

## Algorithms for Static and Dynamic Group Multicast Routing Under Bandwidth Constraints

KUN-CHENG TSAI AND CHYUOHWA CHEN  
*Department of Electronic Engineering*  
*National Taiwan University of Science and Technology*  
*Taipei, 106 Taiwan*

Multicast routing allows network sources to use network resources efficiently by sending only a single copy of data to all group members. In the group multicast routing problem (GMRP), every member of a group is also a source node. The routing algorithm must construct a source-based routing tree for each member of the group which spans all other member nodes without exceeding the capacities of the traversed links. Previous work adopted the direct, intuitive approach by first creating an independent source-based multicast tree for each member node, and then iteratively locating network links whose capacity constraints are violated and eliminating the violation by rerouting the trees. In this paper, we propose a more systematic approach to solve the static GMRP based on the idea of critical pairs. For the dynamic GMRP, we propose the **DGMCP** algorithm. Through extensive experiments, our proposals are shown to outperform previous algorithms in constructing group multicast trees with low costs and high success ratios.

**Keywords:** group multicast routing, Steiner tree, dynamic group multicast routing, QoS routing, constrained Steiner tree

### 1. INTRODUCTION

Multicasting is an important network mechanism for achieving efficient network resource utilization by sending only one copy of data from a source to multiple recipients. Multicast routing algorithms build multicasting trees with the source node as the root, and terminal nodes as leaves so that data may be transmitted to the terminals in an efficient manner. However, finding low cost multicast trees is a difficult problem. The problem of finding the least cost multicast tree in an undirected graph is called the Steiner tree problem (STP), while the problem of finding the least cost tree in a directed graph is referred to as the directed Steiner tree problem (DSTP). The Steiner tree problem has been known to be NP-complete [1] even when the graph is induced by points in the plane, and is MAX SNP-hard [2] for general graphs.

With the emergence of real-time applications, such as videoconferences and on-line games, and peer-to-peer (P2P) applications, such as Napster [3], Mirror Image [4], and other content distribution networks [5], a group of entities may exchange data among themselves. A network layer may provide support for these applications by creating a set of multicast routing trees simultaneously. The problem of finding a set of multicast trees where every member in a group is a terminal as well as a source node is called the group

---

Received June 3, 2003; revised February 24 & June 23, 2004; accepted August 23, 2004.  
Communicated by Chu-Sing Yang.

multicast routing problem (GMRP). The routing algorithm must construct a multicast tree for each member node with each tree rooted at the member and spanning all other member nodes simultaneously. This paper improves upon previous proposals and attempts to shed some light on the GMRP.

In this paper, the network is represented by a directed graph  $G = (V, E, b, c)$ , where  $V$  denotes the set of nodes,  $E$  denotes the set of edges,  $c$  is a cost function on the edges:  $E \rightarrow R^+$ , and  $b$  is a capacity function:  $E \rightarrow R^+$ . That is,  $b(e)$  denotes the capacity of the bandwidth on the edge  $e \in E$ , and  $c(e)$  denotes the cost for one unit flow over the edge. An instance of the GMRP is defined by a multicast group, denoted by a set of nodes  $X = \{v_1, v_2, \dots, v_k\}$ ,  $X \subseteq V$ , and for each member node  $v_i \in X$ , the amount of bandwidth  $B_i$  required by this node, which is used by  $v_i$  to send data to all other member nodes. Following convention, we denote the number of elements in the set  $X$  using the notation  $|X|$ . The problem of group multicast routing is to find a set  $\{T_1, T_2, \dots, T_k\}$  of multicast trees. For each tree  $T_i = (V_i, E_i)$ , where  $V_i \subseteq V$  denotes the set of nodes and  $E_i \subseteq E$  denotes the set of links in the tree,  $T_i$  is rooted at node  $v_i \in X$ , spans all the other member nodes in  $X$ , and satisfies the bandwidth constraint. More formally, GMRP is formulated as

$$\text{Min} \sum_{i=1}^k \sum_{e \in E} c(e) B_i Y_{i,e}, \quad (1)$$

subject to

$$\sum_{i=1}^k B_i Y_{i,e} \leq b(e), \quad \forall e \in E, \quad (1.1)$$

$$\exists p(v_i, v_j) \subseteq T_i \text{ for each pair } (v_i, v_j), v_i \in X, v_j \in X - \{v_i\}, \quad (1.2)$$

and

$$Y_{i,e} = \begin{cases} 1, & \text{if } e \in E_i \\ 0, & \text{otherwise} \end{cases}. \quad (1.3)$$

Constraint (1.1) ensures that the total bandwidth used on each link does not exceed its capacity, and constraint (1.2) ensures that there exists a simple path  $p(v_i, v_j)$  from the source node  $v_i$  to each member node  $v_j \in X - \{v_i\}$  in tree  $T_i$ . Variable  $Y_{i,e}$  defined in (1.3) is a binary indicator variable, which denotes whether the link  $e$  is included in  $T_i$ . Since the problem includes the DSTP as a special case, efficient solutions must be heuristic in nature. In this paper, we propose two effective algorithms to solve the static version of the problem as stated in Eq. (1), and another algorithm for its dynamic extension.

The rest of the paper is organized as follows. In section 2, related works are briefly discussed. Our proposed algorithms are presented in section 3. Results from an extensive set of experiments to compare our proposals with others are discussed in section 4. Finally, section 5 concludes the paper.

## 2. RELATED WORKS

Strategies for solving the GMRP have decomposed the problem into two sub-problems. The first deals with constructing a single source multicast tree satisfying various QoS constraints, while the second deals with constructing a set of multicast trees by repeatedly invoking the single source algorithm so that a set of feasible trees with minimized costs is found for all group members. In general, the focus of the second sub-problem lies in resolving conflict between multiple trees competing for bandwidth on shared links. Solutions for the dynamic GMRP require a third component which deals with joining and leaving events in a dynamic environment.

### 2.1 Single Source Multicast Routing

The theoretical basis for the single source multicast routing is the Steiner tree problem. The formal definitions of undirected and directed versions of the Steiner tree problem are as follows:

#### *Undirected version (STP)*

Given a graph  $G = (V, E)$  with a cost function  $c$  on the edges, and a subset of vertices  $X \subseteq V$  (called terminal nodes or group members), the goal is to find a minimum cost tree that includes all the terminals in  $X$ . The tree may include nodes not in  $X$ .

#### *Directed version (DSTP)*

Given a directed graph  $G = (V, E)$ , a specified root  $v_r \in V$ , and a cost function  $c$  on the edges, and a subset of vertices  $X \subseteq V$ , the goal is to find a minimum cost directed tree rooted at  $v_r$  and spanning all the vertices in  $X$ . The tree may include vertices not in  $X$ , which are called Steiner points.

The Steiner tree problem has been known to be NP-complete [1] even when the graph is induced by points in the plane, and is MAX SNP-hard [2] for general graphs, which means that unless  $P = NP$ , there cannot exist a polynomial time approximation scheme for this problem. We briefly outline the related works for the undirected, directed, and constrained Steiner tree problems below.

#### 2.1.1 Undirected Steiner tree problem

The undirected Steiner tree problem has been very well studied [6-11]. Most proposals are heuristic in nature and approach the Steiner tree problem by approximating a Steiner minimum tree with a minimum spanning tree (MST) of the terminal set. The MST heuristic has an approximation ratio of 2, where the approximation ratio of an algorithm is an upper bound on the ratio between the achieved cost and the optimal cost. In general, the performances of most of the heuristic algorithms in the literature are reasonably good, but they lack sound worst-case approximation guarantees. We briefly describe the KMB and TM algorithms next.

In KMB [6], a complete subgraph of the network, called a closure graph, is first computed with the multicasting group members as nodes, and with edges between each pair of the nodes labeled by the shortest path cost calculated from the original graph. The

algorithm then uses Prim's algorithm [12] to build a minimum spanning tree out of the closure graph. Finally, to obtain the final multicast tree the algorithm replaces the edges of the spanning tree with the corresponding shortest paths from the original network. Another well-known Steiner tree heuristic is the TM algorithm [7]. The TM algorithm first computes the shortest paths from all nodes (including member and non-member nodes) to all the multicast group member nodes in the graph. Multicast tree construction is performed next. The multicast tree is initialized to contain the root node, which is any one of the terminal nodes in an undirected graph. The algorithm then iteratively connects the yet unconnected terminal node, which is nearest to any node on the partially constructed tree, to the tree via the shortest path computed previously. This process is repeated until all terminal nodes are included in the tree. The tree cost performance of TM is better than that of KMB because the TM algorithm not only finds the shortest paths between each pair of member nodes but also computes the shortest path from non-member nodes (i.e., relay nodes) to member nodes in the graph. Takahashi and Matsuyama [7] showed that the approximation ratio of the TM algorithm is at most 2.

More recent approximation algorithms achieve better approximation ratios at the cost of algorithmic complexity [8-10]. For example, the algorithm proposed by Karpinski and Zelikovsky [8] achieved an approximation ratio of 1.644 using a novel technique of choosing Steiner points depending on the possible deviation from the optimal solutions. However, due to its algorithmic complexity, it is difficult to incorporate into our overall algorithm for the GMRP. Therefore, we opt for the much simpler TM algorithm as the single source routing component in our algorithms.

### 2.1.2 Directed Steiner tree problem

In computer networks, network link bandwidths are asymmetric in practice, and cannot be modeled by undirected graphs. Therefore, the directed Steiner tree problem is of wide interest. It is hard to approximate the Steiner tree problem in a directed graph to a factor better than  $\ln k$ , in which  $k$  is the number of terminal nodes, unless  $NP \subseteq DTIME[n^{O(\log \log n)}]$  [13]. Charikar *et al.* [13] also proposed the first approximation algorithm based on solving a more general version of the directed Steiner tree problem **D-Steiner**( $k, v_r, X$ ), which is defined to construct a tree rooted at  $v_r$ , spanning any  $k$  terminals in  $X$  and of minimum possible cost. The proposed approximation algorithm is called  $A_i(k, v_r, X)$ , which gives an  $i(i-1)k^{1/i}$  approximation to **D-Steiner**( $k, v_r, X$ ). The subscript  $i$  controls the size of the search space, with larger values of  $i$  meaning larger spaces are explored. Briefly, the function  $A_i(k, v_r, X)$  repeatedly finds a node  $v_j \in V$  and the path  $p(v_r, v_j)$  from  $v_r$  to  $v_j$  with least cost, and a number  $k', 1 \leq k' \leq k$  such that the density of the tree  $T' = A_{i-1}(k', v_j, X) \cup p(v_r, v_j)$  is the least among all trees of this form. The algorithm combines the set of trees with minimum densities so that at least  $k$  terminals in  $X$  are spanned by the root. Note that  $i = \log k$ ,  $A_i(k, v_r, X)$  is an  $O(\log^2 k)$  approximation ratio algorithm that runs in quasi-polynomial time for the directed Steiner tree problem.

Based on Zelikovsky's and Charikar's works, Roos [14] proposed a simple  $\sqrt{k}$ -approximation algorithm that is particularly attractive due to its algorithmic simplicity. According to Charikar *et al.* [13], a simple  $\sqrt{k}$ -approximation solution can be obtained by choosing a star point, a best intermediate node  $v_{best}$  among all points in  $V$ ,

from which a natural multicast tree can be derived. That is, the directed Steiner tree is formed by connecting the root to  $v_{best}$  via the shortest path, and connecting  $v_{best}$  to all terminals via their respective shortest paths from  $v_{best}$ . Roos' proposal improves on this simple  $\sqrt{k}$ -approximation algorithm and thus is guaranteed to be within a factor  $\sqrt{k}$  of the optimal. We note that even though TM is designed to solve the undirected STP, it is obvious the algorithm can be naturally applied in the DSTP as well. It is of interest to know the relative performance between TM and Roos' algorithm, which we also investigate in the context of DSTP and GMRP.

Various heuristic algorithms without guaranteed theoretical cost performance analysis have also been proposed to solve the DSTP. For example, Shaikh and Shin [15] noted that Dijkstra's algorithm [16] also builds a multicast tree from the source to all other nodes in the network, and proposed the DDMC algorithm [15] based on the Dijkstra's algorithm. DDMC is a straightforward Dijkstra's algorithm with a modified cost  $c[j]$  function, which is associated with a node used in the relaxation process. In DDMC, the cost  $c[j]$  of connecting a new node  $v_j \notin T$  onto the partially constructed tree via link  $e = (v_i, v_j)$ ,  $v_i \in T$ , is defined as

$$c[j] = I(v_i)c[i] + c(e). \quad (2)$$

The indicator function  $I(v_i)$  is 0 if  $v_i$  is a terminal node, and is 1 otherwise. The motivation behind the cost function is the observation that multicast tree cost is different from the shortest path cost from the source node. When  $v_i$  is a terminal node, since the cost of the path to node  $v_i$  in the multicast tree is already included in the final multicast tree, the cost to node  $v_i$  does not need to doubly include the cost to node  $v_i$ . This cost function definition makes nodes which are close to terminal nodes targets to be preferentially connected onto the tree. The result is an efficient shortest path-based algorithm that can create multicast trees with tree costs competitive with other, more elaborate algorithms.

### 2.1.3 Constrained Steiner tree problem

The constrained Steiner tree (CST) problem seeks to find the least-cost multicast tree that satisfies a single or multiple constraints such as end-to-end delay, delay jitter, required bandwidth and packet loss rate. These constraints can be classified as *additive* or *non-additive*. For additive constraints such as delay, delay jitter and packet loss rate, the cumulative constraint weight value of an end-to-end path is given, exactly or approximately, by the *sum* of the individual link weight values along the path. In contrast, for non-additive constraint such as bandwidth, the cost of a path is determined by the value of the weight at the bottleneck link in the path. Theoretically, the single source constrained Steiner tree problem under bandwidth constraint alone is simpler to solve because a filtering step reduces the problem to a non-constrained case. Therefore, many heuristics have focused on the additive QoS constrained Steiner tree problem, including BSMA [17], CBF [18], KPP [19], CKMB [20], and QDMR [21]. Both the KPP and the CKMB algorithms are based on KMB [6] with an extension for the end-to-end delay constraint. CBF [18] is based on the Bellman-Ford algorithm, and has a time complexity that is exponential in the size of the network. Another more efficient class of heuristics for the CST problem is based on Dijkstra's shortest-path algorithm, including QDMR

[21], which is, in turn, based on DDMC [15]. Even though DDMC is an efficient algorithm to create low-cost multicast trees for the unconstrained multicast problem, it may generate trees in which some terminal nodes are connected to the source node via exceedingly long paths. In a delay-constrained environment, the long paths often violate the stated QoS constraints. QDMR replaces the indicator function  $I(v_i)$  in Eq. (2) by the ratio of the cumulative delay from the source to node  $v_i$  to the delay bound. In this manner, node  $v_i$  would be preferentially considered if node  $v_i$  is a terminal node, and the cumulative delay to it is small. QDMR is orders of magnitude faster than the KMB class of algorithms such as KPP, yet constructs trees with comparable, and sometimes slightly better multicast tree cost.

## 2.2 Static Group Multicast Routing

Variations of the GMRP have been investigated in the literature, including GMRP under bandwidth constraints [22-24], GMRP under delay constraints [25, 26], GMRP protocols [27], and the dynamic GMRP [28]. GMRP under QoS constraints has drawn attention with the increasing development of real-time multimedia applications, such as videoconference and on-line games. In this paper, we focus on the bandwidth constrained GMRP, which is useful for video-conferencing types of applications.

Even though bandwidth is a non-additive constraint, and it is easy to reduce the problem to a non-constrained case by a simple filtering step in the single source Steiner tree problem, it is hard to find a solution for GMRP under bandwidth constraint because of the conflict between the trees that compete for the bandwidth on shared links. Various proposals have been made such as Jia and Wang [24], and GTM [22]. Both algorithms are similar in design, with the former using KMB, and the latter using TM as the single source multicast routing algorithm. These algorithms construct the set of multicast trees by sequentially invoking a single source multicast tree algorithm over the member nodes. If any links in the network become *saturated*, i.e., overloaded, in the process, it implies that some multicast trees constructed previously which use these links have to release them and take alternative paths. The algorithms differ in the strategies used to select which previous trees to modify. Later, Low and Wang proposed the FTM algorithm that incorporates a preprocessing stage before actual tree construction in [23]. FTM first uses the breadth first search (BFS) procedure to find paths with maximum bandwidth capacity from each node to all other member nodes. After the BFS procedure, FTM uses the maximum bandwidth of the paths as the path cost for the TM algorithm to build a multicast tree. FTM achieves excellent success ratios for solving the GMRP. In comparison, since tree cost is not taken into consideration in the algorithm, the constructed trees usually have high costs. On the other hand, GTM [22], with its greedy link cost sensitive strategy to tree construction, achieves much lower success ratio than that of FTM, yet can construct trees with lower costs.

The delay constrained group multicast routing problem (DCGMRP), also under bandwidth limitation, has received the most attention in all additive QoS constrained GMRP. Recently, Song and Low proposed the heuristic algorithm DCGMRP\_IA [25, 26] for solving DCGMRP. DCGMRP\_IA is comprised of two stages. In the first stage, the algorithm constructs a set of shortest delay multicast trees for each member in the group using Dijkstra's algorithm [29] as the single source multicast tree construction algorithm,

and the link delay as the cost of the links. Since the initial solution may violate the bandwidth constraint, in the second stage, the algorithm modifies the initial solution in an iterative manner in an attempt to transform those multicast trees that violate the bandwidth constraint into ones that do not. In each iteration, the algorithm locates the busiest link,  $e$ , among all *saturated* links, those whose capacities are exceeded by their usage. The algorithm then determines the set of multicast trees that contain  $e$  as one of its branches. It then attempts to transform one tree  $T$  at a time into one that does not use  $e$  as one of its branches by re-routing  $T$ . Since re-routing previous multicast trees may create other saturated links, the general approach of DCGMRP\_IA is similar to GTM [22] in nature, and is not guaranteed to converge in all cases.

### 2.3 Dynamic Group Multicast Routing

In dynamic group multicast applications, nodes may join or leave a multicast group during the lifetime of the multicast session. A node currently in the multicast group may leave the group by issuing a leave request, and a node may join an ongoing session by issuing a join request. To support such a service, the routing protocol must dynamically update the multicast trees in response to changes in the group membership and ensure that the bandwidth constraint remains satisfied and the total cost of the set of trees is minimized. A typical example of applications requiring this type of service is remote teleconferencing, in which every participant is able to concurrently send and receive video information from all group participants. Since the number and identity of transmitting nodes in such an application changes over time, the ability to support dynamic group memberships is an important feature in the design of group multicast algorithms.

We first describe some previous work on single source dynamic multicast routing, including Waxman's greedy algorithms [30], ARIES [31], and VTDM [32]. Early research on single-source dynamic multicast routing started with the seminal work of Waxman [30]. He divided the DSTP into rearrangement (DSTP-R) and non-rearrangement (DSTP-N) types. For DSTP-N non-rearrangement dynamic multicast routing, Waxman investigated various greedy, polynomial time heuristic algorithms, generically named GRD. These algorithms solve the dynamic Steiner tree problem by attaching new member nodes via shortest path to a nearest node on the current multicast tree. Later, Waxman proposed the weighted version of the generic greedy algorithm, WGA [30], which uses Eq. (3) to decide which on-tree node  $v_j$  should be selected as the attachment node for a new request originating at  $v_i$ . The equation is defined as

$$W(v_i, v_j) = (1 - \omega) \cdot l(v_i, v_j) + \omega \cdot l(v_r, v_i), \quad (3)$$

where the parameter  $\omega \in [0, 0.5]$ ,  $v_r$  is the root node, and  $l(v_i, v_j)$  denotes the distance between the new member node  $v_i$  and a potential attachment node  $v_j$ . The algorithm chooses the node that minimizes the value of the equation as the attachment point. When  $\omega = 0$ , the nearest on-tree node will be selected, and the algorithm becomes GRD. The algorithm reduces to SPT when  $\omega = 0.5$ .

In DSTP-R, of which ARIES is an example, a number of re-arrangements of the multicast tree are allowed when each request is processed. ARIES, a re-arrangeable inexpensive edge-based Steiner algorithm proposed by Fred Bauer [31], makes the mini-

imum necessary modifications for each join and leave request. For each join request, ARIES joins the new member to the existing tree via the shortest path. For each leave request, ARIES deletes the node only if it is a leaf. The algorithm monitors the accumulated damage to the multicast tree within local regions of the tree as nodes are added and deleted, and triggers a rearrangement when the number of changes within a subtree exceeds a threshold.

Lin and Lai proposed the single source dynamic multicast routing algorithm VTDM [32]. The concept of Virtual Trunk (VT) is introduced in their paper. The main idea of the algorithm is that if we use a good static multicast routing algorithm to compute anew the multicast tree for each request in a dynamic multicast routing situation, some nodes and links may repeatedly appear in the series of multicast trees computed by the static multicast routing algorithm. If a dynamic multicast routing algorithm can take note of these nodes and links and use them whenever possible when constructing multicast trees dynamically, hopefully the costs of these dynamic multicast trees will be close to the costs of multicast trees constructed by a good static multicast routing algorithm.

For dynamic GMRP (DGMRP) there has been much less work. Recently, Low, Wang and Ng [28] proposed three heuristic algorithms,  $DGGM_A$ ,  $DGGM_B$ , and MBBPS, to generate low-cost multicast trees for DGMRP. The  $DGGM_A$  and  $DGGM_B$  algorithms (Dynamic Greedy Group Multicast Algorithm) use adaptations of the TM algorithm to construct a multicast tree for the newly joined node, and an adaptation of Waxman's greedy strategy to connect the new node to existing trees. Both algorithms make use of the concept of cheapest constrained path. A path  $p(v_i, v_j)$  between nodes  $v_i$  and  $v_j$  is said to be the cheapest constrained path if it is feasible and has the least cost derived from the links between nodes  $v_i$  and  $v_j$ . Two definitions of derived link costs are considered for each link. In  $DGGM_A$ , the derived cost of each edge is the same as the link cost of the edge, while  $DGGM_B$  defines the derived cost of a link to be equal to the link's cost divided by residual bandwidth of the link. However, since  $DGGM_A$  and  $DGGM_B$  cannot achieve high success rates, the authors also proposed a new algorithm MBBPS (maximum bottleneck bandwidth path selection algorithm) in the same paper. MBBPS searches along maximum bottleneck bandwidth paths instead of paths with low costs, thus increasing the success rate. MBBPS is based on an adaptation of the KMB [6] heuristic. The first step of MBBPS is similar to KMB and constructs a closure graph, which is a complete graph consisting of nodes in the terminal set only, and the cost of each edge in the closure graph being the cost of the cheapest constrained path between each pair of nodes. The bandwidth of each edge in the closure graph is equal to the bottleneck bandwidth of the corresponding cheapest constrained path. The second step of MBBPS for constructing a spanning tree out of the closure graph differs from that of KMB. Starting with a partial tree consisting of only the root node, in each iteration an edge with the most bandwidth between a member node in the partial tree and a node not in the partial tree is selected for inclusion into the partial tree. Once the tree is constructed, all edges in the tree are expanded into the constrained cheapest paths in the original network. The results from their empirical study show that MBBPS performs better in terms of cost and utilization of bandwidth compared to the other two algorithms,  $DGGM_A$  and  $DGGM_B$ , particularly when bandwidth is scarce in the network. We note that these results are counter-intuitive in the face of previous research [20] comparing TM with KMB. The studies [20] have shown that the TM heuristic [7] produces better overall cost perform-

ance than the KMB heuristic [6] for constructing single source multicast trees. In the following, we propose designs based on TM and Roos' algorithms, and show that our designs result in much better performance in terms of tree cost and success rate than MBBPS.

### 3. PROPOSED ALGORITHMS

In general, the heart of any solutions to the GMRP involves a strategy to construct a set of multicast trees based on a single-source multicast routing algorithm and another strategy to resolve the conflict between the trees that compete for the bandwidth on shared links. Note that the two issues are inter-related. Multicast tree construction may route different member nodes through shared links, thus creating competitive situations, and competition resolution involves tree re-routing. Previous work [22] adopted the direct, intuitive resolution approach by iteratively locating each saturated link, and finding alternative routes for trees that traverse the shared link, while our approach is more systematic. For static GMRP, our approach is based on the notion of critical pairs, and we propose a general **GMCP** framework with two instantiated algorithms, **GMCP-TM** and **GMCP-ROOS** (Group Multicasting via Critical Pairs using TM algorithm [7] and Roos' algorithm [14]). For dynamic GMRP, we propose the **DGMCP** (a dynamic version of **GMCP-TM**) algorithm.

We first describe our algorithms for solving the static GMRP. Our approach is based on inferring the reason why some links exceeds their link capacities, and then removes the cause from competition for shared links. Our observation is that there are critical members  $v_i$  whose locations in the network dictate a certain set of links be taken to reach some of the terminal nodes  $X'_i \subseteq X$ . If the bandwidths of these links are used up by others trees, no alternative paths exist to satisfy their requirements. We call the relationship between the member and these particular terminal nodes *critical*, and the tuple  $(v_i, X'_i)$  a *critical pair*. The strategy in both **GMCP-TM** and **GMCP-ROOS** is to route the critical pairs before others so that they have a priority in using the bandwidth on the shared links. The problem now reduces to finding those critical pairs that must be routed before others. We discuss our strategy in later sections. Experiments show that our proposal is more effective on success ratio than GTM [22], and the total tree cost of our algorithm is much lower than that of FTM [23].

For DGMRP, the dynamic group multicast routing problem, the goal is to construct a feasible set of trees while member nodes dynamically join and leave the multicast group by making incremental and localized changes to the existing set of multicast trees. Of the two types of requests, leaving requests can always be satisfied successfully since they involve release of network resources, while processing of join requests may fail due to insufficient network resource availability or algorithm failure to find existing resources. For join requests, our proposed algorithm **DGMCP** makes use of the TM algorithm as the component for creating a multicast tree for the newly joined member. However, since the TM algorithm is not a dynamic routing algorithm, Waxman's greedy algorithm is suitably modified as the mechanism for attaching a new member to existing multicast trees. It will be demonstrated later that our combination of the TM algorithm and Waxman strategies results in an effective strategy for dynamic group multicast routing.

### 3.1 The GMCP-TM and GMCP-ROOS Algorithms for GMRP

The central idea of the **GMCP** framework (with **GMCP-TM** and **GMCP-ROOS** as instances) is to sense conflicting demands and deal with them intelligently. We propose one approach to determine critical competition among demands in **GMCP**. Other strategies to sense conflicts are possible and will be investigated in the future, but our overall algorithm design should remain valid. As mentioned previously, our strategy is to find critical pairs, and construct multicast trees for those critical pairs before others. The **GMCP** algorithm is comprised of three phases, as shown in the pseudo-code in Fig. 1.

```

/* construct a set of multicast trees  $\{T_i \mid v_i \in X\}$  rooted at each member node */
GMCP ( $G, X$ ) { // GMCP-TM and GMCP-ROOS
  // Phase 1: create the initial trees
   $b'(e) = b(e), \forall e \in E;$ 
  for each  $v_i \in X$  {
     $T_i = \mathbf{TM}(v_i, X, G),$  //  $T_i = (V_i, E_i), T_i = \mathbf{ROOS}(v_i, X, G)$  for GMCP-ROOS
     $b'(e) = b'(e) - B_i, \forall e \in E_i;$  }
  // Phase 2: reserve the bandwidth
   $X'_i = \{v_j \mid v_j \in X, v_j \notin V_i\}, \forall v_i \in X;$ 
   $C = \{v_i \mid X'_i \neq \emptyset\};$ 
  if ( $C \neq \emptyset$ ) {
     $b'(e) = b(e), \forall e \in E;$  // restore original bandwidth
    for each  $v_i \in C$  {
       $T'_i = (V'_i, E'_i) = \mathbf{Dijkstra}(v_i, X'_i, G, f_{bw}(), c_i[], B_i);$  //  $T'_i = (V'_i, E'_i)$ 
       $b'(e) = b'(e) - B_i, \forall e \in E'_i;$  }
  // Phase 3: construct the final trees
  for each  $v_i \in X$  {
    if ( $X'_i \neq \emptyset$ ) {
       $b'(e) = b'(e) + B_i, \forall e \in E'_i;$  } // restore the reserved bandwidth
       $T_i = \mathbf{TM}(v_i, X, G);$  //  $T_i = \mathbf{ROOS}(v_i, X, G)$  for GMCP-ROOS
       $b'(e) = b'(e) - B_i, \forall e \in E_i;$  }
    } // end of the block to if ( $C \neq \emptyset$ )
  if ( $\exists T_i \wedge X \not\subset V_i$ ) return failure;
  else return  $T_i, \forall v_i \in X;$ 
}

```

Fig. 1. Pseudo code of **GMCP-TM** and **GMCP-ROOS** algorithms.

We first give a high-level outline of the structure of the algorithm. The goal of the first phase is to find the set of critical pairs. To achieve the goal, a preliminary set of multicast trees is constructed by repeatedly invoking a single source multicast routing algorithm, such as TM or Roos' algorithm, and reserve the required bandwidth for each member node  $v_i$ . For each tree  $T_i = (V_i, E_i)$  constructed for a root node  $v_i$ , let  $X'_i = \{v_j \mid v_j \in X, v_j \notin V_i\}$  be the member nodes not covered by the tree, and let  $C = \{v_i \mid X'_i \neq \emptyset, v_i \in X\}$  be the collection of all such root nodes. Then each  $v_i \in C$  and its associated  $X'_i$  form a

critical pair  $(v_i, X_i')$ . To connect these uncovered members to the trees, the previous strategy was to locate the saturated links that are required by other members competing for the same links, and resolve the competition by forcing some of the trees to release bandwidth on the shared links by finding alternative routes [22]. We consider this strategy too ad hoc, since there does not seem to be a natural order of the choice for the next saturated links for which to resolve the conflict. Furthermore, due to the inherent interactions between routing and conflict resolution, later actions may invalidate previous efforts, making the whole approach unmanageable. Our approach is to reserve the bandwidth for the critical pairs first, forcing less critical members to take other routes. This approach, though simple, yields substantially better results.

In the second phase of **GMCP**, starting from the original network, the bandwidth for each critical pair  $(v_i, X_i')$  is proactively reserved before final tree construction in the third phase. The existence of critical pairs in the multicast group means special attention must be paid to them. The goal of this phase is to reserve bandwidth for the critical pairs along links with *maximum* available bandwidth, so that the non-critical nodes will have residual bandwidth on all links no matter whether they are traveled by the critical pairs or not. This phase makes use of the **Dijkstra** algorithm directly as the single source multicast routing algorithm to construct multicast subtrees for the critical pairs, but employs a cost function that prefers links with more bandwidth to maximize feasibility. The shortest-path tree Dijkstra's algorithm is chosen as our tree construction algorithm instead of **TM** for two reasons. First, since the goal of this phase is to find a feasible tree along maximum bandwidth links for bandwidth reservation, and not cost-optimal trees for the critical pairs, a bandwidth sensitive version of Dijkstra's algorithm is more suitable for our purposes. Secondly, had **TM** been applied instead, it would find essentially the same set of trees as those in the first phase, when it was first invoked, and therefore it would fail to construct trees for the critical pairs *again*, just as in the first phase of **GMCP**. Therefore, **TM** is not applied for this purpose.

In the third phase of **GMCP**, the single source routing algorithm is invoked again to actually construct final multicast trees for all member nodes  $v_i$ , each time using the *complete* member node set  $X$  as the terminal nodes. The key idea for this phase of our algorithm is that, for a node  $v_i$ , if a critical pair  $(v_i, X_i')$  has been found before, we first restore the bandwidth reserved for the critical pair on links in  $T_i'$ , which was constructed in the second phase, before invoking the single source **TM** or Roos' algorithm. Through this operation, the invocation is guaranteed to be able to find a feasible multicast sub-tree for the critical pair, yet it has the freedom to use other links if alternative paths are preferable. At the end of the third phase, if there is any tree that does not cover all members in the multicast group, the complete algorithm fails.

We note that different multicast tree construction sequences may produce different critical pairs. We implemented a set of simulations to investigate the performance difference resulting from different tree construction sequences. Details of the simulations and their results are summarized in [33]. To summarize, though the critical pairs discovered may be different when the multicast trees are constructed in a different sequence, that does not significantly affect the performance of **GMCP** for the following reasons. First, consider the probability of success of the algorithm. We note that when the capacity of the network is sufficient to admit any feasible solution, since **GMCP** usually has a high probability of success, as will be validated in later experiments, **GMCP** will achieve

similar probability of success even if the sequence of tree construction is different. With respect to the tree cost performance, there might be slight variations when different sequences are presented to the **GMCP** algorithm. However, when the number of experiments performed is large, the average tree cost should be comparable across different tree construction sequences. In later experiments, we show our tree cost results together with the confidence intervals to illustrate the tree cost performance variation behavior. Furthermore, in [33], we perform extensive experiments to show that different sequences do not have a dramatic impact on the tree cost performance of the algorithms.

### 3.1.1 The GMCP-TM algorithm

We now describe in more detail the pseudo-code of the three phases of **GMCP-TM**, which is shown in Fig. 1. Variable  $b'(e)$  stores the residual capacity of each link  $e \in E$ . The first phase simply invokes **TM** repeatedly to construct a multicast tree  $T_i = (V_i, E_i)$  for each member node  $v_i \in X$ , and reserves the bandwidth for the returned tree after each invocation. The pseudo-code for **TM** is shown in Fig. 2. **TM** first computes the shortest path from each node  $v_i \in V$  in the network to all nodes in the multicast group using the **Dijkstra** algorithm. Variable  $c_i[]$  is an output variable for the path cost from node  $v_i$  to all other nodes after each invocation of **Dijkstra**. The second step of **TM** does the actual construction of the multicast tree, and begins by initializing the tree to contain the source node only, i.e.  $V' = \{v_r\}$ ; the other members are in  $X$ . It then iteratively selects the path  $p(v_i, v_j)$  (where  $v_i \in V'$  and  $v_j \in X - V'$ ) that has the least cost among all paths. That is, it selects the unconnected group member  $v_j \in X$  which is nearest to any node on the partially constructed tree in  $V'$ , and includes it in the tree. With single source shortest path information stored in  $c_i[]$  for each node  $v_i$ , this step can be easily computed. The path selection process is repeated until all group member nodes are included in the tree. The pseudo-code for **Dijkstra** is shown in Fig. 3. Note that one important parameter for **Dijkstra** is the evaluation function used during the search. Shown in the figure are three functions,  $f_{cost}()$ ,  $f_{bw}()$ , and  $f_{cost\_bw}()$ , used in various phases of **GMCP-TM**. The cost evaluation function  $f_{cost}()$  is used in the first phase of the **TM** algorithm to compute the cheapest paths  $p$  under the bandwidth constraint for all pairs of nodes  $v_i \in V'$  and  $v_j \in X - V'$ , where  $T' = (V', E')$  is the partially constructed tree. The function,  $f_{cost}()$  utilizes Eq. (4) to calculate the estimated cost  $c[j]$  for reaching node  $v_j$  from the root node via node  $v_i$  as follows.

$$c[j] = \begin{cases} c[i] + c(e), & \text{if } c[i] + c(e) < c[j] \\ c[j], & \text{otherwise} \end{cases}, \quad (4)$$

where  $e = (v_i, v_j)$ , and  $c(e)$  denotes the cost of link  $e \in E$ . The rationale for the choice is that, when network resources are abundant, the cost function in Eq. (4) enables the creation of multicast trees with good tree cost performance. In this case, the second and third phases of **GMCP-TM** are not relevant and will not be executed. On the other hand, when resources are scarce, since our purpose is to locate the critical pairs in this phase, this cost is sufficient for the purpose, relegating the task of improving the chances of success to the second phase of **GMCP-TM**. The first and second phases of **GMCP-TM** working

```

/* construct a multicast tree rooted at  $v_r$  and reach out to all other members in  $X$  */
TM ( $v_r, X, G$ ) {
  Dijkstra ( $v_r, X, G, f_{cost}(), c_r[], B_r$ ),  $\forall v_i \in V$ ; //  $B_r$  is the required bandwidth
  // construct multicast tree  $T'$ 
   $V' = \{v_r\}$ ;  $E' = \emptyset$ ;
  while ( $V' - X \neq \emptyset \wedge \exists \text{path from } V' \text{ to } X - V'$ ) {
    Find a path  $p$  from  $V'$  to  $X - V'$ , s.t.  $p = \arg \min_{v_i \in V', v_j \in X - V'} c_i[j]$ ;
     $V' = V' \cup \{v_i \mid v_i \in p\}$ ;
     $E' = E' \cup \{e \mid e \in p\}$ ;
  } // end of the while loop
  return  $T' = (V', E')$ ;
}

```

Fig. 2. Pseudo code of **TM** algorithm.

```

/* construct a tree rooted at  $v_r$  and reach out to all or some of nodes in  $X'$  */
Dijkstra ( $v_r, X', G', f(), c_r[], B$ ) { // and Reverse_Dijkstra function
   $b'[i] = 0, c_r[i] = \infty, \forall v_i \in V$ ;
   $b'[i] = \infty, c_r[r] = 0, Q = \emptyset, V' = E' = \emptyset$ ;
   $Q.push(c_r[r])$ ;
  while ( $Q \neq \emptyset \vee X' \not\subset V'$ ) {
     $c_r[i] = Q.ExtractMin()$ ; // extract  $v_i$  with the connected link  $e$ 
     $V' = V' \cup \{v_i\}$ ;  $E' = E' \cup \{e\}$ ;
    for each link  $e = (v_i, v_j) \vee v_j \notin V'$  { //  $e = (v_j, v_i)$  for Reverse_Dijkstra
      if ( $b'(e) \geq B$ ) //  $B$  is the required bandwidth
        Relax ( $e = (v_i, v_j), f()$ ); } //  $e = (v_j, v_i)$  for Reverse_Dijkstra
    } // end of the while loop
  Prune off all leaves not in  $X'$ .
  return  $T' = (V', E')$ ;
}

Relax ( $e, f()$ ) { // where  $e = (v_i, v_j), f()$  is  $f_{cost}(), f_{bw}()$  or  $f_{cost\_bw}()$ 
  if ( $f(e) < c_r[j]$ ) {
     $c_r[j] = f(e)$ ; // Update  $c_r[j]$ 
    if ( $v_i \in X'$ )  $b'[j] = b'(e)$ ;
    else  $b'[j] = \min(b'[i], b'(e))$ ;
     $Q.push(c_r[j])$ ; // push  $v_j$  with link  $e = (v_i, v_j)$ 
  } // end of the block to if ( $f(e) < c_r[j]$ )
}

 $f_{cost}(e)$  { // for low cost, where  $e = (v_i, v_j)$ 
  return  $c_r[i] + c(e)$ ;
}

 $f_{bw}(e)$  { // for maximum bandwidth, where  $e = (v_i, v_j)$ 
  return  $1/\min(b'[i], b'(e))$ ;
}

 $f_{cost\_bw}(e)$  { // for low cost with more bandwidth, where  $e = (v_i, v_j)$ 
  if ( $v_i \in X'$ ) return  $c(e)/b'(e)$ ;
  else return  $(c_r[i] + c(e))/\min(b'[i], b'(e))$ ;
}

```

Fig. 3. Pseudo code of **Dijkstra** algorithm.

together strike a nice balance between cost optimization and the probability of success. One final note on the **Dijkstra** algorithm is that the algorithm explores only paths that have sufficient bandwidth by checking the condition  $b'(e) \geq B$  in the inner for loop. Therefore, it may return a partial shortest-path tree, unlike the original Dijkstra's algorithm, which always returns a tree that can reach all other nodes in the graph.

At the end of the first phase of **GMCP-TM**, the set of constructed multicast tree may not span all member nodes in the group due to insufficient bandwidth during tree construction. For each node  $v_i$ , let  $X'_i = \{v_j \mid v_j \in X, v_j \notin V_i\}$  be the member nodes not covered by tree  $T_i = (V_i, E_i)$ , and  $C = \{v_i \mid X'_i \neq \emptyset, v_i \in X\}$ . Then each  $v_i \in C$  and its associated  $X'_i$  form a critical pair  $(v_i, X'_i)$ .

In the second phase of **GMCP-TM**, bandwidth for each critical pair  $(v_i, X'_i)$  is proactively reserved before final tree construction. To improve the chances of success of later tree construction, we would like to make the reservations along links with maximum available bandwidth. The strategy is to use the **Dijkstra** algorithm, using  $f_{bw}(\cdot)$  as its evaluation function to construct the reservation tree  $T'_i = (V'_i, E'_i)$  for each critical pair  $(v_i, X'_i)$ . The **Dijkstra** algorithm is chosen as the tree construction algorithm instead of **TM** because the goal of this phase is to reserve bandwidth along links with maximum link bandwidth. Therefore, the bandwidth sensitive version of Dijkstra's algorithm is more suitable for our purpose. It is important to note the cost function  $f_{bw}(\cdot)$  in **Dijkstra** utilizes Eq. (5) to calculate the estimated cost  $c[j]$  for reaching node  $v_j$  from root node  $v_i$  via node  $v_i$  as follows.

$$c[j] = \begin{cases} \frac{1}{\min(b'[i], b'(e))}, & \text{if } \min(b'[i], b'(e)) > \varepsilon \\ \infty, & \text{otherwise} \end{cases} \quad (5)$$

The definition is concerned with link bandwidth only, and enables the second phase of the algorithm to search along paths with more bandwidth, regardless of actual link costs. In Eq. (5),  $e = (v_i, v_j) \in E$ ,  $b'[i]$  denotes the bottleneck bandwidth of the path from the root node to node  $v_i$  along the path in the tree, and  $b'(e)$  stores the residual available link bandwidth on link  $e$ . As seen from the equation, the cost is the inverse of the bottleneck bandwidth between root and node  $v_j$ . Therefore, the expression  $\min(b'[i], b'(e))$  denotes the bottleneck bandwidth of the path from the root node to node  $v_j$  via node  $v_i$ . The parameter  $\varepsilon$  is a threshold used to control the minimum amount of unused bandwidth on the links, which can be used by, for example, other best-effort traffic. To simplify presentation,  $\varepsilon$  is chosen to be 0 in later experiments. After the second **GMCP-TM** phase, bandwidth needed by the critical pairs has been proactively reserved in the network; therefore, later routing actions must consider other routes when routing for non-critical pairs.

In the third phase of **GMCP-TM**, referring to Fig. 1 again, **TM** is invoked again to actually construct final multicast trees for all member nodes, each time using the *complete* member node set  $X$  as destination nodes. The main idea in our algorithm is that, before invoking the **TM** algorithm for node  $v_i$ , if  $(v_i, X'_i)$  is critical, we first restore the bandwidth reserved for the critical pair on links in  $T'_i$  using the statement  $b'(e) = b'(e) + B_i, \forall e \in E'_i$ . Then, the invocation of the **TM** algorithm again, with  $v_i$  as root, is guaranteed to succeed in finding, at minimum, a feasible multicast sub-tree for the critical pair.

It may utilize other links if alternative paths are discovered during the search. At the end of the last phase, if any tree not covering all members in the multicast group exists, the complete algorithm fails.

The complexity of the **Dijkstra** algorithm is  $O(|N|\log|N| + |E|)$  if the network graph uses an adjacency list representation and Fibonacci Heap [34] is used as the priority queue implementation, where  $|N|$  is the number of nodes in the network, and  $|E|$  is the number of links. The complexity of **TM** is  $O(|N|(|N|\log|N| + |E|))$  because the dominant computation in **TM** is the  $|N|$  times **Dijkstra** is invoked to construct the shortest path tree for each node. The time complexity of the **GMCP-TM** algorithm is  $O(|X|(2|N| + 1)(|N|\log|N| + |E|))$ , since **GMCP-TM** invokes the **TM** algorithm  $|X|$  times in the first phase and last phase respectively, and  $|X|$  times **Dijkstra** in the worst case in the second phase. Therefore, the complexity of **GMCP-TM** is around  $O(|N|^3\log|N|)$  when the multicast group contains all nodes in the graph.

### 3.1.2 The GMCP-ROOS algorithm

Our critical pair approach allows different algorithms to serve as the single source multicast routing component. In this subsection, we illustrate the flexibility of our approach by replacing the **TM** algorithm with the algorithm proposed by Roos [14]. Roos' algorithm is selected because of its sound theoretical performance guarantee and improved execution speed compared with Charikar's proposal. The version of **GMCP** in which Roos' algorithm serves as the single source routing algorithm is called **GMCP-ROOS**. Roos' algorithm is based on the following basic scheme for constructing a multicast tree. The basic scheme computes the *best* intermediate node  $v_{best}$ , called a star node, and builds a multicast tree by first connecting the root to  $v_{best}$  via their shortest path, and then connecting  $v_{best}$  to all the other terminals via their respective shortest paths. The best star node is determined by systematically evaluating the cost of all the multicast trees, each naturally created as in the basic scheme with a node in the network as the candidate star node, and finally selecting the one tree with the least cost. According to Charikar [13], the result after one application of the basic scheme is already  $\sqrt{k}$ -optimal. Roos' algorithm improves on this basic scheme by performing further steps to reduce the cost of the constructed multicast tree. Roos improves upon the basic scheme by, after each round of the basic scheme, incorporating into the final multicast tree the path from the root node, via the star node, to the destination  $v_n$  which is closest to the star node in the current tree. Then the algorithm modifies the network graph by creating a virtual link from the root node to each node in that path. The cost and bandwidth are set to be zero and infinity, respectively. The modifications essentially expand the root node into a set that includes the original root node and all nodes on the path, in a spirit very similar to what is performed in Eq. (2) used in the DDMC algorithm. The algorithm then repeats the basic scheme with the modified graph as input until all terminals are exhausted. The performance of Roos' algorithm is guaranteed to be within a factor  $\sqrt{k}$  of the optimal because it improves on an  $\sqrt{k}$ -approximation algorithm. For more details on Roos' algorithm, please consult [14]. Paragraph the pseudo-code of Roos' algorithm is shown in Fig. 4. The algorithm first computes the shortest paths from each node  $v_i \in V$  to all the other nodes in the multicast group using the **Dijkstra** algorithm. Variable  $c_{i[j]}$  is an

```

/* construct a multicast tree rooted at  $v_r$  and reach out to all other members in  $X$  */
ROOS ( $v_r, X, G$ ) {
  Dijkstra ( $v_r, X, G, f_{cost}(\cdot), c_i[], B_r$ ),  $\forall v_i \in V$ ; //  $B_r$  is the required bandwidth
  // actual construction of multicast tree  $T'$ 
   $V' = \{v_r\}$ ;  $E' = \emptyset$ ;
  while ( $|X - V'| > 1$ ) {
    Dijkstra ( $v_r, V, G, f_{cost}(\cdot), c_r[], B_r$ ); //  $B_r$  is the required bandwidth
     $v_{best} = NIL$ ,  $d_{best} = \infty$ ;
    // find the best star node for this iteration
    for each node  $v_i \in V$  {
       $temp = c_r[i] + \sum_{v_j \in X, v_j \notin V'} c_i[j]$ ;
      if ( $temp < d_{best}$ ) {
         $v_{best} = v_i$ ;  $d_{best} = temp$ ; }
    } // end of the for loop
    // Let  $v_n$  be the nearest destination to  $v_{best}$ .
     $v_n = \arg \min_i \{c_{best}[i], v_i \in X, v_i \notin V'\}$ ;
     $p(v_r, v_n) = p(v_r, v_{best}) \cup p(v_{best}, v_n)$ ; // but not including the virtual links;
    add zero-cost, unlimited bandwidth virtual links from  $v_r$  to  $v_i, \forall v_i \in p(v_r, v_n)$ ;
     $T' = T' \cup p(v_r, v_n)$ ;
  } // end of the while loop
  Let  $v_n$  be the only node left in  $X - V'$ .
   $T' = T' \cup p(v_{best}, v_n)$ ;
  Remove the added zero-cost links in  $T'$  and  $G$ .
  return  $T'$ ;
}

```

Fig. 4. Pseudo code of **ROOS** algorithm.

output which contains the path cost from node  $v_i$  to all other nodes  $v_j$  after each invocation of **Dijkstra**. In the second step of Roos' algorithm, construction of the multicast tree begins by initializing the tree to contain the source node only, i.e.  $V' = \{v_r\}$ , and the other group members are stored in set  $X$ . It then computes the best star node  $v_{best}$  in the input graph with the for loop. Now let  $v_n$  be the nearest yet unconnected terminal node with the least path cost to  $v_{best}$ . Let  $p(v_r, v_n)$  denote the path formed by the sub-paths  $p(v_r, v_{best})$  and  $p(v_{best}, v_n)$  except for the virtual links. The algorithm then adds virtual links from  $v_r$  to  $v_i, \forall v_i \in p(v_r, v_n)$  with zero-cost and unlimited bandwidth to the original network graph. The final multicast tree  $T'$  is then updated by including  $p(v_r, v_n)$  as one of its branches  $T' = T' \cup p(v_r, v_n)$ . Then the process continues until all terminals are exhausted. The complexity of Roos' algorithm is  $O(2|N|(|N|\log|N| + |E|))$  because the dominant computation in Roos' algorithm is the  $|N|$  times **Dijkstra** invocation to construct the shortest-path tree for each node and the  $|N|$  times **Dijkstra** invocation to construct the shortest-path tree for the root node in the while loop. The time complexity of the **GMCP-ROOS** algorithm is  $O(|X|(4|N| + 1)(|N|\log|N| + |E|))$  since **GMCP-ROOS** invokes Roos' algorithm  $|X|$  times in its first and last phases, and at most  $|X|$  times the

**Dijkstra** algorithm in the second phase. The complexity of **GMCP-ROOS** is around  $O(|N|^3 \log|N|)$  when the multicast group includes all nodes in the graph.

**3.1.3 An example**

We use the example shown in Figs. 5, 6 and 7 to illustrate the **GMCP-TM** algorithm. The result for the **GMCP-ROOS** algorithm is similar and is omitted. Fig. 5 (a) is the input graph in which nodes 1, 2 and 3 are member nodes in the multicast group. For each link on the network graph, the attached attributes  $(b, c)$  denote available bandwidth and cost associated with the link. Assume the bandwidth requirement is one unit for all members in the group. In the first phase of **GMCP-TM**, taking the graph in Fig. 5 (a) as input, the **TM** algorithm constructs the multicast tree shown in Fig. 5 (d) root at 1. One unit of bandwidth on the links in Fig. 5 (d) is reserved and the available bandwidth of each link in the original network is then updated. The resulting network is shown in Fig. 5 (b). The tree shown in Fig. 5 (e) is generated in a similar fashion for node 2 by the **TM** algorithm. After bandwidth reservation, the resulting network is shown in Fig. 5 (c). Applying the **TM** algorithm again for node 3, it is found that no member nodes in the multicast group can be connected except itself, since no paths with enough bandwidth exist for routing node 3 to nodes 1 and 2. Therefore, at the end of phase 1 **GMCP-TM**, it is discovered that  $(3, \{1, 2\})$  forms a critical pair since the tree rooted at nodes 3 does not cover member nodes 1 and 2 in the multicast group.

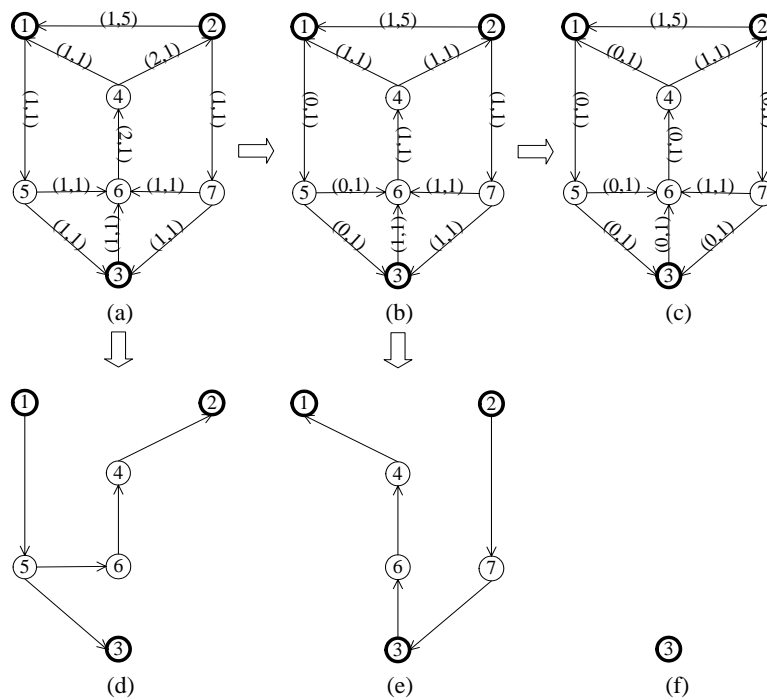


Fig. 5. Example illustrating phase 1 of **GMCP-TM**.

Phase 2 of **GMCP-TM** starts from the original network graph again, reproduced in Fig. 6 (a). To satisfy the requirements of the critical pairs first,  $(3, \{1, 2\})$ , **Dijkstra** is invoked to construct a maximum bottleneck bandwidth tree, the result of which is shown in Fig. 6 (b). One unit of bandwidth is then reserved for this tree. The resulting network is shown in Fig. 7 (a).

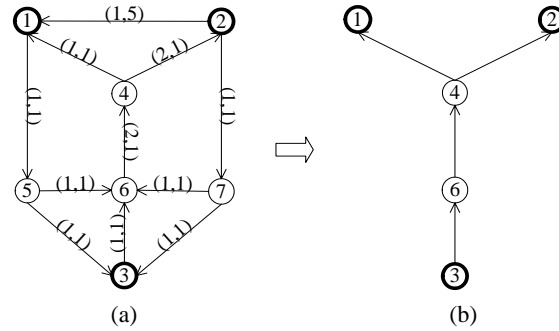


Fig. 6. Example illustrating phase 2 of **GMCP-TM**.

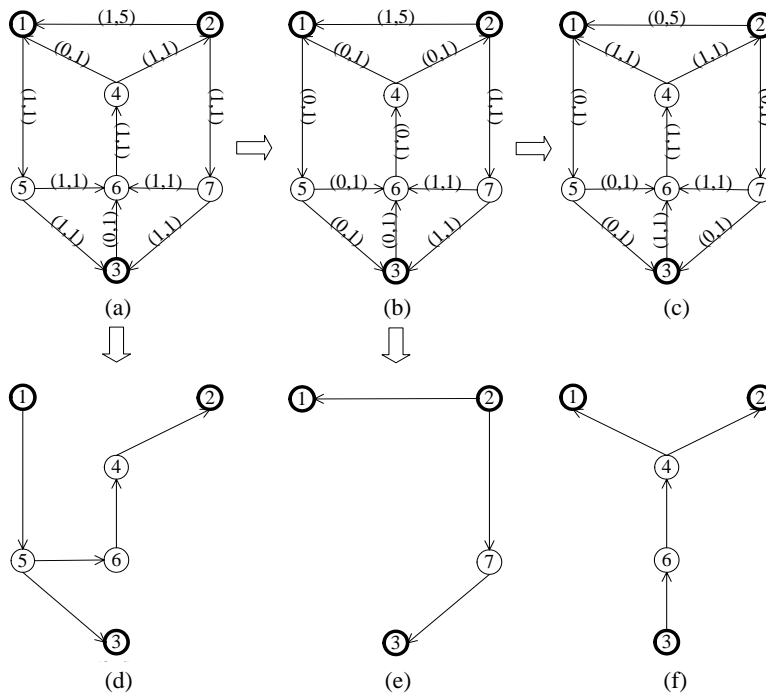


Fig. 7. Example illustrating phase 3 of **GMCP-TM**.

Phase 3 of **GMCP-TM** starts from the network shown in Fig. 7 (a), which is the result from phase 2. The **TM** algorithm is invoked again for each member of the group. For

node 1, the constructed multicast tree generated by the **TM** algorithm is shown in Fig. 7 (d). One unit of bandwidth is reserved for the tree, resulting in the tree shown in Fig. 7 (b). The tree shown in Fig. 5 (e) is generated in a similar fashion for node 2 by **TM** algorithm. The resulting network is shown in Fig. 5 (c). To construct the tree for node 3, one unit of bandwidth is first added to the links in the set  $\{(3, 6), (6, 4), (4, 1), (4, 2)\}$ , which corresponds to the reservation made in phase 2 of **GMCP-TM**. The updated network is shown in Fig. 7 (c). The **TM** algorithm is then invoked and generates the multicast shown tree in Fig. 7 (f) for node 3. **GMCP-TM** has now succeeded in constructing a set of multicast trees for the example.

### 3.2 The **DGMCP** Algorithm for **DGMRP**

In this section, we propose an algorithm, **DGMCP**, a dynamic algorithm to construct a feasible set of trees as member nodes dynamically join and leave the multicast group by making incremental and localized changes to the existing set of multicast trees. In contrast to the static **GMRP** case, the design of the algorithm does not make use of the notion of critical pairs for the following reasons. First, it is infeasible to maintain the critical pair data structure consistently on all network nodes and incrementally in a dynamic environment. Second, in the absence of shared critical pair data structure, complete re-computation of the **GMCP-TM** algorithm each time a node joins or leaves the group is required, which we want to avoid in a dynamic environment. Much research has been performed on ways to avoid re-performing the computation over the complete network graph. The technique called pre-computation in the literature has been proposed specifically to address this problem [35-37]. Third, the existing set of multicast trees should not be destroyed completely as a new member joins the group, and so making incremental changes to the existing set of multicast trees for the new multicast group is more reasonable. The design of **DGMCP** attempts to solve the problem in an efficient, incremental manner, while striving for a minimal increase in the final multicast tree cost.

Of the two types of requests, leaving requests can always be satisfied successfully since they involve releasing network resources, while processing of joining requests may fail due to insufficient network resource availability or failure of the algorithm to find existing resources. For joining requests, two problems must be solved: creating a multicast tree for the newly joined member and attaching the new member to other existing multicast trees. **DGMCP** makes use of the **TM** algorithm as the component for creating a multicast tree for the newly joined member. However, since the **TM** algorithm is not a dynamic routing algorithm, an approach must be devised to attach a new member to existing multicast trees. One strategy is to modify the **TM** algorithm to be dynamic. This strategy is not appropriate for the following reasons. Recall the first step of the **TM** algorithm already computes the shortest paths from all nodes to all members of group  $X$ . In a dynamic environment, the results of the computation must be saved for future request processing to avoid complete re-computation of the first step. Consider the case when a new node  $v_r$  attempts to join an existing group  $X$ . Since the shortest paths from  $v_r$  to  $X$  are already computed, the dynamic version of **TM** may proceed as usual to include the node in the multicast tree in the second step. Then, since group  $X$  has changed, we must again perform the first step of the **TM** algorithm again, to obtain a consistent state for future request processing. This overall strategy has the significant shortcoming of having rela-

tively expensive computation overhead in a dynamic environment since it involves invoking the **Dijkstra** algorithm repeatedly for all nodes in the network. Therefore, a different strategy is adopted in **DGMCP** to attach a new node to other existing trees. Our approach is to find a cheapest feasible path from each multicast tree to the new member node, similar to Waxman's greedy strategy. Even though  $DGGM_A$  and  $DGGM_B$  adopt similar strategies to attach new members, however, we will show later that our combined algorithmic architecture using the different cost functions results in a much higher success rate and lower tree costs.

We next describe the **DGMCP**'s strategy at a high level for handling joining requests when a new node  $v_r \notin X$  attempts to join the multicast group. The strategy is comprised of the following two components:

- (1) Add  $v_r$  into the group,  $X = X \cup \{v_r\}$ . A new multicast tree  $T_r$  with root at  $v_r$  is then constructed, using the **TM** algorithm, and the bandwidth on each link in  $T_r$  is reserved.
- (2) Include  $v_r$  into other already existing multicast trees. For each node  $v_s \in X - \{v_r\}$ , with associated multicast tree  $T_s = (V_s, E_s)$ , we distinguish between the following two cases:
  - (a)  $v_r \in V_s$ : That is,  $v_r$  is already a node on  $T_s$ . No further change is necessary.
  - (b)  $v_r \notin V_s$ : In this case, **DGMCP** must find a cheapest constrained path from the new member node to any node on the current tree by invoking the **Reverse\_Dijkstra** algorithm [38]. The traditional Dijkstra algorithm starts from a source node and finds shortest paths to all the other nodes. The **Reverse\_Dijkstra** algorithm starts from a *sink* node and finds the shortest paths from all the other nodes to it. If a path is found, the bandwidth on each edge on the path is decreased by  $B_s$  units.

Finally, a join request for node  $v_r$  will be rejected if any multicast tree constructed above fails to span all nodes of the new group.

The pseudo-code for **DGMCP** is shown in Fig. 8. The code for join request handling is called **DGMCP\_join**, and corresponds precisely with the description above. We next explain the pseudo-code shown in Fig. 3 for **Reverse\_Dijkstra**. The only difference between **Reverse\_Dijkstra** and the traditional Dijkstra algorithm is that the **Reverse\_Dijkstra** algorithm examines incoming link, rather than outgoing link, during the relaxation operation at a node. Otherwise, the code is exactly the same as **Dijkstra** shown in Fig. 3. Variable  $b'[i]$  denotes the bottleneck bandwidth of the path from node  $v_i$  to the sink node, and  $b'(e)$  stores the residual available link bandwidth for each link  $e \in E$ . More importantly, the critical element in our design is the evaluation function  $f_{cost\_bw}(\cdot)$  used in **Reverse\_Dijkstra**, which calculates the cost to reach the sink node from node  $v_j$  via node  $v_i$ , and is defined by

$$c[j] = \begin{cases} \frac{c(e)}{b'(e)}, & \text{if } v_i \in X \\ \frac{c[i] + c(e)}{\min(b'[i], b'(e))}, & \text{otherwise} \end{cases}, \quad (6)$$

```

DGMCP_init ( $G$ ) {
   $b'(e) = b(e), \forall e \in E;$  // initial available bandwidth
   $V_i = E_i = \emptyset, \forall v_i \in V;$ 
   $X = \emptyset;$ 
}
DGMCP_join ( $v_r$ ) {
   $X = X \cup \{v_r\};$ 
   $T_r = \mathbf{TM}(v_r, X, G);$  //  $T_r = (V_r, E_r)$ 
   $b'(e) = b'(e) - B_r, \forall e \in E_r;$ 
  for each  $v_s \in X - \{v_r\}$  {
    if ( $v_r \notin V_s$ ) { //  $T_s = (V_s, E_s)$  rooted at  $v_s$ 
      Reverse_Dijkstra ( $v_r, V_s, G, f_{cost\_bw}(), c_r[], B_s$ );
      Find a path  $p$  from  $v_i \in V_s$  to  $v_r$ , s.t.  $c_r[i] = \min_{v_j \in V_s} c_r[j].$ 
       $b'(e) = b'(e) - B_s, \forall e \in p$ 
       $V_s = V_s \cup \{\text{all nodes in } p \text{ except } v_i\};$ 
       $E_s = E_s \cup \{\text{all links in } p\};$ 
    } // end of the for loop
  }
  if ( $\exists T_i, v_i \wedge X \not\subset V_i$ , for all  $v_i \in X$ ) {
    Send a leaving event with the node  $v_r$ ; // or call DGMCP_leave ( $v_r$ ) directly
    return failure; } // reject  $v_r$  to join
  else return success;
}
DGMCP_leave ( $v_r$ ) {
   $X = X - \{v_r\};$ 
   $b'(e) = b'(e) + B_r, \forall e \in E_r;$  //  $T_r = (V_r, E_r)$ 
   $V_r = E_r = \emptyset;$ 
  for each  $v_s \in X$  {
    if ( $v_r$  is a leaf in  $T_s$ ) {
      Prune the path  $p$  segment from  $v_r$  to the first branching point in  $T_s$ .
       $b'(e) = b'(e) + B_s, \forall e \in p;$ 
    } // end of the for loop
  }
}

```

Fig. 8. Pseudo code of **DGMCP** algorithm.

where  $e = (v_i, v_j)$ . First, we note that Eq. (6) involves both link cost and link bandwidth simultaneously, unlike **GMCP-TM**, which considers either link cost or link bandwidth, but not both simultaneously. This is because in the dynamic GMRP case, only one invocation of **Reverse\_Dijkstra** is performed. Therefore, its evaluation function must take both link cost and link bandwidth into consideration. Second, the definition of Eq. (6) is motivated by DDMC [15]. Recall that  $b'[i]$  denotes the bottleneck bandwidth of the path from an on-tree node  $v_i$  to the sink node. If  $v_i$  is already one of the multicast member nodes, the cost associated with  $c[i]$  is disregarded, and  $c[j]$  is calculated by weighting the cost of  $c(e)$  with the inverse of the link's available bandwidth  $b'(e)$ . If  $v_i$  is not one of the multicast member nodes,  $c[j]$  is calculated by weighting the combined cost  $c[i] + c(e)$  with the inverse of the bottleneck bandwidth  $\min(b'[i], b'(e))$ . The rationale behind the

weights is that, for two paths with the same cost, the path with more available bandwidth has a higher priority. For two paths with same available bandwidth, the path with lower cost has higher priority in constructing the multicast tree. Furthermore, a node  $v_j$ , neighboring  $v_i$ , would be preferentially considered if node  $v_i$  is a member node, since node  $v_i$  must be included in the multicast tree anyway, and its cost should not be counted twice when routing node  $v_j$ .

We next describe how leave requests are handled when a node  $v_r \in X$  decides to end its participation in the multicast session. We perform the following two steps:

- (1) Update  $X = X - \{v_r\}$  and release all the resources that are allocated for tree  $T_r$  rooted at  $v_r$ . That is, the residual bandwidth of each link on  $T_r$  will be increased by  $B_r$  units.
- (2) Exclude  $v_r$  from other multicast trees. For each tree  $T_s$  where  $v_s \in X$ , we distinguish between the following two cases:
  - (a) If  $v_r$  is a leaf node in  $T_s$ , the path segment starting from  $v_r$  towards the source  $v_s$  and ending at the first branching point on the tree is pruned. The residual bandwidth of each pruned link will be increased by  $B_s$  units.
  - (b) If  $v_r$  is not a leaf node in  $T_s$ , no action needs to be taken.

#### 4. PERFORMANCE EVALUATION

We use Waxman graphs [30] as the directed network model in our simulations to ensure that the routing algorithms are compared fairly with previous work. The nodes are placed randomly on a rectangular grid and links are created using the probability function [30]

$$P(e) = \beta \exp \frac{-d(e)}{\alpha L}, \quad d(e) \geq 0, \quad (7)$$

where  $e = (v_i, v_j)$ ,  $d(e)$  is the distance between nodes  $v_i$  and  $v_j$ , and  $L$  is the maximum possible distance between any two nodes. Parameters  $\alpha$  and  $\beta$ , in the range  $0 < \alpha, \beta \leq 1$ , are used to control various properties of the generated networks. A smaller value of  $\alpha$  increases the number of shorter links relative to longer ones, while a larger value of  $\beta$  results in higher average node degrees. In our simulations, 100 nodes are distributed across a Cartesian plane with coordinates between (0, 0) and (100, 100), corresponding to  $L = 141.42$ , while  $\alpha = 0.2$ , and  $\beta = 0.4$ . Each link in the network is associated with two parameters, bandwidth capacity and cost. We use the distance between node  $v_i$  and node  $v_j$  as the natural cost for link  $e = (v_i, v_j)$ . The bandwidth capacity on each link is allocated as [22]

$$b(e) = b_m + (-1)^r \times r' \bmod b_m, \quad (8)$$

where  $b_m$  is a given mean bandwidth parameter, and  $r$  and  $r'$  are random numbers conceptually uniformly distributed over the entire range of integers. Using this function, no negative values are generated and the bandwidth capacity of all links ranges from 1 to

$2b_m - 1$ . To simplify the simulations, each member of the multicast group requires one unit bandwidth from the network.

An additional requirement is placed on the generated graphs. Each graph must not violate the following necessary conditions for the existence of feasible solutions [22] before a simulation run is performed.

- (1) All member nodes must belong to the same connected component.
- (2) The total input bandwidth capacity for a member node  $v_i$  must be more than  $\sum_{v_j \in X, v_j \neq v_i} B_j$ , which is the least amount of input bandwidth required for the other group members to send data to it.

In the following experiments, we compare the success ratio and total tree cost ratio among the algorithms. The success ratio is defined to be the percentage of successful tree construction among generated graphs, while the quality of the multicast trees built by the algorithms is evaluated as a ratio of the total cost of the constructed trees relative to that of the base simulcast case, where the multicast is simulated using unicast connections. The total cost of the constructed trees is the sum of the cost of all the multicast trees constructed by the algorithms, specifically according to Eq. (1), while the base case total cost is the sum of all the shortest path trees, each one computed using Dijkstra's algorithm with a member node as the root node.

One result worthy of detailed discussion is to the relative performance between the TM algorithm and Roos' algorithm. First note that Roos' algorithm has a theoretical guarantee of being a  $\sqrt{k}$ -approximation algorithm for directed graphs, while TM has a performance bound for only undirected Steiner trees [7]. However, TM consistently outperforms Roos' algorithm in our experiments as shown later. To get more insight into their relative performance, the performance of these two algorithms has been evaluated in more detail with experiments using the OR-Library benchmark sets [39, 40], adapted for the directed STP with a single source node. Complete experimental results are not presented here due to space limitation. Interested readers are referred to [33] for more details. To summarize, TM consistently outperforms the original Roos algorithm in the benchmark sets for directed graphs and randomly generated graphs. The reason will be summarized shortly. However, after making a slight modification to Roos' algorithm, called Roos+, we observe TM and Roos+ have very similar performance across all experimental sets. We computed the percentage of nodes in common between the multicast trees constructed by the two algorithms, and found that the two algorithms often have a surprisingly high percentage of nodes in common, at least 75% on average. Therefore, it is to be expected that the two algorithms should achieve very comparable tree cost performance. Again, more details can be found in our technical report.

We briefly describe the modifications needed below. Recall Roos' strategy requires the determination of the best star point in the current network by computing of the cost of the shortest path from root to a candidate star point, and the cost of the shortest path spanning tree from the candidate star point to cover all the yet un-covered terminal nodes. The problem lies in the computation of the latter cost. To reduce total computation time, the computation is performed only once for each node in the network, at the beginning of

the algorithm by computing the shortest path spanning tree from each node to all the network terminal nodes, as can be seen from the for loop at the beginning of the procedure in Fig. 4. However, since Roos' algorithm modifies the network graph after each iteration, the shortest path tree computation from the candidate node to all the yet uncovered network nodes is valid only for the first iteration. This optimization renders the performance of Roos' algorithm suboptimal, as demonstrated in [33]. Therefore, the modification involves moving the invocation of Dijkstra's algorithm for all nodes at the beginning of the algorithm into the first action performed within the while loop, thereby correcting the inaccuracy in the original Roos algorithm. However, this simple modification substantially increases the computation time of the algorithm, losing the time efficiency advantage of the original algorithm.

#### 4.1 Performance Evaluation for GMRP

We will compare the success ratio and total tree cost ratio among the algorithms. The success ratio is defined as the percentage of successful tree constructions among the generated graphs. Fig. 9 shows the success ratio of the algorithms versus the mean bandwidth. Each data point is the average of 500 runs. The number of group member nodes is fixed at 30. The performance of **GMCP-TM** and **GMCP-ROOS** is shown relative to that of FTM and GTM. Observe that the success ratio of every algorithm increases with increasing mean link bandwidth. This phenomenon is expected, since as the mean bandwidth increases, it becomes harder for each of the edges to become saturated, and the chance of finding paths with sufficient bandwidth capacity increases. From Fig. 9 we can see that the success ratios of **GMCP-TM** and **GMCP-ROOS** are almost identical because both algorithms are based on the same idea of critical pairs. The success ratios of **GMCP-TM** and **GMCP-ROOS** are significantly higher than those of GTM and FTM. Even though FTM algorithm is designed to find only feasible solutions by always selecting the path with maximum bottleneck bandwidth, **GMCP-TM** and **GMCP-ROOS** still manage to achieve higher success ratios than FTM.

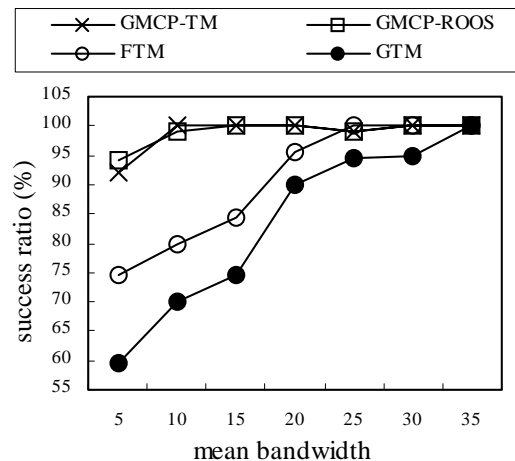


Fig. 9. Success ratio vs. mean bandwidth.

The quality of the multicast trees built by the algorithms is evaluated next. In addition, we plot the function  $k^{0.8}/k$ , which is derived from the power law investigated in general multicast routing [41]. The power law function  $L(i, k) = \frac{L_i}{\ell_i} \propto k^{0.8}$  gives an explicit formula for the average tree cost of a single multicast tree to reach  $k$  randomly chosen network locations from any given source node  $v_i$ .  $k$  is the multicast group size,  $v_i$  is the root node of the tree,  $\ell_i$  is the average cost of unicast routing paths from  $v_i$  to every member node, and  $L_i$  is the total cost of the multicast distribution tree, defined as the summation of edge costs of all links that make up the tree. Therefore, the cost ratio for a constructed multicast tree with root node  $v_i$  would be calculated as

$$\frac{L_i}{\sum_{v_j \in X} c(p(v_i, v_j))} \cong \frac{\ell_i \times k^{0.8}}{k \times \ell_i} = \frac{k^{0.8}}{k}, \quad (9)$$

where  $X \subseteq V$  is the terminal set, and  $|X| = k$ . The expression  $\sum_{v_j \in X} c(p(v_i, v_j))$  denotes the summation of the cost of the shortest paths from the source  $v_i$  to the  $k$  terminal nodes, where  $c(p(v_i, v_j)) = 0$  if  $v_i = v_j$ . Since the average cost of each shortest path is  $\ell_i$ ,  $\sum_{v_j \in X} c(p(v_i, v_j))$  can be approximated by  $k \times \ell_i$ . Then the total cost ratio for all constructed multicast trees in the group is

$$\sum_{v_i \in X} \frac{L_i}{\sum_{v_j \in X} c(p(v_i, v_j))} \cong \sum_{v_i \in X} \frac{\ell_i \times k^{0.8}}{k \times \ell_i} = k^{0.8}. \quad (10)$$

Therefore, the value of  $k^{0.8}/k$  is a reasonable approximation of the average per tree cost ratio according to the power law.

In Fig. 10, we show the average per tree cost ratio, shown along with its 95% confidence interval, versus multicast group size. The mean bandwidth is fixed at 30 with the number of multicasting nodes ranging from 10 to 70. We first observe that the tree cost ratio of every algorithm decreases as group size increases. This phenomenon is expected because as the group size increases, more links can be shared by the members, which results in relatively lower tree costs compared to the base-line Dijkstra case, which does not utilize shared links. The performance of **GMCP-TM** is better than that of **GMCP-ROOS** in our simulations.

Fig. 11 shows the overall tree cost ratio versus the mean bandwidth along with the corresponding 95% confidence intervals. The group size is fixed at 30 with the mean bandwidth ranging from 5 to 35. The performance of **GMCP-TM**, **GTM**, and **GMCP-ROOS** is consistent with the intuition that, with more bandwidth available, all algorithms should be able to build multicast trees with more shared branches. However, the tree cost ratios of **GMCP-TM**, **GMCP-ROOS** and **GTM** decrease only slightly because the dominant factor influencing the tree cost ratio performance is the group size. It is interesting to note that as mean bandwidth increases, the tree cost ratio of **FTM** increases

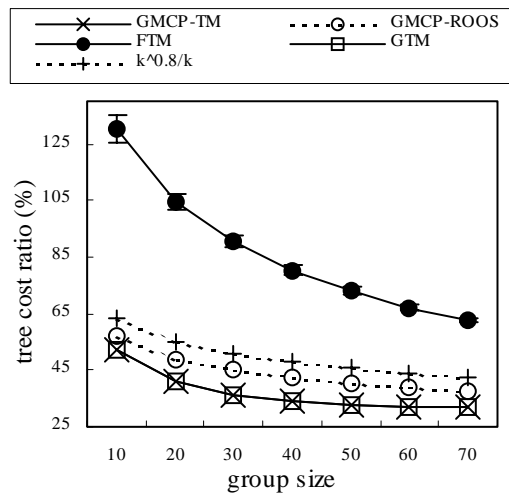


Fig. 10. Tree cost ratio (%) vs. group size.

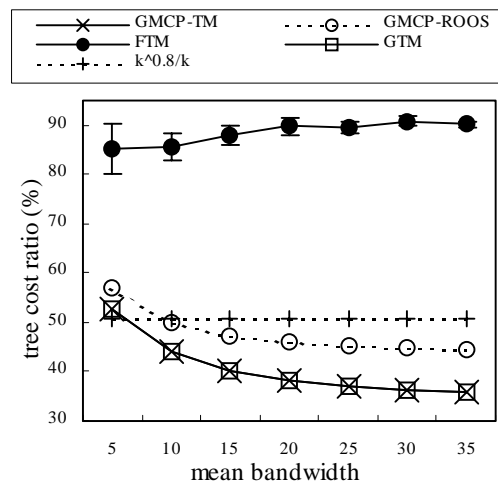


Fig. 11. Tree cost ratio (%) vs. mean bandwidth.

instead of decreasing. The reason is that the FTM algorithm always selects the path with maximum bottleneck bandwidth to construct the tree due to its focus on finding only feasible solutions. Therefore, the algorithm is induced to branch out with the availability of more bandwidth. In Figs. 10 and 11, **GMCP-TM** and **GTM** consistently exhibit almost the same performance, but the performance of **GMCP-ROOS** is not as good as that of **GMCP-TM**, and that of **FTM** is the worst. However, while **GMCP-TM** achieves the same tree cost performance as **GTM**, it achieves that with much better probability of success, as illustrated in Fig. 9.

We have also implemented the **GMCP-CHARIKAR** algorithm, a revised version of **GMCP-TM**, in which Charikar *et al.* proposed algorithm [13] serves as the single

source routing algorithm in the framework. Details of the simulations and their results are shown in [33]. In particular, we implemented a set of simulations to investigate the performance difference resulting from different values of parameter  $i$  in algorithm  $A_i(k, v_r, X)$ , which is, as described briefly before, an approximation procedure for constructing a directed Steiner tree rooted at  $v_r$  and spanning  $k$  of the terminals in set  $X$ . Complete experimental results are not presented here due to space limitation; interested readers are referred to [33] for more details. To summarize, the performance of **GMCP-CHARIKAR** is slightly better than that of **GMCP-TM** and **GMCP-ROOS** when the parameter  $i$  in  $A_i(k, v_r, X)$  is higher than or equal to  $\lceil \ln k \rceil$ , where  $k = |X|$ , and  $\lceil \ln k \rceil$  is the smallest integer value greater than  $\ln k$ .

#### 4.2 Performance Evaluation for DGMRP

In this subsection we compare the performance of **DGMCP** and **MBBPS** with respect to their success ratio and tree cost ratio. Since results in [28] indicate that the performance of **MBBPS** is better than that of **DGGM<sub>A</sub>** and **DGGM<sub>B</sub>**, the performance of the latter two algorithms will not be included. The same network model and parameters used in static **GMRP** simulations will be used in dynamic **GMRP** simulations, except that 200 node networks are used to simulate a larger number of join and leave requests. The generated graphs are tested until all nodes belong to a connected component. A sequence of join and leave requests is generated according to a probability function  $P_{join}(k)$ , as defined in [30], that determines whether the request is a join request or a leave request. The probability function  $P_{join}(k)$  is

$$P_{join}(k) = \frac{\lambda(|N| - k)}{\lambda(|N| - k) + k(1 - \lambda)}, \quad (11)$$

where  $k$  is the current number of nodes in the multicast group,  $|N|$  is the number of nodes in the network, and  $\lambda$  is a parameter in the range  $(0, 1)$ . A larger  $\lambda$  value means join requests are more likely to be generated; hence there will be more multicast members created in the network. When the generated event is a join request, a node is randomly chosen from the set of nodes that are not in the current multicasting group for addition into the group. If a leave request is generated, a member node will be randomly selected from the current group for removal.

For the performance measures, the success ratio result includes the successful completion of join events only since leave events will always succeed. The join success ratio is defined as the number of successful join events divided by the total number of join events, where the total number of requests generated in the simulations is 100. To put into perspective the performance of both our proposed static and dynamic algorithms, we also include the result of the static **GMCP-TM** in following figures.

Fig. 12 shows the join success ratio vs. the joining probability parameter  $\lambda$ , where the mean bandwidth is fixed at 6. We can see that as  $\lambda$  increases, the success ratio of every algorithm decreases. This is because a larger value for  $\lambda$  results in more group members in the multicasting group. Therefore, a larger number of multicast trees must be created in the network. With network resources being fixed, the probability for all algorithms to find feasible multicast trees must become smaller. The figure also shows that

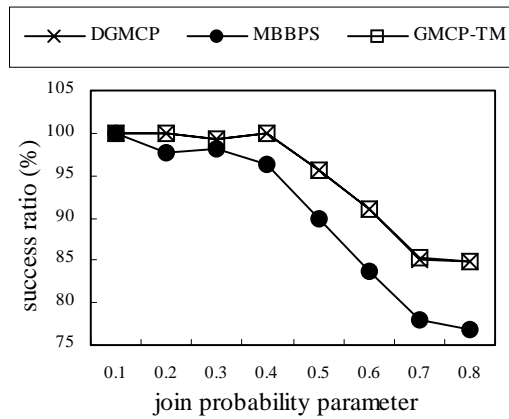


Fig. 12. Join success ratio vs. join probability.

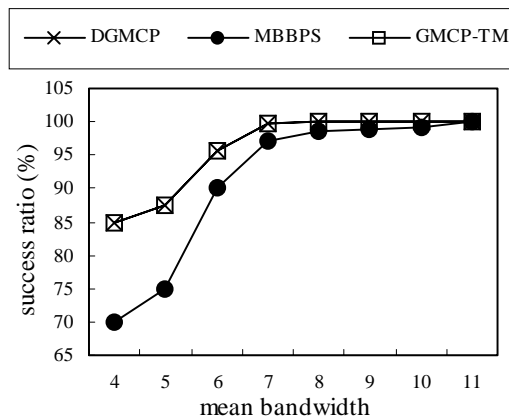


Fig. 13. Join success ratio vs. mean bandwidth.

the join success ratio of **DGMCP** is slightly higher than **MBBPS** and almost equal to that of static **GMCP-TM**'s. Considering that **MBBPS** is designed to focus entirely on constructing feasible multicast trees, the performance of **DGMCP** is excellent.

Fig. 13 shows the join success ratio vs. the mean bandwidth, where joining probability parameter  $\lambda$  is fixed at 0.5. We can see that as the mean bandwidth increases, the success ratio of every algorithm increases. This is because larger mean bandwidth values indicate more abundant network bandwidth resources. Therefore, the probability of success for adding new members into the multicasting group improves. The figure also shows that the joining success ratio of **DGMCP** is slightly higher than **MBBPS**'s and almost equal to static **GMCP-TM**'s.

Fig. 14 shows the total tree cost ratio and the corresponding 95% confidence interval vs. the joining probability parameter  $\lambda$ , where the mean bandwidth is fixed at 6. Observe that the tree cost ratio decreases as the joining probability parameter  $\lambda$  increases. This is because as  $\lambda$  increases, more members are admitted into the multicast group.

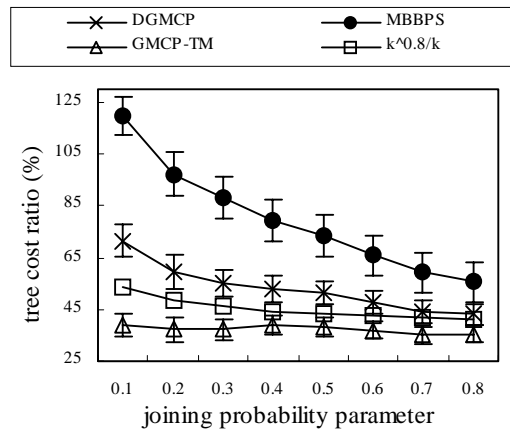


Fig. 14. Tree cost ratio vs. join probability.

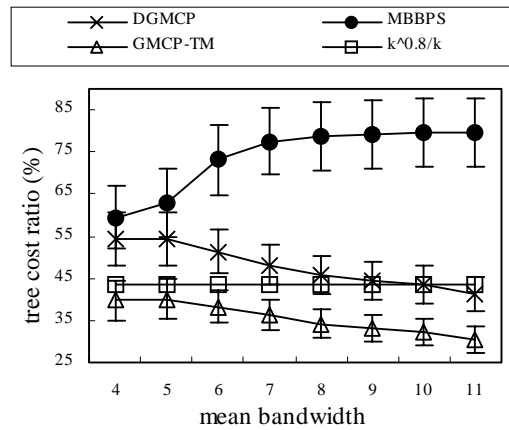


Fig. 15. Tree cost ratio vs. mean bandwidth.

Therefore, more links can be shared by the group members, resulting in relatively lower tree costs compared to the base-line Dijkstra case, which does not utilize shared links. **DGMCP**'s tree cost ratio is always lower than **MBBPS**'s, with **GMCP-TM** showing the best performance among the three. Furthermore, as  $\lambda$  increases, the cost ratio of **DGMCP** approaches that predicted by the power law  $k^{0.8}/k$ , and that of the static **GMCP-TM**.

Fig. 15 gives total tree cost vs. the mean bandwidth, with  $\lambda$  fixed at 0.5. Each data point is shown along with the corresponding 95% confidence intervals. Not surprisingly, as mean bandwidth increases, the tree cost ratio of **DGMCP** decreases since more links can be shared by the group members as capacity of the links increases. As mean bandwidth increases, the cost ratio of **DGMCP** approaches that predicted by the power law  $k^{0.8}/k$ . In contrast, it is seen that the performance of **MBBPS** exhibits the peculiar trend that the cost ratio performance increases with increasing mean bandwidth. This is a consequence of its search strategy, which always selects the path with maximum bottleneck

bandwidth as it constructs the multicast tree. Clearly, as large numbers of links with ample bandwidth appear in the network, the MBBPS strategy will tend to spread the selected links over large areas of the network instead of sharing the links as much as possible. This is a significant drawback with the MBBPS approach, which sacrifices tree cost while striving for feasibility. On the other hand, our approach can achieve much better tree cost performance without sacrificing the goal for maximizing feasibility.

## 5. CONCLUSIONS AND FUTURE WORK

Three algorithms are proposed in this paper for solving the GMRP and DGMRP. For the static GMRP, results from the experiments show that the proposed **GMCP-TM** and **GMCP-ROOS** algorithms can construct multicast trees with higher success ratios and lower cost than FTM [23]. In addition, it can always construct multicast trees of comparable quality but with higher success ratios than GTM [22]. For the DGMRP, our algorithm, **DGMCP**, also performs significantly better than MBBPS. Our proposals are also very efficient, since they are based on Dijkstra's algorithm and the TM algorithm. Our results are important in view of recent emergence of peer-to-peer network applications, such as massive multiplayer online games, and video conferencing. Our proposals provide effective tools for possible network layer support for P2P applications that have bandwidth constraints.

For future work, we intend to investigate the group multicast problem under other kinds of constraints. For example, the packet-copying capacity of each switching node is limited due to technological restrictions. The packet-copying capacity constraint of a node can be represented as an egress degree constraint. The problem of finding a set of multicast trees under these constraints can be modeled as a *Degree Constrained Group Multicast Routing Problem*. On the other hand, the ingress degree constraint is often considered inherently associated with networks supporting group multicasting, since multicasting requires routers to build  $O(|X|)$  source-specific multicast routing trees for each group member. The routing algorithm must consider the resources required by the routers. Strictly speaking, this issue is somewhat at odds with to the routing algorithm issue addressed in this paper. Research addressing this issue has been discussed in [42, 43]. We are also interested in extending our work to deal with multiple additive QoS constraints, such as delay, delay jitter and packet loss rate.

## REFERENCES

1. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Company, California, 1979.
2. M. Bern and P. Plassmann, "The Steiner problems with edge lengths 1 and 2," *Information Processing Letters*, Vol. 32, 1989, pp. 171-176.
3. Napster, LLC, <http://www.napster.com>, 2003.
4. Mirror Image® Internet, Inc., <http://www.mirror-image.com>, 2004.
5. B. Krishnamurthy, C. Wills, and Y. Zhang, "On the use and performance of content distribution networks," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2001, pp. 169-182.

6. L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for Steiner trees," *Acta Informatica*, Vol. 15, 1981, pp. 141-145.
7. H. Takahashi and A. Matsuyama, "An approximate solution for the Steiner problem in graphs," *Mathematica Japonica*, Vol. 24, 1980, pp. 573-577.
8. M. Karpinski and A. Zelikovsky, "New approximation algorithms for the Steiner tree problems," *Journal of Combinatorial Optimization*, Vol. 1, 1997, pp. 47-65.
9. P. Berman and V. Ramaiyer, "Improved approximations for the Steiner tree problem," *Journal of Algorithms*, Vol. 17, 1994, pp. 381-408.
10. A. Zelikovsky, "Better approximation bounds for the network and euclidean Steiner tree problems," Technical Report CS-96-06, University of Virginia, 1996.
11. A. Zelikovsky, "An  $11/6$  approximation algorithm for the network Steiner problem," *Algorithmica*, Vol. 9, 1993, pp. 463-470.
12. R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, Vol. 36, 1957, pp. 1389-1401.
13. M. Charikar, C. Chekuri, T. Y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li, "Approximation algorithms for directed Steiner problems," *Journal of Algorithms*, Vol. 33, 1999, pp. 73-91.
14. S. Roos, "Scheduling for ReMove and other partially connected architectures," Technical Report 1-68340-44(2001)-05, Delft University of Technology, 2001.
15. A. Shaikh and K. G. Shin, "Destination-driven routing for low-cost multicast," *IEEE Journal on Selected Areas in Communications*, Vol. 15, 1997, pp. 373-381.
16. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, "Introduction to algorithms," *MIT Press*, 1990.
17. Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves, "A source-based algorithm for delay-constrained minimum-cost multicasting," in *Proceedings of IEEE INFOCOM*, 1995, pp. 377-385.
18. R. Widyonon, "The design and evaluation of routing algorithms for real-time channels," Technical Report ICSI TR-94-024, International Computer Science Institute, University of California at Berkeley, 1994.
19. V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Transactions on Networking*, Vol. 1, 1993, pp. 286-292.
20. Q. Sun and H. Langendoerfer, "An efficient delay-constrained multicast routing algorithm," *Journal of High-Speed Networks*, Vol. 7, 1998, pp. 43-55.
21. I. Matta and L. Guo, "QDMR: an efficient QoS dependent multicast routing algorithm," in *Proceedings of 5th IEEE Real-Time Technology and Applications Symposium*, 1999, pp. 213-222.
22. C. P. Low and N. Wang, "An efficient algorithm for group multicast routing problem with bandwidth reservations," *Computer Communications*, Vol. 23, 2000, pp. 1740-1746.
23. C. P. Low and N. Wang, "On finding feasible solutions to group multicast routing problem," *IEICE Transactions on Communications*, Vol. E85-B, 2002, pp. 268-277.
24. X. Jia and L. Wang, "Group multicast routing algorithm by using multiple minimum Steiner trees," *Computer Communications*, Vol. 20, 1997, pp. 750-758.
25. C. P. Low and X. Song, "On finding feasible solutions for the delay constrained group multicast routing problem," *IEEE Transactions on Computers*, Vol. 51, 2002, pp. 581-588.

26. X. Song and C. P. Low, "On finding feasible solutions for delay and bandwidth constrained group multicast routing problem," *IEEE International Conference on Telecommunications*, 2002.
27. H. Tanioka, K. Kinoshita, and K. Murakami, "Multipoint-to-multipoint routing for multimedia communication service," in *Proceedings of IEEE International Conference on Communications*, Vol. 2, 2002, pp. 1248-1252.
28. C. P. Low, N. Wang, and J. M. Ng, "Dynamic group multicast routing with bandwidth reservations," *International Journal of Communication Systems*, Vol. 15, 2002, pp. 655-682.
29. E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerical Mathematics*, Vol. 1, 1959, pp. 269-271.
30. B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, Vol. 6, 1988, pp. 1617-1622.
31. F. Bauer and A. Varma, "ARIES: a rearrangeable inexpensive edge-based on-line Steiner algorithm," in *Proceedings of IEEE INFOCOM '96*, 1996, pp. 361-368.
32. H. Lin and S. Lai, "VTDM – a dynamic multicast routing algorithm," in *Proceedings of IEEE INFOCOM '98*, 1998, pp. 1426-1432.
33. K. T. Tsai, "Empirical comparisons of the TM and ROOS algorithms for DST and GMRP problems," Technical Report, Dept. of Electronic Engineering, National Taiwan University of Science Technology, <http://cchen1.csie.ntust.edu.tw/students/tsai/TM-ROOS-report.doc>, 2004.
34. M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM*, Vol. 34, 1987, pp. 596-615.
35. A. Orda and A. Sprintson, "QoS routing: the precomputation perspective," *IEEE INFOCOM 2000*, Vol. 1, 2000, pp. 128-136.
36. G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, and S. K. Tripathi, "Intra-domain QoS routing in IP networks: a feasibility and cost/benefit analysis," *Special Issue of IEEE Network on Integrated and Differentiated Services*, Vol. 13, 1999, pp. 42-54.
37. A. Orda and A. Sprintson, "Precomputation schemes for QoS routing," *IEEE/ACM Transactions on Networking*, Vol. 11, 2003, pp. 578-591.
38. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Inc., 1993.
39. J. E. Beasley, "An SST-based algorithm for the Steiner problem in graphs," *Networks*, Vol. 19, 1989, pp. 1-16.
40. C. Duin and S. Voss, "Efficient path and vertex exchange in Steiner tree algorithms," *Networks*, Vol. 29, 1997, pp. 89-105.
41. G. Phillips, S. Shenker, and H. Tangmunarunkit, "Scaling of multicast trees: comments on the chuang-sirbu scaling law," in *Proceedings of the ACM SIGCOMM*, 1999, pp. 41-51.
42. J. H. Cui, J. Kim, D. Maggiorini, K. Boussetta, and M. Gerla, "Aggregated multicast – a comparative study," *Special Issue of Cluster Computing: The Journal of Networks, Software and Applications*, Baltzer Science Publisher, 2003.
43. J. H. Cui, D. Maggiorini, J. Kim, K. Boussetta, and M. Gerla, "A protocol to improve the state scalability of source specific multicast," in *Proceedings of IEEE GLOBECOM 2002*, 2002, pp. 17-21.



**Kun-Cheng Tsai (蔡坤成)** received the M.S. degree in Electrical Engineering from Chung Hua College, Taiwan in 1995. During 1997-2002, he was with the Multimedia and Communications Research Laboratory of the University of Science and Technology in Taiwan. He is now studying real-time multimedia communication and wireless network systems.



**Chyouhwa Chen (陳秋華)** is an Associate Professor in Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taiwan. He received his Ph.D. degree from State University of New York at Stony Brook. His research interests include computer network, traffic management, multimedia communications, and issues in multicast routing.