Exact algorithm for delay-constrained capacitated minimum spanning tree network

Y.J. Lee and M. Atiquzzaman

Abstract: The delay-constrained capacitated minimum spanning tree (DC-CMST) problem of finding several broadcast trees from a source node is discussed. While the traditional CMST problem deals with only the traffic capacity constraint served by a port of the source node, and delay-constrained minimum spanning tree (DCMST) considers only the maximum end–end delay constraint, the DC-CMST problem deals with both the mean network delay and traffic capacity constraints. The DC-CMST problem consists of finding a set of minimum cost spanning trees to link end-nodes to a source node satisfying the traffic requirements at end-nodes and the required mean delay of the network. In the DC-CMST problem, the objective function is to minimise the total link cost. A dynamic programming-based three-phase algorithm that solves the DC-CMST problem is proposed. In the first phase, the algorithm generates feasible solutions to satisfy the traffic capacity constraint. It finds the CMSTs in the second phase, and allocates the optimal link capacities to satisfy the mean delay constraint in the third phase. Performance evaluation shows that the proposed algorithm has good efficiency for any network with less than 30 nodes and light traffic. The proposed algorithm can be applied to any network regardless of its configuration, and used for the topological design of local networks and for efficient routing algorithms capable of constructing least cost broadcast trees.

1 Introduction

Network topology is defined as the configuration of connecting computers in a network. The topology affects important factors such as the communication cost and transmission speed of a network. Thus, topology discovery is an important area of research in computer networks.

Modern computer networks consist of ‘backbone’ networks that serve as the major highways to transfer large volumes of communication traffic, and ‘local networks’ that feed traffic between the backbone node and end-user nodes. Several researchers have proposed algorithms for discovering the topology of the Internet backbone [1–5]. A local area network (LAN) connected to a backbone node can be regarded as an end-user node. A local network therefore consists of a backbone node and several end-user nodes hanging off a LAN.

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. Therefore the local network consists of a backbone node (source) and several trees or rings that cover all end-user nodes to satisfy the constraints on traffic volume. Topology discovery is required to find all the trees and rings in a local network that satisfies the constraints. Issues related to topology discovery for local network has been classified into two problems in the literature: capacitated minimum spanning tree (CMST) [6, 7] and delay-constrained minimum spanning tree (DCMST) [8] problems that have been solved using heuristic algorithms. The CMST problem is known to be NP complete, and consists of finding a set of minimum spanning trees rooted at the source node satisfying the traffic constraint. The DCMST problem finds the least cost broadcast trees rooted at the source node satisfying the given maximum end-to-end delay constraint. Details of the two problems and related works are given in Section 2. To summarise, the CMST and the DCMST problems consider the traffic capacity and the end-to-end delay constraint, respectively. However, communication delay between end nodes deteriorates quality of service (QoS) of users as the network size grows. In real situation, the traffic capacity or the number of ports in the source node (e.g. router or switching hub) is limited. If the solution wants to satisfy the end-to-end delay requirement, the depth of trees tends to short and the number of trees is larger than the number of ports that can be served by the source node. This means that some trees cannot be served because of the limited number of ports in the source node despite satisfying the end-to-end delay constraint. The use of mean network delay instead of end-to-end delay leads to more efficient utilisation of network resources such as the limited router ports.

To solve the above issue, it is necessary to consider the traffic capacity and the mean network delay constraints in a common framework. We therefore present the Delay-Constrained CMST (DC-CMST) problem. The authors are not aware of any previous work on DC-CMST. The objective of this paper is to formulate the DC-CMST problem and to develop an exact algorithm to solve the problem. We propose a dynamic programming based exact algorithm to solve the DC-CMST problem. To investigate the feasibility of the algorithm for real

doi:10.1049/iet-com:20060544

Paper first received 31st August 2006 and in revised form 19th July 2007

Y.J. Lee is with the Department of Technology Education, Korea National University of Education, Darak-Ri, Gangnae-Myon, Chungwon, Chungbuk 363-791, Korea

M. Atiquzzaman is with School of Computer Science, University of Oklahoma, 200 Felgar Street, Norman OK 73019, USA

E-mail: lyj@knue.ac.kr

networks, we carry out a performance evaluation. Our proposed algorithm solves the DC-CMST problem in three phases: First, it uses dynamic programming to generate feasible solutions to satisfy the traffic capacity constraint. Secondly, it finds CMSTs based on the matching procedure. Thirdly, it assigns link capacities to the set of capacitated spanning trees optimally in order to satisfy the desired mean network delay constraint. Computational complexity shows that our algorithm can be effectively applied to problems where the number of nodes is less than 30.

Our performance evaluation results demonstrate that the time complexity of our proposed exact algorithm is large when the number of nodes in the local network is large. We therefore suggest that our algorithm can be used for small local networks (less than 30 nodes), whereas heuristic methods can be used for large local networks (however, no heuristic method has yet been reported). The main contributions of this paper are: (i) formulate the DC-CMST problem, (ii) propose and evaluate its exact solution and (iii) depending on the size of the local network, determine the threshold in choosing between heuristic methods and the exact algorithm.

The rest of the paper is organised as follows: Section 2 describes the previous related work in this area. Section 3 presents the mathematical formulation of the DC-CMST problem. Section 4 describes our proposed dynamic programming based modelling and exact algorithm for the DC-CMST problem. Section 5 evaluates the computational complexity of our proposed algorithm using an analytical model. Finally, concluding remarks are given in Section 6.

2 Previous related research

To aid in better understanding the DC-CMST problem formulation in Section 3, we first familiarise the readers with the formulation of CMST and DCMST problems and related work in the literature.

2.1 CMST problem

The CMST problem (Fig. 1) is concerned with finding a set of minimal spanning trees, which are used to form the broadcast trees for real time traffic, such as multimedia. The end-user nodes are linked together by a tree connected to a port in the backbone node, where the links connecting the end-user nodes have finite capacity, that is, they can handle limited amount of traffic. This translates to restricting the number of end-user nodes served by a single tree. The solution of the CMST problem results in a collection of trees that serve all end-user nodes with a minimal connection cost.

We now consider the modelling of the CMST problem for the topological design of a tree network. Assume that there is graph, $G = (V, A)$, where $V = \{0, 1, 2, \cdots, n\}$, traffic requirement at node is τ_i $(i \in V - \{0\})$ and link cost between node i and node j is d_{ij} $((i, j) \in A)$. ε represents the maximum traffic served by single tree. Index 0 represents the source node. The problem formulation is described by (1).

$$\begin{aligned} &\text{Minimise} \sum_{i,j} d_{ij}x_{ij} \\ &\text{S.T.} \\ &\sum_{i\in \text{tree}(\alpha)} \tau_i x_{ij} \le \varepsilon \\ &\sum_{i,j} x_{ij} = n \\ &x_{ij} = 0 \text{ or } 1 \end{aligned} \tag{1}$$

The objective of the CMST problem is to find a collection of least-cost spanning trees rooted at node 0. tree(α) ($\alpha = 1, 2, \ldots,$ lcnt) is the spanning tree whose one of nodes is connected the source node. 'lcnt' is the number of disjoint trees covering n nodes. x_{ij} represents link between node i and j $(i, j$: $i = 1, 2, \ldots, n; j = 1, 2, \ldots, n)$. If link (i, j) is included in any tree (tree(α)) of the solution, then x_{ij} is set to 1. Otherwise, it is set to 0. Total traffic volume per tree $(\sum_{i\in \text{tree}(\alpha)} \tau_i x_{ij})$ is less than equal to the given traffic constraint (ε). A particular case occurs when each τ_i is equal to one. In that case, the constraint means that no more than ε nodes can belong to any tree of the solution. With this constraint, we can confine the fault to the tree only in which it occurred. Since the number of links in the CMST problem should be equal to the number of nodes, the sum of x_{ij} $(\sum_{i,j} x_{ij})$ is equal to n.

2.2 DCMST and similar problems

We now discuss the DCMST problem (Fig. 2) that has an upper bound on end-to-end delay to satisfy QoS requirements, important criteria in real time applications, and finds least cost spanning trees. In the problem, the maximum end-to-end delay of a spanning tree should be less than or equal to a given delay constraint.

Formulation of the DCMST problem is given by

$$\begin{aligned} &\text{Cost}(T(s)) \le B \\ &\text{Max\_Delay}(T(s)) \le \delta \end{aligned} \tag{2}$$

$T(s)$ and Cost($T(s)$) represent the spanning tress rooted at the source node (s) and the minimum total cost among all possible spanning trees, respectively. B shows the upper bound cost. Max_Delay($T(s)$) and δ represent the maximum end-to-end delay for $T(s)$ and a given delay constraint, respectively [8]. As shown in (2), the DCMST

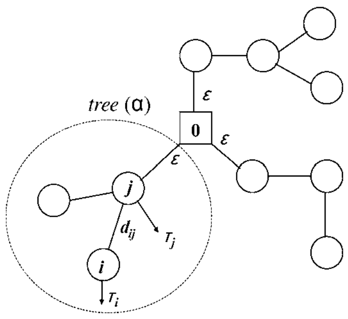

Fig. 1 CMST problem

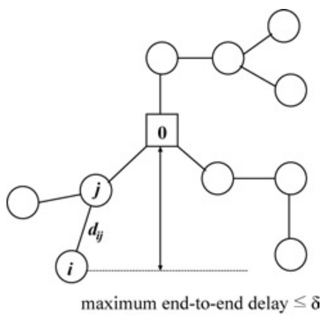

Fig. 2 DCMST problem

problem finds the set of minimum spanning tress satisfying the maximum end-to-end delay constraint. These approaches however do not always lead to efficient network resource utilisation, since the use of least-delay paths does not necessarily optimise sharing of the network links. Minimising the overall cost of the multicast tree is a key issue in the deployment of multimedia applications in distributed environments [9].

Several other research efforts aimed at finding a low-cost broadcast tree (spanning tree) and multicast tree [10, 11] that satisfy the constraints imposed by QoS requirements in a centralised environment with a single source. Reeves and Salama [12] deal with the delay-constrained unicasting problem. Karaman and Hassanein [13] and Zhengying *et al.* [14] have proposed an algorithm for multiple source delay-constrained multipoint communication problem and an algorithm for the delay-constrained multicast routing for dynamic multicast groups, respectively. Kun *et al.* [15] concentrated on finding multicast trees that meet end-to-end delay and delay variation constraints. Tseng *et al.* [16] proposed a genetic algorithm for solving the problem of constructing a degree and DMST for multimedia broadcasting on overlay networks. In such a communication scheme, QoS requirement limits the transmission time and the number of receivers to which each node can transmit. Li *et al.* [9] developed a heuristic to construct multicast trees that minimise the cost, while bounding the end-to-end delay from the source to every multicast nodes.

Most of the above works are based on heuristic methods, and have not considered the traffic capacity constraint that each port of the backbone node can handle only limited amount of traffic in real applications. In addition, the above problems need the network configuration such as link connectivity, link cost and the desired delay. On the other hand, the proposed DC-CMST problem is to find the network configuration composed of several trees to satisfy the mean network delay constraint in order to share the network resource among end nodes. Since the definition of our proposed DC-CMST problem is different from those of DCMST and other similar problems, solutions to DCMST and similar problems cannot be applied to the DC-CMST problem.

3 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. Indexes of nodes are numbered as $\{0, 1, 2, \ldots, n\}$. 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 $(1, n)$ satisfying the delay constraint. End-user nodes can be switching hub or host and originates traffic.

While the CMST and the DCMST problems deal with traffic capacity and maximum end-to-end delay constraints, respectively, the DC-CMST problem deals with both the traffic capacity and the network mean delay constraints simultaneously.

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

The DC-CMST problem is represented in Fig. 3. In the rest of this section, we formulate the DC-CMST problem.

3.1 Notations

The following notations will be used to formulate the DC-CMST problem:

n: the number of nodes excluding the source node
0: the index of source node
N': set of node indexes excluding 0
$N' = \{1, 2, \ldots, n\}$
N_j: set of node indexes excluding j
$N_j = N' - \{j\} = \{1, 2, \ldots, j-1, j+1, \ldots, n\}$
d_{ij}: distance (cost) from node i to node j
$\boldsymbol{D}$: distance (cost) matrix
$\boldsymbol{D}'$: shortest connected distance matrix
ρ: unit cost of link capacity
d_k: traversal distance cost of link k in topology obtained by the second phase of DC-CMST algorithm $(k = 1, \ldots, n)$
S: subset of N_j composed of k nodes (cardinality of $S = k$)
ε: maximum traffic served by single tree
τ_i: traffic amount generated at each node i $(i = 1, 2, \ldots, n)$
$\psi_k(j, S)$: the minimum distance (cost) for a tree that covers set S composed of k nodes from the source node to node j to satisfy the constraint ε (k represents the number of nodes included in the set S)
$P_{\rm m}$: set, $\{j\} \cup$ S which is composed of the same elements ignoring the order
$\psi_k'(P_{\rm m})$: the minimum cost among $\psi_k(j, S)$ corresponding to the set, $P_{\rm m}$
$R_{\rm m}$: optimal node sets corresponding to $\psi_k'(P_{\rm m})$ considering the order
tree(α): set of trees connecting of end-node in $R_{\rm m}$ to the source node $(\alpha = 1, 2, \ldots, l)$
$\psi_{{\rm tree}(\alpha)}$: total link cost corresponding to tree(α)
R: set of optimal subtrees
Φ: mean delay time of network (s)
Φ_k: mean delay time of link k (s)
ν: total traffic between source-destination pairs
$1/\mu$: average packet length (bits/packet)
Θ_k: the capacity of link k obtained by the second phase of DC-CMST algorithm $(k = 1, \ldots, n)$ (bits/s)
λ_k: traffic flow on link k (packets/s)
δ: desired mean delay time of network (s)
$G_{\rm cost}$: total link cost in the third phase of DC-CMST algorithm
F: optimal cost

3.2 Assumptions

We make the following assumptions to formulate the DC-CMST problem.

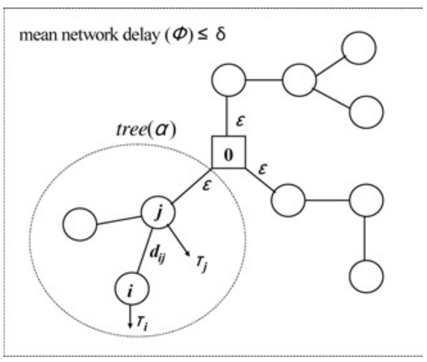

Fig. 3 *DC-CMST problem*

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 subtree (Max $\{\tau_i\} \leq \varepsilon$, $1 \leq i \leq n$).
3. The traffic at a node is not splintered.
4. The total traffic exceeds the maximum traffic served by a tree ($\sum_{i=1}^{n} \tau_i > \varepsilon$).
5. An arrival of packet is based on a Poisson distribution.
6. Service time of packets is exponentially distributed.

3.3 Problem formulation

We formulate the DC-CMST problem in order to satisfy the mean delay constraint (total network mean delay time is less than δ) as well as the traffic capacity limitation imposed by the existing CMST problem. The mean delay time (Φ) of the network is given by (3) [17]. The reason that the number of links is n in (3) is because the solution of the DC-CMST problem is a set of spanning trees that has n links when the number of nodes is n.

$$\Phi = \frac{1}{v}\sum_{k=1}^{n} \lambda_k \Phi_k \tag{3}$$

If each link k can be regarded as an independent M/M/1 queue, mean delay time (Φ_k) at link k, from queuing theory, is given by

$$\Phi_k = \frac{1}{(\mu\Theta_k - \lambda_k)} \tag{4}$$

Substituting (4) into (3) gives the network mean delay

$$\Phi = \frac{1}{v}\sum_{k=1}^{n} \lambda_k \left[\frac{1}{(\mu\Theta_k - \lambda_k)}\right] \tag{5}$$

The total link cost is assumed to be a linear function as given by

$$G_{\text{cost}} = \sum_{k=1}^{n} \rho\Theta_k d_k \tag{6}$$

Having defined the network mean delay (5) and the objective function (6), the DC-CMST problem can be formulated by (7)–(11).

$$\text{Minimise} \quad G_{\text{cost}} = \sum_{k=1}^{n} \rho\Theta_k d_k \tag{7}$$

S.T.

$$\frac{\lambda_k}{\mu} \leq \Theta_k \tag{8}$$

$$\Phi = \frac{1}{v}\sum_{k=1}^{n} \lambda_k \left[\frac{1}{(\mu\Theta_k - \lambda_k)}\right] \leq \delta \tag{9}$$

$$\sum_{i\in \text{tree}(\alpha)} \tau_i x_{ij} \leq \varepsilon \tag{10}$$

$$\sum_{i,j} x_{ij} = n \quad x_{ij} = 0 \text{ or } 1 \tag{11}$$

The objective function of the DC-CMST problem is to find a collection of trees with minimal link cost (7) subject to the following three constraints: (i) average traffic flow on the link should be smaller than the capacity of the link (8), (ii) mean delay time of network has to be dropped within allowable mean delay time (δ) (9) and (iii) maximum traffic flow on one tree must be below ε (10). If the number of nodes that a tree takes charge of is smaller than a specific value, we change τ_i to 1, $\forall i$ in (10). x_{ij} represents link between node i and j (i, j: $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, n$). If link (i, j) is included in any tree (tree(α)) of the solution, then x_{ij} is set to 1. Equation (11) represents that n nodes have to be included in the solution.

4 Solution for the DC-CMST problem

Having formulated the DC-CMST problem in the previous section, we now develop solution for the problem using dynamic programming. Our solution consists of three phases: subtree generation, matching and link capacity allocation.

4.1 Subtree generation phase

To solve the DC-CMST problem using dynamic programming, we define stage variables and state variables.

- Stage variable, k ($k = 1, 2\ldots$), is the number of nodes to form a subtree rooted at the source node to any arbitrary node j.
- State variables j and S are the node index to be connected and the set of node indexes included in the subtree in order to connect node j, respectively. Then, using the principle of optimality, the recurrence relation can be obtained as shown in (12).

In (12), $\psi_k(j, S)$ represents the least cost to connect node j with the subtrees of which the number of nodes included in S is k to connect node j. $\psi_k(j, S)$ is set to infinity when the sum of all the traffic exceeds ε.

To obtain a feasible solution, we compute $\psi_1(j, S)$ for all (j, S) satisfying $\sum_{q\in S}\tau q + \tau j \leq \varepsilon$ by using ψ_0. Then, $\psi_2(j, S)$ is computed using ψ_1. By repeating this procedure, we reach the phase where for all (j, S), $\forall(j, S)$ $\sum_{q\in S}\tau q + \tau j > \varepsilon$.

$$\begin{aligned}
&\text{If} \sum_{q\in S} \tau_q + \tau_j \leq \varepsilon \\
&\psi_k(j, S) = \underset{q\in S, q\neq j}{\text{Min}} [\psi_{k-1}(q, S - \{q\}) + d_{qj}], \quad k = 1 \\
&\psi_k(j, S) = \underset{q\in S, q\neq j}{\text{Min}} [\psi_{k-1}(q, S - \{q\}) + d_{qj}, \\
&\qquad \sum_{q\in S, q\neq j} \psi_1(q, \{j\}) - (k-1)d_{j0}], \quad k \geq 2 \quad (S \subseteq N_j) \\
&\text{else} \\
&\psi_k(j, S) = \infty
\end{aligned} \tag{12}$$

Since boundary condition represents the cost to connect directly from the source node to node j without intermediate nodes, binary condition is defined by (13).

$$\psi_0(j, -) = d_{0j} \tag{13}$$

Since $\psi_k(j, S)$ is infinity for all (j, S) at such a phase, we set $L = k$. This means that the tree cannot be extended any further. So, subtrees obtained at the previous stage k $(0, 1, 2, \ldots, L-1)$ are feasible solutions. In (12),

$\sum_{q\in S, q\neq j}\psi_1(q, \{j\}) - (k-1)d_{j0}$ represents the case where the tree branches off. For example, assume that there are five nodes including source node (index 0) and all the links between nodes are connected. Now, we connect node 2 and source node. For $k = 3$, (2) can be expressed as: $\psi_3(2, \{1, 3, 4\}) = \mathrm{Min}[\psi_2(1, \{3, 4\}) + d_{12}, \psi_2(3, \{1, 4\}) + d_{32}, \psi_2(4, \{1, 3\}) + d_{42}, \psi_1(1, \{2\}) + \psi_1(3, \{2\}) + \psi_1(4, \{2\}) - 2d_{02}]$ and corresponding feasible connections are depicted in Fig. 4. Since the optimal connection of the circle was already obtained in the previous stage ($k = 2$), it is sufficient to consider only S to connect node 2 (Fig. 4*a*–*c*). Now, we consider the situation where the tree branches off. Since $\psi_1(1, \{2\}) + \psi_1(3, \{2\}) + \psi_1(4, \{2\}) = \psi_0(2, -) + d_{21} + \psi_0(2, -) + d_{23} + \psi_0(2, -) + d_{24} = 3\psi_0(2, -) + d_{21} + d_{23} + d_{24} = 3d_{02} + d_{21} + d_{23} + d_{24}$, we have included d_{02} three times (Fig. 4*d*). Hence, the cost of Fig. 4*d* becomes $\psi_1(1, \{2\}) + \psi_1(3, \{2\}) + \psi_1(4, \{2\}) - 2d_{02}$. From the above inferences, in the kth stage, the branch cost to node j becomes $\sum_{q\in S, q\neq j}\psi_1(q, \{j\}) - (k-1)d_{j0}$.

4.2 Matching phase

At stage k of the subtree generation phase, $\psi_k(j, S)$ represents the cost of the tree that is composed of the same elements as $j \cup S$, but the order of elements included in the set is different.

Among $\psi_k(j, S)$, the minimum cost $\psi'_k(P_\mathrm{m})$ is computed. That is, the cost for P_m at stage k, $\psi'_k(P_\mathrm{m})$ is as follows.

$$\begin{aligned}&\psi'_0(P_\mathrm{m}) = \psi_0(j, -), \quad k = 0\\ &\forall(j, S) \text{ such that } P_\mathrm{m} - \{\{j\} \cap S\} = \varnothing \qquad (14)\\ &\psi'_k(P_\mathrm{m}) = \mathrm{Min}[\psi_k(j, S)], \quad k = 1, 2, \ldots, L-1\end{aligned}$$

The value of $\psi'_k(P_\mathrm{m})$ from (14) represents the cost of the tree, which is composed of the same node indexes of different order. Node set of the optimal policy R_m corresponds to $\psi'_k(P_\mathrm{m})$ R_m is the set of node indexes included in the tree rooted from the source node. Of course, node 0 is not included in R_m.

We find the set of trees containing the source node by adding the source node index (0) to the end-side indexes of R_m as (15).

$$\begin{aligned}\mathrm{tree}(\alpha) = \{0 - R_\mathrm{m}\}, \forall R_\mathrm{m} \text{ such that } \cup R_\mathrm{m}\\ = N' \text{ and } R_i \cap R_j = \varnothing (i \neq j)\end{aligned} \qquad (15)$$

4.3 Link capacity allocation phase

Now, we compute $\psi_{\mathrm{tree}(\alpha)}$ that represents cost for each tree(α) ($\alpha = 1, 2, \ldots, l$) after allocating link capacities optimally and computing the total cost($\sum \rho d_k \Theta_k$). In order to obtain the optimal link capacity for each link k, we use fixed routing to, find the mean arrival rate (λ_k: $k = 1, 2, \ldots, m$) of links on the capacitated tree (tree(α): $\alpha = 1, 2, \ldots, l$). The reason to use fixed routing is that, in order to transfer the traffic to the backbone node, each node in the trees has to send it to the neighbour node on the tree. The formulation for satisfying the mean network delay constraints is represented as

$$\text{Minimise} \quad G_{\mathrm{cost}} = \sum_{k=1}^{n} \rho\Theta_k\, d_k$$

$$\text{S.T.} \qquad (16)$$

$$\frac{1}{v}\sum_{k=1}^{n}\frac{\lambda_k}{\mu\Theta_k - \lambda_k} = \delta$$

The optimal solution of (16) is obtained using the Lagrange multiplier as shown in (17).

$$\Theta_k = \frac{\lambda_k}{\mu}\left[1 + \frac{1}{v\times\delta}\sum_{j=1}^{m}\frac{\sqrt{\rho\lambda_j d_j}}{\sqrt{\rho\lambda_k d_k}}\right] \qquad (17)$$

By adding the $\rho d_k \Theta_k$ for each tree(α), we can find the $\Psi_{\mathrm{tree}(\alpha)}$.

$$\psi_{\mathrm{tree}(\alpha)} = \sum_{k=1}^{m}\rho d_k \Theta_k \qquad (18)$$

where k is the index of each link included in the tree(α).

The global optimal value F is the least value among $\Psi_{\mathrm{tree}(\alpha)}$ ($\alpha = 1, 2, \ldots, l$).

$$F = \underset{\alpha=1,2\ldots,l}{\mathrm{Min}}\left[\psi_{\mathrm{tree}(\alpha)}\right] \qquad (19)$$

4.4 Optimal DC-CMST algorithm

Fig. 5 shows the optimal DC-CMST algorithm based on above model.

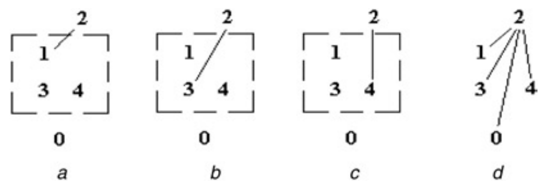

Fig. 4 *Feasible connections for* $\psi_3(2, \{1, 3, 4\})$

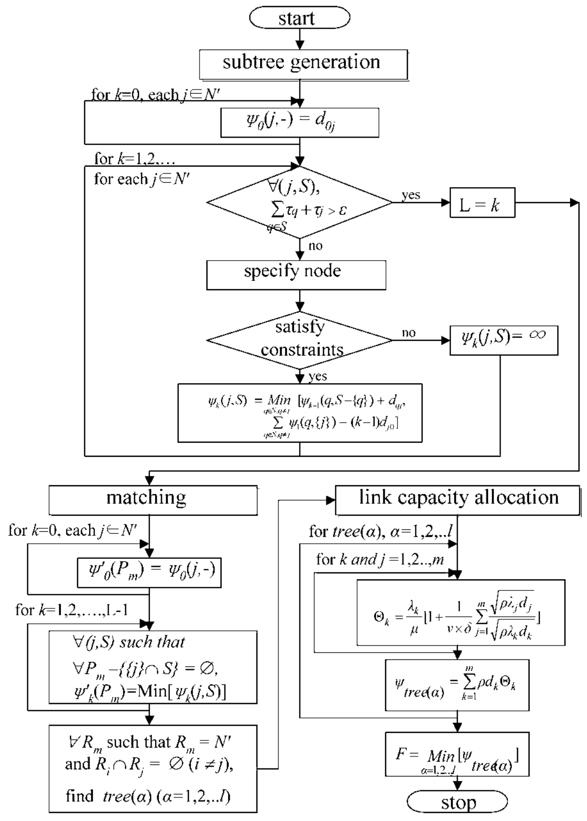

Fig. 5 *Optimal DC-CMST algorithm*

Table 1: Distance matrix

	0	1	2	3	4
0	0	3	3	5	10
1		0	6	4	8
2			0	3	5
3				0	7
4					0

5 Performance evaluation

Having formulated the DC-CMST problem and its solution in Sections 3 and 4, in this section, we evaluate the performance of the algorithm in terms of its computational complexity.

5.1 Numerical example

To facilitate better understanding our DC-CMST algorithm, we provide below a simple example. We consider the problem in which maximum traffic (ε) per tree is three, traffic requirement at each node is one ($\forall i,\ \tau_i = 1$), and mean network delay constraint (δ) of the network is 1 ms. Also, assume that mean packet length ($1/\mu$) is 1000 bits/packet, with ρ set to 1. Index of the source node is 0. Distance matrix (D) is symmetric as shown in Table 1.

The steps for computing the optimal DC-CMST algorithm for subtree generation (phase 1) is shown in Table 2, and for matching (phase 2) is shown in Table 3. The optimal policy in Table 2 shows the node index with the least cost in the previous stage to connect node j. The numbers in ( ) in Tables 2 and 3 represent branched nodes, while those in { } in Table 3 represent R_m.

Table 4 shows the result after allocating link capacities to the capacitated minimum spanning trees obtained in phase 2. Optimal costs for two trees {0-1} and {0-2-(3-4)} are 3.0 and 11.0, respectively. Capacities and delay times for each link to meet the total network mean delay ($\delta = 1.0$ ms) are also presented.

5.2 Performance of optimal DC-CMST algorithm

We present the following Lemmas in order to show the properties of the proposed algorithm.

Lemma 1: Optimal DC-CMST algorithm produces the optimal solution.

Proof: In the subtree generation phase, we enumerate the feasible subtrees using the optimality principle of dynamic programming. So, there can be no other feasible subtrees except our solutions. In the matching and link capacity allocation phase, we first find partitions of which unions compose of the node set, $N' = \{1, 2, \ldots, n\}$. Next, we generate trees composed of the above partitions. These trees are found by adding the index of the source node to indexes of both end nodes included in the partition. They are candidates for optimal solutions. Then, we find all the costs of tress by allocating the optimal link capacities satisfying the mean delay constraint (δ). These costs are sub-optimal solutions because they represent the optimal cost for the specific corresponding tress. Finally, since we select the least cost solution among sub-optimal solutions, it is natural for that solution to be the global optimal solution.□

Lemma 2: Time complexity of DC-CMST algorithm is $O(kV^2)$, $1 \le k \le 2^L$

Table 2: Phase 1: subtree generation

step		$\psi_k(j, S)$	Optimal policy
1	$k = 0$	$\psi_0(1, -) = d_{01} = 3$	0
		$\psi_0(2, -) = d_{02} = 3$	0
		$\psi_0(3, -) = d_{03} = 5$	0
		$\psi_0(4, -) = d_{04} = 10$	0
2	$k = 1$	$\psi_1(1, \{2\}) = \psi_0(2, -) + d_{21} = 3 + 6 = 9$	2
		$\psi_1(1, \{3\}) = 9,\ \psi_1(1, \{4\}) = 18$	3, 4
		$\psi_1(2, \{1\}) = 9,\ \psi_1(2, \{3\}) = 8,$	1, 3
		$\psi_1(2, \{4\}) = 15,\ \psi_1(3, \{1\}) = 7,$	4, 1
		$\psi_1(3, \{2\}) = 6,\ \psi_1(3, \{4\}) = 17$	2, 4
		$\psi_1(4, \{1\}) = 11,\ \psi_1(4, \{2\}) = 8,$	1. 2
		$\psi_1(4, \{3\}) = 12$	3
	$k = 2$	$\psi_2(1, \{2,3\}) = \mathrm{Min}[\psi_1(2, \{3\}) + d_{21},\ \psi_1(3, \{2\}) + d_{31},$ $\psi_1(2, \{1\}) + \psi_1(3, \{1\}) - d_{01}] = \mathrm{Min}[8 + 6,\ 6 + 4,$ $9 + 7 - 3] = 10$	3
		$\psi_2(1, \{2,4\}) = 16,\ \psi_2(1, \{3,4\}) = 15$	4, (3, 4)
		$\psi_2(2, \{1,3\}) = 9,\ \psi_2(2, \{1,4\}) = 14,$	3, (1, 4)
		$\psi_2(2, \{3,4\}) = 11$	(3, 4)
		$\psi_2(3, \{1,2\}) = 12,\ \psi_2(3, \{1,4\}) = 16,$	1, 1
		$\psi_2(3, \{2,4\}) = 15$	4
		$\psi_2(4, \{1,2\}) = 14,\ \psi_2(4, \{1,3\}) = 14,$	1, 3
		$\psi_2(4, \{2,3\}) = 13$	2
	k = 3	$\forall(j, S)$, since $\psi_k(j, S) = \infty$, we let L = 3.	

Table 3: Phase 2: matching

step		$\psi_k'(P_m)$	R_m
1	$k = 0$	$\psi_0'(\{1\}) = \psi_0(1, -) = 3$	{1}
		$\psi_0'(\{2\}) = 3$	{2}
		$\psi_0'(\{3\}) = 5$	{3}
		$\psi_0'(\{4\}) = 10$	{4}
2	$k = 1$	$\psi_1'(\{1,2\}) = \mathrm{Min}[\psi_1(1,\{2\}), \psi_1(2,\{1\})] = 9$	{1,2}
		$\psi_1'(\{1,3\}) = 7$, $\psi_1'(\{1,4\}) = 11$,	{1,3}, {1,4}
		$\psi_1'(\{2,3\}) = 6$,	{2,3}
		$\psi_1'(\{2,4\}) = 8$, $\psi_1'(\{3,4\}) = 12$	{2,4}, {3,4}
	$k = 2$	$\psi_2'(\{1,2,3\}) = \mathrm{Min}[\psi_2(1, \{2,3\}), \psi_2(2, \{1,3\}), \psi_2(3, \{1,2\})]$	{1,3,2}
		$= \mathrm{Min}\ [10, 9, 12] = 9$	
		$\psi_2'(\{1,2,4\}) = \mathrm{Min}[16, 14, 14] = 14$	{2,4,1}
		$\psi_2'(\{1,3,4\}) = \mathrm{Min}[15, 16, 14] = 14$	{1,3,4}
		$\psi_2'(\{2,3,4\}) = \mathrm{Min}[11, 15, 13] = 11$	{2,(3,4)}
3		$\psi'(\{1,3,2\}) + \psi'(\{4\}) = 9 + 10 = 19$	
		$\psi'(\{2,4,1\}) + \psi'(\{3\}) = 14 + 5 = 19$	
		$\psi'(\{1,3,4\}) + \psi'(\{2\}) = 14 + 3 = 17$	
		$\psi'(\{2,(3,4)\}) + \psi'(\{1\}) = 11 + 3 = 14$	
		$\psi'(\{1,2\}) + \psi'(\{3,4\}) = 9 + 12 = 21$	
		$\psi'(\{1,3\}) + \psi'(\{2,4\}) = 7 + 8 = 15$	
		$\psi'(\{1,4\}) + \psi'(\{2,3\}) = 11 + 6 = 17$	
		$\mathrm{Min}[19, 19, 17, 14, 21, 15, 17] = 14$	
4		tree(α): {0-1}, {0-2-(3, 4)}	

Proof: Consider a network with n nodes. This network can be modelled as a graph $G = (V, E)$ where V denotes a set n vertices corresponding to the network nodes. In each ($k = 1, 2, \ldots, L$) stage of subtree generation phase in the algorithm, we have to add ${}_{n-1}\boldsymbol{C}_k$ for n nodes, and compare ${}_{n-1}\boldsymbol{C}_k$ for the previous stage and n nodes. Therefore, the number of additions $= n\sum_{k=1}^{L} k \cdot {}_{n-1}\boldsymbol{C}_k = n(n-1)\sum_{k=1}^{L}((n-2)!/(k-1)!(n-1-k)!) = n(n-1)\sum_{k=1}^{L} {}_{n-2}\boldsymbol{C}_{k-1} = n(n-1)2^L$ and the number of comparisons $= n\sum_{k=1}^{L}(k-1)\cdot{}_{n-1}\boldsymbol{C}_k =$ the number of additions $- n\sum_{k=1}^{L} {}_{n-1}\boldsymbol{C}_k = n(n-1)\sum_{k=1}^{L} k\cdot{}_{n-2}\ \boldsymbol{C}_{k-1} - n\sum_{k=1}^{L} {}_{n-1}\boldsymbol{C}_k \simeq n(n-1)2^L - n(2^L - 1) = n(n-2)2^L$. In the matching phase, we do not consider the sequence of nodes unlike in the subtree generation phase. It is natural for this to make the amount of computations less in the most case. However, since there might exist the case that the amount of computation is same according to the traffic capacity constraint ($\sum_{i=1}^{n} \tau_i \leq \varepsilon$) regardless of the consideration for node sequence, we can state that the number of additions and comparisons in the matching phase is same as those in the subtree generation phase at most. Thus, the upper bound for the number of additions and comparisons in the matching phases are $n(n-1)2^L$ and $n(n-2)2^L$, respectively. Time complexity of link capacity allocation is O(1). Therefore, total execution time of DC-CMST algorithm is = execution time of subtree generation phase + execution time of matching phase + execution time of link capacity allocation phase = maximum $\{\mathrm{O}\ [n(n-1)2^L + n(n-2)2^L], \mathrm{O}[n(n-1)2^L + n(n-2)2^L], \mathrm{O}(1)\} = \mathrm{O}(kV^2)$, $1 \leq k \leq 2^L$. □

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 by 100. Computational experiments were carried out on an IBM-PC using the C language.

Fig. 6 represents the number of computations of DC-CMST algorithm for two different cases (heavy traffic, $\varepsilon \simeq 1/2\sum_{i=1}^{n} \tau_i$ and light traffic, $\varepsilon \simeq 1/4\sum_{i=1}^{n} \tau_i$). In the light traffic and $n = 30$, the number of additions and multiplications are 111 360 and 107 520, respectively.

Table 4: Phase 3: Link capacity allocation

Tree	Link (k $i - j$) k	i-j	Link cost (d_k)	Traffic flow (λ_k, pkts/s)	Capacity (Θ_k, Mbps)	Delay time (Φ_k, ms)
0-1	1	0-1	3.0	1.0	2.37	0.42
0-2- (3, 4)	2	0-2	3.0	3.0	41.01	0.24
	3	2-3	3.0	1.0	23.70	0.42
	4	2-4	5.0	1.0	23.70	0.42
	Mean network delay (δ), 1ms					
	Optimal total cost (F), 14.0					

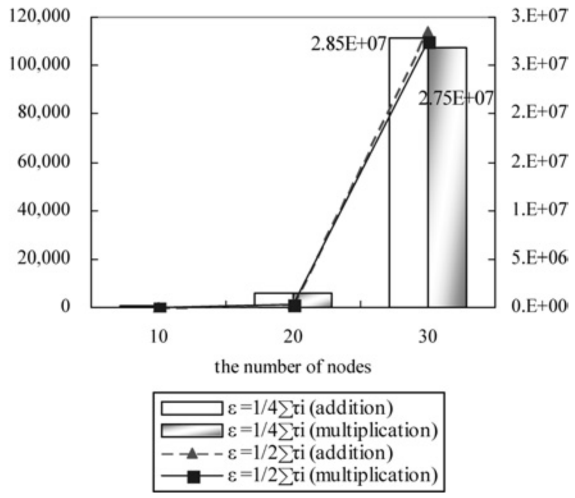

Fig. 6 *The number of computations*

These values are so small compared to the performance of modern computer. In the heavy traffic and $n = 30$, the number of additions and multiplications are increased to 2.85×10^7 and 2.75×10^7, respectively. There is no problem for current PC to execute theses computations in short time.

Fig. 7 depicts that the number of nodes and maximum ε corresponding to the sum of the number of additions and comparisons for number of nodes, being 30 and $\varepsilon = 29$. In Fig. 7, we observe that the theoretical number of computations for ($n = 30$, $\varepsilon = 29$) and ($n = 100$, $\varepsilon = 25$) are almost the same, but the real execution time is different. This is because $k \times {}_nC_{n-1}$ storage spaces are required in each stage k to store $\Psi_k(j, S)$, that is, as the number of nodes becomes large, memory access time increases sharply because the main memory cannot maintain all the results from the previous computation.

We now compute the amount of memory space. In order to store $\Psi_k(j, S)$ at each stage k in the feasible tree generation phase, we need $n \cdot {}_{n-1}C_k$ storage space. Fig. 8 shows the storage space requirements for $\Psi_k(j, S)$ according to varying k when $n = 30$ and 40. In the figure, we can find that the amount of memory space is maximum at $k = n/2$. At $n = 30$, maximum memory space requirements is 2.3×10^9 (2.3 GB). Since the main memory size of current PC is more than 1 GB(1.0×10^9), it can satisfy the 2.3-GB space requirements. The space requirements however increase up to 2.8 TB (2.8×10^{12}) at worst case when $n = 40$. This value is very large from the viewpoint of the space amount of current PC. Therefore we cannot maintain the entire computations at the previous stage of

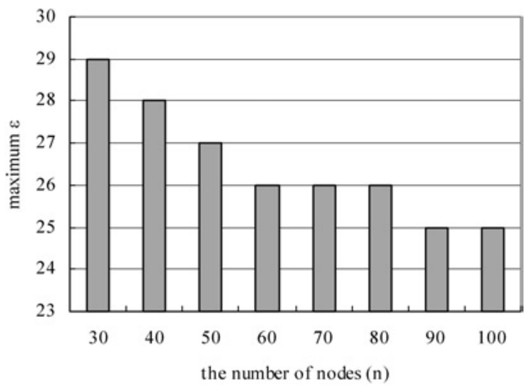

Fig. 7 *Relationship between maximum ε and n*

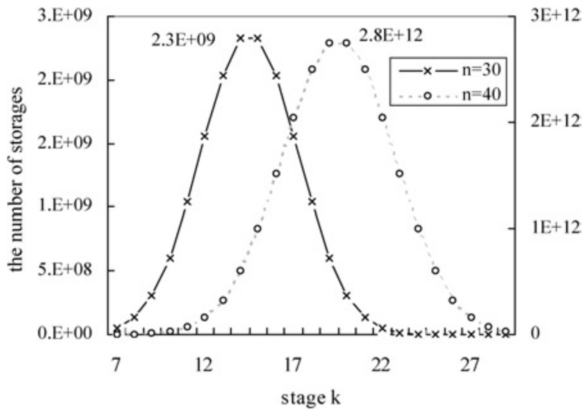

Fig. 8 *Storage space requirements for $\Psi_k(j,S)$*

dynamic programming in the main memory. This leads to reduction in the efficiency of algorithm. We can, of course, use the auxiliary memory such as hard disk instead of main memory to store the previous results. However, in this case, transfer time between main memory and auxiliary memory sharply increases the execution time of the algorithm. In addition our algorithm needs large main memory for the execution efficiency. Because of current PC's main memory limitation, our algorithm is effective when the number of nodes is less than 30 at the worst case.

On the basis of the above reasons, we generated n Poisson random numbers as traffic requirements (τ_i: $i = 1, 2, \ldots, n$) and used 10, 20 and 30 as the number of nodes (n). Maximum traffic (ε) handled in a single tree was used for three different cases, including heavy traffc, $\varepsilon = 1/2\sum_{i=1}^{n} \tau_i$; medium traffc, $\varepsilon = 1/3\sum_{i=1}^{n} \tau_i$ and light traffic, $\varepsilon = 1/4\sum_{i=1}^{n} \tau_i$. For each case, 10 problems were randomly generated. Fig. 9 depicts the average execution time of our proposed algorithm.

From Figs. 8 and 9, it is seen that the proposed DC-CMST algorithm is affected by node traffic and ε, and is effective when the number of nodes is less than 30.

In order to compare other algorithms, the savings rate (SA) is defined by (20), where A is the cost obtained by applying link capacity allocation phase to the topology of the EW solution – benchmark algorithm for CMST problem – and B is the cost of the modified CMST

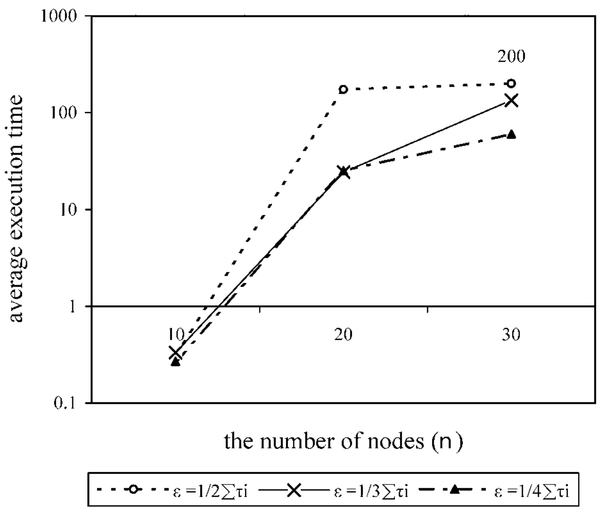

Fig. 9 *Average execution time* (s)

Table 5: Comparison of algorithms

Algorithm	Application problem	Need to modify	Solution method	Time complexity	Savings rate (*SA*)	The number of nodes (*n*)
EW algorithm	CMST	yes	heuristic	$O(V^2 \log V)$	benchmark	Large ($n > 100$)
Sharma algorithm	CMST	yes	heuristic	$O(V \log V)$	worse	Large ($n > 100$)
Gavish algorithm	CMST	yes	heuristic	$O(V^3)$	1–5 (%)	Large ($n > 100$)
Lee algorithm	CMST	yes	exact	$O(2^L V^2)$	exact	Small ($n \le 30$)
least-cost heuristic	DC-CMST	no	heuristic	$O(V^2 \log V)$	1-5 (%)	Large ($n > 100$)
optimal DC-CMST	DC-CMST	no	exact	$O(kV^2)$, $1 \le k \le 2^L$	exact	Small ($n \le 30$)

heuristics to meet the mean delay constraint [18].

$$SA = \frac{(A - B)}{A} \times 100 \tag{20}$$

Table 5 depicts the comparison of several algorithms, including the proposed optimal DC-CMST algorithm. It should be noted that the algorithms – EW, Sharma, Gavish [8] and Lee [7] – deal only with the CMST problem which does not consider the mean delay time constraint. In order to compare the performance of algorithms, we modified the above algorithms using the link capacity allocation phase so that they can satisfy the mean delay constraint required for the solution of the DC-CMST problem. Sharma algorithm can be applied only when the locations of nodes are given by the coordinates of plane, and every traffic requirement is one (Table 5). Gavish algorithm shows the results when every traffic requirement is one. If the traffic requirements are different, the results are inferior to that of the benchmark solution. For a complete graph, since the time complexity of Gavish algorithm is $O(\delta V^3)$, the computing time increases more sharply than the least-cost DC-CMST heuristic [8] with a time complexity of $O(V^2 \log V)$. Lee algorithm [7] gives the optimal solution for the CMST problem with small number of nodes. However, it does not consider the mean delay constraint, and hence cannot be applied to the DC-CMST problem without modification. To summarise, all the above algorithms, except least-cost heuristic, should be modified in order to apply them to the DC-CMST problem.

Our proposed algorithm shows good efficiency when the sum of the traffic is much smaller than ε. Since L becomes small in such a case, the amount of computations for our optimal DC-CMST algorithm also becomes small. The proposed optimal DC-CMST algorithm is affected by node traffic and ε, and is effective in the case when the number of nodes is less than 30. Since the execution time increases exponentially for more than 30 nodes, it is desirable to use the heuristic method, such as the least-cost heuristic, in such case.

6 Conclusions

In this paper, we presented the problem formulation and an exact algorithm for the DC-CMST problem, which has not been reported in any previous work. The proposed exact algorithm minimises the total cost to discover the capacitated spanning tree topology with additional mean network delay constraint. It consists of generating the feasible subtrees using dynamic programming, finding the capacitated minimum spanning trees by using the matching procedure and allocating the optimal link capacities to meet the network mean delay constraint. The algorithm can be applied to any network regardless of its configuration.

Computational complexity analysis and performance evaluation shows that the proposed algorithm is effective when the number of nodes is less than 30 and the total traffic volume is much smaller than the maximum traffic to be served by a port of the source node. Our proposed algorithm can be used by network designers for the topological design of local networks, or centralised networks with a small number of nodes. In addition, it can be used to discover the broadcast trees for real-time multimedia traffic.

Future work consists of developing the solution space relaxation algorithm in the feasible tree generation phase and the matching phase. In addition, it is desirable to develop a heuristic algorithm applicable to local networks with large number of nodes and an alternative exact algorithm for the small networks with further reduced computation time.

7 Acknowledgment

This work was supported in part by the 2007 research grant from Korea National University of Education.

8 References

1 Astic, I., and Festor, O.: ‘A hierarchical topology discovery service for IPv6 networks’. Network Operations and Management Symp., IEEE/IFIP, 2002, pp. 497–510

2 Bejerano, Y., Breibart, M., and Rastogi, R.: ‘Physical topology discovery for large multi subnet networks’. INFOCOM, 2003, pp. 342–352

3 Breitbart, Y., Garofalakis, M., Martic, C., Seshadri, S., and Silberschatz, A.: ‘Topology discovery in heterogeneous IP networks’. INFOCOM, 2000, pp. 265–274

4 Huffaker, B., Plummer, D., Moore, D., and Claffy, K.: ‘Topology discovery by active probing’. Applications and the Internet (SAINT) Workshops, 2002, pp. 90–96

5 Donnet, B., Raoult, P., and Friedman, T.: ‘Deployment of an algorithm for large-scale topology discovery’, *IEEE J. Sel. Areas Commun.*, 2006, **24**, (12), pp. 2210–2220

6 Lee, Y., and Atiquzzaman, M.: ‘Least cost multicast spanning tree algorithm for local computer network’ Lecture Notes in Computer Science 2005, vol. 3619, pp. 268–275

7 Lee, Y.: ‘Multimedia traffic distribution using capacitated multicast tree’, Lecture Notes in Computer Science, 2006; **4317**, pp. 212–220

8 Salama, H., Reeves, D., and Viniotis, Y.: ‘The delay-constrained minimum spanning tree problem’. 2nd IEEE Symp. on Computers and Communications, 1997, pp. 699–703

9 Li, S., Melhem, R., and Znati, T.: ‘An efficient algorithm for constructing delay bounded minimum cost multicast trees’, *J. Parallel Distrib. Comput.*, 2004, **64**, (8), pp. 364–372

10 Mala, C., and Selvakumar, S.: ‘Construction of an optimal multicast tree for group communication in a cellular network using genetic algorithm’, *Comput. Commun.*, 2006, **29**, (6), pp. 3306–3312

11 Xu, Z., and Chen, L.: ‘An effective algorithm for delay-constrained dynamic multicasting’, *Knowledge-Based Syst.*, 2006, **19**, (1), pp. 172–179

12 Reeves, D., and Salama, H.: ‘A distributed algorithm for delay-constrained unicast routing’, *IEEE/ACM Trans. Netw.*, 2000, **8**, (2), pp. 239–250

13 Karaman, A., and Hassanein, H.: ‘DCMC- delay-constrained multipoint communication with multiple sources’. IEEE

International Symp. on Computers and Communication, 2003, pp. 467–472

14 Zhengying, W., Bingxin, S., and Ling, Z.: ‘A delay-constrained least-cost multicast routing heuristic for dynamic multicast groups’, *Electron. Commun. Res.*, 2002, **2**, (2), pp. 323–335

15 Kun, Z., Heng, W., and Feng-Yu, L.: ‘Distributed multicast routing for delay and delay variation-bounded steiner tree using simulated annealing’, *Comput. Commun.*, 2005, **28**, (1), pp. 1356–1370

16 Tseng, S., Huang, Y., and Lin, C.: ‘Genetic algorithm for delay- and degree-constrained multimedia broadcasting on overlay networks’, *Comput. Commun.*, 2006, **29**, (7), pp. 3625–3632

17 Zheng, Y., and Akhtar, S.: ‘Networks for computer scientists and engineers’ (Oxford University Press, 2000), pp. 364–372

18 Lee, Y., and Atiquzzaman, M.: ‘Least cost heuristic for the delay-constrained capacitated minimum spanning tree problem’, *Comput. Commun.*, 2005, **28**, (11), pp. 1371–1379