Efficient Implementations of a Delay-Constrained Least-Cost Multicast Algorithm

Gang Feng, Kia Makki, and Niki Pissinou

Abstract: **Constrained minimum Steiner tree (CMST) problem is a key issue in multicast routing with quality of service (QoS) support. Bounded shortest path algorithm (BSMA) has been recognized as one of the best algorithms for the CMST problem due to its excellent cost performance. This algorithm starts with a minimum-delay tree, and then iteratively uses a k-shortest-path (KSP) algorithm to search for a better path to replace a "superedge" in the existing tree, and consequently reduces the cost of the tree. The major drawback of BSMA is its high time complexity because of the use of the KSP algorithm. For this reason, we investigate in this paper the possibility of more efficient implementations of BSMA by using different methods to locate the target path for replacing a superedge. Our experimental results indicate that our methods can significantly reduce the time complexity of BSMA without deteriorating the cost performance.**

Index Terms: **Multicast routing, constrained minimum Steiner tree problem, QoS routing, constrained unicast routing.**

I. INTRODUCTION

Multicast routing allows a source to send information to multiple destinations concurrently [1]. It has attracted a lot of attention in the last few years [2]–[4] due to the emergence of many new applications such as video conferencing, tele-education, and interactive multimedia game, etc., in which multicast routing protocols play a critical role. The major advantage of multicast routing lies in its capability of saving network resources since only one copy of message needs to be transmitted over a link shared by paths leading to different destinations.

Multicasting routing generally consists of three major tasks: (1) set up a tree that spans the source and all destinations, (2) reserve network resources such as bandwidth, buffer, and routing table so that information can be forwarded to all destinations along the paths specified by the multicast tree, and (3) dynamically manage the tree as members leave or join the multicasting group.

So far a number of multicast routing protocols have been proposed. In terms of how the initial multicast tree is established, they can be classified into two categories, *centralized* and *distributed* [1]. A centralized multicast protocol [5] assumes that the source node has the state information of the entire network and thus the multicast tree can be computed within the source node. To do this, a link-state protocol such as OSPF [6] is assumed to be available to provide the state information. In contrast, a distributed multicast routing protocol [7] requires other nodes to share the responsibility for computing the multicast tree, and generally a distance vector protocol [8] is employed to provide necessary routing information. While most people believe that distributed routing protocols are more robust than their centralized counterparts, we should also notice that as the underlying protocols for providing routing information, link-state protocols are more reliable and have better scalability than distance-vector protocols [9].

Initial research on multicast routing mainly focused on the problem of finding a minimum Steiner tree (MST) [3]. Since the MST problem is NP-complete [10], major efforts were dedicated to developing efficient heuristics that can produce a low-cost tree in reasonable time complexity [11]. The protocols based on these heuristics cannot provide additional quality of service (QoS) guarantees such as constrained delay and/or delay-jitter, and thus are called *QoS-oblivious protocols* [12]. In recent years, however, due to the fact that newly emerging multimedia applications have very strict QoS requirements, research interests have shifted to develop *QoS-sensitive protocols* [12] and a lot of work has been published. Nonetheless, a number of related issues still need further explorations.

For the above reason, we restrict our attention in this paper to the problem of finding a delay-constrained minimum-cost multicast tree, assuming that the source has the global state information provided by a link-state routing protocol. This particular problem, known as constrained minimum Steiner tree (CMST) problem, is NP-complete [10] and has been extensively studied in the literature [3]. Of all the previously proposed heuristics for the CSMT problem, the bounded shortest path algorithm (BSMA) proposed by Zhu *et al.* [4], [5] had the most profound influence on latter works. It has been demonstrated by computer simulation that on average BSMA can determine a constrained multicast tree with a very low cost [3], [4], [7], and [13]. There is no other heuristic currently available that can outperform BSMA in terms of the average cost of the resulting tree, and it has been frequently used as a criterion to measure the performance of other heuristics [7], [13]. The disadvantage of BSMA, however, is its high time complexity. It is probably the most time consuming one among all heuristics that have been proposed [3], [7], and [13].

A critical step in BSMA is to find a delay-constrained least-cost (DCLC) path. In the original implementation of BSMA, this is done by using a loopless k-shortest-path (KSP) algorithm [14] to enumerate the shortest paths in the order of increasing cost, and pick up the first path that satisfies all delay constraints. Since the value of k needs to be very large in order to achieve a

Manuscript received May 22, 2002; approved for publication by July 18, 2002, Division III Editor, Byung G. Kim, 2002.

G. Feng is with Department of Electrical Engineering at University of Wisconsin, Platteville, WI 53818, e-mail: gfeng@eng.fiu.edu.

K. Makki and N. Pissinou are with Telecommunications & Information Technology Institute, Florida International University, Miami, FL 33174. e-mail: {kia, pissinou}@eng.fiu.edu.

Work supported in part by NSF under grants No. 0196557 and No. 0123950.

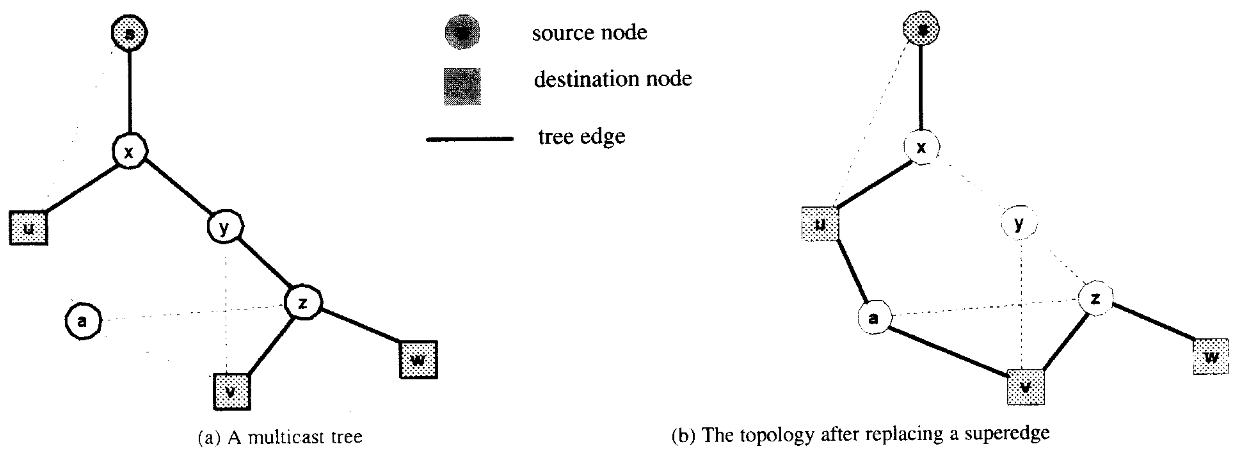


Fig. 1. Illustration of multicast tree and replacement of superedge.

good cost performance, this step may take a lot of computation time. On the other hand, we have seen many efficient exact and approximate algorithms for the DCLC problem published in recent years [15]–[17]. It is very likely that these algorithms can be used in BSMA to reduce its time consumption. For this reason, we explore alternative implementations of BSMA in this paper based on two recently proposed algorithms for the DCLC problem, and investigate the impact of such implementations on the resulting cost and time performance.

The remainder of this paper is organized as follows. The CMST problem is formally defined in Section II. Then, we briefly describe the heuristic BSMA in Section III. In Section IV, we suggest two alternative implementations of BSMA. Section V gives a review on some closely related work. In Section VI, the performance of the alternative implementations is investigated through extensive simulation, and Section VII concludes this paper.

II. NOTATION AND PROBLEM DEFINITION

A *network* is represented by a directed graph $G(V, E)$, where V is the set of nodes, and E is the set of links.

A *weight* w defines a number $w(e) \in R_0^+$ associated with each link e, i.e., $w : E \to R_0^+$. In particular, weight $d : E \to R_0^+$ is called *delay*, while $c : E \to R_0^+$ is called *cost*.

A *path* is a finite sequence of non-repeated nodes $\mathbf{p} = (v_0, v_1, \cdots, v_k)$, such that, for $0 \leq i < k$, there exists a link from v_i to v_{i+1}, i.e., $(v_i, v_{i+1}) \in E$. A link $e \in \mathbf{p}$ means that path $\mathbf{p}$ passes through link e. The delay and the cost of a path $\mathbf{p}$ are thus given by $d(\mathbf{p}) = \sum_{e \in \mathbf{p}} d(e)$ and $c(\mathbf{p}) = \sum_{e \in \mathbf{p}} c(e)$, respectively.

Given a multicast tree T spanning a source s and a set of destinations D, let $\mathbf{p}_T(s, v)$ denote the path on T from s to destination $v \in D$. The cost of the multicast tree T is defined by $c(T) = \sum_{e \in T} c(e)$. The set of destinations D is also called a multicast group with each destination being a group member.

Definition 1: [CMST Problem] Given a source s, a set of destinations D, a delay upper bound Δ_v for destination $v \in D$, the CMST problem needs to find a tree T spanning $D \cup \{s\}$ such that $c(T)$ is minimized subject to $d(\mathbf{p}_T(s, v)) \leq \Delta_v$, $\forall v \in D$.

The following two definitions are necessary to describe the alternative implementations of BSMA.

Definition 2: [DCLC Problem] Given a source, a destination, and a delay upper bound $\mathcal{D}$, the DCLC problem needs to find a path $\mathbf{p}$ from the source to the destination such that $c(\mathbf{p})$ is minimized subject to $d(\mathbf{p}) \leq \mathcal{D}$.

Definition 3: [DCC Problem] Given a source, a destination, a delay upper bound $\mathcal{D}$, and a cost upper bound $\mathcal{C}$, the delay-cost-constrained (DCC) problem needs to find a path $\mathbf{p}$ from the source to the destination such that $d(\mathbf{p}) \leq \mathcal{D}$, and $c(\mathbf{p}) \leq \mathcal{C}$.

III. HEURISTIC BSMA

Heuristic BSMA [4] solves the CMST problem by starting with a feasible solution and then gradually improving the solution. It includes two basic steps: (1) construct a minimum-delay tree, and (2) iteratively reduce the cost of the tree with the delay constraints always being satisfied.

The minimum-delay tree is constructed using Dijkstra's shortest path algorithm [18]. In case that the delay upper bound for a destination is so tight that even the least-delay (LD) path from source s to the destination cannot satisfy it, the QoS requirements have to be renegotiated. Once an initial feasible solution is obtained, however, the BSMA iteratively improves the solution by finding better paths to replace superedges on the current tree. A superedge is defined as a path on the tree such that if all edges and internal nodes of the path are taken out the tree becomes exactly two subtrees. For instance, in the tree shown in Fig. 1(a), only the following paths are superedges: $s \to x$, $x \to u$, $x \to y \to z$, $z \to v$, and $z \to w$. In order for a new path to be qualified to substitute a superedge, it must have a cost no more than the cost of the original superedge, and after the substitution we must still have a tree structure with the delay constraints being satisfied.

Fig. 2 is a high-level description of BSMA describing more clearly how the heuristic originally proposed in [4] works. After finding the minimum-delay tree, all superedges are identified and unmarked. The heuristic then enters an iterative procedure. At a particular iteration, the unmarked superedge with the high-

```
Heuristic BSMA($G, s, D, \Delta$)
Input:
        $G(V, E)$: graph
        $s$: source; $D$: set of destinations
        $\Delta$: set of delay upper bounds for destinations
Output:
        a delay-constrained tree spanning $D \cup \{s\}$
    $j = 0$
    $T_j \leftarrow$ minimum-delay tree
    unmark all superedges of $T_j$
    loop{
        $\mathbf{p}_h \leftarrow$ the highest-cost unmarked superedge
        if ($\mathbf{p}_h$=NULL) then return $T_j$
        mark superedge $\mathbf{p}_h$
        Obtain $T_j^1$ and $T_j^2$ by removing $\mathbf{p}_h$ from $T_j$
        $\mathbf{p}_s \leftarrow$ delay-constrained least-cost path between $T_j^1$ and $T_j^2$
        $T_{j+1} \leftarrow T_j^1 \cup \mathbf{p}_s \cup T_j^2$
        if $\mathbf{p}_s \neq \mathbf{p}_h$ then
            unmark all superedges
        $j = j + 1$
    }
```

Fig. 2. High-level description of heuristic BSMA.

est cost, denoted by $\mathbf{p}_h$, is taken out from the current tree, and thus we obtain two subtrees. A certain algorithm is further employed to identify the DCLC path, denoted by $\mathbf{p}_s$, between the two subtrees. Here the DCLC path is defined as the least cost (LC) path satisfying all delay constraints. In case that $\mathbf{p}_s$ and $\mathbf{p}_h$ are different, all superedges of the new tree are identified and unmarked. The heuristic stops when all superedges are marked.

As mentioned earlier in Section I, the DCLC path between two subtrees $\mathbf{p}_s$ is found using a KSP algorithm. To do that, two additional nodes are introduced, one connected to all nodes in one subtree and the other connected to all nodes in the other subtree, with all new connections having zero cost. The KSP algorithm is then employed to enumerate the paths between the two nodes in the order of increasing cost, and $\mathbf{p}_s$ is the first path that satisfies all delay constraints.

The BSMA described in Fig. 2, however, works well only if the network has symmetric link costs. Otherwise, the total cost of the tree may increase after replacing $\mathbf{p}_h$ with $\mathbf{p}_s$. This point can be illustrated in Fig. 1(b). Consider the initial multicast tree in Fig. 1(a). Suppose at certain iteration, $\mathbf{p}_h$ is $x \rightarrow y \rightarrow z$, and the DCLC path $\mathbf{p}_s$ between the two subtrees after removing $\mathbf{p}_h$ is $u \rightarrow a \rightarrow v$. Then, the new tree topology is shown in Fig. 1(b). Even though $\mathbf{p}_s$ may have a smaller cost than $\mathbf{p}_h$, the new tree may still have a higher total cost than the previous one if the cost of path $u \rightarrow a \rightarrow v \rightarrow z$ is greater than the cost of path $x \rightarrow y \rightarrow z \rightarrow v$. The reason behind this is because the two links $v \rightarrow z$ and $z \rightarrow v$ may have different costs in a directed graph.

Obviously, the above problem can be resolved by checking if the total cost of the tree will increase or not before deciding to make a replacement, but this may further complicate the implementation and increase the time complexity. For this reason, we make a slight modification on BSMA in our following implementations. Suppose T_j^1 is the subtree that contains the multicast source s after removing $\mathbf{p}_h$ at iteration j, and t is the end node of $\mathbf{p}_h$ on T_j^2. Our modification is to let $\mathbf{p}_s$ be the DCLC path between subtree T_j^1 and node t instead of the DCLC path between the two subtrees. The rest of the heuristic is kept the same. Obviously, BSMA with such modification can avoid the problem mentioned in the previous paragraph. The DCLC path between T_j^1 and node t can be found in a way similar to the one for finding the path between two subtrees except that now only one additional node is needed to connect all nodes on T_j^1.

For the purpose of consistency, throughout this paper we will refer the original implementation of BSMA to the one that uses a KSP algorithm to find the DCLC path between T_j^1 and node t by enumerating the shortest paths in the order of increasing cost. Unless otherwise stated, T_j^1 will be the subtree that contains the source s after removing a superedge, T_j^2 will be the other subtree, and node t will be the end node of the superedge on T_j^2.

IV. ALTERNATIVE IMPLEMENTATIONS OF BSMA

A. Motivation

The original BSMA uses a KSP algorithm to locate the DCLC path, requiring a total time complexity of $O(k|V|^3 \log |V|)$. In order to find such a path, the value of k could be extremely large, as also mentioned in [5]. The authors of [5] claim that the time complexity can be controlled by taking k as an input parameter, and by setting k to a smaller value we can trade off the cost performance against a lower time complexity. However, for a particular distribution of link weights and delay upper bounds we never know what value k should take so that the time consumption is acceptable. Even if the time complexity is controlled to a satisfactory level, we may doubt that maybe a simpler heuristic can achieve the same cost performance in even less time.

In view of this, one may immediately agree that the best method to solve the conflict between the cost performance and the time complexity is to use a faster algorithm to find the DCLC path. We may not require such an algorithm to be an exact algorithm that can always find the optimal path. Instead, as long as it can find the optimal path with a very high probability in a relatively low time complexity, we may expect that the modified version of BSMA based on such algorithm may achieve a very satisfactory performance in both cost and time.

Around a decade ago, the work on the DCLC problem was very limited. This was probably one of the reasons for the original implementation of BSMA. In the past few years, however, there has been a significant advance in research on finding least-cost path subject to one or more constraints [16], [19]. In the following two subsections, we will briefly review two algorithms for the DCLC problem and furthermore describe how these algorithms can be incorporated into BSMA to find the DCLC path between subtree T_j^1 and node t.

B. Exact Algorithm E_DCLC for the DCLC Problem

DCLC problem is NP-complete [10]. Therefore, any exact algorithm for this problem may be very time consuming when the network size becomes relatively large. However, there do exist several exact algorithms that have proven practically efficient when the network is of moderate size, e.g., the constrained Bellman-Ford (CBF) algorithm proposed in [17]. Here we describe another exact algorithm E_DCLC, which is based

```
E_DCLC($G(V, E), Src, Dst, c, d, \mathcal{D}$)
p ← least delay path
if ($d(\mathbf{p}) > \mathcal{D}$) then
    return NULL
q ← least cost path
if ($c(\mathbf{p}) = c(\mathbf{q})$) then
    return p
$\alpha = \frac{\mathcal{D}-d(\mathbf{p})}{c(\mathbf{p})-c(\mathbf{q})}$
compute $w(e) = d(e) + \alpha c(e)$ for each link $e$
set $DCLC_{max} = \mathcal{D}+\alpha c(\mathbf{p})$
loop{
    r ←the current shortest path w.r.t. $w$ found by a KSP algorithm
    if (r = NULL) then
        return p /* no more path */
    if ($w(\mathbf{r}) > DCLC_{max}$) then
        return p /* p is the optimal solution */
    if ($c(\mathbf{r}) < c(\mathbf{p})$ and $d(\mathbf{r}) \leq \mathcal{D}$) then
        p ← r
        update $DCLC_{max}$
}
```

Fig. 3. Exact algorithm E_DCLC for the DCLC problem.

on a KSP algorithm. Our previous computer simulation demonstrates that on average, E_DCLC takes less time than CBF if the KSP algorithm is implemented through sophisticated data structure [20].

E_DCLC is closely related to one of our recent works. In [21], we proposed an exact algorithm E_MCOP to find the least cost path subject to multiple constraints. The basic idea of E_MCOP is to first construct an aggregate weight w by linearly combining all the weights, and then use a KSP algorithm to check the shortest paths w.r.t. w one at a time to find the optimal solution.

E_DCLC is a special case of E_MCOP because for the DCLC problem there are only two link weights, delay d, and cost c. Therefore, we only need one parameter α to form the aggregate weight $w(e) = d(e) + \alpha c(e), \forall e \in E$. Once we have the aggregate weight for each link, a KSP algorithm can be similarly used to search for the optimal solution.

A more precise description of E_DCLC is shown in Fig. 3. Assuming that we need to find the DCLC path from source Src to destination Dst with a delay upper bound of $\mathcal{D}$, the algorithm starts by checking the feasibility of the delay constraint. If even the least delay (LD) path $\mathbf{p}$ has a delay greater than $\mathcal{D}$, we cannot find a feasible path at all. Otherwise, the algorithm finds the least cost (LC) path $\mathbf{q}$. In the case that $\mathbf{q}$ and $\mathbf{p}$ have the same cost, $\mathbf{p}$ must be the optimal solution. If $\mathbf{p}$ is neither infeasible nor optimal, the algorithm computes the value of parameter α, the aggregate weight $w(e)$ for each link e, and a quantity $DCLC_{max}$. A KSP algorithm is then employed to check the shortest paths w.r.t. w one at a time. If there is no more path available, or the aggregate weight of the current shortest path $\mathbf{r}$ exceeds $DCLC_{max}$, we can conclude that $\mathbf{p}$ is the optimal path, and the algorithm stops. On the other hand, if $\mathbf{r}$ is a feasible path with a lower cost than that of $\mathbf{p}$, we let $\mathbf{r}$ replace $\mathbf{p}$ and update $DCLC_{max}$.

Two issues need to be clarified. First, the value of α is given by

$$\alpha = \frac{\mathcal{D} - d(\mathbf{p})}{c(\mathbf{p}) - c(\mathbf{q})},$$

where $\mathbf{p}$ and $\mathbf{q}$ are the LD and LC paths, respectively. The reason for setting α to this value is because it guarantees that the smallest number of infeasible paths will be checked before the algorithm terminates. Due to space limit, please see [21] for more details. Second, at a particular iteration, if $w(\mathbf{r}) > DCLC_{max}$, we can conclude that $\mathbf{p}$ is the optimal path. This is because if $d(\mathbf{r}) + \alpha c(\mathbf{r}) > \mathcal{D}+\alpha c(\mathbf{p})$, then either $\mathbf{r}$ is an infeasible path or $\mathbf{r}$ has a higher cost than $\mathbf{p}$. This also applies to any subsequent shortest path since it must have an aggregate weight no less than $w(\mathbf{r})$.

```
NR_DCLC($G(V, E), Src, Dst, d, c, \mathcal{D}$)
q ← least cost path
if ($d(\mathbf{q}) \leq \mathcal{D}$) then
    return q
p ← least delay path
if ($d(\mathbf{p}) > \mathcal{D}$) then
    return NULL
if ($c(\mathbf{p}) \neq c(\mathbf{q})$) then
    set $\mathcal{C} = c(\mathbf{p}) - \epsilon$
    repeat
        r ← H_DCC($G(V, E), Src, Dst, d, c, \mathcal{D}, \mathcal{C}$)
        /*solved as a DCC problem */
        if (r ≠ NULL) then
            p ← r
            set $\mathcal{C} = c(\mathbf{p}) - \epsilon$
    until r = NULL
return p
```

Fig. 4. Approximate algorithm NR_DCLC for the DCLC problem.

C. Approximate Algorithm NR_DCLC for the DCLC Problem

NR_DCLC is an approximate algorithm for the DCLC problem proposed in [22]. As an approximate algorithm, it may not be able to find a feasible (or optimal) solution even if there exists one or more feasible solutions. However, computer simulation demonstrates that this algorithm can obtain a feasible solution with a very high probability [19], and moreover in most cases the obtained solutions are optimal [22]. The time complexity of this algorithm is very low. It only needs to run Dijkstra's algorithm 5 or 6 times on average and a maximum number of 20 times even when the network size is relatively large [22].

A high-level description of this algorithm is shown in Fig. 4. Like E_DCLC, NR_DCLC also starts by checking the LC path and the LD path. If the LC path is a feasible path, then it is also the optimal solution. If the LD path is not feasible, then no feasible path is available. If none of the above conditions is true, the algorithm enters a loop to search for the best solution. The searching procedure is done by first converting the DCLC problem to a DCC problem and then using a procedure H_DCC to search for a better solution by gradually tightening the cost upper bound.

As shown in Fig. 4, we assume the delay and the cost upper bounds of the DCC problem are denoted by $\mathcal{D}$ and $\mathcal{C}$, respectively ($\mathcal{D}$ is also the delay upper bound of the original DCLC problem). Initially, $\mathcal{C}$ is given by $c(\mathbf{p}) - \epsilon$, where ϵ is a small positive number. If H_DCC finds a feasible solution $\mathbf{r}$ to the DCC problem, $\mathbf{r}$ must satisfy both the delay and the cost constraints. This means that $\mathbf{r}$ is a feasible solution to the original DCLC problem and it has a lower cost than $\mathbf{p}$. By replacing $\mathbf{p}$ with $\mathbf{r}$ and updating the cost upper bound, the algorithm can continuously search for better solutions until no feasible path is returned by H_DCC.

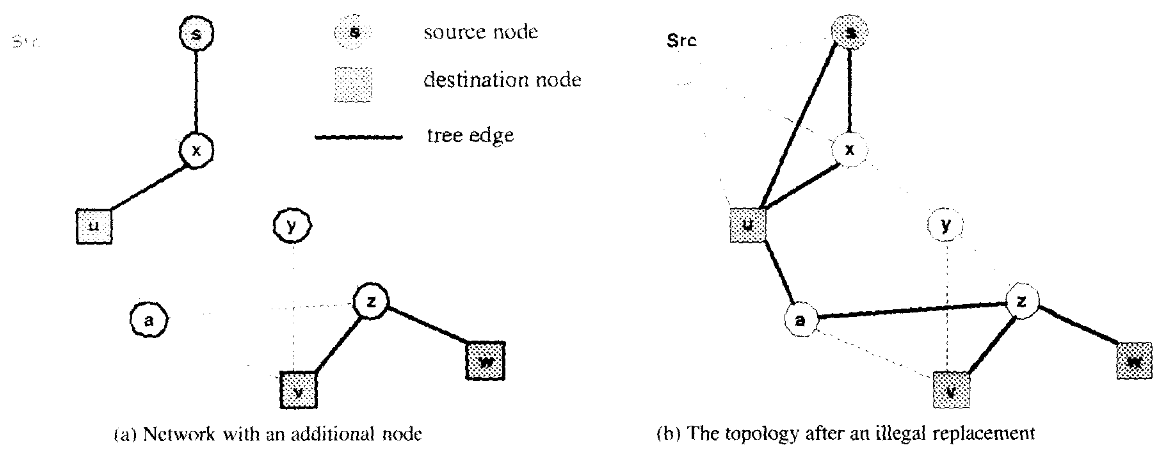

Src
s
source node
destination node
tree edge
x
u
y
a
z
v
w
(a) Network with an additional node
Src
(b) The topology after an illegal replacement

Fig. 5. Illustration of illegal replacement of superedge.

H_DCC uses a nonlinear Lagrange relaxation technique to search for a feasible solution to the DCC problem. It runs Dijkstra's algorithm (with modifications) no more than two times. Computer simulation indicates that its success ratio of finding a feasible solution is very high if such a path exists. For more details, please see [19], [22].

D. Implementation of BSMA based on E_DCLC or NR_DCLC

Recall that in BSMA we need to find a DCLC path between subtree T_j^1 and an end node t of a superedge. At first glance, one may think E_DCLC or NR_DCLC can be straightforwardly used to find such a path. However, the DCLC path in BSMA, after joining the two subtrees, must satisfy multiple delay constraints set by different destinations. For instance, in order for a path $u \to a \to z$ to be qualified to replace the superedge $x \to y \to z$ in Fig. 1(a), it must meet two delay constraints, i.e., the delay of path $s \to u \to a \to z \to v$ must be no greater than Δ_v, and the delay of path $s \to u \to a \to z \to w$ must be no greater than Δ_w. In contrast, algorithms E_DCLC and NR_DCLC assume that a deterministic delay upper bound is known for a given pair of source and destination.

For the above reason, before we can use E_DCLC or NR_DCLC to find a candidate path to replace a given superedge, we need to determine three arguments: Source Src, destination Dst, and the delay upper bound $\mathcal{D}$. $\mathcal{D}$ should take a value such that all multicast delay constraints must be satisfied for any path between Src and Dst with a delay no more than $\mathcal{D}$ to be used to replace the superedge. We choose Src and Dst the same way as in the original implementation described in Section III, namely, let Src be the additional node connected to all nodes on subtree T_j^1, and Dst be the end node t of the superedge on subtree T_j^2. The connection between the additional node and a node x on T_j^1 should have a cost of zero and a delay equal to the delay of the path on T_j^1 from the multicast source s to x. The upper bound $\mathcal{D}$ should be computed as follows:

$$\mathcal{D} = \min \left\{ \Delta_v - d\left(\mathbf{p}_{T_j^2}(t, v)\right) |\forall \text{ group member } v \text{ on } T_j^2 \right\},$$

where t is the end node of the superedge on T_j^2, $\mathbf{p}_{T_j^2}(t, v)$ is the path on T_j^2 from node t to node v.

There is another issue we need to take care of before running E_DCLC or NR_DCLC. If the DCLC path is obtained using one of the two algorithms without considering this issue, the resulting topology may not be a tree structure. Let us still use the tree shown in Fig. 1(a) to illustrate this point. Again, suppose we need to find a DCLC path to replace superedge $x \to y \to z$. After connecting the additional node to subtree T_j^1, the network is shown in Fig. 5(a). It is possible that the DCLC path returned by E_DCLC or NR_DCLC is $Src \to s \to u \to a \to z$. If we use this path to connect the two subtrees, the resulting topology would be the one show in Fig. 5(b), which is not a tree. To avoid this problem, we must ensure that the returned DCLC path crosses exactly one node on the two subtrees (obviously, the second node of the returned path must be a node on T_j^1). This can be done by trimming all incoming links of the nodes on T_j^1, except the link from the additional node, and all incoming links of the nodes on T_j^2, except the end node t of the superedge. The purpose of trimming links is to find the DCLC path. They should be recovered after running E_DCLC or NR_DCLC.

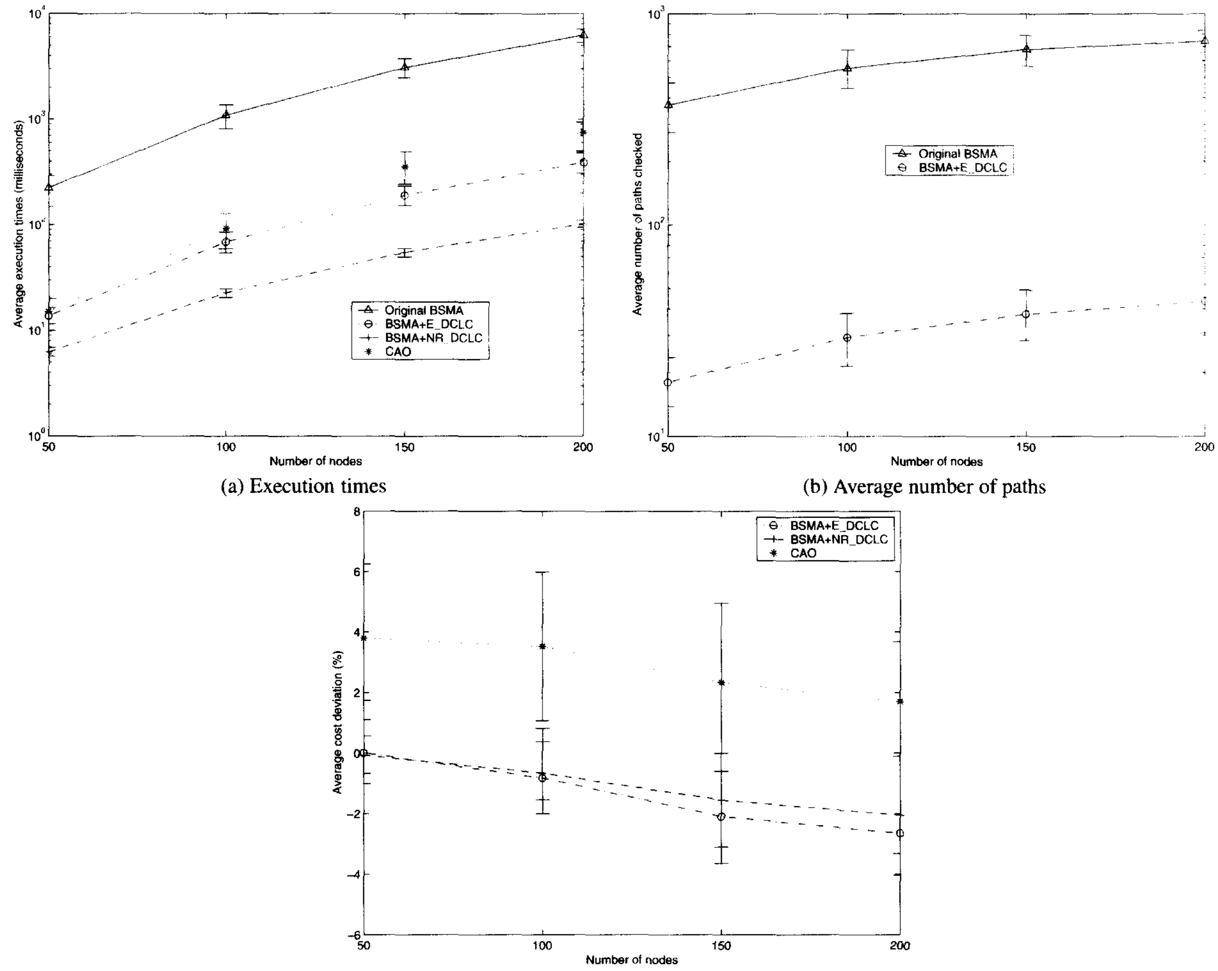
V. RELATED WORK

In this Section, we give a brief review on prior work on delay-constrained multicast and unicast routing.

As one of the earliest works on CMST problem, Kompella *et al.* proposed a heuristic in [23], which is called KPP by later researchers and in which the link delay and delay upper bound are assumed to be integers. The time complexity of KPP is $O(\Delta|V|^3)$, where Δ is delay upper bound for all destinations.

In [17], Widyono proposed four heuristics based on a constrained Bellman-Ford (CBF) algorithm, which is an exact algorithm for the DCLC problem. Among the four heuristics, the constrained adaptive ordering (CAO) was proved to have excellent performance. The basic idea of CAO is to use CBF to find a DCLC path between a partial tree and each of the destinations that have not yet been added to the tree, and the destination that requires the minimal additional cost is chosen to add to the partial tree. This procedure is repeated until all destinations are included in the tree.

In [3], Salama *et al.* made a performance comparison between



(a) Execution times

(b) Average number of paths

(c) Average cost deviation

Fig. 6. Performance comparison with four destinations.

several heuristics including BSMA, CAO, KPP, etc. Comparison results indicated that BSMA on average can achieve a multicast tree with the lowest cost, but takes much more time than other heuristics. CAO has the second best cost performance and requires less computation time than BSMA and KPP for networks with 100 nodes or less.

In [24], Hong *et al.* proposed a heuristic very similar to CAO except that an approximate algorithm BG [25] was used to find the DCLC path. Statistical data indicate that this heuristic is very efficient in both time and cost performance, but unfortunately the authors did not compare it with other heuristics in their paper. Nevertheless, since CAO uses an exact algorithm to find the DCLC path, we believe that CAO should have better cost performance.

In [13], Matta *et al.* proposed a heuristic QDMA, which has very low time complexity. On average it is about 50 times faster than CAO and 500 times faster than BSMA. The cost of the resulting multicast tree is about 10% worse than that of BSMA.

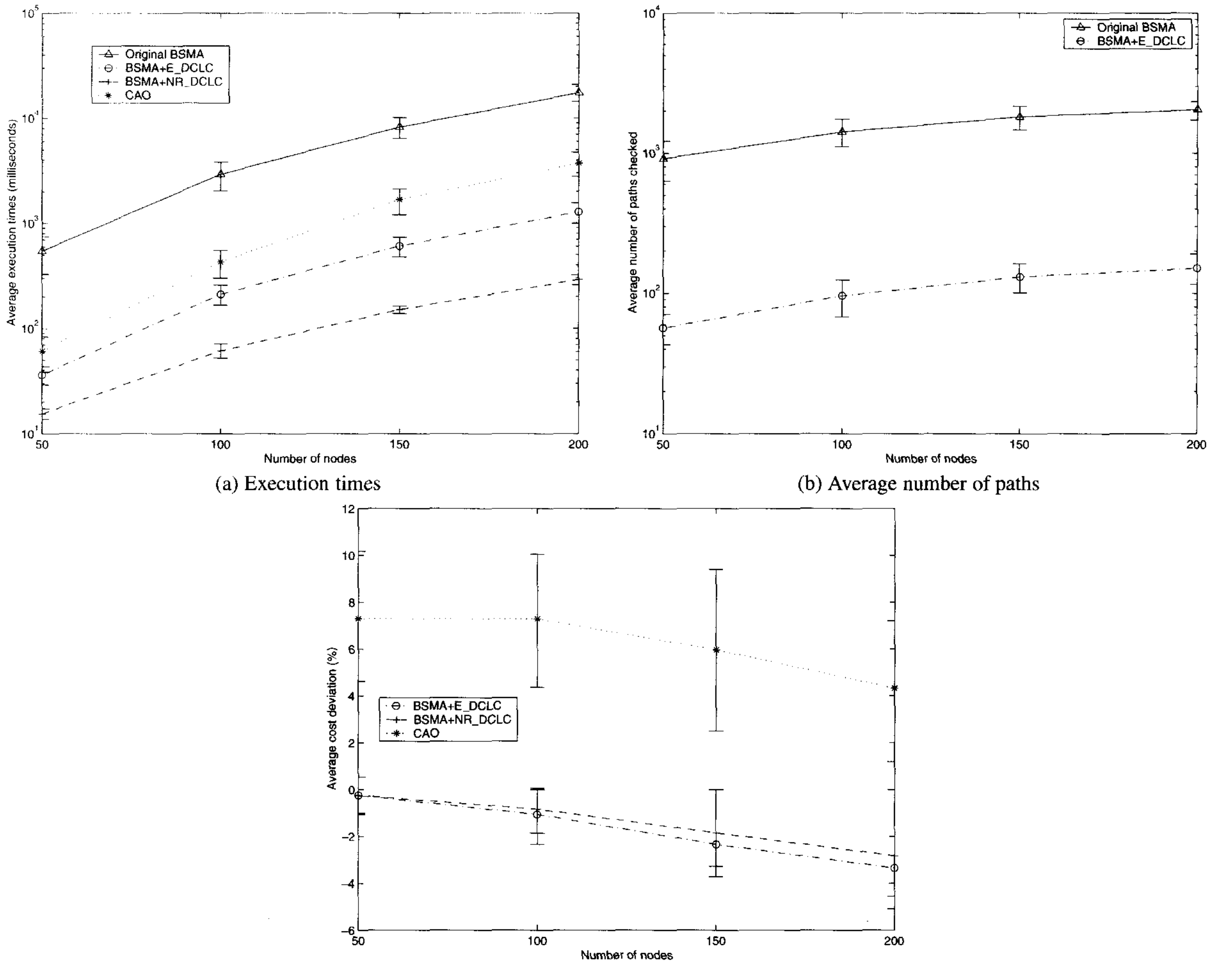
There are many other heuristics for the CMST problems, but previous research results [3] demonstrate that they either have worse cost performance or need more computation time than certain heuristic mentioned above.

Since in this paper algorithms for the DCLC problem play a critical role in our alternative implementations of BSMA, we also briefly review the current status in this area. In recent years, both exact algorithms and heuristics have been proposed to solve the DCLC problem. Probably E_DCLC described in this paper (and its generalization E_MCOP in [21]) and CBF mentioned earlier are the most efficient two exact algorithms currently available for the DCLC problem. In the following we give a brief review on previously proposed approximate algorithms.

The earliest work for the DCLC problem probably should be accredited to Hassin's two ϵ-optimal approximation algorithms [26] which can produce solutions with their costs less than $1+\epsilon$ times the cost of the optimal solution. These two algorithms, albeit very efficient in finding feasible solutions, have very high time complexities [19].

In [16], Jüttner *et al.* proposed a heuristic algorithm based on linear Lagrange relaxation technique. Its basic idea is to first construct an aggregate weight by linearly combining delay and cost, and then use Dijkstra's algorithm to find the corresponding shortest path. This heuristic is in essence the same as the approximate algorithm DCLC-IA-II proposed in [15] and the BG in [25], and it is renamed as LR_DCLC in [22].

In [22], Feng *et al.* compared the performance of heuristics



(a) Execution times

(b) Average number of paths

(c) Average cost deviation

Fig. 7. Performance comparison with eight destinations.

NR_DCLC and LR_DCLC. Simulation results demonstrate that both heuristics have excellent performance. They can find optimal solutions with very high probability in very low time complexity. In comparison, NR_DCLC on average can find a solution with a lower cost but needs slightly more computation time than LR_DCLC.

VI. PERFORMANCE EVALUATION

In this Section, we investigate the performance of the BSMA, based on three types of implementations: The original implementation, and the two alternative implementations based on E_DCLC and NR_DCLC, respectively. The two alternative implementations will be denoted by "BSMA+E_DCLC" and "BSMA+NR_DCLC," respectively, in the following figures. In view of the performance comparison results in [3], CAO is also tested as a reference.

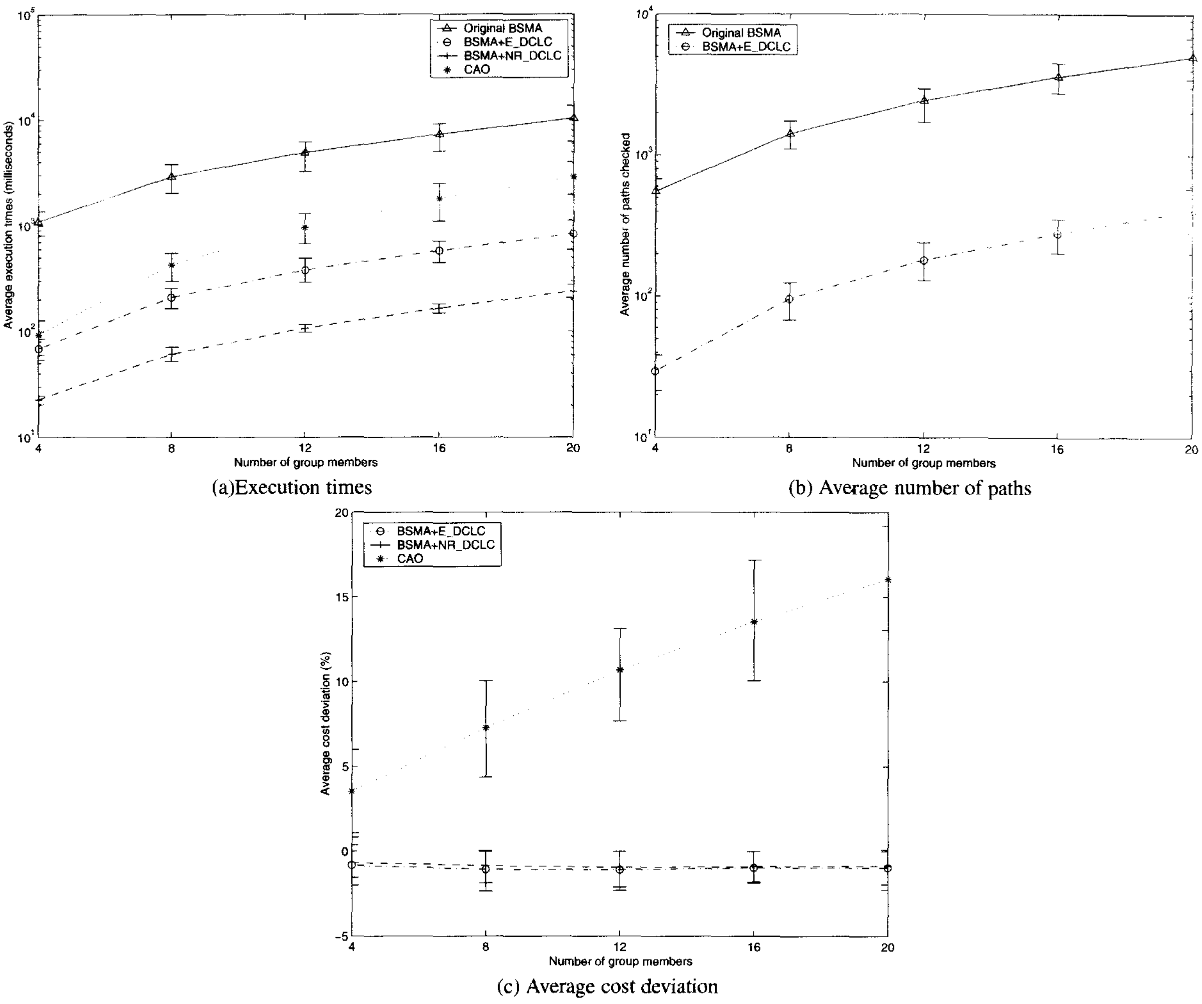
Two types of network topologies are used to test these heuristics. The topologies used in Figs. 6–8 are generated using Waxman's method, which has been used extensively in previous works, while the topologies used in Fig. 9 are generated based on the Tansit-Stub model proposed by Zegura *et al.* [27]. Waxman's method is used here for the purpose of making a fair comparison with previous works, while Zegura's method is used in order to test the performance of these heuristics in real communication networks. This is because Tansit-Stub model has proved to be able to more accurately characterize real Internet topologies.

With Waxman's method, all nodes are first randomly distributed in a square, and links between nodes u and v are generated with probability

$$P(u,v) = \beta \exp\left(\frac{-d(u,v)}{\alpha L}\right),$$

where $d(u,v)$ is the Manhattan distance between nodes u and v, and L is the maximum possible distance between any two nodes. In our simulation, four types of networks with nodes of 50, 100, 150, and 200 are generated, and parameters (α,β) for the four types of network are $(0.4,0.3)$, $(0.2,0.3)$, $(0.2,0.25)$, and $(0.2,0.2)$, respectively. After a network topology is generated, we let link delay be the physical distance between the two end nodes, and link cost equal the link delay times a random number in $(0,1]$.

For a given size of a network with a specified number of group members, we generated 100 network topologies, for which 100 sources and the corresponding multicast groups are randomly generated. For a given source and its multicast group, we assume all destinations have the same delay upper bound given



(a)Execution times (b) Average number of paths (c) Average cost deviation

Fig. 8. Performance comparison for 100-node networks with different numbers of destinations.

by

$$\max\{d(\mathbf{p}(s,v))|v \in D\},$$

where $\mathbf{p}(s,v)$ is the LD path from source s to destination v.

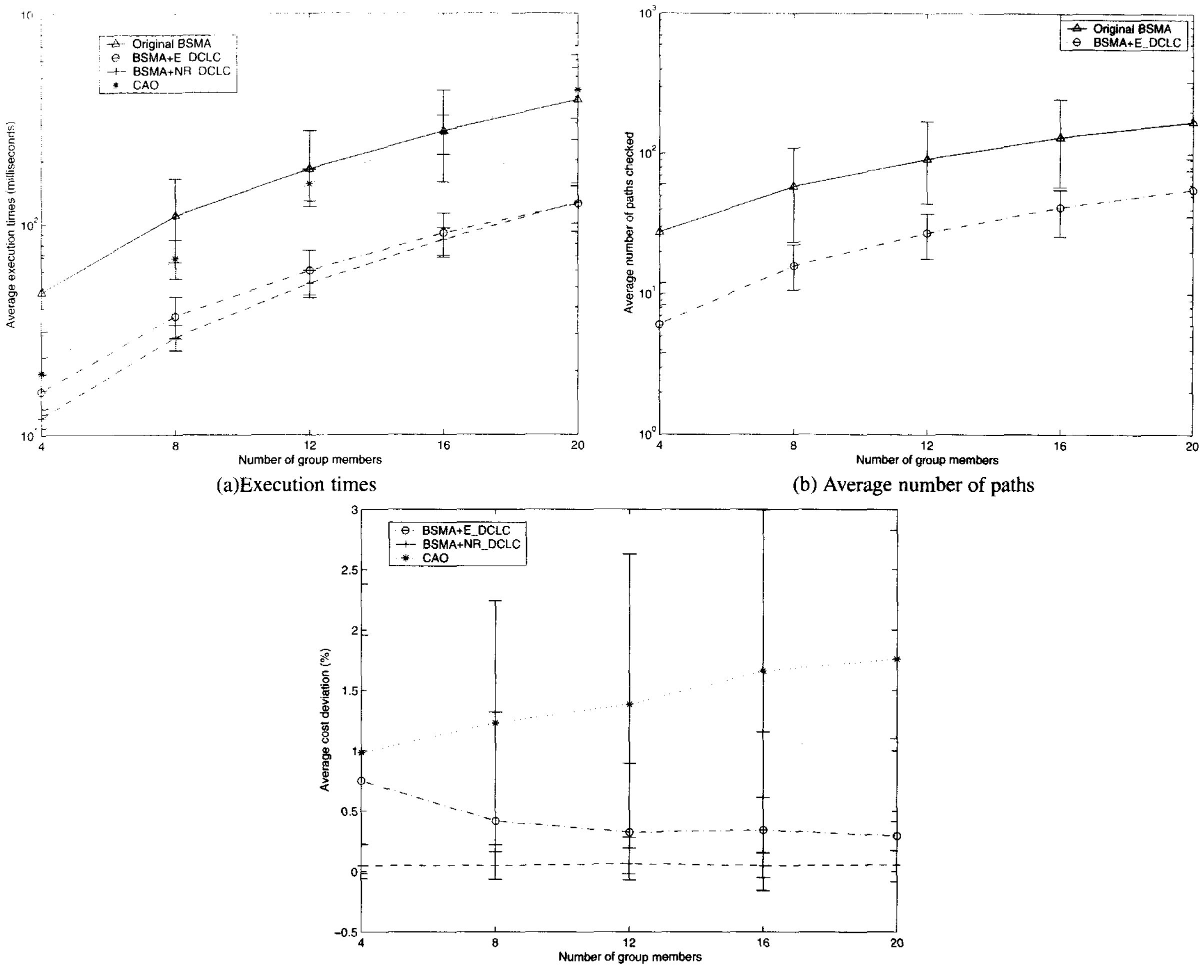
For each multicast routing request, heuristics BSMA, BSMA+E_DCLC, BSMA+NR_DCLC, and CAO are used independently to find a multicast tree. Based on the results returned by each heuristic, two performance measures with 95% confidence intervals are computed. The first performance measure is the *average computation time* for solving a single CMST instance. The second is called *average cost deviation* from the original BSMA. For a given CMST problem, the cost deviation of heuristic $\mathcal{A}$ is defined by

$$\frac{c\left(T_{\mathcal{A}}\right)-c\left(T_{BSMA}\right)}{c\left(T_{BSMA}\right)} \times 100\%,$$

where $c\left(T_{\mathcal{A}}\right)$ is the cost of the tree returned by heuristic $\mathcal{A}$, and $c\left(T_{BSMA}\right)$ is the cost of the tree returned by BSMA with original implementation.

For BSMA and BSMA+E_DCLC, an additional performance measure, i.e., *average number of paths* checked by the KSP for a single CMST instance, is also computed. In addition, to ensure that the simulation can be finished in reasonable time, BSMA, and BSMA+E_DCLC are only allowed to check up to 200 of the shortest paths for replacing a superedge. All simulations were run on a Dell Pentium IV computer.

Fig. 6 shows the performance measures when the number of destinations is four. From Fig. 6(a) we may see that BSMA+E_DCLC requires much less computation time than the original BSMA, and even requires less time than CAO. In comparison with BSMA+E_DCLC, BSMA+NR_DCLC can further reduce the execution time. Fig. 6(b) shows the average number of paths checked by the KSP algorithm. We can see that on average BSMA+E_DCLC needs to check much smaller number of paths than the original BSMA. This is also the reason why BSMA+E_DCLC is more computationally effective. Furthermore, with the 200-path limit, in certain cases, the original BSMA may not be able to find the optimal path, while BSMA+E_DCLC can. Therefore, on average, BSMA+E_DCLC may achieve better cost performance than BSMA. This is demonstrated in Fig. 6(c), which shows that in most cases the average cost deviation of BSMA+E_DCLC is less than zero. From our previous definition of the cost deviation, we know this means that on average, BSMA+E_DCLC can find a multicast tree with lower cost than BSMA. In addition, we should notice that on average, BSMA+NR_DCLC can also find solutions better than BSMA, even though slightly worse than



(a)Execution times (b) Average number of paths

(c) Average cost deviation

Fig. 9. Performance comparison for 100-node Transit-Stub topologies with different numbers of destinations.

BSMA+E_DCLC. The reason that the average cost deviation of BSMA+E_DCLC or BSMA+NR_DCLC decreases with the increase of network size is because for larger networks the limit of 200 paths has a more severe impact on the cost of the solution of BSMA.

Fig. 7 shows the performance measures when the number of destinations is eight. Comparing Figs. 7 and 6, we may see that for the two cases (4 vs. 8 destinations) the performance measures are very similar, except that all heuristics need more computation time when the number of destinations increases.

Fig. 8 shows the performance measures for 100-node networks with different numbers of destinations. Figs. 8(a) and 8 (b) are easy to understand since more destinations require more computation time, as mentioned earlier. Fig. 8(c) shows that the average cost deviation of CAO becomes higher and higher with the increase of destinations, and we notice that this complies with the results in previous works [3]. On the other hand, BSMA+E_DCLC and BSMA+NR_DCLC have almost fixed cost deviations relative to BSMA. This is because BSMA, BSMA+E_DCLC and BSMA+NR_DCLC use the same basic idea to find the multicast tree. Their only difference lies in the way of finding the DCLC path, and the impact of this difference cannot be reflected unless we change the network size or the upper limit of the total number of paths to be checked.

Fig. 9 also shows the performance measures for 100-node networks with different numbers of destinations. However, the topologies used in this case are generated using the Transit-Stub model [27], as mentioned earlier in this section. The link weights, and the delay upper bound are generated using the same method as the one described earlier. Since the topologies generated in this case are always symmetric, and the number of links is much less than the one in our previous experiments, we notice that the time consumption is much less compared with the case where topologies are generated using Waxman's method. This can be illustrated by comparing Fig. 8(a) and Fig. 9(a). On the other hand, it should also be noted in Fig. 9(a) that the two alternative implementations still take much less time than the original implementation. From Fig. 9(c), we can see that even though the solutions obtained by the alternative implementations have slightly higher cost than those given by the original BSMA, their difference is very small (less than 1%).

From these results, we may conclude that the advantage of BSMA+E_DCLC or BSMA+NR_DCLC over the original BSMA is unquestionable. They also considerably outperform CAO in both time and cost performance. It should be noted that previous works have already demonstrated that CAO can outperform KPP.

VII. CONCLUSION

We have demonstrated in this paper that the proposed alternative implementations of BSMA can very efficiently solve the constrained minimum Steiner tree problem. They not only significantly reduce the computation time, but also obtain solutions with lower costs in case there is an upper limit on the number of paths that can be checked by a KSP algorithm. Compared with some other heuristics, our implementations also have obvious advantages. We even believe that they are very competitive when compared with many other heuristics if we take into account both the cost performance and the time complexity.

VIII. ACKNOWLEDGEMENT

This work was supported in part by NSF under grants No. 0196557 and No. 0123950.

REFERENCES

[1] S. Chen and K. Nahrstedt, "An overview of quality of service routing for next-generation high-speed networks: Problems and solutions," *IEEE Network*, pp.64–79, Nov./Dec. 1998.

[2] K. Makki, N. Pissinou, and O. Frieder, "Efficient solutions to multicast routing in communication networks," *Mobile networks and Applications*, vol. 1, pp. 221–232, 1996.

[3] H. F. Salama, D. S. Reeves, and Y. Viniotis, "Evaluation of multicast routing algorithms for real-time communication on high-speed networks," *IEEE J. Select. Areas Commun.*, vol. 15, no. 3, pp. 332–345, Apr. 1997.

[4] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves, "A source-based algorithm for delay-constrained minimum-cost multicasting," *IEEE INFOCOM'95*, vol. 1, 1995, pp. 377–385.

[5] M. Parsa, Q. Zhu, and J. J. Garcia-Luna-Aceves, "An iterative algorithm for delay-constrained minimum-cost multicasting," *IEEE/ACM Trans. Networking*, vol. 6, no. 4, pp. 461–474, 1998.

[6] J. Moy, "OSPF version 2, "Stands track RFC 2328, IETF, Apr. 1998.

[7] X. Jia, "A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks," *IEEE/ACM Trans. Networking*, vol. 6, no. 6, pp. 828–837, Aug. 1998.

[8] C. Hedric, "Routing information protocol," RFC 1058, 1988.

[9] *Internetworking Technology Handbook*, Cisco online documentation, available at http://www.cisco.com/univercd/cc/td/doc /cisintwk/ito_doc/ index.htm.

[10] M. R. Garey and D. S. Johnson, *Computers and intractability, a guide to the theory of NP-completeness*, Freeman, San Francisco, 1979.

[11] L. Kyou, G. Markowsky, and L, Berman, "A fast algorithm for Steiner trees," *Acta Info.*, vol. 15, no. 2, pp. 141–145, 1981.

[12] M. Faloutsos, A. Banerjea, and R. Pankaj, "QoSMIC: Quality of service sensitive multicast internet protocol," *SIGCOMM '98*, Sept. 1998, pp.144–153.

[13] I. Matta and L. Guo, "QDMR: An efficient QoS dependent multicast routing algorithm," *J. Commun. Networks*, vol. 2, no. 2, 2000.

[14] E. L. Lawler, *Combinatorial optimization: networks and matroids*, Holt, Rinehart and Winston, 1976.

[15] G. Feng and C. Doulgeris, "Fast algorithms for delay-constrained least-cost unicast routing," shortened version presented on *INFORMS'2001*, Miami Beach, Nov. 2001,
available at http://www.students.miami.edu/~gfeng/.

[16] A. Jüttner *et al.*, "Lagrange relaxation based method for the QoS routing problem," *INFOCOM'2001*, Alaska, 2001.

[17] R. Widyono, "The design and evaluation of routing algorithms for real-time channels," Tech. Rep. ICSI TR-94-024, University of California at Berkley, International Computer Science Institute, June 1994.

[18] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[19] T. Korkmaz and M. Krunz, "Multi-constrained optimal path selection," *INFOCOM'2001*, Alaska, 2001.

[20] D. Eppstein, "Finding the *k* shortest paths," *SIAM J. Computing*, vol. 28, no. 2, pp.652–673, 1998.

[21] G. Feng *et al.*, "Heuristic and exact algorithms for QoS routing with multiple constraints," *IEICE Trans. Communications*, accepted for publication, available at http://www.students.miami.edu/~gfeng/.

[22] G. Feng *et al.*, " Performance evaluation of delay-constrained least-cost QoS routing algorithms based on linear and nonlinear Lagrange relaxation," *ICC'02*, vol. 4, New York, 2002, pp. 2273–2278.

[23] V. Kompella, J. C. Pasquale, and G. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 286–292, 1993.

[24] S. P. Hong, H. Lee, and B. H. Park, "An efficient multicast routing algorithm for delay-sensitive applications with dynamic membership," *INFOCOM'98*, 1998, pp. 1433–1440.

[25] D. Blokh and G. Gutin, "An approximate algorithm for combinatorial optimization problems with two parameters," IMADA preprint PP-1995-14 (Department of Mathematics and Computer Science, Univ. of Southern Denmark), 1995.

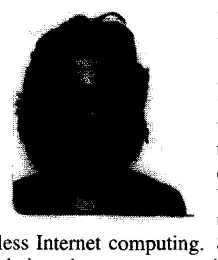
[26] R. Hassin, "Approximation scheme for the restricted shortest path problem," *Mathematics of Operation Research*, 17(1): 36–42, Feb. 1992.

[27] E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork", *Infocom'96*, Mar. 1996, pp. 594–602.

[28] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Select. Areas Commun.*, 6(9): 1617–1622, Dec. 1988.

Gang Feng received his B.E. and M.E. from University of Electronic Science and Technology of China in 1992 and 1995 respectively, and Dr. Eng. from Beijing University of Posts and Telecommunications in 1998, all in electrical engineering. Between 1998 and 2001, he was studying in University of Miami, where he was granted Ph.D. in December 2001. Between July 2001 and August 2002, he was working in Telecommunications and Information Technology Institute at Florida International University as a Research Associate. Since August 2002, he has been with the faculty in Depart of Electrical Engineering at University of Wisconsin, Platteville. His major research interests include routing in next generation broadband wired/wireless networks, optical networking, combinatorial optimization, neural networks, and evolutionary computation.

Kia Makki received his Ph.D. in computer science from the University of California, Davis, and two M.S. degrees from The Ohio State Universities and West Coast University. He is currently a tenure professor with an endowed chair professorship at Telecommunications and Information Technology Institute in Florida International University. His technical interests include wireless and optical telecommunications. He has over hundred refereed publications and his work has been extensively cited in books and papers. He has been Associate Editor, Editorial Board Member, Guest Editor of several International Journals including Editorial Board Member of several telecommunication journals. He has been involved in numerous conferences as a Steering Committee Member, General Chair, Program Chair, Program Vice-Chair, and Program Committee Member



Niki Pissinou received her Ph.D. in Computer Science from the University of Southern California, her M.S. in Computer Science from the University of California at Riverside, and her B.S.I.S.E. in Industrial and Systems Engineering from The Ohio State University. She is currently as a tenured professor and the director of the Telecommunication and Information Technology Institute at FIU. She has published over hundred refereed publications and has received best paper awards. She has also co-edited seven volumes and is the author of an upcoming book on wireless Internet computing. She has also served as a steering committee, general chair and program committee member of over hundred programs and organizational committees of IEEE and ACM sponsored technical conferences. She has also refereed for a wide variety of conferences and journals, and has served as a reviewer on NSF and NASA panel reviews including the KDI and ITR panels. She has been the editor of eight journals including IEEE TKDE and guest editor of seven journals. She has received seven achievements awards and an invited speaker and keynote speaker of conferences.