. Mean delay link capacity allocation algorithm (MDLCAA)

step 1: Compute λ_k by using the fixed routing.
(Note that all *path*(*p*)'s are concerned at the initial call from NEA
but only two *path*(*p*)'s are concerned after the second call from NEA or NSA.)
step 2: Compute the link capacity(C_k) of link *k* on the topology by using Eq. (13).
step 3: Compute the network cost(D_{cost}) by using Eq. (4)
step 4: Return D_{cost} to NEA or NSA.

3.3. Node shift algorithm (NSA)

NSA uses information (node sets, link sets, cost, etc.) obtained in NEA, and improves the solution obtained from NEA or EW solution by moving a node included in a node set of a specific tree to a different tree. If the sum of traffics of all node sets is equal to MAX (or if the number of nodes is equal to MAX), NSA is not carried out. Otherwise, trade-offs for node pair (*i*,*j*) belonging to the different trees are computed, and if node *j* can be moved to the other tree, node *j* is moved into *node*(*sub*1) to which node *i* belongs. Then, node *j* is deleted from *node*(*sub*2), and the new node set is found. Trade-off criterions for NSA are defined by Eq. (14).

$$\text{If } sub1 \neq sub2 \text{ and } \sum_{m \in node(sub1)} q_m + q_j \leq MAX$$

$$ms_{ij} = d_{ij} - d_{\max},$$

where $d_{\max} = maximum\{edge(sub1),\ edge\ (sub2)\}$

else

$$ms_{ij} = \infty \tag{14}$$

Under the condition that *sub1* is not equal to *sub2*, if sum of traffics on the tree, *sub1* and traffic of node *j* is larger than *MAX*, the node can not be moved. Then, the corresponding trade-off, ms_{ij} is set to ∞. Otherwise, ms_{ij} is set to $d_{ij} - d_{max}$. Starting from node pair (*i*,*j*) with the least negative ms_{ij}, we find new node sets by moving *j* to the tree inclusive of node *i*. The procedure of comparing the existing cost (NEW_{cost}) with the cost (D_{cost}) returned from MDLCAA and applying the MST algorithm is similar to that of NEA. The NSA algorithm is as follows.

Algorithm 4. Node shift algorithm (NSA)

step 1: If $kcnt(p) = MAX$ for all *p*, stop.
step 2: If node *i* and *j* are included in the different *node*(*p*) respectively and for loop *p* containing node *i*, $kcnt(p)+1$ is less than or equal to *MAX*, Compute the trade-off criterion (ms_{ij}) for node pair (*i*,*j*) by using Eq. (14).
Otherwise, set $ms_{ij} = \infty$.
If all ms_{ij} are positive, stop

step 3: While there is (i,j) such that $ms_{ij}<0$,
Move node j to $node(p)$ containing node with the minimum negative ms_{ij}.
Obtain two $path(p)'$s by applying the MST algorithm.
Call **MDLCAA** on the modified two $path(p)'$s and find D_{cost}.
step 4: If $D_{cost} \geq NEW_{cost}$,
Restore node (i,j) to the previous state. set $ms_{ij} = \infty$ and repeat step 3.
set $ms_{ij} = \infty$ and repeat step 3.
Otherwise, set $NEW_{cost} = D_{cost}$ and $ms_{ij} = \infty$ and goto step 2.

3.4. Property of the Least-Cost DC-CMST Heuristic

In order to determine various properties (such as memory and time complexity) of the Least Cost DC-CMST Heuristic, we present the following Lemmas.

Lemma 1. No cycles are included in the solution by the Least-Cost DC-CMST Heuristic, and all trees included in the solution are minimum spanning trees.

Proof. The Least-Cost DC-CMST Heuristic uses node sets of minimum spanning trees obtained by the EW solution. In the case that node sets are exchanged or moved, it applies the minimum spanning tree algorithm to the changed node sets. Since a minimum spanning tree has no cycles, it is natural that no cycles are included in the solution of the Least-Cost DC-CMST Heuristic, resulting in every tree being a minimum spanning tree.

Lemma 2. All elements of the trade-off matrix in NEA become negative in a finite number of steps.

Proof. We assume that ks_{ij}*'s* are positive for some (i,j). For node pair (i,j) with positive ks_{ij}, NEA sets ks_{ij} to $-\infty$ after exchanging node i for node j. In the worst case, if all node pairs are exchanged with each other, all ks_{ij}'s are set to $-\infty$. Since the trade-off matrix has finite number elements, all elements of the matrix become negative in finite steps.

Lemma 3. Memory complexity of the Least-Cost DC-CMST Heuristic is $O(n^2)$.

Proof. d_{ij}, ks_{ij}, ms_{ij} $(i=1,\ldots,n; j=1,\ldots,n)$ are stored as two-dimensional data in memory. Therefore, NEA, NSA, and MDLCAA have memory complexity of $O(n^2)$, respectively. Since memory complexity of the EW solution is $O(n^2)$, the total memory complexity of the Least-Cost DC-CMST Heuristic is also $O(n^2)$.

Lemma 4. Time complexity of MDLCAA is $O(n)$.

Proof. We apply only MDLCAA of the Least-Cost DC-CMST Heuristic in order to find the solution that considers mean delay time using the trees obtained by any CMST algorithm. Because the MST obtained by any algorithm for the CMST problem has n links, the maximum number of links that we must compute in MDLCAA is n. Time complexity for each expression in MDLCAA is $O(1)$, and time complexity to get the maximum value of link load is the same as the number of links. Thus, total execution time complexity is maximum $[O(1), O(n)] = O(n)$.

Lemma 5. When MDLCAA is applied to the EW solution, time complexity is $O(n^2 \log n)$.

Proof. Because we have to add execution time of MDLCAA to the execution time of the EW solution, the total execution time must take the maximum of the two times. Thus, the total time complexity is maximum $[O(n^2 \log n), O(n)] = O(n^2 \log n)$.

Lemma 6. Time complexity of the Least-Cost DC-CMST Heuristic is $O(n^2 \log n)$ if the number of nodes to be included in a tree is limited to MAX.

Proof. First, we consider NEA of the Least-Cost DC-CMST Heuristic. The time complexity of the EW solution is $O(n^2 \log n)$. Since the maximum number of links as input to step 1 is n, and time complexity of MDLCAA is $O(n)$ by Lemma 4, the complexity of step 1 is $O(n)$. In step 2, ks_{ij}'s are computed for node pair (i,j), $i \in node(sub1)$, $j \in node(sub2)$, and the maximum computed number is $1/2(n-MAX)(n+MAX-1)$. So, the time complexity is $O(n^2)$. In the worst case, step 3 must carry out MDLCAA and step 2 iteratively until ks_{ij}'s are positives. In this case, the maximum number of iterations for $ks_{ij}>0$ is n^2. First, time complexity for exchanging node i and j with the maximum positive ks_{ij} is $O(n^2)$. Since it is enough to apply the MST algorithm to two changed node sets and time complexity of MST is $O(E \log E)$ [19] (E is the number of edges, which takes on values MAX and $1/2MAX(MAX+1)$ for sparse and complete graphs, respectively), time complexity for finding MST trees is $O(n^2)$. MDLCAA is applied to two changed trees, so the maximum number of links to be computed in MDLCAA is $2MAX$. In the worst case, MDLCAA can be repeated up to the number of ks_{ij}'s. So, the time complexity is $O(n^2)$. After all, time complexity of step 3 in NEA is maximum $[O(n^2), O(n^2), O(n^2)] = O(n^2)$. From the above results, time complexity for NEA is maximum [complexity of step 1, complexity of step 2, complexity of step 3] = maximum $[O(n), O(n^2), O(n^2)] = O(n^2)$. Similarly, time complexity for NSA is $O(n^2)$. Therefore, total execution time of the the Least-Cost DC-CMST Heuristic = EW solution's execution time + NEA's execution time + NSA's execution time = maximum $[O(n^2 \log n), O(n^2), O(n^2)] = O(n^2 log\ n)$.

4. Performance evaluation

In this section, we present the performance evaluation of the proposed heuristic in terms of execution time and total cost. The performance evaluation was carried out using several networks defined according to the location of

the source node. The co-ordinates of nodes were randomly generated in a square grid of dimension 100×100. Assuming the co-ordinates of the two nodes as (x_1,y_1) and (x_2,y_2), the distance between the two nodes is given by Eq. (15).

$$d_{ij} = \left\lfloor \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + 0.5 \right\rfloor \quad (15)$$

Based on the location of the source node, the following two types of tests were considered:

TC test: Test where the source node is at the center, i.e. (50,50)
TE test: Test where the source node is at the upper left edge i.e. (0,0)

The savings rate (SR) is defined by Eq. (16), where A is the cost obtained by applying MDLCAA to the topology of the EW solution, and B is the cost of the Least-Cost DC-CMST Heuristic presented in this paper.

$$SR = (A - B)/A \times 100 \quad (16)$$

In order to compare running time, we define CPU time Ratio (CR) as in Eq. (17) where $T(EW)$ is the CPU time of the EW solution, and $T(DC\text{-}CMST)$ is the CPU time inclusive of the execution time of NEA, NSA, and second NEA in the Least-Cost DC-CMST Heuristic. The reason for adding one is to account for the execution time of the EW solution.

$$\text{CR} = T(\text{DC-CMST})/T(\text{EW}) + 1 \quad (17)$$

4.1. Computational experience

We assumed that traffic at a node has Poisson distribution Letting X be an exponential random number and v as the average traffic rate of network, X can then be expressed as:

$$X = \frac{1}{v}\ln\left(\frac{1}{1-U}\right) \quad (18)$$

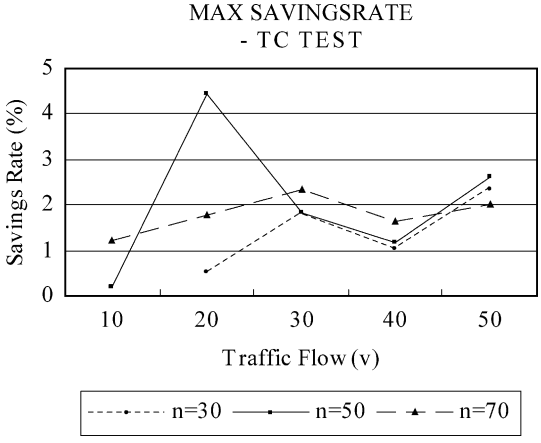

Fig. 2. Maximum savings rate (SR) for TC test.

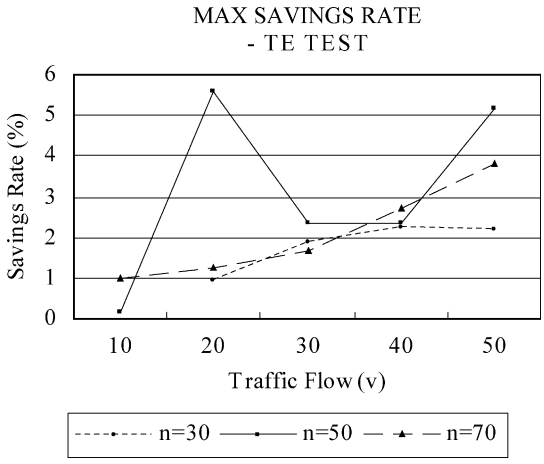

Fig. 3. Maximum savings rate (SR) for TE test.

In Eq. (18), U is a random variable of uniform distribution within interval [0,1]. Using Eq. (18), n Poisson random numbers were generated and used as traffic requirements at nodes. $v = 10$, 20, 30, 40, 50 and $n = 30$, 50, 70 were used. Maximum traffic handled in a tree (MAX) was used from the maximum value among the Poisson random numbers generated by the simulation to $n/4$. TC and TE tests were executed for each case. The maximum savings rate for TC and TE tests in 10 runs, using Eqs. (16) and (17), are shown in Figs. 2 and 3, respectively.

In Figs. 2 and 3, the maximum savings rates (SR) are 4.5 and 5.5% for TC and TE tests, respectively.

The CPU time ratio (CR) for TC and TE tests in 10 runs using Eq. (17) are represented in Figs. 4 and 5, respectively. From Figs. 4 and 5, CPU time ratio is 1.4–2.54 and 1.2–2.03 for TC and TE tests, respectively. Thus, the Least-Cost DC-CMST Heuristic is more efficient for TE test than TC test in both the savings rate and the CPU time ratio.

4.2. Analysis of results

As can be shown from Figs. 2–5, savings rate and CPU time ratio of the Least-Cost DC-CMST Heuristic for TE test

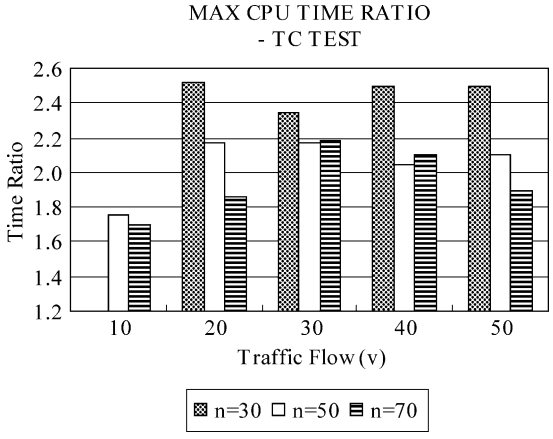

Fig. 4. Maximum CPU time ratio (CR) for TC test.

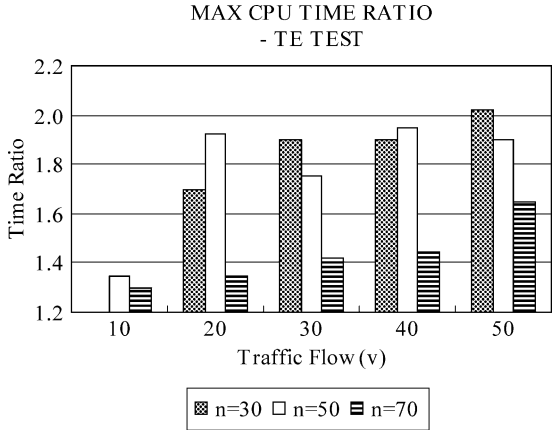

Fig. 5. Maximum CPU time ratio (CR) for TE test.

are better than those for TC test. Increasing *MAX* and *n* have no effect on the savings rate. From the above results, we can state that it is difficult to derive the mathematical expression describing the relationship between the solution and *n*, and solution and *MAX*. In addition, solution of the Least-Cost DC-CMST Heuristic is affected by locations of the nodes.

Table 1 depicts the comparison of four different algorithms, including the Least-Cost DC-CMST Heuristic. It should be noted that the algorithms in [5,8,20] deal only with the CMST problem which does not consider the mean delay time constraint. In order to compare the performance of the four algorithms, we modified the algorithms in [5,8,20] by applying MDLCAA (used in the Least-Cost DC-CMST Heuristic) so that they can satisfy the mean delay constraint required for the solution of the DC-CMST problem. In the table, modified algorithm [20] can be applied only when the locations of nodes are given by the coordinates of plane, and every traffic requirement is one. Modified algorithm [8] represents the results when every traffic requirement is one. If the traffic requirements are different or *MAX* is not a power of two, the results are inferior to that of the EW solution. For a complete graph, since the time complexity of modified algorithm [8] is $O(MAXn^3)$, the computing time increases more sharply than the Least-Cost DC-CMST Heuristic with a time complexity of $O(n^2 \log n)$.

It is known that though the Least-Cost DC-CMST Heuristic can not guarantee an optimal solution, it does not limit the type of traffic requirements, has a relatively small time complexity, and considers time delay constraint affecting quality of service of end users. Hence, the Least-Cost DC-CMST Heuristic is more suitable for practical environment than other algorithms.

Table 1
Comparison of algorithms

Algorithm	Memory complexity	Time complexity	Savings rate (SR)	CPU time ratio (CR)
Modified algorithm [5]	$O(n^2)$	$O(n^2 \log n)$		
Modified algorithm [20]	$O(n)$	$O(n \log n)$	Worse	Same
Modified algorithm [8]	$O(n^2)$	$O(n^3)$	1–5	3–4
Least-Cost DC-CMST Heuristic	$O(n^2)$	$O(n^2 \log n)$	1–5	1.5–2

5. Conclusions

In this paper, we formulated the DC-CMST problem and presented a heuristic solution for the problem, which has not been reported in any previous work. The proposed Least-Cost DC-CMST Heuristic minimizes the total cost to discover the capacitated spanning tree topology with additional mean network delay constraint. This heuristic improves previous solutions through node exchange and shift between trees based on the trade-off criteria suggested in this paper. In addition, it allocates link capacities optimally within a given mean delay. Several properties about the performance of our algorithm were derived, and through experiments, it was shown that the proposed algorithm produces better solutions than those of the modified previous CMST algorithms to solve the DC-CMST problem.

Our proposed algorithm can be used by network designers for the topological design of local networks with a large number of nodes. It can also be used to discover the broadcast and multicast trees for real-time multimedia traffic. Future extension of this work consists of more efficient heuristic algorithms in terms of memory, time complexity, and cost.

References

[1] I. Astic, O. Festor, A hierarchical topology discovery service for IPv6 networks, IEEE/IFIP Network Operations and Management Symposium 2002; 497–510.
[2] Y. Bejerano, M. Breitbart, R. Rastogi, Physical topology discovery for large multi subnet networks, INFOCOM 2003; 342–352.
[3] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, A. Silberschatz, Topology discovery in heterogeneous IP networks, INFOCOM 2000; 265–274.
[4] K.M. Chandy, T. Lo, The capacitated minimum spanning tree, networks, Networks 3 (1973) 173–181.
[5] L.R. Esau, K.C. Williams, On teleprocessing system design, part II, IBM System Journal 5 (3) (1996) 142–147.
[6] B. Gavish, Augmented based algorithm for centralized network design, IEEE Transactions on Communication 33 (12) (1985) 1247–1257.
[7] B. Gavish, Topological design of centralized computer networks formulations and algorithms, Networks 12 (1982) 355–357.
[8] B. Gavish, K. Altinkemer, Parallel savings heuristic for the topological design of local access tree networks, INFOCOM 1986; 130–139.
[9] B. Huffaker, D. Plummer, D. Moore, K. Claffy, Topology discovery by active probing, Applications and the Internet (SAINT) Workshops 2002; 90–96.

[10] A. Karaman, H. Hassanein, DCMC—delay-constrained multipoint communication with multiple sources, IEEE International Symposium on Computers and Communication 2003;.
[11] A. Kershenbaum, R.R. Boorstyn, Centralized teleprocessing network design, Networks 13 (1983) 279–293.
[12] H.C. Lin, H.L. Lai, S.C. Lai, Automatic link layer topology discovery of IP networks, ICC '99, IEEE International Conference on Communications 1999; 1034–1038.
[13] H.C. Lin, Y. Wang, C.H. Wang, C.L. Chen, Web-based distributed topology discovery of IP Networks, 15th International Conference on Information Networking 2001; 857–862.
[14] H.C. Lin, S.C. Lai, P.W. Chen, An algorithm for automatic topology discovery of IP networks, 1998 IEEE International Conference on Communication 1998; 1192–1196.
[15] C.H. Papadimitriou, The complexity of the capacitated tree problem, Networks 8 (1978) 217–230.
[16] D.S. Reeves, H.F. Salama, A distributed algorithm for delay-constrained unicast routing, IEEE/ACM Transaction on Networking 8 (2) (2000) 239–250.
[17] H.F. Salama, D.S. Reeves, Y. Viniotis, The delay-constrained minimum spanning tree problem, Second IEEE Symposium on Computers and Communications 1997; 699–703.
[18] H.F. Salama, D.S. Reeves, Y. Viniotis, Evaluation of multicast routing algorithms for real-time communication on high-speed networks, IEEE Journal on Selected Areas in Communications 15 (3) (1997) 332–345.
[19] R. Sedgewick, Algorithms, Addison-Wesley, Reading, MA, 1989. pp. 452–461.
[20] R. Sharma, Design of an economic multi-drop network topology with capacity constraints, IEEE Transactions on Communication 31 (4) (1983) 590–591.
[21] Y. Zheng, S. Akhtar, Networks for Computer Scientists and Engineers, Oxford University Press, NY, USA, 2000. pp. 364–372.
[22] W. Zhengying, S. Bingxin, Z. Ling, A delay-constrained least-cost multicast routing heuristic for dynamic multicast groups, Electronic Commerce Research 2 (2002) 323–335.
[23] Q. Zhu, M. Parsa, J. Garcia-Luna-Aceves, A source-based algorithm for delay-constrained minimum-cost multicasting, INFOCOM '95 1995; 377–385.