Degree-Constrained Multicasting in Point-to-Point Networks

Fred Bauer

Anujan Varma

UCSC-CRL-95-08
February 17, 1995

Computer Engineering Department
University of California
Santa Cruz, CA 95064

E-mail: {fred,varma}@cse.ucsc.edu

Abstract

Establishing a multicast tree in a point-to-point network of switch nodes, such as a wide-area ATM network, is often modeled as the NP-complete Steiner problem in networks. In this paper, we study algorithms for finding efficient multicast trees in the presence of constraints on the copying ability of the individual switch nodes in the network. We refer to this problem as the *degree-constrained multicast tree problem* and model it as the *degree-constrained Steiner problem in networks*. Steiner heuristics for the degree-constrained case are proposed and their simulation results for sparse, point-to-point networks are presented. The results are compared with respect to their quality of solution, cost (running time), and the number of test cases for which no solution could be found.

The results of our research indicate that efficient multicast trees can be found in large, sparse networks with small multicast groups even with limited multicast capability in the individual switches. Some of the Steiner heuristics tested yielded degree-constrained multicast trees within 5% of the best heuristic solution found in most of the cases. Even when the fanout of each switch node was restricted to 2, the heuristics we used were able to generate efficient multicast trees in almost all our test networks. Surprisingly few test networks were unsolvable. In those cases where no solution was found by a heuristic, backtracking solved many of the remaining cases. Among the heuristics we used, degree-constrained versions of simple path-distance heuristics such as SPH and SPH-R provided the best tradeoffs between quality of solution and cost.

Keywords: Degree-constrained multicasting, Steiner problem in networks, multicast switch design, ATM multicast.

This research is supported by the Advanced Research Projects Agency (ARPA) under Contract

1 Introduction

Multicasting, defined here as the ability to connect multiple nodes in a point-to-point network by a low-cost tree, is likely to take an increasingly important role in data networks. Many existing networks already support multicasting. For example the Internet MBONE service, a popular conferencing tool, already uses the multicast support recently added to the Internet [7]. Many emerging standards for point-to-point packet-switched networks, notably ATM, Frame Relay and SMDS, include support for multicasting. Future applications such as audio and video conferencing, replicated database updating, and distributed resource discovery will rely on the ability of the network to perform multicast communication. Thus, multicasting will likely be an essential part of future networks.

The cost of multicasting quickly becomes unacceptable for many applications if a separate copy of data is transmitted from the source to each recipient. Transmitting common copies of data over a multicast tree is the preferable method. However, determining the optimal multicast tree for a graph, in general, is a difficult problem. Previous authors have established that the multicast tree problem may be modeled as the *Steiner problem in networks* [3, 4, 5, 12, 27], referred to hereafter as the SPN, and that explicit solutions are prohibitively expensive. For example, two popular explicit algorithms, the *spanning tree enumeration algorithm* and the *dynamic programming algorithm* [27], have algorithmic complexity of $O(p^2 2^{(n-p)} + n^3)$ and $O(n3^p + n^2 2^p + n^3)$, respectively, where n is the number of nodes in the graph and p the number of multicast members. A number of good, inexpensive heuristics exist for the SPN and have been reviewed extensively elsewhere [5, 12, 16, 20, 19, 26, 27]. Some have been shown through analysis to produce solutions no worse than twice the optimal solution [27]. Our empirical evidence indicates that these heuristics find solutions much better than twice the optimal with reasonable speed in most cases.

Finding a multicast tree is complicated by the likely heterogeneous nature of the multicast environment. Switches in a point-to-point network will likely vary in their ability to support multicasting. Some switches may not support multicasting; others may be limited in the number of multicast copies they can reasonably make [29]. The multicast capability of each switch is represented in this paper by a degree-constraint. Thus, a degree constraint of d implies that the corresponding switch is able to forward copies of an incoming packet to a maximum of $d-1$ output ports. The problem of how to create the initial multicast tree when switch multicast capabilities vary is hereafter referred to as the *degree-constrained multicast tree problem*. We model the degree-constrained multicast tree problem as the *degree-constrained Steiner problem in networks* (DCSP), first described by S. Voss [25].

The degree-constrained Steiner problem in networks is of particular interest to high-speed networks since multicast copies must be made quickly, typically by hardware. Whereas a low-speed switch might reasonably approximate infinite copy-capability, high-speed switches may have limited copy capabilities due to their speed constraints. Even when the switches allow multicasting to an arbitrary number of destination ports, there are several advantages in limiting the number of copies made by each switch. For example, some packet-switch architectures implement multicasting by circulating copies of packets through the switch fabric multiple times [22]. Thus, keeping the degree small reduces the number of passes

No. F19628-93-C-0175 and by the NSF Young Investigator Award No. MIP-9257103. A shortened version of this paper appears in *Proc. IEEE INFOCOM '95*, Boston, April 4–6, 1995, pp. 369–376.

needed through the switch fabric. In addition, a degree-constrained multicast tree also distributes the load more evenly among the nodes in the network than an unconstrained tree. This has two benefits: (i) the task of making multicast message copies is shared among more nodes, and (ii) the damage inflicted on the tree by the failure of a single node is reduced.

Formally, the DCSP as used in this paper is defined as follows.

GIVEN: A simple, undirected, connected graph $G = (V, E)$ with n nodes, non-negative edge cost $c_{i,j}$, p multicast members $Z \subseteq V$, and node degree constraints $k_i \geq 2$.

FIND: A multicast tree T such that the degree of each node $d_i \leq k_i$ and its total cost is minimum among all possible choices of the tree satisfying the degree constraints.

The degree-constrained Steiner problem in networks is a relatively new problem and is addressed in only a handful of published papers [13, 21, 23, 24]. The degree-constrained Steiner problem is NP-complete [24] and contains the NP-complete problem of determining a degree-constrained spanning tree [9, 15]. Furthermore, finding a solution to the DCSP is also NP-complete [24]. In practice, however, the heuristics we tested rarely failed to find a solution in our test networks.

Although a multicast tree generated by a DCSP heuristic might be an *optimal* solution to the graph, it need not be. Since finding the optimal tree for a graph is prohibitively expensive, we compare heuristic results for each graph against the best tree found by *any* heuristic. Some heuristics in this paper generate more than one tree. Each such heuristic returns the best tree generated as its solution.

Few DCSP heuristics exist in the literature [21, 24]. A previous paper by Tode, et al. [21] treats the DCSP as an optimization problem. It presents two DCSP heuristics that minimize the *average* degree of the multicast tree. It also includes a discussion of how to modify the heuristics to limit the *maximum* degree in the multicast tree. However, no provision is made for networks of switches with *dissimilar* degree-constraints.

This paper presents several heuristics and simulation results for solving the degree-constrained multicast tree problem on sparse, point-to-point, networks of switches with varying multicast capacity. We believe such models describe typical wide-area networks such as the Internet and future ATM networks. We restrict our analysis to sparse networks for two reasons: (i) they are more representative of real point-to-point networks, and (ii) they are inherently more difficult to solve because fewer alternative Steiner trees exist in a sparse network than in a dense one. Similarly, the simulated multicast groups are small relative to the size of the network, reflecting likely multicast applications. The heuristics compared in this paper are centralized algorithms based on popular published heuristics modified to handle degree-constraints. They are compared on the basis of three criteria: quality of solution, cost (running time), and the number of unsolved networks. Note that these results are not specific to any particular type of point-to-point network such as ATM, but may be applied to any point-to-point network matching these assumptions.

The results of our research indicate that efficient multicast trees can be found in large, sparse networks even with limited multicast capability in the individual switches. Some of the Steiner heuristics tested yielded degree-constrained multicast trees within 5% of the best heuristic solution found in almost all of the networks tested. Surprisingly few test networks were unsolvable. Even with a degree-constraint of 3 (that is, a maximum fanout of 2 per

switch), the heuristics we used were able to generate efficient multicast trees in almost all our test networks. In those cases where no solution was found by a heuristic, backtracking solved many of the remaining cases. Relatively cheap heuristics often produced high quality results while other algorithms of greater complexity rarely justified their additional effort. Understandably, dense networks pose less of a challenge as compared to sparse networks since more alternative paths exist between multicast members in the former. As an example consider the two extremes, a tree and a complete graph. A complete graph has many alternative Steiner trees, some of which may be degree-constrained. A tree, in contrast, has only one Steiner tree and it may not be degree-constrained.

While other constraints such as jitter, delay, and link capacity are important in constructing multicast tress, we focus specifically on the problem of degree-constraints. Papers such as [1, 14, 17] typify published approaches to multicasting with constraints other than the degree of individual nodes. For example, Ammar et al. address the multicast tree problem as a flow problem, allowing for asymmetric capacities between source and destinations [1]; Jiang includes link capacity constraints in his Steiner heuristics [14]; and Kompella et al. include delay constraints in their Steiner heuristics [17].

The remainder of this paper is organized as follows. Section 2 describes the evaluated heuristics. Section 3 discusses the simulation methodology. Section 4 presents the simulation results and compares the heuristics based on these results. Finally, Section 5 concludes the paper with a discussion of the results and directions for future research.

2 Algorithms for the Degree-Constrained Steiner Problem in Networks

This section describes algorithms and methods to solve the DCSP. It begins with a summary of unconstrained Steiner methods and ends with modified heuristics and graph reductions specific to the DCSP.

We use the following definitions in the paper: Z is the set of multicast destinations, S is the set of non-multicast nodes $V - Z$, $P_{i,j}$ is the shortest path between nodes i and j, and $d_{i,j}$ is the distance of the shortest path between nodes i and j. Graph distances will be defined as follows. The distance between nodes is the distance of the shortest path between them. Likewise, the distance between a node and a tree is the distance of the shortest path between the node and any node in the tree. Finally, the distance between two trees is the distance of the shortest among all paths between any node in one tree and any node in the other tree.

Smith and Winter [19] divide Steiner heuristics into a morphological structure similar to the one shown in Figure 1. At the highest level, heuristics are divided between those that are path-distance heuristics (PDH) and others. Path-distance heuristics rely on distance calculations and iteratively improve an initial partial solution using appropriately chosen shortest paths between multicast members until the partial solution contains all the multicast nodes. Path-distance heuristics are further divided between variants of the shortest-path heuristic and other path-distance heuristics. Table 1 summarizes known results on the asymptotic time-complexities of these algorithms, together with the known upper bounds on the ratio of the quality of solution produced to that of an optimal solution.

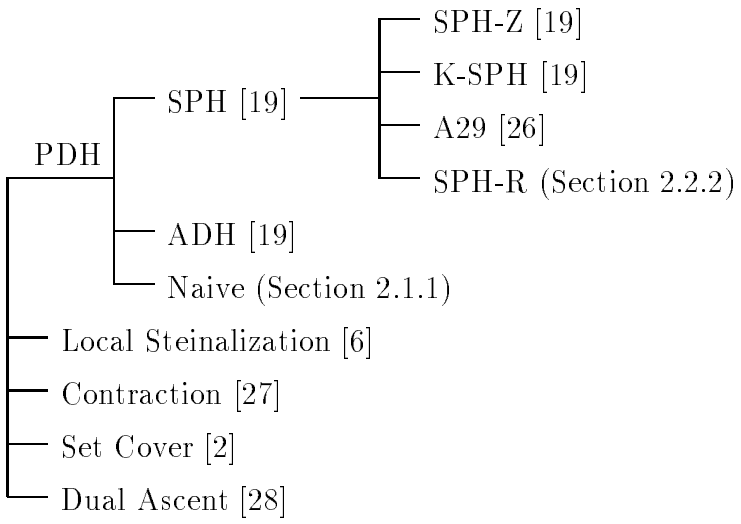

Figure 1: Morphological structure of Steiner heuristics.

2.1 Unconstrained Steiner Tree Heuristics

Many Steiner heuristics have been proposed in the literature [12, 19, 26, 27] and form a suitable basis for DCSP-specific heuristics. In this section, we first examine the heuristics for the unconstrained Steiner problem listed in Table 1. These heuristics are then modified to handle degree constraints and compared by way of simulation.

Heuristic	Time Complexity	Solution Bound
SPH	$O(pn^2)$	$2X$
SPH-Z	$O(p^2n^2)$	$2X$
K-SPH	$O(n^3)$	$2X$
ADH	$O(n^3)$	$2(1-\frac{1}{p})X$
Dual Ascent	N.A.	N.A.
Set Cover	N.A.	N.A.
Local Steinalization	N.A.	N.A.
Contraction	N.A.	$2(1-\frac{1}{p})X$

Table 1: Characteristics of published Steiner heuristics.

2.1.1 Heuristic Naive

For comparison sake, we first consider the simplest possible Steiner algorithm that we name *Naive*. Heuristic Naive starts with an arbitrary multicast member as the multicast tree. It then repeatedly connects another random multicast member to the multicast tree by the shortest path between the new member and the multicast tree until all the members are in the multicast tree. We expected this heuristic to give the worst results; as it sometimes does. However, we discovered that it often produces quite respectable results. This matches Doar's results [8]. We treat heuristic Naive as the baseline from which to compare other Steiner heuristics. We describe heuristic Naive as follows.

1. Initialize subtree T to an arbitrary Z-node.
2. Connect subtree T with an arbitrary Z-node $\notin T$ by the shortest path.
3. If a Z-node exists $\notin T$, go to step 2.

2.1.2 Shortest Path Heuristic (SPH)

The shortest-path heuristic [20] produces surprisingly good results and has many variants as shown in Figure 1. SPH initializes the multicast tree to an arbitrary multicast member. It then joins the next closest multicast member to the multicast tree by the shortest path between the multicast member and the tree. The algorithm terminates when all members have joined the tree. Note that this algorithm differs from heuristic Naive because multicast members join in the order determined by their distance to the multicast tree, rather than in random order.

This solution may be improved even further using the following technique [19]. Find the graph induced by the nodes in the solution, derive the induced graph's minimum spanning tree using a method such as Prim's algorithm, and prune the minimum spanning tree of non-multicast member leaves. The resulting tree is an improved solution. This improvement is equally applicable to any of the SPH variants.

The basic SPH algorithm is described as follows:

1. Initialize subtree T to an arbitrary Z-node.
2. Connect subtree T with the closest Z-node $\notin T$
3. If a Z-node $\notin T$ exists, go to step 2.
4. Further improve the solution
   (a) Let graph G' be the subgraph of G induced by T's nodes. Let U be a minimum spanning tree of G'.
   (b) Repeatedly prune S-leaves of U.

2.1.3 Heuristic SPH-Z

This variant of SPH applies the basic SPH algorithm described in Section 2.1.2 once for each possible choice of the starting Z-node, returning the best solution found.

2.1.4 Heuristic K-SPH

Unlike the previous heuristics, K-SPH, the Kruskal-based shortest-path heuristic [18], starts with the forest of multicast member nodes. It repeatedly joins the two closest multicast member subtrees until a single tree spanning all multicast members remains. K-SPH's algorithm is described as follows.

1. Initialize T to be the forest of Z-nodes.
2. Connect the two closest subtrees in T by their shortest path.
3. If T is disconnected, go to step 2.
4. Improve the solution by constructing a minimum spanning tree of the subgraph induced by T's vertices in G and pruning its S-leaves.

2.1.5 Heuristic ADH

The average distance heuristic is a generalization of K-SPH. Like K-SPH, the algorithm starts with the forest of multicast member nodes. It repeatedly connects the *three* closest multicast member components through the *most central* node (defined to be the node with the least average distance to all multicast members). ADH terminates when a single tree remains, spanning all multicast members. Its algorithm is described as follows:

1. Initialize T to be the forest of Z-nodes.
2. Connect the closest and the second-closest subtrees in T to the most central node by their shortest paths.

   The most central node is determined as follows: For each node v, order the subtrees of T by their distance to node v in nondecreasing order $T_1, T_2, \cdots T_k$. Let d_i represent the distance between v and T_i. The most central node is the node with the smallest value for

   $$f(v) = \min_{2 \le r \le k} \left\{ \sum_{i=1}^{r} \frac{d_i}{r-1} \right\} = \min \left(d_1 + d_2, \frac{d_1 + d_2 + d_3}{2}, \cdots, \frac{d_1 + d_2 + d_3 + \cdots + d_k}{k-1} \right).$$

   Note that if v belongs to a subtree, that subtree will be first on the list with $d_1 = 0$. Fortunately $f(v)$ need not be evaluated in full for each node. If $v \in Z$, $f(v) = d_2$. If $v \in S$,

   $$f(v) = \frac{d_1 + d_2 + \cdots + d_r}{r-1}$$

   for the smallest value of r such that

   $$\frac{d_1 + d_2 + \cdots + d_r}{r-1} < \frac{d_1 + d_2 + \cdots + d_{r+1}}{r}.$$

3. If T is disconnected, go to step 2.
4. Improve the solution as discussed in step 4 of the SPH heuristic.

2.1.6 Heuristic Dual Ascent

Wong [28] describes an exact solution and a heuristic to the SPN. The Dual Ascent heuristic finds a solution to the SPN in five steps:

1. First, convert the undirected graph G into a directed graph G' by substituting every undirected edge by two directed edges of equal weight in opposite directions.

2. Next, build a directed subgraph A from G' which contains a solution. This step terminates when at least one multicast node is *connected* to every other multicast member. Note that in directed graphs, node i is connected to node j if a directed path exists from i to j. Further, if two nodes are connected to each other, they are said to be *strongly connected*. The steps to build auxiliary graph A are as follows.

   (a) Create directed subgraph A with all the nodes of directed graph G', but none of the edges.
   Let $G' = (V, E)$. Let $A = (V, \emptyset)$.

   (b) Initialize cost matrix S to the edge weights of G'.
   For all edges of G', set $s_{i,j} = c_{i,j}$.

   (c) Pick a *root component* R of A.
   A root component is a *maximal strongly connected component* that contains a Z-node and from which no Z-node *dangles*. A node dangles from a component if the node is connected to the component, but the reverse is not true. A strongly connected component is a set of nodes in which every node in the set is strongly connected to every other node in the set. A maximal strongly connected component is a strongly connected component that is not a subset of any other strongly connected component.

   (d) Pick a multicast member z from the root component R.

   (e) Pick the cheapest edge (m, n) out of z's cut set.
   The cut set of z are those edges not yet chosen for A which bridge a node connected to z to a node not connected to z in A. Let $C(z)$ be the set of nodes connected to node z. The cut set for z is

   $$E(z) = \{(i,j) | (i,j) \in G', (i,j) \notin A, i \notin C(z), j \in C(z)\}.$$

   Choose (m, n) such that $s_{m,n} = \min\{s_{i,j} | (i,j) \in E(z)\}$

   (f) Having found the cheapest cut-set edge, deduct its cost from z's remaining cut set edges.
   For all edges in $E(z)$, set $s_{i,j} = s_{i,j} - s_{m,n}$.

   (g) Update subgraph A by adding the minimum cut set edge.
   $A = A \bigcup (m, n)$

   (h) If root components remain, go to step 2(c).

3. Next, convert A to an undirected graph U.

   Start with $U = (V, \emptyset)$. For every directed edge in A, add an equivalent, undirected edge to U. This will make U a multigraph. Delete all redundant edges. Note that all edges between the same pair of nodes will have the same weight.

4. Next, find U's minimum spanning tree T.

   If U is a small enough subgraph, the minimum spanning tree represents a reasonable approximation of the Steiner tree. However, if U is not significantly smaller than G', then the minimum spanning tree may be a poor approximation of the Steiner tree. We know this to be true since simply taking the spanning tree and pruning it to be the Steiner tree can, in the worst case, produce a solution that departs from the optimal by a factor of $|S| + 1$ [26].

5. Finally, prune all non-multicast member leaves from T. The result is the solution.

2.1.7 Local Steinalization Heuristic

Chen [6] describes a method to determine the optimal node through which to connect *three* multicast members. Using this method recursively, he outlines two heuristics, TPS1 and TPS2, which find Steiner solutions. Heuristic TPS1, like SPH, expands an initial solution one component at a time until all multicast members are part of the solution. Heuristic TPS2, like K-SPH, starts with a forest of multicast members and connect the closest three components until all multicast members are part of a single multicast tree.

The optimal node to connect the three components must lie within an area defined by the shortest-path distances among the three components. Given three components A, B and C, finding the optimal node to interconnect them involves the following steps:

1. Determine the shortest path $P_{A,B}$ between components A and B.

2. Determine the shortest path $P_{C,d}$ between component C and path $P_{A,B}$.

   Let d be the endpoint of path $P_{C,d}$ which lies on path $P_{A,B}$.

3. Find the optimal node i to connect components A, B, and C.

   Let $d_{i,j}$ be the distance along the shortest path between i and j. Choose node i such that

$$\min\{d_{A,i} + d_{B,i} + d_{C,i} \mid d_{A,i} \leq 2d_{A,B}, d_{B,i} \leq 2d_{A,B}, \text{and } d_{C,i} \leq d_{C,d}\}.$$

Heuristic TPS1 expands an initial multicast tree one multicast member at a time until the multicast tree includes all multicast members. Each additional multicast member is added to the tree by breaking the existing multicast tree into two components and combining with the two old multicast components with the new multicast node through the optimal intermediate node.

Heuristic TPS2 connects a forest of multicast members by building a single tree out of the forest, three components at a time. Each set of three components is connected through an optimal node determined as above. If, at the end, only two components remain, they are connected using the shortest path between them.

2.1.8 Set Cover Heuristic

Aneja [2] describes an integer linear programming approach to the SPN by posing the SPN as the *set cover problem*. Given a finite set of elements X and a number of subsets of those elements F, the set cover problem attempts to find a minimum set of subsets $C \subseteq F$ that includes all the elements of X. Such a minimum set C is called a *cover*. A cover is also a *non-redundant cover* if no proper subset is also a cover.

The algorithm repeatedly solves set cover problems, using an evaluation function on each successive solution to determine if a better solution exists. If so, it adds constraints to the set cover problem and solves again. As the algorithm converges on an optimal solution to the set cover problem, it also converges on an optimal solution to the SPN. In its unaltered form, this algorithm is an explicit SPN algorithm. The Set Cover heuristic accelerates the rate of convergence by modified edge weights and adding constraints. The heuristic result is a (sub-optimal) solution. Unfortunately, this algorithm is very complex and requires many optimizations to run efficiently. Because it is so complex and because other, simpler heuristics produced good solutions, we elected to leave this heuristic out of our evaluation.

2.1.9 Contraction Heuristic

The Contraction heuristic [27] recursively collapses *neighborhoods classes* of multicast nodes into a single multicast super-node. A neighborhood of a node i is defined to be the nodes of G that lie within a given distance r from i. Two neighborhoods are *reachable* from each other if they are adjacent or if there is a path between them that lies entirely within neighborhoods of multicast nodes. A *neighborhood class* is a maximal set of multicast neighborhoods reachable from one another.

Using the cost of the cheapest edge adjacent to a multicast node as the given distance r, the algorithm repeatedly collapses each neighborhood class defined by r into a super-node. When only one such super-node remains, the heuristic solves each previously defined neighborhood class using another Steiner heuristic and pieces together the solution from the resulting Steiner subtrees. An overview of the heuristic follows.

1. Let $r =$ the least cost of any edge adjacent to a Z-node in G.
2. Let $C_1, C_2, \cdots C_t$ denote the t neighborhood classes of Z-nodes.
3. If $t > 1$, modify G by contracting the t neighborhoods. Go to step 1.
4. Determine the solution
   (a) Find the Steiner subtree for each neighborhood class found using another Steiner heuristic
   (b) Assemble the Steiner tree out of the neighborhood class subtrees

2.2 Degree-Constrained Steiner Tree Heuristics

The eight DCSP heuristics compared in this paper are a mix of unconstrained Steiner heuristics and heuristics specific to the DCSP. The unconstrained heuristics are essentially unchanged aside from the exceptions listed below. However, a number of factors differentiate

s_1	s_2	s_3	s_4	s_5
$s_1 \in Z$				
$s_1 \notin G$				
$s_1 \in Z$	$s_2 \in Z$			
$s_1 \in Z$	$s_2 \notin G$			
$s_1 \notin G$	$s_2 \in Z$			
$s_1 \notin G$	$s_2 \notin G$			
$\vdots$				
$s_1 \notin G$	$s_2 \notin G$	$s_3 \notin G$	$s_4 \notin G$	$s_5 \notin G$

Table 2: Subgraphs examined during backtracking.

Steiner heuristics running in a constrained environment from those in an unconstrained environment.

First, connecting more than two components is much more complicated in the constrained case than the unconstrained one. For example, in heuristic ADH, three components are merged at each step. In the unconstrained case, it is sufficient to track the shortest path between each component and the central node. In the constrained case, however, one or more such shortest paths may exhaust the degree-constraint of nodes in the connecting tree, rendering it infeasible. In short, connecting greater than two components becomes a smaller version of the DCSP with all of its complications. Thus, each heuristic used in this paper merges at most two components of a graph together by their shortest path.

Second, the topology of the network graph G may change during a heuristic because of the partial solution generated so far. This is because a node's degree-constraint may be exhausted by a partial solution, eliminating the node and its remaining edges from consideration. This changed topology can and does alter the shortest path information for remaining steps. As a consequence, heuristics must constantly reevaluate shortest paths between nodes.

Third, since finding a solution to the DCSP is NP-complete [24], a given DCSP heuristic may not be able to find a solution even though one exists. Consequently, all of our DCSP heuristics employ backtracking. Our backtracking strategy is inspired by Beasley's Lagrangean relaxation approach [3] in which Beasley applies a Steiner heuristic to each subgraph of G created by forcing every non-member node in or out of the solution. To force a non-member node to be in a solution, Beasley converts it temporarily to a member node. To force a non-member node out of any solution, Beasley temporarily deletes that node from graph G. To examine every such combination, Beasley's explicit algorithm looks at 2^s subgraphs where s is the number of non-multicast nodes. Since non-multicast nodes are the majority of nodes in our networks, we cannot use this approach. Instead, we use only five "central" non-multicast nodes $s_1, s_2, \ldots, s_5$, chosen such that their average shortest-path distances to the multicast members are the lowest among all the non-multicast nodes. During backtracking, we examine the $2^1 + 2^2 + \cdots + 2^5 = 62$ subgraphs created by forcing these nodes in or out of the solution, as shown in Table 2. If no solution is found in these subgraphs, the heuristic terminates without a solution.

2.2.1 Modified Steiner Tree Heuristics

Although the modified Steiner heuristics in this section are similar to their unconstrained equivalents, they differ in at least one important way. All DCSP heuristics must frequently reevaluate shortest path information when building a solution as explained in Section 2.2. Most of the heuristics needed no further modification with the exceptions noted below:

ADH: ADH connects *only* the closest subtree to the most central node. This is because of the difficulty of connecting greater than two components as explained in Section 2.2. ADH's step 2 becomes:

2. Connect the closest subtree to the most central node v by *its* shortest path.

Dual Ascent: Modified Dual Ascent differs from it unconstrained equivalent because it does not find the degree-constrained minimum spanning tree to its subgraph. Instead, it uses SPH to generate a Steiner tree from the subgraph. Thus, this Dual Ascent step becomes:

4. Next, Use heuristic SPH to find a Steiner tree T for U.

Local Steinalization: Because the Local Steinalization heuristic combines three components at a time and because of the difficulties of combining multiple components outlined in Section 2.2, this heuristic is not included in the simulations.

Set Cover: As described in Section 2.1.8, the Set Cover heuristic is not included in our simulation because of the sheer complexity of the algorithm. Ultimately, it is doubtful whether the quality of solution for the Set Cover heuristic would justify its cost.

2.2.2 Constrained Heuristics

In addition to the above unconstrained heuristics, we simulate two DCSP-specific heuristics, A29 and SPH-R. Heuristic A29 was first proposed by Voss [24] and heuristic SPH-R is our own contribution. They are described below.

A29: Heuristic A29 [24] is the only published DCSP heuristic of which we are aware. It is also a variant of SPH. It first adds enough edges of infinite weight to make G a complete graph and then applies heuristic SPH. Since G is complete, A29 will always find a degree-constrained Steiner tree, although the solution may contain infeasible edges (edges of infinite weight). A29's algorithm is described as follows.

1. Construct the complete graph G'. Let every edge (i, j) common to both graphs G and G' have equal cost. Let all other edges have cost ∞.

2. Apply heuristic SPH to G'.

3. Delete all edges in T of cost ∞.

   Note that T may not be connected and may thus be infeasible.

SPH-R:

Heuristic SPH-R is our own variation of SPH. SPH-R like SPH-Z repeatedly applies SPH to the graph G for different starting points. However, SPH-R terminates the first time it generates a solution. We observed that SPH-R rarely iterates more than a few times.

2.3 Graph Reductions

Before applying a heuristic, a graph may be examined for easily solved subgraphs and modified so as to reduce the run-time of the heuristic. These modifications based on easily solved subgraphs are referred to as graph reductions. The strategy employed is to identify such subgraphs, solve them, save their results, and track the cost of the partial solution. The modifications reduce the size of the initial graph, reducing subsequent heuristic run-time. Our empirical evidence suggests that on sparse networks even simple graph reductions may reduce their size as much as 15%. Graph reductions have been extensively reviewed elsewhere [10, 11, 27].

A short list of such reductions follows.

1. S degree 1.

   A non-member node that is a leaf of graph G cannot be a part of the solution and may be deleted. Thus, if G contains S-nodes of degree 1, delete each such node and its adjacent edge. Note that this reduction reduces the degree of the neighbor node by one and may convert a formerly degree-constrained node into an unconstrained one.

2. Z degree 1.

   A leaf Z-node and its adjacent edge must be part of the Steiner tree. Let y denote the single neighbor of such a node. Thus, if G contains a Z-node z of degree 1, take the following steps.

   (a) Delete node z and edge (z, y) from G.

   (b) Save node z and its adjacent edge for use later in the solution.

   (c) If neighbor node y is a S-node, convert it to be a Z-node.
   This step ensures that G's solution will also include node y, solution edge (z, y)'s connection point.

   (d) Decrease node y's degree constraint by one.
   This step reserves node y's degree-constraint for later use when constructing the solution. Note that if node y's degree-constraint falls below zero, no solution exists.

   (e) After finding a solution using a DCSP heuristic, reconnect the saved edges and Z-nodes to the solution and reconvert former S-nodes.

3. S degree 2.

   If an S-node connects exactly two nodes, it may be replaced by an edge. Thus, if G contains an S-node s of degree 2 connecting two nodes x and y, replace the node s and the two edges connecting x and y through node s by an edge (x, y) with cost $c_{x,y} = c_{x,s} + c_{s,y}$. If, as a result of this addition, two edges now connect nodes x and y, delete the more expensive edge. Note that this reduction reduces the degree of the two neighbor nodes x and y by one and may convert either or both from a degree-constrained node into an unconstrained one.

4. Reachability Test (RT).

   Unlike the previous reductions, the reachability test improves upon a solution by eliminated those non-multicast members that cannot be part of a better solution. Consequently, this reduction is not a pre-processing step, but rather an improvement of an existing DCSP heuristic's solution. It eliminates nodes and edges by examining every node outside the solution and eliminating those whose distance to multicast nodes makes them ineligible to be in the solution. This distance criterion is based on a non-multicast member's distance to the closest, second closest and farthest multicast member.

   Formally, let c_f be the cost of a solution, let U be its nodes, and let c_t be the cost of the partial solution collected so far. For any such $i \in V - U$, let

$$c_{max} = \max\{c_{i,z}, z \in Z\};$$

$$c_{min} = \min\{c_{i,z}, z \in Z\};$$

$$c_{2nd} = \min\{c_{i,z}, z \in Z - \{z_{min}\}\}.$$

   Node i and its adjacent edges may be eliminated if

$$c_t + c_{max} + c_{min} + c_{2nd} \geq c_f.$$

Each of the graphs used in our simulations were first reduced by the following three reductions: S degree 1, Z degree 1 and S degree 2. Reduction RT was discarded because it was both expensive and ineffective. It was expensive because it required a solution to start with. Furthermore, it failed to eliminate even a single node in the test networks simulated. Detailed results on the effectiveness of the reductions are presented in Section 4.

3 Evaluation Methodology

Thus far, the discussion has centered on descriptions of degree-constrained Steiner heuristics and graph reductions. Here, we turn to our evaluation methodology. In this section we describe our network model and our heuristic simulator.

3.1 Network model

Because our choice of existing networks and multicast applications was small, we chose to compare DCSP heuristics using randomly generated networks. Each of the heuristics was run on a total of 2000 test networks. Each of the 2000 networks is a sparse 200-node network with a degree-constraint on 50 or 75% of its nodes. We consider an n-node graph to be sparse when less than 5% of the possible $\binom{n}{2}$ edges are present in the graph. We believe such a graph describes a plausible network of point-to-point nodes in a WAN because a large multicast network is loosely interconnected and may have many degree-constrained nodes. Likewise, the simulated networks have 10 or 30% of its nodes in the multicast group because

multicast applications running on such a WAN are likely to involve only a minority of nodes in the network.

The 2000 networks were generated to resemble networks in a manner similar that of Doar [8]. Each of n nodes is distributed across a Cartesian coordinate plane with minimum and maximum coordinates $(0,0)$ and $(2n,2n)$, creating a forest of n nodes spread across this plane. In all of our test graphs, the number of nodes n was set at 200. The nodes are then connected by a random spanning tree. This tree is generated by iteratively considering a random edge between nodes and accepting those edges that connect distinct components. The remaining redundant edges of the graph are chosen by examining each possible edge (x,y) and generating a random number $0 \leq r < 1$. If r is less than a probability function $P(x,y)$ based on the distance between x and y, then the edge is accepted. The weight of each edge is its rectilinear distance plus a small constant. We used the probability function

$$P(x,y) = \beta e^{\frac{-d_{x,y}}{2\alpha n}},$$

where $d_{x,y}$ is the rectilinear distance between nodes x and y [8]. The parameters α and β govern the density of the graph. Increasing α increases the number of connections to nodes far away and increasing β increases the number of edges from each node. After some experimentation, the graphs in this simulation were generated using $\alpha = 0.10$ and $\beta = 0.20$. These values produced graphs of realistic density and degree-distribution.

We performed four different simulations on each generated graph by varying the multicast group size and percentage of degree-constrained nodes, each in two ways. The number of multicast nodes was chosen as either 20 or 60 of the 200 nodes; the number of degree-constrained nodes was selected as either 100 or 150. Results are presented for all four combinations for each graph. Degree-constrained nodes in each case were chosen randomly and assigned a random degree constraint between 2 (no multicast capability) and one less than the degree of the node. Similarly the nodes in a multicast group are chosen randomly in each case. The random numbers were chosen from a uniform distribution. To ensure fairness, each heuristic was run on the same 2000 networks.

In additions to the above simulations where the degree-constraints of individual nodes were varied randomly, we also performed a set of experiments by limiting the degree constraint of each node in the network to 3 (that is, at most one additional copy per node). The goal of these experiments was to investigate whether the heuristics are able to generate efficient solutions even with very limited multicast capability in the individual nodes. These experiments were run on the same 200-node graphs described in the previous paragraphs.

3.2 The Heuristic Simulator

Each of the heuristics was implemented on top of our degree-constrained Steiner problem simulation platform, designed to provide the level playing field upon which to base comparisons. It supplies the basic graph manipulation routines used by the heuristics such as adding and deleting edges, and is written in MAINSAIL, a machine-independent ALGOL-like language. Since the simulator runs on a variety of platforms, including SUN, DEC and IBM workstations, CPU time was a poor metric upon which to base comparisons. Instead, we use source language statement counts as supplied by MAINSAIL's profiler, excluding

operating system sensitive operations such as file I/O. The heuristics compared use identical source code on all platforms. Because all the heuristics share much of the same code, statement counts provide a means to measure relative computing cost of each heuristic while minimizing the influence of coding style.

Each simulated heuristic uses the same steps to insure uniformity of solution: First, the network is evaluated for trivial cases; it is then pre-processed using the graph reductions Z degree 1, S degree 1, and S degree 2; finally, the heuristic is run on the reduced graph.

4 Simulation Results

In this section we present the results from simulations of the heuristics for both degree-constrained and unconstrained cases on 2000 test networks. These heuristics are compared on the basis of three criteria: cost, quality of solution, and number of cases in which the heuristic was unable to find a solution. Cost is the run-time of each heuristic measured in source-level (MAINSAIL) language statements. The quality of solution is measured by the ratio of the cost of the heuristic's solution (the sum of the edge weights) to the cost of the best solution found among all the heuristics simulated. The quality of solution for a heuristic is compared with the best solution found because (i) the cost of finding an optimal solution for a 200-node graph is prohibitive and (ii) the optimal solutions we found for a small number of graphs using Beasley's Lagrangean Relaxation algorithm [3] often had the same cost as the the best solution found by the heuristics. The last criterion gives the number of networks for which a heuristic could not find a solution even with backtracking.

To verify and calibrate simulation results, we first compared our results obtained from running unconstrained versions of the heuristics with those published by Smith [19] and Voss [26]. Of the heuristics compared, only one — Dual Ascent — did not match the published results. That only one heuristic differed significantly from published results is interesting since our simulations focused on a much narrower range of networks than those considered by Smith or Voss. Each of our networks is large, sparse and has a small multicast group. By contrast, Smith and Voss simulated a mix of sparse and dense networks with small to large multicast groups. In Voss' simulations [26], heuristic Dual Ascent found the best solution as much as 80% of time. In our simulations, however, the algorithm rarely produced the best solution; in some cases the solutions were as much as 15% off when compared with the best solution among all the heuristics. This behavior is due to two reasons: First, our simulation networks differ from Voss's networks in that they are always sparse, reducing the number of choices possible when building the subgraph in the Dual Ascent heuristic. The subgraph often contained as many nodes as the original graph, reducing Dual Ascent's effectiveness. Second, all of our sample networks have 200 nodes with a small multicast group-size — 20 or 60 nodes. Voss' simulation results are based on sample graphs with 20 to 60 nodes and a wide distribution of multicast sizes. This is important because Dual Ascent uses a minimum spanning tree (MST) of the subgraph to find a Steiner tree. Previously published results have shown that the approach based on pruning the MST works well for large multicast groups, but poorly for small groups [26]. The published worst-case ratio between a solution using the MST and an optimal solution is $|S|+1$ where $|S|$ is the number of non-multicast members. Thus, it is not surprising that Dual Ascent's quality of solution

Heuristic	Number of solved networks without backtracking	Number of solved networks with backtracking	Number of unsolved networks
SPH-Z	1999	0	1
SPH-R	1999	0	1
SPH	1995	0	5
A29	1995	0	5
ADH	1992	1	7
Naive	1990	9	1
K-SPH	1970	23	7
Dual Ascent	1951	42	7

Table 3: Success rates of degree-constrained heuristics on 2000 test networks.

was poor for sparse 200-node networks with 20 or 60 multicast members.

Table 3 summarizes the success rates of the degree-constrained heuristics based on simulations on 2000 test networks. As expected, heuristics run on the sample networks with degree-constraints sometimes had difficulty in finding solutions. This is because finding a solution for the degree-constrained case is also NP-complete [24]. In practice, however, most of the degree-constrained graphs could be solved by the heuristics directly or through backtracking. In fact, each of the 2000 test networks could be solved by at least one of the heuristics. The maximum number of cases missed by any single heuristic was 7.

To further verify the ability of the heuristics to find solution with more stringent degree constraints, we also tested the heuristics on some of the graphs by imposing a fixed degree-constraint of 3 for each node. Table 4 summarizes the success rates of these simulations on 1000 test networks. As expected, both the number of cases in which backtracking was needed and the number of unsolved cases increased as a result of the small degree constraint. However, the increases were barely noticeable in most cases. Indeed, SPH-R and SPH-Z solved all 1000 test networks even without backtracking. This leads us to conclude that some of the tested heuristics will be able find multicast trees in large, sparse networks even with limited multicast fanout in the individual nodes.

4.1 Cost and Quality of Solution

Since the differences in the success rates of the individual heuristics were small, the remaining two criteria — quality of solution and cost — then become the more important factors to consider. Figures 2 and 3 summarize the tradeoff between these two criteria. The results cluster into five distinct groups: simple shortest-path heuristics SPH, SPH-R and A29; repetitive shortest path heuristic SPH-Z; Kruskal-based shortest-path heuristics K-SPH and ADH; heuristic Dual Ascent; and heuristic Naive. Tables 5 and 6 summarize how often each heuristic produced a solution equal to the best heuristic solution found and Figures 4 and 5 show quality of solution distributions for the degree-constrained and unconstrained cases.

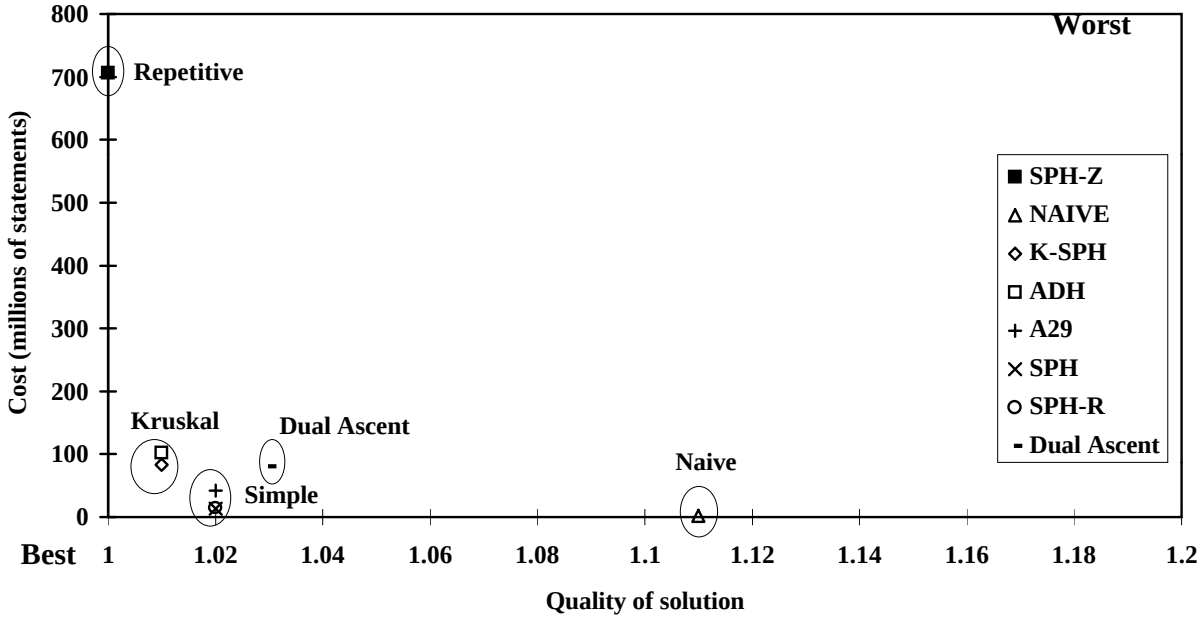

Figure 2: Cost vs. quality of solution for degree-constrained multicast.

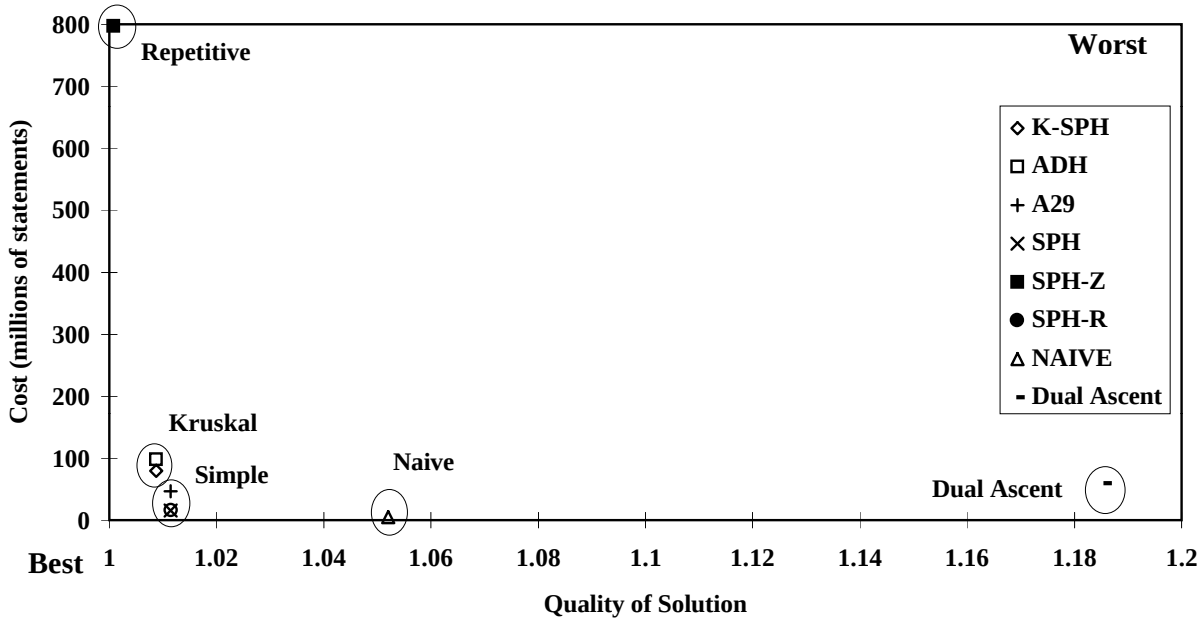

Figure 3: Cost vs. quality of solution for unconstrained multicast.

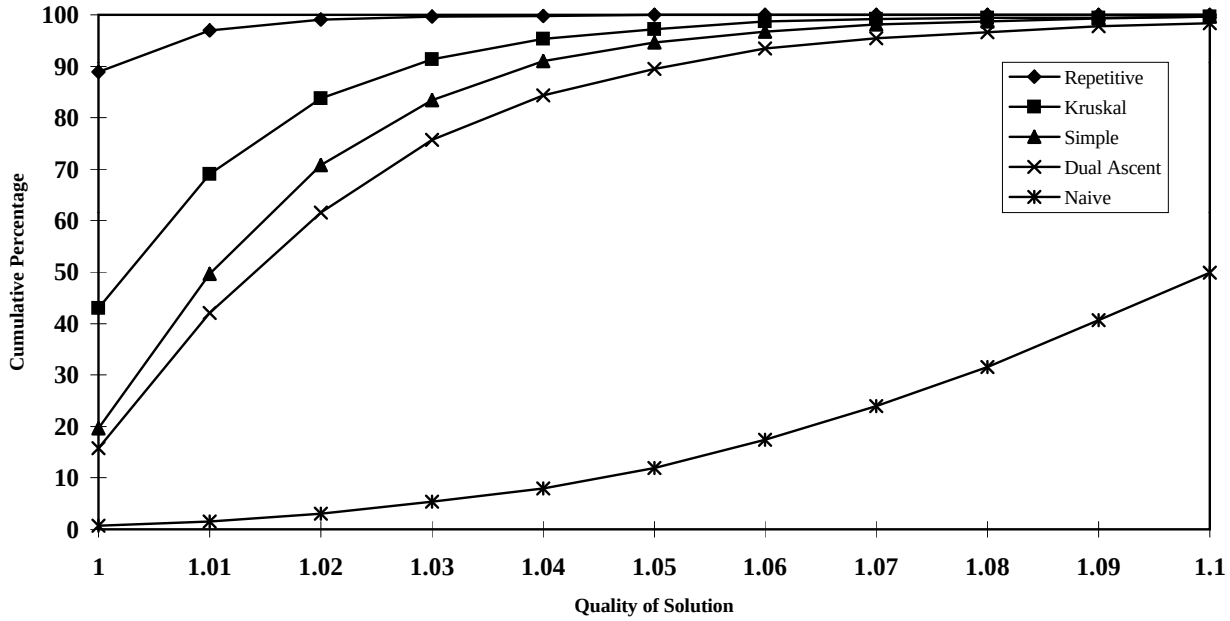

Figure 4: Degree-constrained quality-of-solution distributions.

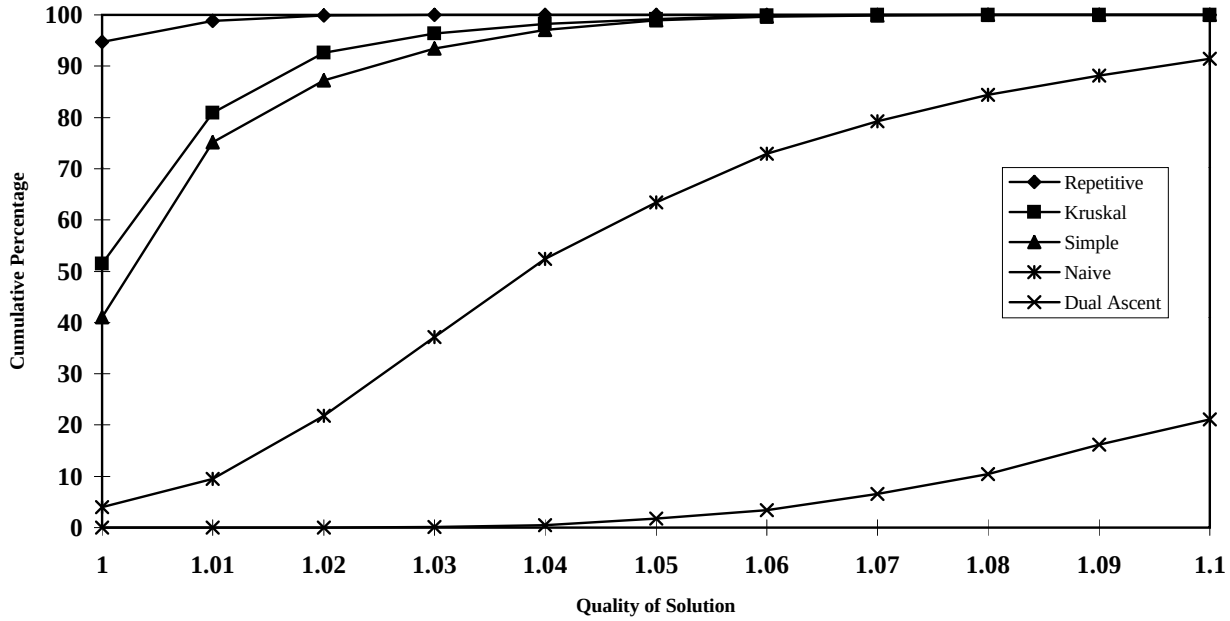

Figure 5: Unconstrained quality-of-solution distributions.

Heuristic	Number of solved networks without backtracking	Number of solved networks with backtracking	Number of unsolved networks
SPH-Z	1000	0	0
SPH-R	1000	0	0
SPH	988	8	4
A29	988	8	4
ADH	984	7	9
K-SPH	984	7	9
Naive	976	24	0
Dual Ascent	950	43	7

Table 4: Success rate of degree-constrained heuristics on 1000 test networks with a fixed degree-constraint of 3.

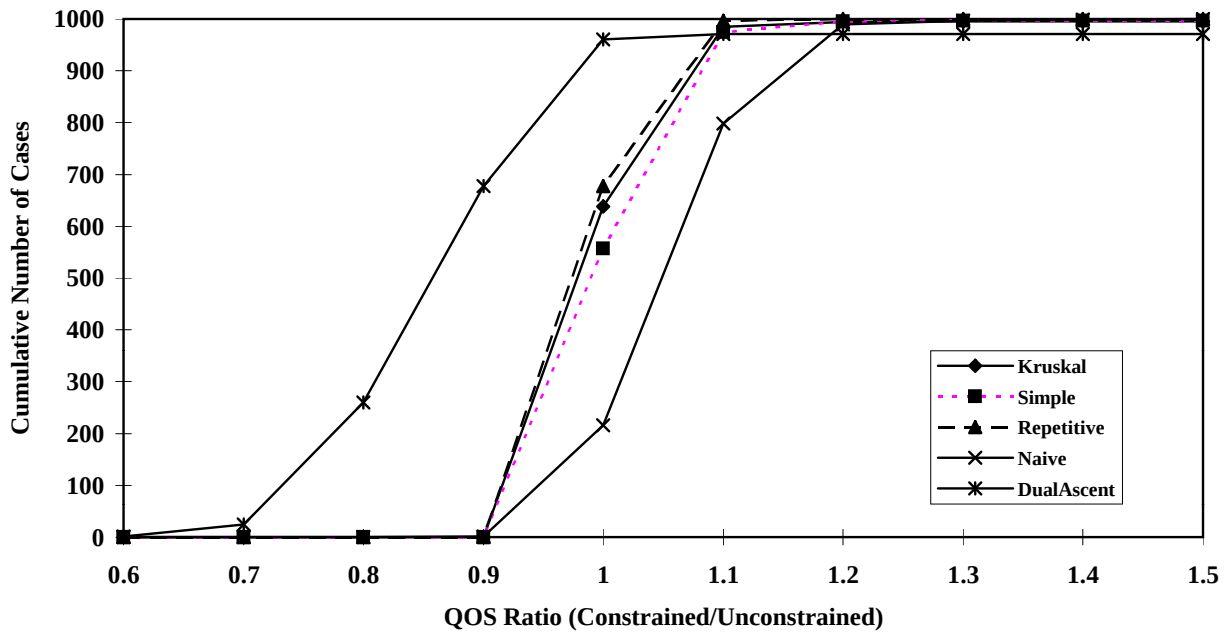

Figure 6: Quality of solution ratios for random degree-constraints.

Heuristic	Frequency of best solution (%)
SPH-Z	93.20
ADH	55.55
K-SPH	55.55
SPH	32.95
A29	32.95
SPH-R	30.80
Dual Ascent	27.35
Naive	1.05

Table 5: Frequency of best results produced by individual degree-constrained heuristics on 2000 test networks.

Heuristic	Frequency of best solution (%)
SPH-Z	48.60
ADH	34.10
K-SPH	34.10
SPH	29.75
A29	29.75
SPH-R	29.40
Naive	3.10
Dual Ascent	0.00

Table 6: Frequency of best results produced by unconstrained heuristics on 2000 test networks.

Figure 6 shows the ratio of constrained heuristic quality of solution to unconstrained heuristic quality of solution for each cluster. On comparing cost versus quality of solution for the degree-constrained and unconstrained heuristics (Figures 2 and 3), the average quality of the degree-constrained solutions was within 5% of that of the corresponding unconstrained solution for all the heuristics except Dual Ascent and Naive. It is interesting to observe that the average quality of solution of heuristic Naive degraded as a result of imposing degree constraints, whereas that of Dual Ascent actually improved. The difference between the quality of solution of heuristic Dual Ascent's unconstrained and degree-constrained cases may be explained by comparing the way both algorithms derive the Steiner tree from its subgraph. The unconstrained version uses a minimum spanning tree (MST) algorithm while the degree-constrained version uses the shortest path heuristic (SPH). Since the worst case ratio between a solution and an optimal solution for the SPH is $2X$ and for the MST is $|S|+1$ (where S is the number of non-multicast members) [26], this difference is not surprising.

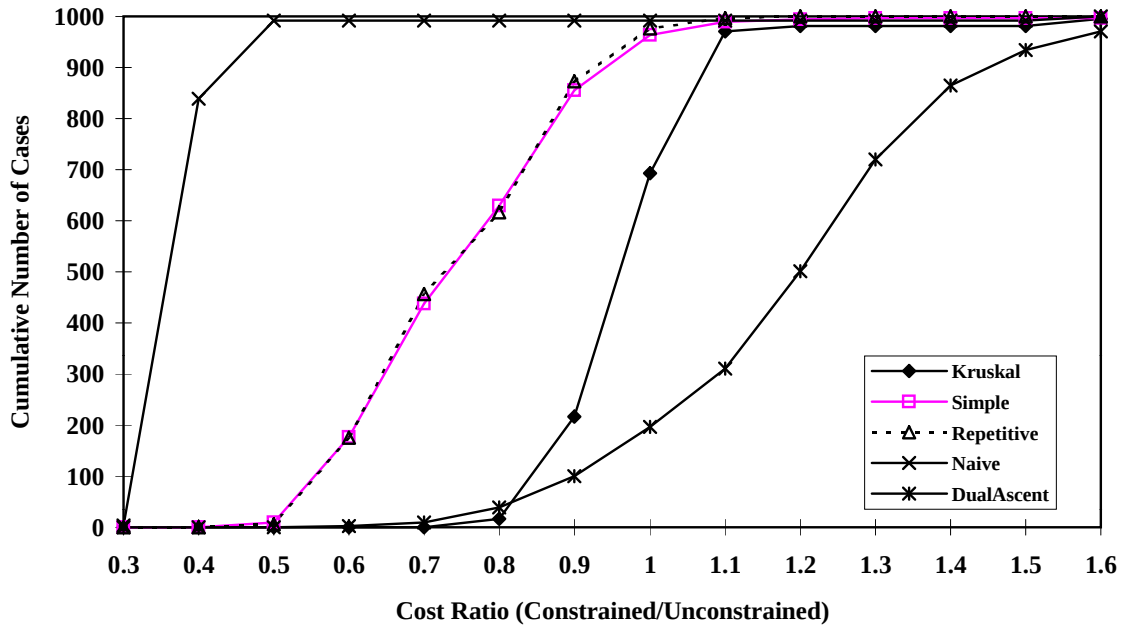

Figure 7: Run-time ratios for random degree-constraints.

Cluster	Degree-Constrained	Unconstrained
Repetitive	705.88	797.08
Kruskal	81.57	79.17
Dual Ascent	72.70	59.13
Simple	13.11	15.25
Naive	1.94	4.55

Table 7: Average cost of heuristics for the degree-constrained and unconstrained cases (in millions of statements).

Each cluster's quality of solution for the unconstrained case in Figure 3 remained essentially unchanged for all the clusters except Naive and Dual Ascent. Heuristic Naive yielded a significant number of solutions of inferior quality in the degree-constrained case that were worse than the best solution by 10% or more. With Dual Ascent, on the contrary, most of the solutions in the degree-constrained case were within 5% of the best solutions found, while the majority of solutions in the unconstrained case were worse by 10% or more.

On comparing the average cost (running time) of the heuristics for the unconstrained and degree-constrained cases (Table 7), it may be observed that imposing the degree constraints had only a small effect on the running time of the individual heuristics. Figure 7 shows the ratio of run times for the degree-constrained versus unconstrained versions of each heuristic cluster. The run-time of clusters Naive, Simple, and Repetitive actually improved slightly as a result of reducing the number of possible paths between nodes in the degree-constrained case. The run-time of the degree-constrained version of cluster Kruskal, however, remained

close to that of its unconstrained version. This is because the heuristics in this cluster must re-compute the shortest path between pairs of components every time degree-constraints force a topology change. This offsets any advantage gained by the reduced number of paths between components. In the case of Dual Ascent, the run-time actually increased in the degree-constrained case because of the cost of finding a degree-constrained subgraph as compared to an unconstrained one. Note that for any degree-constrained heuristic, backtracking increases the run-time over its unconstrained equivalent heuristic. However, because backtracking occurred so infrequently (a maximum 42 out of 2000 cases), it did not have a significant effect on the results.

The results presented so far demonstrate the cost versus quality-of-solution tradeoffs in the choice of a heuristic for degree-constrained multicast. At one end of the scale, cluster Repetitive (heuristic SPH-Z) yielded the uncontested best solutions. However, it finds these solutions at much higher cost than the other clusters. Clusters Naive and Dual Ascent, although much cheaper, varied wildly in the average quality of solutions between the degree-constrained and unconstrained cases. Neither appeared to be a suitable choice. Thus, the best tradeoffs are found in the remaining clusters Simple and Kruskal. Cluster Simple (heuristics SPH, SPH-R and A29) yielded the cheapest solutions of the two. Despite the simplicity of these algorithms, they produced solutions on average within 2% of the best solution found. Among the three heuristics in cluster Simple, SPH and SPH-R are favored; Heuristic A29's additional complexity only resulted in higher cost. Cluster Kruskal (heuristics K-SPH and ADH) retained its relative middle position for both degree-constrained and unconstrained graphs. It yielded better quality of solution on the average, while costing a fraction of cluster Repetitive. However, its disadvantage was the larger number of cases it could not solve, as shown in Table 3. Of the two heuristics in cluster Kruskal, heuristic K-SPH is favored because of its relative simplicity.

While heuristic Naive performed poorly relative to the other clusters in the comparison, it is the easiest heuristic to update with new members: New members join simply by using the shortest paths between themselves and the multicast tree. If quality of solution is of less concern than quickly updating membership, Naive is a suitable candidate.

Figure 8 show the ratio of constrained heuristic quality of solution to unconstrained heuristic quality of solution for a uniform degree-constraint of 3. The degradation in the quality of solution when all nodes have a uniform degree-constraint of 3 was well within 10% in most cases, except for heuristics Naive and Dual Ascent. Similarly, the run-time ratio for a uniform degree-constraint of 3 (Figure 9) is also relatively unchanged from that for the case with random degree-constraints. These results lead us to believe that both the cost and quality of solution of the heuristics in the cluster Simple are relatively insensitive to the degree constraints, and provide efficient solutions for the vast majority of multicast problems in large, sparsely-connected networks.

4.2 Effectiveness of Graph Reductions

The graph reductions applied, simple as they might seem, achieved remarkable results. We restricted ourselves to graph reductions S degree 1, S degree 2, and Z degree 1 because these were of reasonable cost and could be applied to the degree-constrained case. We had to discard many other graph reductions [10, 11, 27] because they eliminated nodes and edges

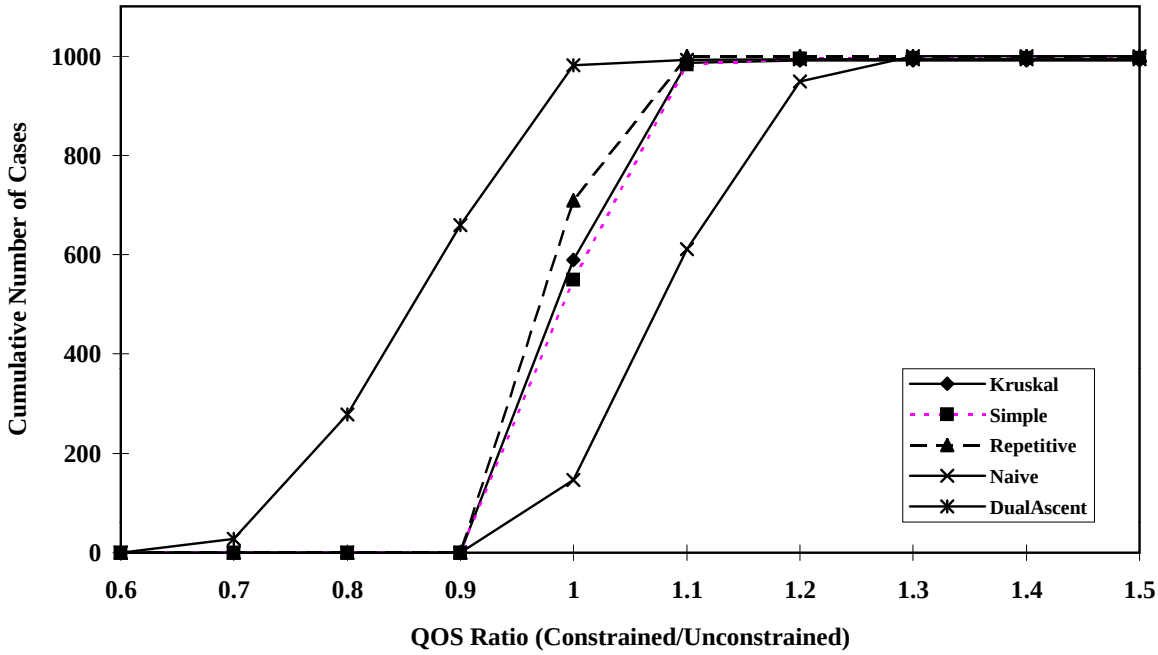

Figure 8: Quality of solution ratios when maximum degree-constraint = 3.

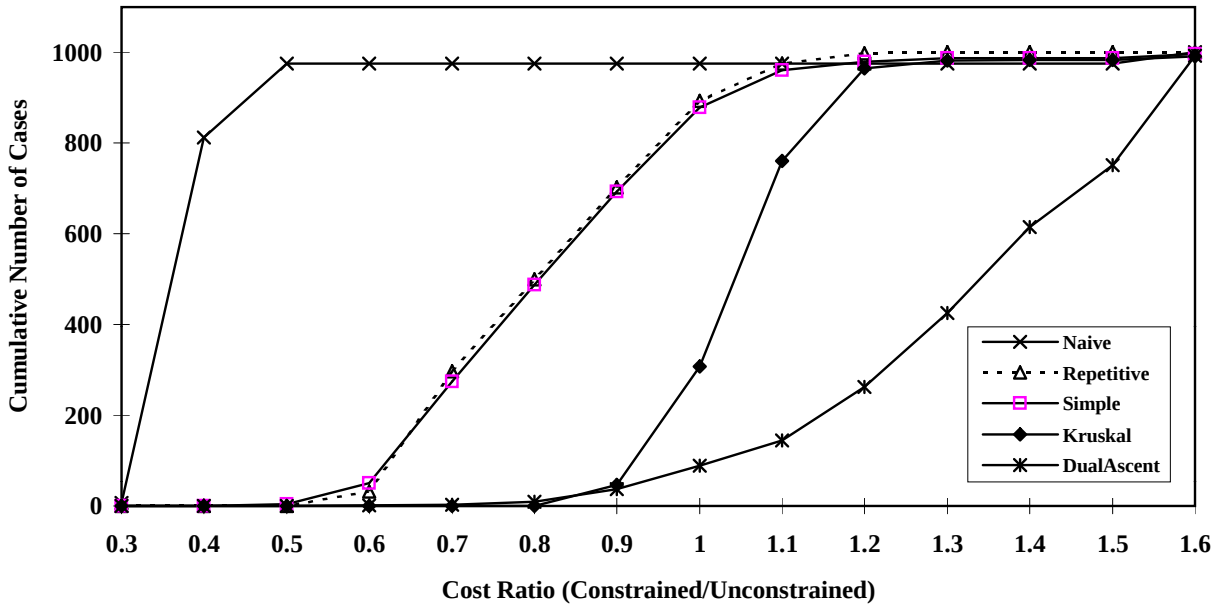

Figure 9: Run-time ratios when the maximum degree-constraint = 3.

needed in the degree-constrained case. For example, some graph reductions identify shortest paths between multicast members as solution edges, eliminating longer paths. However, in the degree-constrained case, the multicast tree containing this shortest path may violate degree constraints of nodes in its path. Consequently, a longer path may be part of the final solution edge and should not be eliminated by the reduction. Of the reductions left, we also discarded those whose cost outweighed their benefit. For example, the reachability test (RT), outlined in Section 4, is as expensive as heuristic SPH, yet did not improve any of our sample networks. In fact, all of the reductions employed in our simulation together required far less than 1% of heuristic SPH's cost and achieved much better results.

Figure 10 displays the histograms for nodes and edges reduced as a result of applying the graph reductions to the 200 test networks we used. The reductions might reduced an average graph in our simulations by 7% of its nodes and 3% of its edges. Figure 11 shows the percentage of solution edges identified by the reductions. In our experiments, the reductions on average identified at least one solution edge for approximately 4% of the multicast members.

Performing the reductions first has two positive effects on the subsequent heuristic. First, some solutions of inferior quality are eliminated. Our empirical evidence indicates this has a positive effect on the heuristics, particularly those in cluster Simple (SPH, SPH-R and A29). Solution quality for this cluster in some cases improved from 12% to within 5% of the best heuristic solution found. Second, the run-time of heuristics also improves. Our results indicate an improvement of 10–20% in heuristic run-time for many cases. This fits the 17–20% improvement predicted using the time complexity for heuristics ADH, SPH and SPH-Z [27].

5 Concluding Remarks

In this paper we studied the degree-constrained multicast tree problem as applied to point-to-point networks and evaluated the effectiveness of several heuristic algorithms based on modifications of algorithms for the unconstrained Steiner-tree problem. These heuristics for finding degree-constrained multicast trees were compared in terms of their cost (running time), quality of solution, and the number of networks they could not solve.

Our results show that many of the Steiner heuristics tested yielded degree-constrained multicast trees within 5% of the best heuristic solution found in almost all of the networks tested. Surprisingly few of our networks were unsolvable. In those cases where no solution was found by a heuristic, backtracking solved many of the remaining cases. In addition, relatively cheap heuristics produced high quality results while others of greater complexity did not justify their additional effort.

Of the heuristics simulated, simple Steiner heuristics such as SPH and SPH-R emerged as the clear winners with an attractive balance between the conflicting objectives of solution quality and algorithm complexity. The next least expensive heuristics K-SPH and ADH often gave better solutions at moderate, extra expense. Heuristic Naive, our expected worst Steiner heuristic, often did produce the worst solution; however, it also produced many solutions of surprisingly high quality. This result matches that of Doar and Leslie [8]. The quality of solution of the more complex heuristics A29 and Dual Ascent rarely justified their

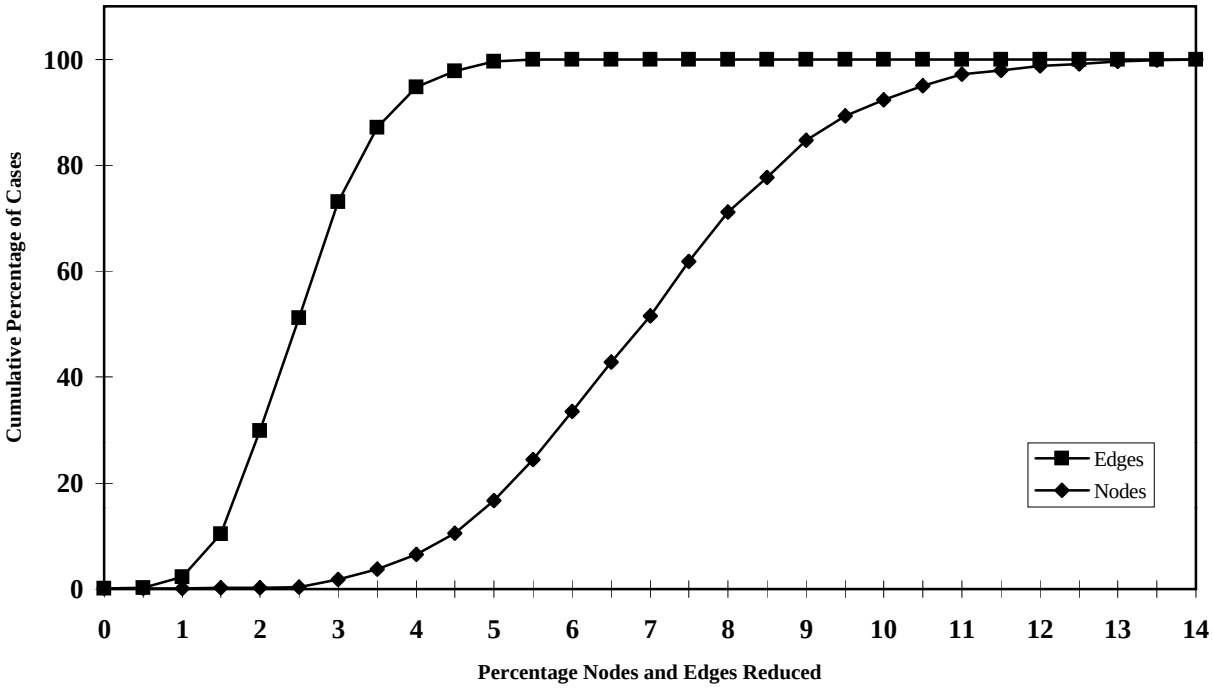

Figure 10: Distribution of nodes and edges identified by graph reductions.

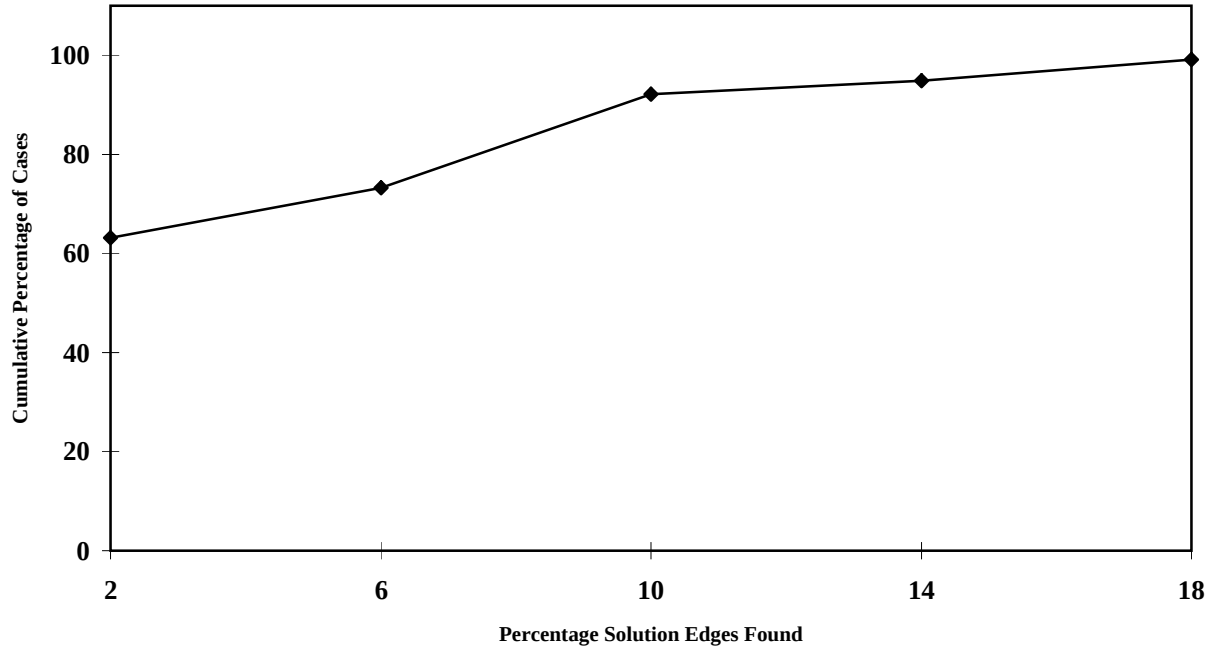

Figure 11: Distribution of the percentage of multicast members for which at least one solution edge was identified by graph reductions.

additional effort.

As expected, dense networks posed much less of a challenge to degree-constrained Steiner heuristics than sparse networks. In fact, our degree-constrained heuristics easily solved all the dense networks we tested without backtracking. Consequently, our focus in future research will continue to be on sparse networks.

An interesting result from our simulations is that the need for unlimited multicast capability in the individual switches of a wide-area network (such as an ATM network) may be overstated. The median degree-constraint in our simulations with random degree-constraints was 3. Our additional simulations with a fixed degree-constraint of 3 (that is, a fanout of 2 per node), also produced similar results. Thus, our results show that ATM switches in a large wide-area network need not be designed with unconstrained multicast capability. Instead, the ability to make even one additional multicast copy (degree-constraint of 3) would allow our DCSP heuristics to find a solution in many, perhaps even the overwhelming majority, of cases.

All the degree-constrained multicast heuristics described in this paper are centralized algorithms. We are currently working on distributed algorithms that do not require each of the network nodes to store the entire topology of the network. We are also working on algorithms for incrementally updating the multicast tree when nodes join and leave the tree.

References

[1] M. Ammar, S. Cheung, and C. Scoglio. "Routing multipoint connections using virtual paths in an ATM network," in *Proc. IEEE INFOCOM*, San Francisco, CA, Apr. 1993, pp. 98–105.

[2] Y.P. Aneja. "An integer linear programming approach to the Steiner problem in graphs," *Networks*, vol. 10, pp. 167–178, 1980.

[3] J. Beasley. "An SST-based algorithm for the Steiner problem in graphs," *Networks*, vol. 19, pp. 1–16, 1989.

[4] L. Berry. "Graph theoretic models for multicast communications," in *Traffic theories for new telecommunications services ITC Specialists Seminar*, Adelaide, Australia, Sep. 1989, pp. 95–99.

[5] K. Bharath-Kumar and Jaffe. "Routing to multiple destinations in computer networks," *IEEE Transactions on Communications*, vol. COM-31, no. 3, pp. 343–351, Mar. 1983.

[6] N. Chen. "New algorithms for Steiner tree on graphs," in *1983 IEEE International Symposium on Circuits and Systems*, Newport Beach, CA, May 1983, pp. 1217–1219.

[7] S. Deering. "Multicast routing in internetworks and extended LANs," *Computer Communication Review*, vol. 18, no. 4, pp. 55–64, Aug. 1988.

[8] M. Doar and I. Leslie. "How bad is naive multicast routing?," in *Proc. IEEE INFOCOM*, San Francisco, CA, Apr. 1993, pp. 82–89.

[9] R. Douglas. "NP-completeness and degree restricted spanning trees," *Discrete Mathematics*, vol. 105, pp. 41–47, 1992.

[10] C. Duin and A. Volgenant. "An edge elimination test for the Steiner problem in graphs," *Operations Research Letters*, vol. 8, pp. 79–83, 1989.

[11] C. Duin and A. Volgenant. "Reduction tests for the Steiner problem in graphs," *Networks*, vol. 19, no. 5, pp. 549–567, Aug. 1989.

[12] F. Hwang and D. Richards. "Steiner tree problems," *Networks*, vol. 22, pp. 55–89, 1992.

[13] F. Hwang, D. Richards, and P. Winter. *The Steiner Tree Problem.* New York: North-Holland, 1992.

[14] X. Jiang. "Path finding algorithms for broadband multicast," in *Third Int'l Conf. on High Speed Networking*, O. Spaniol and A. Danthine, ed., New York: North-Holland, 1991, pp. 153–164.

[15] D. Johnson. "The NP-completeness column: an ongoing guide," *Journal of Algorithms*, vol. 6, no. 3, pp. 434–451, Sep. 1985.

[16] V. Kompella, J. Pasquale, and G. Polyzos. "Multicasting for multimedia applications," in *Proc. IEEE INFOCOM*, New York, NY, May 1992, pp. 2078–2085.

[17] V. Kompella, J. Pasquale, and G. Polyzos. "Multicast routing for multimedia communications," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 286–292, Jun. 1993.

[18] J. Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proc. Amer. Math. Soc.*, vol. 7, pp. 48–50, 1956.

[19] M. Smith and P. Winter. "Path-distance heuristics for the Steiner problem in undirected networks," *Algorithmica*, vol. 7, no. 2-3, pp. 309–327, 1992.

[20] H. Takahashi and A. Matsuyama. "An approximate solution for the Steiner problem in graphs," *Math. Japonica*, vol. 24, no. 6, pp. 573–577, 1980.

[21] H. Tode, Y. Sakai, M. Yamamoto, H. Okada, and Y. Tezuka. "Multicast routing algorithm for nodal load balancing," in *Proc. IEEE INFOCOM*, New York, NY, May 1992, pp. 2086–2095.

[22] J. Turner. "An optimal nonblocking multicast virtual circuit switch," in *Proc. IEEE INFOCOM*, Toronto, Canada, Jun. 1994, pp. 298–305.

[23] S. Voss. "A survey of some generalizations of Steiner's problem," in *Proc. of the First Balkan Conference on Operational Research*, 1988.

[24] S. Voss. *Steiner-Probleme in Graphen.* Frankfurt/Main: Hain, 1990, pp. 179–184.

[25] S. Voss. "Problems with generalized Steiner problems," *Algorithmica*, vol. 7, no. 2-3, pp. 333–335, 1992.

[26] S. Voss. "Steiner's problem in graphs: Heuristic methods," *Discrete Applied Mathematics*, vol. 40, pp. 45–72, 1992.

[27] P. Winter. "Steiner problem in networks: A survey," *Networks*, vol. 17, no. 2, pp. 129–167, 1987.

[28] R. Wong. "A dual ascent approach for Steiner tree problems on a directed graph," *Mathematical Programming*, vol. 28, pp. 271–287, 1984.

[29] W. De Zhong, Y. Onozato, and J. Kaniyil. "A copy network with shared buffers for large-scale multicast ATM switching," *IEEE/ACM Transactions on Networking*, vol. 1, no. 2, pp. 157–165, Apr. 1993.