On the Construction of Low Cost Multicast Trees with Bandwidth Reservation

Dirceu Cavendish [*], Aiguo Fei, Mario Gerla

Department of Computer Science
University of California,Los Angeles CA 90024
{dirceu,afei,gerla}@cs.ucla.edu

Raphael Rom

Faculty of Electrical Engineering
Israel Institute of Technology Haifa 32000 Israel
rom@ee.technion.ac.il

Abstract

This report presents routing algorithms for many-to-many multicast connections with bandwidth requirements. The objective is to compute a connectivity structure among multicast members of low cost, taking into account their bandwidth requirements. We first study the complexity of routing multicast connections with bandwidth reservation. Based on derived impossibility results, we propose a cluster strategy to connect multicast members, called Bandwidth Constrained Steiner Tree (BCST) strategy. We show that BCST strategy is the best polynomial time algorithm for such routing problems, in the sense of worst case performance. We propose two BCST algorithms, and claim hard bounds on cost and bandwidth overload. We study the proposed algorithms' performance on randomly generated multicast routing problems, and compare them against two well known routing strategies for multicast connection: Shortest Path Tree (SPT), where a SPT is built per multicast source, taking into account bandwidth requirements; and Minimum Spanning Tree (MST), where a single Steiner Tree is built to provide connectivity among multicast members, regardless of their bandwidth requirements. Our simulation experiments show that BCST algorithms strike a balance between blocking probability and tree cost, when compared against both SPT and MST algorithms.

[*] This work was partially supported by CNPq under Grant 201597/93-4(NV)

Contents

1 Introduction

One of the new challenges brought on by multimedia network technologies, such as ATM and the next generation Internet, is routing with resource reservations. By incorporating resource reservation into a routing algorithm, the chance that the resources required for carrying a given connection be available at connection setup time are greatly increased. Therefore, sucessful response to connection requests requiring network resources depends heavily on the routing algorithm used.

In a QOS environment, routing algorithms must provide routes which conform with users end-to-end requirements while minimizing the cost of a connection. A connection cost is defined as the sum of the costs of individual links participating in the connection. Link costs can be assigned in various ways. Generally, they should reflect the desirability of using a given link as part of a connection path. As such, they are often referred to as administrative weights in the ATM world. QOS requirements translate into path constraints for a routing algorithm, and impact its computation complexity. Complexity is a major concern in network scalability since the routing algorithm must sometimes be run in real time, specially in case of multicast connections. A given application may not afford to wait too long for a routing algorithm to compute a QOS suitable path, before a connection setup signalling can be initiated. Moreover, if route computation takes too long, topology information in which this computation is performed is more likely to be outdated, leading to a higher probability of having a connection setup failure.

The first, and perhaps most important network resource is bandwidth. We assume a link state routing strategy, as in PNNI ATM technology or in Internet OSPF used as a Interior Gateway routing Protocol (IGP), where each route computation site knows about resources along the links of the entire network. The problem of routing unicast (one-to-one) unidirectional connections with bandwidth reservation was shown to be solved by a Dijkstra type algorithm [Wang95] for link state routing. For many-to-many type of connections, however, to our knowledge there is no routing algorithm in the literature which includes bandwidth reservation.

Multicast traffic is typically carried through one (Core Based Tree) or more (Distance Vector Multicast Routing Protocol) multicast trees [Deering96], for efficiency of network resources. We address the problem of constructing multicast trees, so as to provide not only connectivity among members of a multicast connection, but also bandwidth necessary to carry the traffic injected by multicast members. The report is organized as follows. Section 2 describes related work. Section 3 formally defines our problems of interest, and presents some hard complexity results. Section 4 uses the hard results to design polynomial algorithms with worst case guarantees. Section 5 presents simulation results to illustrate the algorithms behavior and performance. Conclusions are drawn in the last section.

2 Related Work

As mentioned earlier, the most important network resource to be allocated by a QOS network is bandwidth. Unicast (pt-pt) routing has been shown to be of polynomial complexity [Wang95], and thus scalable with the size of the network. The idea is to disregard any link which does not have enough available bandwidth to carry the bandwidth required by the single source in the computation of a minimum cost path. Using the same strategy for point-to-multipoint (pt-mtp) connections, a minimum cost spanning tree can be constructed with the same polynomial complexity. However, for truly multicast connections (mtp-mtp), we are indeed facing a NP-Complete problem, as will be shown shortly.

Although the problem of constructing Steiner trees is hardly new (e.g. [Smith83]), its study has intensified in the last ten years due to its direct applicability to multimedia networks [Frank85, Makki95]. Moreover, multimedia connections are likely to add some constraints in feasible trees, such as bandwidth and delay. Research work on the construction of Steiner Trees with such constraints is less abundant in the literature. Among these, [Jiang91] surveys algorithms which are able to compute a set of Steiner Trees whose total bandwidth capacity matches the needs of multicast members. The work is limited, however, in a sense that it assumes that every multicast member has the same bandwidth needs, and does not provide performance guarantees of any kind. [Ravi95] establishes hard results, and proposes algorithms for the construction of constrained Steiner Tree problems, where the constraints are the *diameter* of the tree and the *maximum degree* of a vertex. It also provides worst case guarantees for algorithms performance. Our contribution extends that work in providing algorithms for constructing low cost Steiner Trees with *bandwidth* constraints and worst case guarantees, for many-to-many casting applications.

3 Theoretical Results

We model our network as a directed graph $G(V, E)$, where V is the set of nodes and E the set of directed *edges*. We refer to a *link* (i, j) as the pair of edges (i, j) and (j, i). We associate two values with every edge $(i, j) \in V$: a positive weight $w(i, j)$ and a nonnegative (bandwidth) capacity $b(i, j)$. Let $S \subseteq V$, $D \subseteq V$ be source and destination subsets of V, such that $\forall i \in S$, we associate $B_i \geq 0$ as the bandwidth needed by source i.

Define a multipoint connectivity structure (MCS) $\sigma(V', E')$ as a connected subgraph of $G(V, E)$ containing at least the nodes $S \cup D$ and having at least one path from each node $s \in S$ to every node $d \in D$. The bandwidth and weights associated with an edge in σ are those associated with the original edges of G.

Let $S(i, j) \subseteq S$ be the subset of sources contributing flow to edge (i, j). Then:

Definition 1 *An edge (i, j) is said to be underloaded if:*

$$b(i, j) \geq \sum_{p \in S(i,j)} B_p \qquad (1)$$

otherwise, the link is said to be overloaded.

A link is underloaded if both edges comprising it are underloaded. An MCS is called *feasible* if all its edges are underloaded. The *weight* or *cost* of an MCS is the sum the weights of its edges.

In this paper we focus on searching and investigating low cost, feasible, MCSs.

3.1 The bidirectional connection

The simplest multipoint-to-multipoint (mtp-mtp) connection that one may conceive is a bidirectional unicast connection. In this case, $S = D = \{s, d\}$. If we allow the paths $p(s, d)$ and $p(d, s)$ to be distinct, the problem of finding a minimum cost feasible connectivity structure can be reduced to computing two single source shortest-path problems with bandwidth constraint ([Wang95]). More formally:

Algorithm1:

1. Create the subgraph $G'(V', E')$ by pruning off edges $(i, j) \in V$ such that $b(i, j) < B_s$.
2. Compute a shortest path $m(s, d)$, between the source s and destination d.
3. Decrease the bandwidth of every edge $(i, j) \in m(s, d)$ by B_s.
4. Create the subgraph $G''(V'', E'')$ by pruning off edges $(i, j) \in V$ such that $b(i, j) < B_d$.
5. Compute a shortest path $m(d, s)$, between the source d and destination s.
6. Construct an MCS the includes all the nodes and edges comprising both $m(d, s)$ and $m(s, d)$.

Theorem 1 *Algorithm1 finds a feasible minimum cost MCS for $S = D = \{s, d\}$.*

Proof of Theorem 1 *If the MCS is formed by two disjoint paths, the Theorem follows by applying [Wang95]'s result for each "half" connection, as if they were separate problems. So, let's assume that minimum paths $m(s, d)$ and $m(d, s)$ share some edges(s). However, the underload condition of the edge guarantees that the referred edge will be present in the second shortest-path computation, which will find the second minimum path $m(d, s)$.*

Let us require now that the MCS be a single bidirectional path connecting (s, d). To proceed, we need the following simple definitions:

Definition 2 *The length $d^c(s, d)$ of a path $p(s, d)$ under cost function c is defined as:*

$$d_p^c(s, d) = \begin{cases} \sum_{(i,j) \in p(s,d)} c(i, j) & \\ \quad \textit{if } \forall (i, j) \in p(s, d), \qquad b(i, j) \geq B_s \textit{ and } b(j, i) \geq B_d & \\ \infty & \\ \quad \textit{otherwise} & \end{cases}$$

i.e. the path length is the sum of its link lengths if its edge components are all underloaded. Otherwise, the path length is assumed to be ∞.

Definition 3 *The shortest path between (s, d) with respect to the cost function c is:*

$$\delta^c(s,d) = \min_{p \in P} d^c(s,d)$$

where P is the set of all paths $p(s, d)$.

We now propose a Dijkstra type of algorithm to solve the single bidirectional path min-cost problem. The pseudo-code follows:

3.1.1 BD-Dijkstra Pseudo-code

BD-DIJKSTRA G(V,E)

BD-Initialize(G,s);
$S \leftarrow \{\}$, $Q \leftarrow V[G]$;
while $Q \neq 0$ **do**
    u= BD-Extract-Min(Q, d);
    $S \leftarrow S \cup \{u\}$;
    For each vertex $v \in Adj[u]$ **do**
        BD-Relax$(B_s, B_d, u, v, b(u,v), b(v,u), c(u,v))$;

BD-Initialize(G,s)

For each vertex $v \in V[G]$
    Do $d[v] \leftarrow \infty$; $\pi[v] = NIL$;
$d[s] \leftarrow 0$;

BD-Extract-Min(Q,d)

$d_{min} \leftarrow \infty$; $u \leftarrow NAN$;
For $i \in Q$ **do**
    If $(d[i] < d_{min})$
        { $d_{min} \leftarrow d[i]$; $u \leftarrow i$;}
RETURN u;

BD-Relax$(B_s, B_d, u, v, b(u,v), b(v,u), c(u,v))$

If $((B_s < b(u,v))\text{and}(B_d < b(v,u)))$
  $\{w(u,v) = c(u,v);\}$
else
  $\{w(u,v) = \infty;\}$
If $(d[v] > d[u] + w(u,v))$
  $d[v] \leftarrow d[u] + w(u,v);$
  $\pi[v] \leftarrow u;$

3.1.2 BD-Dijkstra analysis

As usual in Dijkstra type algorithms, S is a set of vertices whose current shortest path is maintained, and Q is a priority queue with vertices $i \in V - S$ with current distance $d[i]$. Each vertex u has a pointer $\pi[u]$ to its previous vertex in the current shortest path, which is initially set to NIL. BD-Extract-Min fetches the vertex outside S which is closest to the source, and BD-Relax updates vertices distances to the shortest ones, as is standard in Dijkstra's algorithm. The twist here is that BD-Relax tests if there is enough bandwidth on the link before consider it for relaxation. Notice that this is possible only because at this time, the direction in which the link will be used in the path is already defined. The reason why this link pruning is not done in advance is precisely because the direction in which the links could be used for connectivity is not known up until they are inserted in the path candidate, which is done by BD-Relax. So, BD-Dijkstra uses a cost function w which is defined during run time only.

We now prove the following theorem:

Theorem 2 *BD-Dijkstra algorithm computes $\delta^c(s,d)$ shortest path as defined above.*

Although we can prove the algorithm from scratch, we will rather build our proof on top of the correctness of the original Dijkstra algorithm [Cormen90]. In the course of the proof, we differentiate between w and c edge costs. We need the following lemmas:

Lemma 1 *BD-Dijkstra is a usual Dijkstra algorithm with respect to the edge cost function w. It, therefore, computes $\delta^w(s,d)$.*

Proof of Lemma 1 *The proof is based on the fact that each link is accessed by the algorithm only once[1], by BD-Relax, whereby its cost value is determined and remains fixed throughout the rest of the computation. Therefore, after a first run of the algorithm, all edge costs w are determined (notice that w is non-negative, as required). Thus, one can easily see that the running of a regular Dijkstra algorithm on the edge costs just defined by BD-Relax is guaranteed to compute the same path as the one computed by BD-Dijkstra. From the regular Dijkstra algorithm, this path is the shortest path in w.*

Lemma 2 *Throughout the execution of the algorithm, for every vertex $v \in V$, $d[v]$, the current distance from the source s to vertex v, is non-increasing.*

Proof of Lemma 2 *The lemma follows from the regular Dijkstra algorithm.*

The last and most important lemma we need for the Theorem proof is:

Lemma 3 *During the execution of the algorithm, for every path $p(s,u)$ built by BD-Dijkstra, $d[u] = d^c_p(s,u)$.*

Proof of Lemma 3 *A path is built by sucessively calling BD-Relax, since this is the only place where $\pi[v]$ gets assigned. Using the previous lemma, it is easy to see that this assignment occurs only if, for each edge $(i,j) \in p$, $w(i,j) < \infty$. But then $\forall (i,j) \in p, w(i,j) = c(i,j)$. Summing up over all edges, we obtain $d[u] = d^c_p(s,u)$.*

The last lemma dictates that every path computed in w by BD-Dijkstra has identical length in c. Conversely, it is easy to see that any path with finite length in c has identical length in w. It remains to be proved that the minimum path computed in w by BD-Dijkstra is identical to the minimum path in c, or $\delta^w(s,d) = \delta^c(s,d)$.

[1] If a link could be relaxed more than once, even if in opposite directions, loops could be formed. But we know Dijkstra algorithm is loop free for non-negative edge costs

Proof of Theorem 2 *Suppose that the minimum paths, p_o^c, p_o^w for the two cost functions are different, thereby with different costs. We have:*

$$\delta^c(s,d) = \sum_{edge \in p_o^c} c(i,j) \tag{2}$$

$$\delta^w(s,d) = \sum_{edge \in p_o^w} w(i,j) \tag{3}$$

For sake of contradiction, assume:

$$\delta^c(s,d) < \delta^w(s,d)$$

By the previous lemma, however, path p_o^c has a cost given by:

$$\begin{aligned} d^c_{p_o^c}(s,d) &= \delta^c(s,d) \\ &= \sum_{edge \in p_o^c} w(i,j) \\ &< \delta^w(s,d) \end{aligned} \tag{4}$$

But this implies that there is a path, p_o^c, with lower cost in w than $\delta^w(s,d)$, which contradicts lemma 1.

The complexity of BD-Dijkstra algorithm is identical to the regular Dijkstra algorithm, and is $O(N \log N)$, where N is the number of vertices. However, it is worth noticing that the original Dijkstra algorithm outputs shortest paths from a source to all other network vertices, or a Shortest Path Tree, while BD-Dijkstra solves a *single shortest path* only. This is essentially due to the fact that the optimality principle may be violated in this problem. This principle basically states that subpaths of shortest paths are themselves shortest paths. More precisely, we can prove that:

Lemma 4 *In the bidirectional shortest path problem, subpaths of shortest paths are not necessarily shortest paths.*

Proof of Lemma 4 *Let $m(i,j)$ be the shortest path between vertices i and j, with respective bandwidth requirements B_i, B_j. Moreover, let k be an intermediate vertex on this shortest path, $k \in m(i,j)$, with bandwith requirement $B_k > B_j$. Let there be a link (r,q) on the subpath $p(i,k)$ of the shortest path $m(i,j)$ such that its edge bandwidth $b(q,r)$ is $B_k \geq b(q,r) \geq B_j$. Then, it is easy to see that path $p(i,k)$ is not even a feasible path connecting i and k.*

It is easy to see, therefore, that an all shortest bidirectional path with bandwidth constraints has $O(M^2 N \log N)$ complexity, where $M = |S \cup D|^2$.

3.2 Multicast Tree Problems

We now focus our attention to larger S and D sets. We are interested on a particular Multipoint Connectivity Structure (MCS), called multicast tree, which we now define:

Definition 4 *A multicast tree (mtree) $MT(E', V')$ is an* **acyclic** *MCS $\sigma(E', V'), E' \subseteq E, V' \subseteq V$ providing connectivity to every $m \in S \cup D$.*

One can easily see that an mtree is a Direct Acyclic Graph (DAG). Mtrees inherit the same feasibility definition as for any MCS. Regarding the construction of feasible mtrees, we may devise two problems:

Problem 1 *Construct a feasible mtree.*

Problem 2 *Construct a feasible mtree of minimum cost.*

Generic minimum cost tree problems are known as Steiner Tree problems. Steiner Tree problems are known to be NP-Complete. Problems of such nature with additional constraints are called constrained Steiner Tree problems. Our approach, therefore, is to provide polynomial time algorithms,

[2]We can prove that an all shortest bidirectional path with bandwidth constraints has $O(M^2 N \log N)$ complexity by using the known fact that a single shortest path has $O(N \log N)$ complexity plus lemma 4. A worst case analysis leads to the desired claim

for sake of scalability, with *worst case performance* guarantees. In what follows, we establish theoretical hard results about the mtree problems defined previously from which worst case guarantees can be drawn. These are results regarding both feasibility and cost. We start with the former.

Theorem 3 *The problem of finding a feasible multicast tree is NP-Complete.*

Proof of Theorem 3 *By reduction from the PARTITION problem. Given a generic instance X of the PARTITION problem, we construct an instance G of the (Bandwidth, Steiner tree) problem in polynomial time, as follows. Let the graph G have a vertex x_i for each corresponding x_i element of X. Let there be six extra vertices: m, m, o, p, q, d. For each x_i element, we attach an edge of infinite bandwidth between itself and vertex m, and another between itself and vertex n. We also attach infinity bandwidth edges between vertices o and q, p and q, and between q and d. Moreover, we attach an edge of bandwidth equal to $(\sum_{x_i \in X} s(x_i))/2$ between vertices m and o, and between vertices n and p. All other possible edges are assumed to have zero bandwidth. Finally, let $S = X$ be the source set, where $B_{x_i} = s(x_i)$, for each $x_i \in X$, and $D = d$ the destination set. The arrangement for a particular instance of the PARTITION problem is illustrated in Figure 1.*

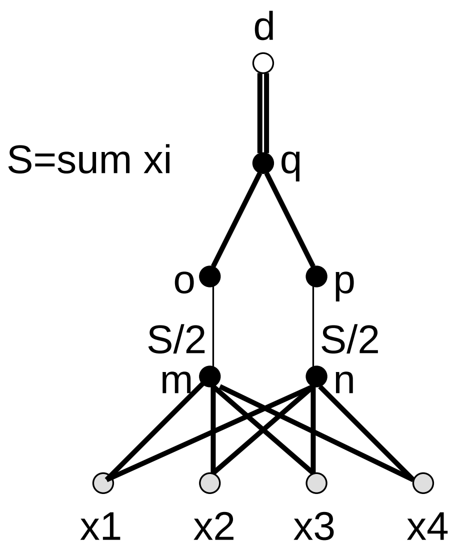


Figure 1: Reduction from PARTITION to feasible multicast problem

It is easy to see that any feasible mtree will define a solution for the PARTITION problem.

Regarding cost, we use a recent hard result about the minimum set cover problem, which is:

Theorem 4 *[Feigne96]Unless $NP \subseteq DTIME(n^{\log\log n})$, the MIN SET COVER problem, with a universe of size k, cannot be approximated to better than a $\ln k$ factor.*

In a similar approach to [Ravi95], we use an approximation preserving reduction from the MIN SET COVER problem to 2 problem, to prove the following theorem:

Theorem 5 *Unless $NP \subseteq DTIME(n^{\log\log n})$, given an instance of the (Bandwidth, Total Cost, Steiner tree) problem with $k = \mid S \cup D \mid$ sites, there is no polynomial-time approximation algorithm that outputs a feasible Steiner Tree, which costs at most R times that of the minimum cost Steiner Tree, for $R < \ln k$.*

Proof of Theorem 5 *Let (T, X) be a generic instance of the MIN SET COVER problem, where $T = \{t_1, t_2, \cdots, t_n\}$, $X = \{X_1, X_2, \cdots, X_m\}$, $X_i \subseteq T$, and c_i be the cost associated with set X_i. We construct an instance G of the (Bandwidth, Total Cost, Steiner tree) problem in polynomial time, as follows. Let the graph G have a vertex t_i for each corresponding t_i element of T, a vertex x_i for each set X_i, and two extra vertices, n, and s. For each set X_i, we attach an edge between vertices n and x_i of cost c_i, and bandwidth $(B, 0)$[3]. Moreover, for each element t_i and set X_j such that $t_i \in X_j$, we attach an edge of cost 0 and bandwidth $(B, 0)$. We also attach a similar edge between s and n. All other edges are assumed to have infinity cost and $(0, 0)$ bandwidth available. Finally, let $S = \{s\}$ be the source set, $D = \{t_1, t_2, \cdots, t_k\}$ the destination set, and $B_s = B$. The arrangement for a particular instance of the MIN SET COVER ($T = \{t_1, t_2, \cdots, t_7\}$, $X = \{X_1, X_2, X_3, X_4\}$, $X_1 = \{t_1, t_2, t_3\}$, $X_2 = \{t_3, t_4, t_5\}$, $X_3 = \{t_5\}$, $X_4 = \{t_6, t_7\}$) is shown in Figure 2.*

[3] B in the up-down direction, 0 in the opposite direction

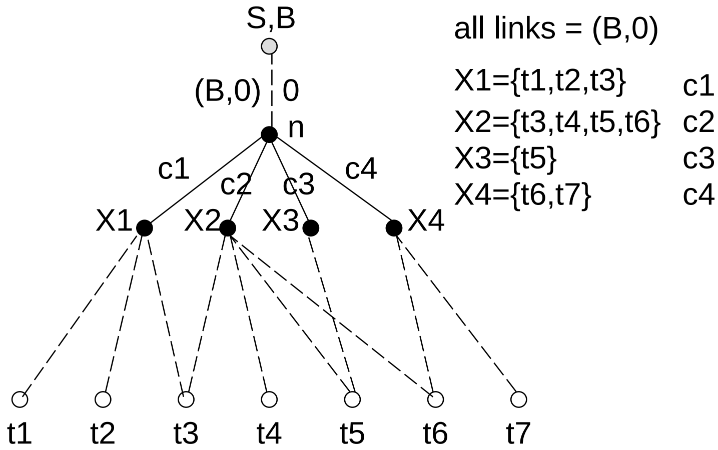


Figure 2: Reduction from MIN SET COVER to feasible minimum cost multicast problem

It is easy to see that any feasible mtree must contain a path from t_i to n, using an edge (x_j, n) for some X_j such that $t_i \in X_j$, for all i. Therefore, any feasible mtree provides a feasible solution of equivalent cost to the original set cover instance. The proof now follows from the previous theorem.

Based on Theorem 5, we advocate the use of a cluster type of polynomial time algorithm to solve problem 2, in a similar approach of [Ravi95], since such strategy is likely to provide a ln factor bound on the mtree cost.

4 A cluster algorithm for low cost mtrees

The basic idea is to design an algorithm that works in phases. In each phase, the algorithm connects a given number n of clusters together which were previously disconnected. The algorithm will stop after $\log_n K$ phases, for a multicast group of size K. Let OPT be the minimum cost feasible tree with respect to cost function c, and let OPT_c be its cost. Then, to guarantee a tree whose cost s is no greater than $\log_n K \times OPT_c$ the cost of the partial connectivity we are adding in each phase must be proportional to OPT_c. If this is true, Theorem 5 tells us we can not do better than that with a polynomial time algorithm. Thus, the cluster strategy is an excelent candidate to produce low cost mtrees in polynomial time, at least in the worst case sense. We now present a family of algorithms, called Bandwidth Constrained Steiner Tree algorithms, or BCST-algorithms. A discussion of the algorithms design follows.

BCST-algorithms

1 Initialize the set of clusters in phase 1, Φ_1, to contain $\mid S \cup D \mid$ singleton sets, one for each member of the multicast connection. Define the single vertex as the "center" of its respective cluster. $i = 1$.

2 Repeat until there remains a single cluster in Φ_i:

 2.1 Construct a complete graph $G_i(E_i, V_i)$ as follows: $v \in V_i$ iff v is a center of a cluster in Φ_i. Between every pair of vertices $(x, y), x, v \in V_i$, include an edge in E_i of cost defined as the minimum path cost between these two vertices in which the bidirectional bandwidth constraint defined by the clusters bandwidths is required. Theorem 2 states that the algorithm to find such path, BD-Dijkstra, is polynomial.

 2.2 Find a minimum cost matching of largest cardinality in G_i.

 2.3 For each edge (x, y) selected in the matching, merge their respective clusters C_x, C_y, by adding the path in the original graph corresponding to that edge to form cluster C_{xy}. The vertex (edge) set of the newly formed cluster is defined to be the union of the vertices (edges) of its cluster components, plus the vertices (edges) of the connecting path.

 2.4 Elect a new vertex center for each newly formed cluster. $i = i + 1$.

BCST-Algorithms maintain a set of clusters Φ_i in each phase i. Each cluster is formed by multicast members which are already connected among themselves. Notice that we have referred to BCST as a *family* of algorithms because certain steps are not completely specified. They are basically: the specification of a cluster bandwidth, in step (2.1); the election of the cluster head, in step (2.4).

We first argue why we have decided for a merging between two clusters and not more, for the greater the number of clusters connected in each phase the lesser the number of phases. In step (2.1), we need to find minimum paths connecting such clusters. If $n > 2$, we face the problem of finding minimum cost trees, which in itself needs an exponential time algorithm. For $n = 2$, BD-Dijkstra meets our needs, hence we settle for pairing clusters together in each phase, through a minimum cost matching of largest cardinality. The minimum cost matching comes into play in providing cost bounds, as we will see shortly.

The second consideration is regarding the nature of the connecting paths between two clusters. Shall they be only of source-destination nature? Or shall destination-destination and source-source connecting paths be also considered? We argue that when considering connecting paths, any of the types described must be considered indiscriminately, and the reason is two fold: Firstly, if we favored any type of path, it is easy to see that the resulting number of phases might be greater than $\log_2 k$, hence we wouldnt be able to use the hard results presented earlier. Secondly, favoring any type of path would make invalid the following lemma, which we will see shortly is paramount in bounding the cost of the resulting mtree.

Lemma 5 *[Klein95, Ravi93] Let T be a tree with an even number of marked vertices. Then, there is a pairing $(v_1, w_1), \cdots, (v_k, w_k)$ of the marked vertices such that the paths $p(v_i, w_i)$ in T are edge-disjoint.*

In particular, if the tree is an edgeweighted tree, a pairing of marked vertices that minimizes the sum of the lengths (cost) of the tree-path between these vertices are edge-disjoint. This is why we need to find a minimum cost matching in step (2.2). Notice that connecting source-source or destination-destination pairs does not pose any major problem to the mtree construct . In fact, it is likely that two sources or destinations are close to one another in a mtree. Moreover, multicast members are likely to be both sources and destinations.

In order to study hard bounds on both cost and overload, let there be a solution for Problem 2. Namely, let there be at least one feasible tree connecting the multicast members. Figure 3 gives an example of such a tree.

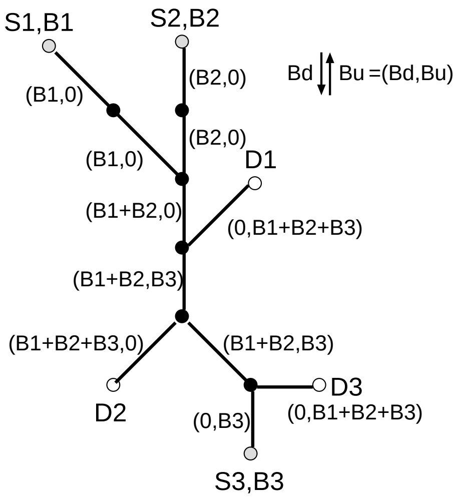


Figure 3: Feasible mtree. Only link bandwidths are shown

It is not difficult to see that bounds on cost and overload are inter related. Thus lets define various strategies for the amount of bandwidth B_p (bandwidth of path p) required by paths connecting clusters together. We can devise two extreme path bandwidth requirements: i) $B_p = 0$, or reserve nothing; ii) $B_p = \sum_{s \in S} B_s$, or reserve everything. In the first one, we use lemma 5 to argue that any tree found by such an algorithm has a total cost MT_c no greater than $\log_2 k \times OPT_c$.

Lemma 6 *A reserve nothing BCST-algorithm outputs a mtree* $MT_c \leq \log_2 k \times OPT_c$.

Proof of Lemma 6 *Lemma 5 is used to provide a (loose) bound of* OPT_c *on the cost of the additional paths in each algorithm phase. Each shortest path between two end points can be exactly found, since this is a regular shortest path problem with no constraints. Since there are at most* $\lceil \log_2 k \rceil$ *phases, the lemma follows.*

Notice, however, that the overload of a given edge belonging to the resulting mtree can be as large as the total bandwidth required by all sources. This is so because a zero-bandwidth edge next to a destination, which would require a bandwidth of $\sum_{s \in S} B_s$ may be part of the mtree. Therefore, for the reserve nothing strategy, if the algorithm outputs a feasible tree, the cost bound provided by lemma 6 is guaranteed. However, for heavy loaded networks, it is likely that the output mtree be unfeasible.

In the reserve everything algorithm, trivially any mtree found is feasible. However, we can not guarantee any cost bounds. This is so because edges belonging to the theoretical minimum cost tree which do not necessarily have bandwidth of $\sum_{s \in S} B_s$ will be ignored in the tree computation. Therefore, the cost of the resulting mtree will have no relation with OPT_c. Notice, however, that the reserve everything algorithm does not require a feasibility check, which is necessary for any other algorithm with overload bounds greater than zero.

We next explore the possibility of doing better than these two strategies. The following remarks about feasible mtrees are due.

Remark 1 *Any source-destination path connecting a given (s, d) pair in a feasible mtree must have at least B_s available in this path.*

Remark 2 *Any source-source or destination-destination pair in a feasible mtree does not necessarily require any available bandwidth in its connecting path.*

Remark 1 is obvious. Remark 2 comes from the fact that in case two sources (destinations) are placed next to each other on a "tip" of a feasible mtree, there is one flow direction in which no bandwidth is necessary. Remark 2 does not help improving the general algorithm, but remark 1 does. Moreover, for this new bandwidth strategy, which we call minimum bandwidth BCST-algorithm (MINBCST), BD-Dijkstra can be used to find the shortest paths between (s, d) with B_s, B_d bandwidth constraints in polynomial time.

So, we can rewrite lemma 6 as:

Lemma 7 *A MINBCST-algorithm outputs a mtree $MT_c \leq \log_2 k \times OPT_c$.*

Proof of Lemma 7 *It follows from remarks 1, 2, and lemma 6.*

We next claim that we can not trade off performance bounds on cost for bounds on bandwidth overload any further. Namely:

Lemma 8 *Any BCST-algorithm with cost bound of* $\log_2 k \times OPT_c$ *cannot have bandwidth overload bound lower than* $\sum_{s \in S} B_s$.

Proof of Lemma 8 *In a destination-destination path, according to remark 2, we must allow for zero bandwidth paths connecting any two multicast members. Clearly, the optimal mtree has an edge next to each destination which requires a bandwidth of* $\sum_{s \in S} B_s$. *Therefore, the overload of such an edge can be as large as this amount.*

Conversely, we have the following lemma:

Lemma 9 *Any BCST-algorithm which requires more bandwidth from paths connecting multicast members than what is prescribed by remarks 1 and 2 can not provide any bounds on the cost of its resulting mtree.*

Proof of Lemma 9 *Consider the last phase of a BCST-algorithm. In connecting the last two clusters, a link next to a given source s can have as little bandwidth as* $(B_s, 0)$, *by the previous remarks. If we require more bandwidth than what is prescribed by these remarks, the link under consideration must be ruled out in the search for our connecting path. All other paths may have cost as large as infinity, which leads to the unboundness result.*

The last two lemmas are somehow complementary, in the sense that if we try to improve cost bounds any further, we cannot claim any bounds on link overload, and vice versa. We are left, therefore, with a family of algorithms, each of which is obtained by varying the amount of bandwidth required by a cluster when a new phase starts. However, the only one over which we can claim hard bounds on cost is the MINBCST-algorithm. In order to study the effect of the amount of bandwidth required at each cluster connecting path, we define the Maximum Bandwidth BCST-algorithm (MAXBCST) as being a BCST-algorithm in which the bandwidth of a cluster is the sum of bandwidths required by the cluster's components. In the next section, we present simulation studies of both algorithms, comparing their average performance in terms of the ability of finding a feasible tree, and the cost of the tree found.

5 Simulation Study

The only non-straightforward step of the BCST-algorithm presented earlier is the minimum cost matching of largest cardinality computation (step (2.2)). Although the minimum cost matching problem is in $PTIME$, we have chosen an approximation algorithm by Goemans at al. [Goemans93], with worst case of twice the minimum cost matching, so that we can solve relatively large multicast tree problems. Their algorithm runs in $O(|V|^2 \log |V|)$ time, which is the same as BD-Dijkstra for all bandwidth constrained shortest path, used in step (2.1). Therefore, a Minimum Bandwidth BCST-algorithm implementation in this section gives us a cost bound of $MT_c \leq 2 \log_2 k \times OPT_c$, where OPT_c is the minimum cost of a feasible Steiner Tree.

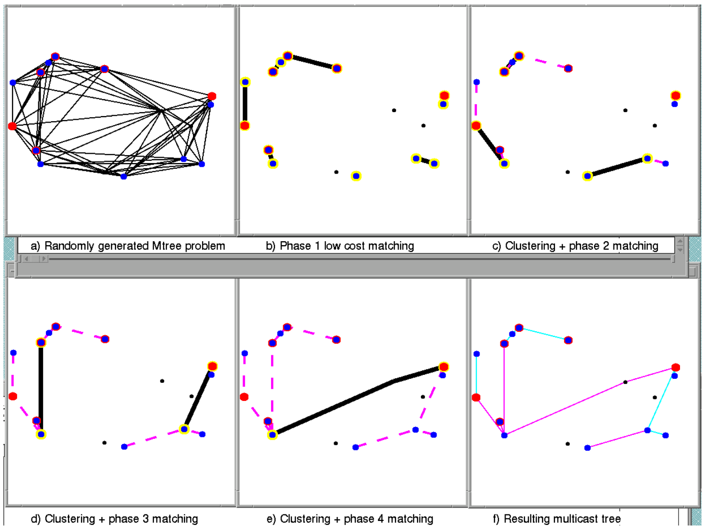


Figure 4: Four phases of a BCST-algorithm

A typical run of the algorithm, for $|V| = n = 16$, is depicted in Figure 4. In the upper left square (a), a randomly generated instance of a 16 node topology is depicted. The next four squares (b,c,d,e) show a sequence of four phases that the algorithm goes through to build the multicast tree.

The smallest dots are network vertices which are neither sources nor destinations. Dots surrounded by a gray circle are cluster heads in a given phase. Links in these middle squares are sets of shortest paths between cluster heads which define a low cost matching. The lower right square (f) depicts the final multicast tree. Although the quality of the figure has suffered greatly for losing its colors, the overall behavior of the algorithm can still be perceived.

5.1 BCST-Algorithms Performance

For simplicity, we have adopted the Euclidean distance between any two vertices i, j as the cost of link (i, j). This also comes handy because the minimum cost matching algorithm used requires that link costs satisfies the triangularity inequality. Each problem instance is generated as follows. $|V| = n$ vertices are randomly placed on a one by one grid. Among $n(n-1)/2$ links of the complete graph formed by the pairwise distances, a connectivity factor C_f is used to prune off the $(1 - C_f) * n(n-1)$ longest links. Edge bandwidths are independently chosen from a $U(B_{min}, B_{max})$ uniform distribution. A vertex is designated as a source with probability p_s and a destination with probability p_d (hence the number of sources and destinations has a binomial distribution). Source traffic rates (i.e, required bandwidths) are chosen from a $U(0, B_s)$, where B_s is computed by:

$$B_s = \frac{B_{min} + B_{max}}{n * p_s} \times h_f \tag{5}$$

Equation 5 is such that h_f gives a percentage of the average link bandwidth per source as compared with the average source bandwidth. In other words, $B_T = n \times B_s/2$ being the total average bandwidth requested by the multicast connection, and $B_e = (B_{min} + B_{max})/2$ the average edge bandwidth, $h_f = B_T/B_e$ is a hardness factor for a given problem. Heavy load and light load situations can be simulated by varying h_f within the $(0, 1)$ interval.

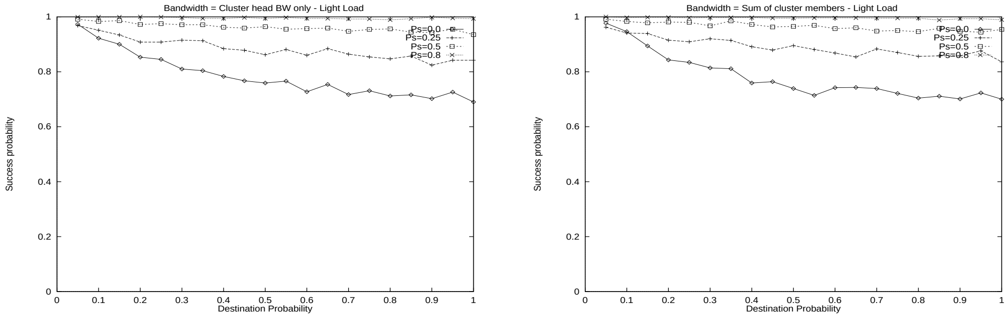
Because instances of the multicast tree problem are randomly generated, in some of the cases a feasible multicast tree may not exist. Since we are attempting to solve an NP-complete problem with BCST, a polynomial time algorithm, we are interested only in cases in which a solution indeed exists. In order to be sure that the problem instance is solvable, we embed a feasible mtree by "adjusting" some edge bandwidths [4]. This allows us to be sure that a failure to find a solution is due to BCST-algorithm behavior, and not to the fact that the algorithm faces an impossible instance of the problem. This procedure, in general, does not affect much the randomness of the problems generated.

On the other hand, ideally, we also want to make assertions about cost, and not only feasibility. Thus, the embedded solution cost should have minimum cost. Although a feasible Steiner Tree can be generated in polynomial time, such a task requires the adjustment of edge costs of potentially the entire graph in order to ensure cost minimality of the embedded tree. This fact, together with the requirement that link costs must obey the triangularity inequality, would "deform" the graph characteristics and randomness greatly (affecting even graph connectivity, since some links would eventually have to be eliminated from the problem in order to ensure that the triangularity inequality holds). Comparisons between various statistically differing cases, then, become difficult. Instead, we have opted for embedding a feasible mtree chosen by computing a Minimum Spanning Tree (MST), and then pruning off the unnecessary branches. Although the resulting Steiner Tree's cost minimality cannot be guaranteed, its cost serves as a comparative measure with the cost of the tree obtained by BCST-algorithms.

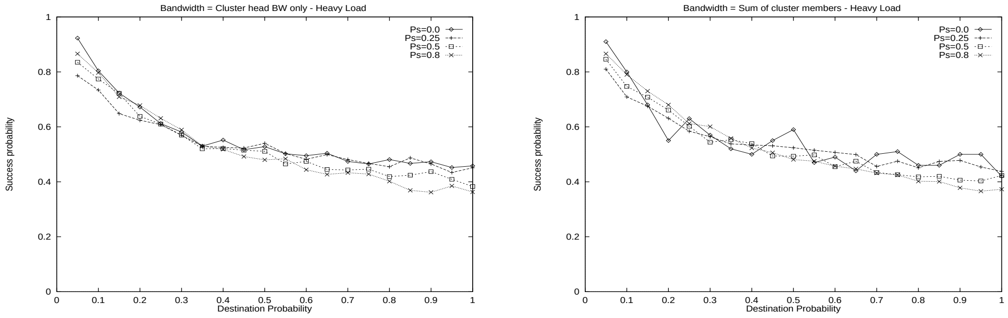
In the set of figures that follow, each point represents an average of 1000 randomly generated instances of size $|V| = 32$ vertices. By varying h_f, we are able to simulate light and heavy network load situations, as mentioned before. By controlling p_s and p_d, we can simulate multicast connections ranging from broadcasting (one-to-many) to video-conferencing (many-to-many). We have also experimented with various graph connectivity degrees. Since graph connectivity seems to have a scaling effect on the comparative results, we present the most representative results only (with fixed graph connectivity), for sake of conciseness.

[4] The algorithm to embed a feasible tree is similar to the one to check feasibility, and is depth first search type of algorithm (so polynomial)

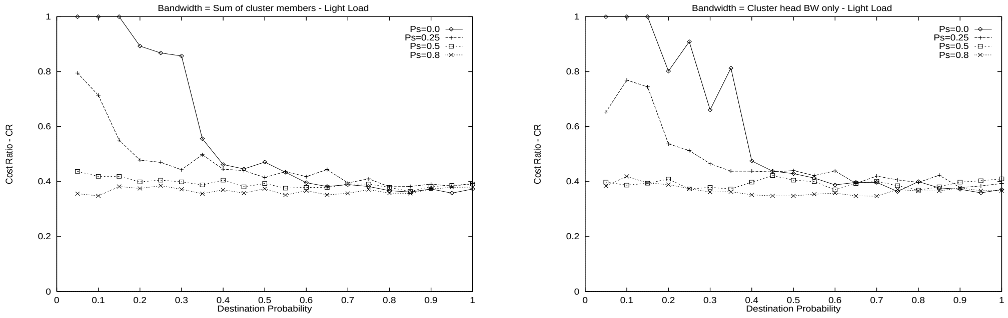
MINBCST MAXBCST



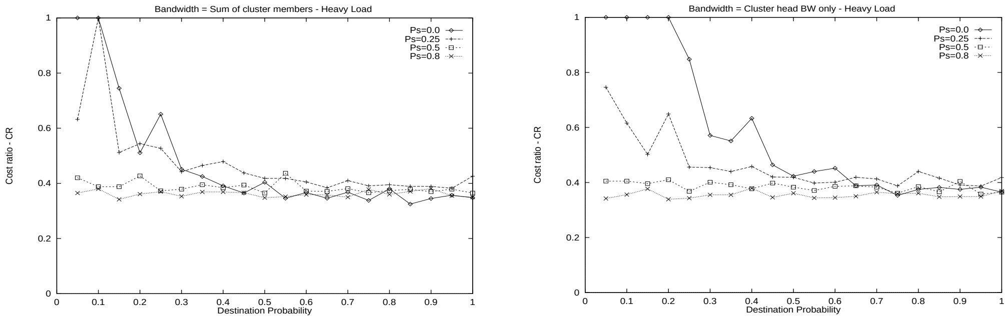
a) Success Rate under light load



b) Success Rate under heavy load



c) Cost Ratio under light load



d) Cost Ratio under heavy load

Figure 5: BCST-Algorithms Performance

Figures 5 shows the performance of MINBCST-algorithm and MAXBCST-algorithm under light and heavy network load. For a given average number of sources (easily obtained from the binomial dirtribution, given p_s), as the number of destinations increases (determined by p_d), the success rate falls, as expected, since one has to build larger trees. Furthermore, the curves for various source probabilities show that, under heavy load (Figure 5b), the number of sources participating in the multicast group does not affect the success rate of the algorithms. Under light load (Figure 5a), however, the impact of additional sources in the success rate is more pronounced. The overall behavior is that the success rate decreases with the size of the multicast group. Notice that both algorithms perform very similarly.

Regarding cost performance, Figures 5 c,d depict in their Y-axis the ratio $CR = \max[MT_c/(\log_2 |S \cup D| \times MT_c^*)]$, where MT_c is the cost of a mtree computed by the BCST-algorithm, MT_c^* is the cost of the embedded mtree, and the max operator is over all problem instances generated. The reason for defining CR this way is two fold: it shows how close our solution is from the embedded mtree; it promptly shows any violation of the cost bounds defined in the previous section [5]. Namely, a $CR > 1$ points to an instance of the problem in which the computed mtree violates the bounds established by lemma 7.

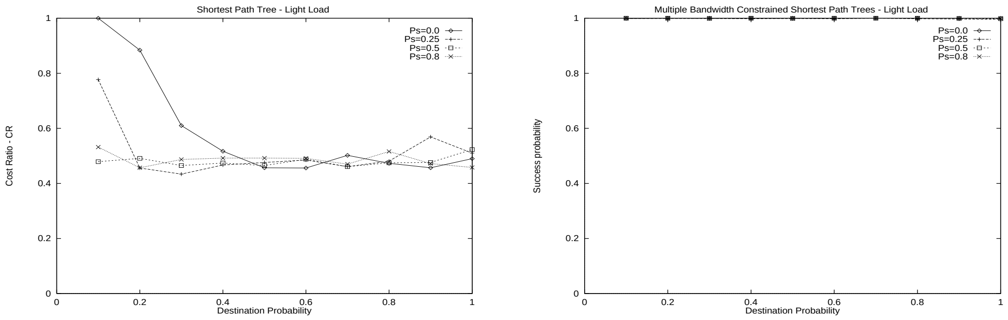
Figures 5 c and d depict the cost ratio of the worst tree found by both MINBCST and MAXBCST algorithms. Notice that the costs presented are of trees which were checked to be feasible only. These figures show that the number of sources has impact on cost only when the number of destinations is relatively small. Basically the larger the size of the multicast group, the cheaper the trees built. Notice that MINBCST-algorithm is more sensitive to the number of multicast members. If we recall that the worst tree ratio is being plotted only, MAXBCST-algorithm smooth behavior for all multicast sizes is quite remarkable.

In general, MINBCST and MAXBCST- algorithms perform very similarly regarding both success rate and cost. Basically, we have noticed that, although MAXBCST-algorithm fails more often on constructing a mtree, the overall success rate is similar to the one presented by MINBCST-algorithm. Surprisingly enough, the cost of the mtrees built by both algorithms are comparable,

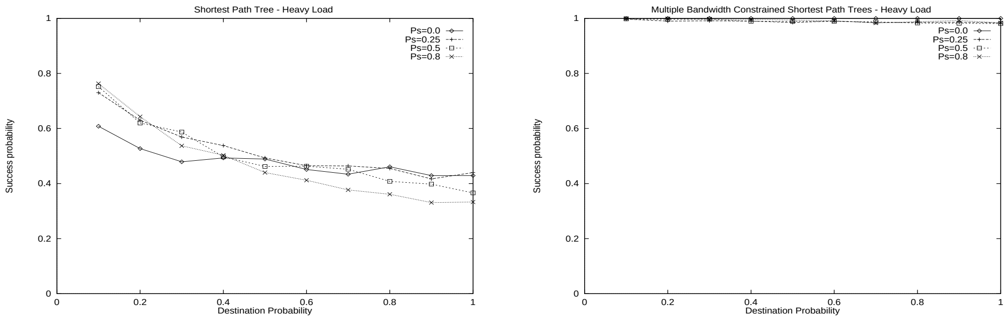
[5] The precise CR definition for the simulation code executed should be $CR = \max[MT_c/(2 \times \log_2 |S \cup D| \times MT_c^*)]$, due to the minimum matching approximation algorithm used. However, as can be inferred from the cost results, a matching very close to the minimum is always found, perhaps due to the graph connectivity factor used

even under heavy load situations. Essentially, the difference between the behavior of these two algorithms lies in the aggregation of bandwidth to form a cluster. Thus, they diverge more in behavior as they progress along their phases. So, in the first phase they are exactly the same, in the second their cluster bandwidths diverge because the MAXBCST-algorithm aggregates all bandwidth of the cluster members into a cluster head, whereas MINBCST-algorithm leaves the cluster bandwidth as the cluster head bandwidth only. Since we can roughly say that, in the second phase, half of the mtree links have been already chosen, due to the log characteristic of both BCST-algorithms, the small difference in the value of their cluster bandwidths will be affecting the selection of half the mtree links only. In the third phase, more bandwidth differences are expected, but they will affect the selection of 1/4 of the mtree links only. Thus the number of link selections affected by the different bandwidth strategies decreases in a log fashion, while the cluster bandwidth differences increase with a power of 2 of the current phase. These two effects "cancel" one another, on average, giving similar performance for both algorithms, independently of the network load condition.

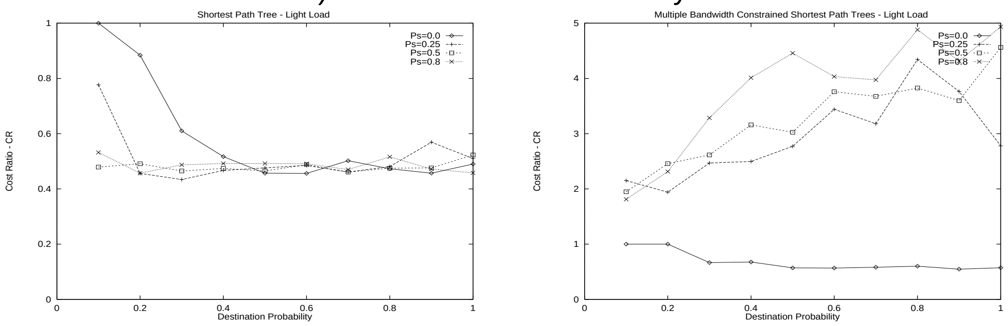
SPTree MSPTree



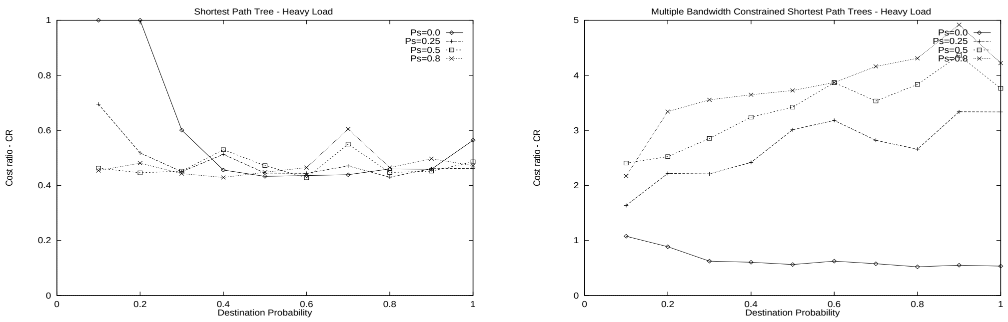
a) Success Rate under light load



b) Success Rate under heavy load



c) Cost Ratio under light load



d) Cost Ratio under heavy load

Figure 6: Alternative Algorithms Performance

For comparative purposes, we have also simulated two popular algorithms for building multicast trees. The first algorithm, hereafter called Shortest Path Tree (SPTree), builds a multicast tree by computing a Shortest Path Tree per multicast member, and picking the minimum cost among these trees. Since bandwidth requirements are not taken into account in the shortest path computation, a feasibility check is performed. Tree costs are reported for feasible trees only. The second algorithm, hereafter called Minimum Spanning Tree with BandWidth constraints (MSTree-BW) builds a Steiner Tree per source (root) by computing a Spanning Tree among links that have available bandwidth greater than the root bandwidth requirement, and pruning off unnecessary branches. Success rates are computed, together with the cost of the set of Steiner Trees computed for each problem instance solved.

Figures 6 a and b show the success rate of both algorithms under heavy and light network load, respectively. Notice that MSPTree-BW has a very impressive success rate, regardless of the size of the multicast group, its composition (source/destination numbers), and network traffic load. However, this ability in finding multicast trees does not come for free. Figures 6 c and d show the ratio of the tree cost vs the cost of the embedded, feasible solution, as earlier defined[6]. Notice that the costs of these multicast trees are extremely high, when compared with our embeded solution.

On the other hand, the success rate of SPTree algorithm is not so good as of MSPTree, due to the fact that SPTree does not pay attention to multicast members bandwidth requirements. It computes, however, much cheaper MCS than MSPTree algorithm. When compared with BCST-algorithms, SPTree algorithm outputs a feasible tree less often (Figs. 5 a and b) and with higher cost, specially for large size multicast groups (Figs. 5 c and d). Overall, BCST-algorithms have performance in between these two heuristics reagarding both cost and success rate.

6 Conclusion and Future Work

In this document, we have analysed the problem of computing a Steiner Tree which is constrained by bandwidth requirements of its connecting vertices. This problem finds applications in multimedia networks, in which members of a multicast connection, with well defined bandwidth re-

[6]Notice the change of scale of the Y-axis for the MSPTree algorithm curves

quirements, wish to communicate. We have presented hard results about the problem, and have used them to propose polynomial time algorithms. In particular, one of the algorithms proposed - MINBCST-algorithm - does have worst case performance guarantees. We have proved that MINBCST-algorithm is the best polynomial time algorithm regarding cost performance. We have also shown trade offs between feasibility and cost measures.

Work is now under way on the implementation of BCST-algorithms in a real time, distributed network environment. We plan to extend the basic algorithm to handle also the dynamic membership problem (join/leave), as well as the problem of resource reuse among sources (eg., in voice conferencing, where bandwidth overbooking is possible on a link basis, since only few speakers can be active at a time).

The problem of finding a bandwidth feasible Steiner Tree without considering cost at all is still an open issue. Although we have proven this to be indeed an NP-Complete problem, the algorithms here presented were driven mainly by a cost minimazation strategy. As exemplified by the Minimum Spanning Tree with bandwidth constraints (MSPTree), other strategies may present better success rates (probability of finding a feasible tree when one exists) than BCST-algorithms if cost is not a concern. A key issue to be probed further is the design of polynomial time algorithms which maximizes success rate regardless of cost.

References

[Cavendish97] D. Cavendish, A. Fei, M. Gerla, R. Rom "On the Construction of Low Cost Multicast Trees with Bandwidth Reservation," *UCLA Technical Report Number 970043*, December 1997.

[Cormen90] Cormen, Leiserson, and Rivest, "Introduction to Algorithms", McGraw-Hill, 1990.

[Deering96] S. Deering et al., "The PIM Architecture for Wide-area Multicast Routing", *IEEE/ACM Transactions on Networking*, vol. 4, pp.153-162, April 1996.

[Feigne96] U. Feigne, "A Threshold of $\ln n$ for Approximating Set Cover", *Proceedings of the 28th Annual ACM Symposium on the Theory of Computation*, 1996.

[Frank85] A. Frank, L. Wittie, and A. Bernstein, "Multicast Communication in Network Computers", *IEEE Software*, Vol. 2, No. 3, pp 49-61, 1985.

[Goemans93] H. N. Gabow, M. X. Goemans and D. P. Williamson, "An Efficient Approximation Algorithm for the Survivable Network Design Problem", *In Proceedings of the Third MPS Conference on Integer Programming and Combinatorial Optimization*, pp. 57-74, 1993.

[Jiang91] X. Jiang, "Path Finding Algorithms for Broadband Multicast", *Third Intl. Conf. on High Speed Networking*, pp. 153-164, North-Holland, 1991.

[Hassin92] R. Hassin, "Approximation Schemes for the Restricted Shortest Path Problem", *Mathematics of Operations Research* , vol. 17, no.1, pp.36-42, 1992.

[Klein95] P. N. Klein and R. Ravi, "A Nearly Best-possible Approximation for Node-weighted Steiner trees", *Journal of Algorithms*, No. 19, pp. 104-115, 1995.

[Kruskal56] J. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", *Proceedings of American Mathematics Society*, vol.7, pp. 48-50, 1956.

[Makki95] K. Makki, N. Pissinou, O. Frieder, "On Multicast in a Communication Network", *Proceedings of the ACM SIGCOMM*, pp. 647-654, 1995.

[Ravi95] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, H. B. Hunt, "Bicriteria Network Design Problems", *Proceedings of 22nd International Colloquium on Automata Languages and Programming*, LNCS 944, pp. 487-498, 1995.

[Ravi93] R. Ravi, M. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, "Many Birds with One Stone: Multi-objective Approximation Algorithms", *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pp. 438-447, 1993.

[Smith83] V. J. Rayward-Smith, "The Computation of Nearly Minimal Steiner Trees in Graphs", *Intl. J. Math. Educ. Sci. Tech.*, Vol. 14(1), pp. 15-23, 1983.

[Wang95] Z. Wang and J. Crowcroft, "Bandwidth-Delay Based Routing Algorithms", *GLOBECOM'95*, pp. 2129-2133.