Distributed Algorithms for Multicast Path Setup in Data Networks

Fred Bauer

Anujan Varma

UCSC-CRL-95-10
August 16, 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, can be modeled as the NP-complete Steiner problem in networks. In this paper, we introduce and evaluate two distributed algorithms for finding multicast trees in point-to-point data networks. These algorithms are based on the centralized Steiner heuristics, the *shortest path heuristic* (SPH) and the *Kruskal-based shortest path heuristic* (K-SPH), and have the advantage that only the multicast members and nodes in the neighborhood of the multicast tree need to participate in the execution of the algorithm. We compare our algorithms by simulation against a baseline algorithm, the pruned minimum spanning-tree heuristic, which is the basis of many previously published algorithms for finding multicast trees. Our results show that the competitiveness (the ratio of the sum of the heuristic tree's edge weights to that of the best solution found) of both of our algorithms was on the average 25 percent better in comparison to those produced by the pruned spanning-tree approach. In addition, the competitiveness of our algorithms was, in almost all cases, within 10 percent of the best solution found by any of the Steiner heuristics considered, including both centralized and distributed algorithms. Limiting the execution of the algorithm to a subset of the nodes in the network results in an increase in convergence time over the pruned spanning-tree approach, but this overhead can be reduced by careful implementation.

Keywords: Multicasting, Steiner problem in networks, Distributed algorithms.

This research is supported by the Advanced Research Projects Agency (ARPA) under Contract No. F19628-93-C-0175 and by the NSF Young Investigator Award No. MIP-9257103. This paper was presented in part at GLOBECOM '95, Singapore, November 1995.

1 Introduction

Many future applications of computer networks such as distance education, remote collaboration, and teleconferencing will rely on the ability of the network to provide multicast services. Indeed, many recent standards for packet-switched networks, notably ATM, Frame Relay and SMDS, include support for multicasting. Thus, multicasting will likely be an essential part of future networks.

Multicasting is sometimes supported in a point-to-point packet network by setting up a multicast tree connecting the members of the multicast group. We concern ourselves in this paper with networks that use virtual circuit routing, such as ATM and Frame Relay. In such a network, a multicast virtual circuit is set up from the source of the multicast to the destinations before data transmission occurs. Determining this optimal multicast tree for the virtual circuit is a difficult problem. Previous authors have established that the multicast tree problem can be modeled as the *Steiner problem in networks* [2, 3, 8, 20], referred to hereafter as the SPN, and that finding explicit solutions in large networks is prohibitively expensive. For example, two popular explicit algorithms, the *spanning tree enumeration algorithm* and the *dynamic programming algorithm* [20], have algorithmic complexities 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, centralized heuristics exist for the SPN and have been reviewed extensively elsewhere [3, 8, 10, 15, 16, 17, 20]. Some have been shown through analysis to produce solutions no worse than twice the optimal solution [20]. Empirical evidence from our previous papers indicate that these heuristics find solutions much better than twice the optimal with reasonable speed in most cases [1].

Most of the algorithms proposed in the literature for SPN are serial in nature. However, a few distributed heuristics exist [4, 12]. Many of these algorithms are based on reducing the SPN to the minimum spanning tree problem, referred to here as the MST, and using a distributed minimum spanning tree algorithm such as the one described by Gallager, Humblet, and Spira [6]. A Steiner tree is created by pruning the minimum spanning tree by removing subtrees containing no multicast members. For example, Chen, et al. [4] finds a Steiner tree by applying a distributed minimum spanning tree algorithm twice. First the algorithm is applied to the original graph. This first minimum spanning tree is used to create a *shortest path forest* composed of disjoint trees and edges that together form a connected subgraph of the original graph. The distributed minimum spanning tree algorithm is then applied a second time to this subgraph. The solution is obtained by pruning unnecessary leaves and branches from this second minimum spanning tree. Likewise, Kompella, et al. [12] describe two distributed versions of earlier centralized heuristics proposed by the same authors [11]. Both of these distributed heuristics first build a *constrained Steiner tree* that reflects the combined criteria of cost and delay. A distributed MST algorithm is applied to this constrained Steiner tree and the solution tree is pruned. The two heuristics differ in their criteria for choosing edges while constructing the MST.

Distributed Steiner heuristics based on a minimum spanning tree algorithm suffer from two drawbacks: First, all the nodes in the network must participate in the execution of the algorithm. This may be impractical in a large network with sparse multicast groups. Second, the theoretical upper bound on *competitiveness* of a pruned MST to that of an optimal Steiner tree has been shown to be $s+1$, where s is the number of non-multicast nodes [17]. Here competitiveness is defined to be the ratio of the sum of the heuristic tree's edge weights to that of an optimal tree [9, 18, 19].

Other equivalent terms for this measure include *inefficiency* and *quality of solution*. Thus the competitiveness of a multicast tree decreases with the size of the multicast group. In comparison, the equivalent theoretical upper bound for the shortest path heuristic (SPH) for the Steiner tree problem is $2(1-\frac{1}{p})$ [20], where p is the size of the multicast group. Our empirical evidence suggests that pruned MST heuristics often produce solutions of inferior quality as compared to those produced by shortest path Steiner heuristics.

In this paper, we present two distributed algorithms for the Steiner problem in networks. The algorithms are based on the shortest path heuristic (SPH) and the Kruskal-based shortest path heuristic (K-SPH), described in [1]. We analyze their message and convergence-time complexities and compare their simulation results against those from a pruned MST algorithm. We choose the distributed MST algorithm due to Gallager, Humblet, and Spira [6] as our baseline algorithm for comparison. This algorithm is perhaps the simplest of all pruned MST algorithms, yet produces Steiner trees that are representative of other, more elaborate pruned MST heuristics such as those described in [4, 12]. The distributed heuristics are compared on the basis of three criteria: competitiveness, number of messages exchanged, and convergence time.

Our simulations of the algorithms are performed on a large set of sparse, randomly-generated network topologies. We use the distance propagation delay model, using the distance between nodes as the weight of the edge between them. 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, in general, fewer solutions 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 such as video conferencing, distance learning, resource discovery and replicated database updating. Note that our results are not specific to any particular type of network such as ATM, but may be applied to any virtual circuit-based point-to-point network.

The remainder of this paper is organized as follows. Section 2 introduces and analyzes our two distributed heuristics, based on the centralized Steiner heuristics K-SPH and SPH, respectively. Section 3 compares the algorithms in terms of their competitiveness, convergence time, and number of messages exchanged. We show that the distributed shortest-path heuristics produced multicast trees with competitiveness within 5% of that of the best solution found by any heuristic, both centralized and distributed, in more than 90 percent of our test networks. In contrast, only 0.3% of the solutions produced by the pruned MST algorithm fell within 5% of the best solutions in terms of their cost (sum of edge weights). Finally, Section 4 concludes the paper with a discussion of the results.

2 Steiner Tree Heuristics

In this section, we summarize previous Steiner heuristics and introduce the two distributed Steiner heuristics.

Before continuing, we make the following basic definitions and notations. 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, $d_{i,j}$ is the distance of the shortest path between nodes i and j as well as the weight of the edge between nodes i and j, and $C(T)$ is the cost of tree T (the sum of T's edge weights). Graph distances will be defined as follows: The distance between two 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. As in [6], we append the weight of an edge or path with the index of its destination node in determining shortest paths so that, in case of a tie, the actions of the individual nodes would be consistent. Since we do not allow multiple edges between the same pair of nodes, this ensures that all the nodes select the same edge or path, given the same set of edge weights.

To be suitable for distributed implementation, a heuristic must satisfy four criteria. It must (i) use the existing routing information available at each node in the network, (ii) use minimal computational and network resources, (iii) require a minimum of coordination between neighbors, and (iv) limit itself to nodes directly involved in the multicast. Of the centralized heuristics evaluated in [1], we chose the following four heuristics as candidates for distributed implementation: the *shortest-path heuristic* (SPH), a variant of SPH known as SPH-Z, the *Kruskal-based shortest-path heuristic* (K-SPH), and the *Average distance heuristic* (ADH). Each of these heuristics is described in [15]. A brief summary of heuristics SPH and K-SPH follows.

SPH

Heuristic SPH, introduced in [16], initializes the multicast tree to an arbitrary multicast member. It then grows the tree by successively adding 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 the multicast members have joined the tree.

K-SPH

Heuristic K-SPH, introduced in [13], differs from SPH in the manner in which the multicast tree is expanded. Instead of growing the tree one node at a time, the algorithm joins subtrees pairwise repeatedly until all the multicast nodes are part of a single tree. The algorithm initially starts with Z subtrees, each a multicast member itself. In the expansion step, it finds two subtrees that are closest to each other and joins them along the shortest path between them to form a single subtree. This heuristic is a refinement of the average distance heuristic first proposed by Rayward-Smith [14].

2.1 Distributed Steiner Heuristics

After further consideration, only two of the four heuristics, SPH and K-SPH, remain as suitable candidates for distributed implementation. Both heuristics SPH-Z and ADH fail our criteria for conversion. Although heuristic SPH-Z initially appears attractive as a distributed heuristic, its component distance information for each of the Z instances could be distinct. This forces as many as Z copies of component information at each node and a virtual storm of network messages before convergence. Likewise, ADH fails our criteria because its calculation of the most central node requires excessive overhead for coordination between nodes in the network. In addition, our earlier results indicate that, on the average, the solutions produced by K-SPH and ADH are of nearly identical quality [1]. Of these two heuristics, K-SPH is the more attractive candidate because of its relative simplicity and lower running time.

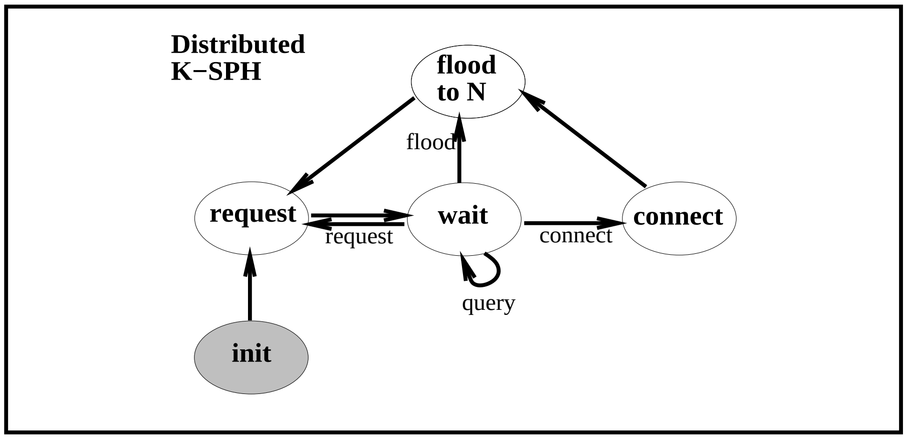


Figure 1: The finite state machine for fragment leaders in distributed K-SPH.

Distributed heuristics SPH and K-SPH are designed to run as asynchronous, independent processes running one per node in a network.

We assume that each such node knows its shortest path to all other nodes in the network. Each distributed heuristic assumes that each node in the network is a router; the routing tables in each such node is up-to-date; all shortest paths are *symmetric* in the sense that nodes i and j share shortest paths between them; no topology changes occur during the execution of the algorithm; the network is connected; every node has a unique index; each multicast member has knowledge of the indices of all other multicast members; and each multicast member is able to determine the distance to every other node from its routing table.

Heuristic SPH is inherently a serial algorithm, since there is only one subtree expanding itself at any time during the execution of the algorithm and nodes must join the tree serially. Heuristic K-SPH, on the other hand, allows many of the join operations to proceed in parallel. The latter, however, is substantially more difficult to parallelize because of the significant amount of coordination that may be needed while combining subtrees. In the following, we present distributed K-SPH first, followed by a similar distributed implementation of SPH.

2.1.1 Distributed Heuristic K-SPH

Like its centralized version, distributed K-SPH starts with a forest of Z multicast members (Z-nodes) and connects them pairwise into successively larger subtrees until a single multicast tree remains or no further connections are possible. We refer to the subtrees during the execution of the algorithm as *fragments*. Thus, at the beginning of the algorithm, there are Z fragments, each a trivial subtree consisting of one Z-node. As the algorithm proceeds, these fragments grow into increasingly larger collections of multicast members.

At any instant during the execution of the algorithm, each node in the network is either part of a fragment or has not been yet been included in the multicast tree. Note that every Z-node is always a fragment node and every non-member node (S-node) is initially a non-fragment node. When two fragments merge, the nodes in both fragments and the nodes that lie on the path connecting them become the fragment nodes of the new, merged fragment.

```
while fragments remain do

    # algorithm discovery step
    update fragment information for nearby fragments
    $f \leftarrow$ closest fragment

    # algorithm connection step
    request merger with fragment $f$
    if merger requests exchanged with fragment $f$ then
        attempt connection to fragment $f$
        if connection established then
            merge fragments
        else
            restore original fragments
        end if
    end if
end while
```

Figure 2: Pseudocode for fragment leaders in distributed K-SPH.

Each fragment has a *fragment leader* coordinating the activities of the fragment. This fragment leader is the fragment Z-node with the lowest index. Each fragment leader executes the finite state machine shown in Figure 1. Initially, each multicast member is the leader of its own one-node fragment; when two fragments merge, leadership is assigned to the fragment leader with the lower index. To identify fragments uniquely, each fragment has the same index as its leader and each fragment node is aware of its fragment index.

At the highest level, each fragment, guided by its leader, executes the pseudocode shown in Figure 2. During the execution of the algorithm, each fragment attempts to merge with its closest neighboring fragment. This is accomplished in two steps — a *discovery* step and a *connection* step. During the discovery step, the leader gathers and updates its information on other fragments and graph nodes. Based on the information gathered, it determines the closest fragment to merge with. During the connection step, it communicates with the closest neighbor fragment's leader, requesting a merge. This closest fragment leader is simply the Z-node with the same index as the closest fragment. If accepted, the leader with the lowest index attempts to connect the two fragments. Regardless of the outcome (the request is rejected, the fragments are connected, or the connection attempt fails), the cycle repeats until the algorithm terminates.

Distributed K-SPH processes running on each node rely on the shortest path information assumed available at its node, as well as information maintained by the fragment leaders. Each node also stores the index of its fragment. Initially, only multicast nodes have a fragment index —its own index. Each leader maintains additional shortest path information for its fragment. This information augments the shortest path information at each node. For example, the leader stores only the distance, and the head and tail of the shortest path between its fragment and every other fragment. The additional details necessary to build the path between fragments is stored at the

```
# request merger with fragment $f$
do
    send request to fragment $f$'s leader
    wait for response
until accept or reject

leader ← this fragment index < fragment $f$'s index
if accept and leader then
    # attempt connection to fragment $f$
    send connect message to head of shortest path

    wait for connection success or failure
    if failure then
        send reject to fragment $f$'s leader
end if
```

Figure 3: Fragment leader pseudocode for the connection step in distributed K-SPH.

head of the path, a node in the leader's subtree. Note that the shortest path between fragments need not start or end at a leader node.

2.1.2 The finite state machine of heuristic K-SPH

State init: When distributed K-SPH starts, each Z-node, the leader of its own trivial one-node fragment, already knows its distance to every other fragment as provided by the initial distance tables and no discovery step is necessary. Instead, each distributed K-SPH leader starts with the connection step, described below.

States request, wait, and connect: States *request, wait* and *connect* in Figure 1 comprise the connection step. During this step, each leader attempts to connect its fragment with the closest fragment, known as its *preferred fragment* (Figure 3 shows the pseudocode for this step). It does so by sending a *merge request* message to the leader of the preferred fragment (That is, the Z-node with the same index as the preferred fragment). A leader receives one of three responses to its request: *accept*, *reject*, or *busy*.

busy A fragment leader returns the busy response when a request arrives during its discovery step. Upon receiving a busy response, a fragment will retransmit its merge request.

reject A fragment node returns a reject message when (i) it receives a connections request from a fragment other than its preferred fragment, (ii) when a connection attempt fails, or (iii) when it is no longer a fragment leader.

accept A fragment leader returns an accept response when it exchanges merge requests with its preferred fragment. Once an accept message is sent, the fragment may not leave the connect step or accept a request from another fragment until the connection attempt completes.

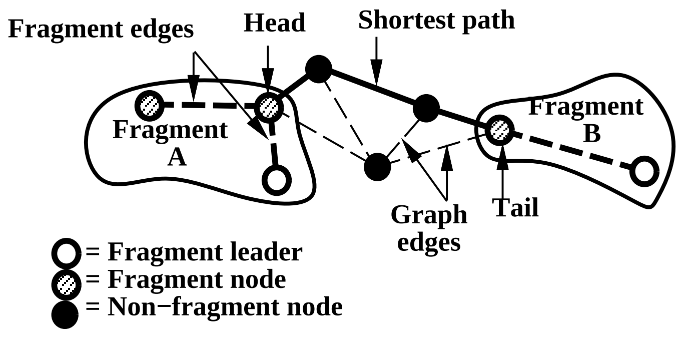


Figure 4: Example of fragments A and B merging in distributed K-SPH.

Connecting fragments: Upon receiving the accept response, the fragment leader with the lowest index attempts to connect the two fragments together using a shortest path. To do so, a connect message is sent down this shortest path. The message may either reach the target fragment or be *blocked*; blocking occurs when the message reaches a node in a third fragment before reaching the target fragment.

Figure 4 illustrates a successful connection where fragments A and B are connected by their shortest path. This path has its head in one fragment, its tail in the other, and passes through only non-fragment nodes. The connect message stops at the first node in the target fragment it reaches and sends a status message back along the same path.

If the connect message is blocked, the blocked node returns a status message back along the shortest path. Each intermediate node upon receiving this status message reverts to its previous non-fragment status. Upon receiving the status message, the initiating fragment leader sends a reject message to the other fragment leader.

After the connection step, fragment leaders enter the discovery step.

The discovery step: The discovery step accomplishes three tasks: (i) it informs every node in the fragment of its new fragment index, (ii) it gathers fragment information about nodes close to the fragment, and (iii) it refreshes its information on shortest paths to other fragments. The pseudocode for the discovery step is shown in Figure 5. Each fragment leader achieves these tasks by performing a multicast on its fragment rooted at itself. In the multicast message, the leader includes the fragment index, the distance to the preferred fragment and shortest paths to other fragments. As each node in the fragment receives the multicast, it updates its fragment index, queries *nearby* nodes and passes the multicast message to its children. Nearby nodes are defined to be those nodes that lie within the shortest distance from this fragment to the preferred fragment. Nearby nodes are queried for fragment index information. The objective of queries to nearby nodes is to find fragment nodes closer than those already known by the leader. Figure 6 illustrates a case where this is useful. Queries could be sent to all nodes in the graph, but are limited to nearby nodes for two reasons: (i) a set distance avoids broadcast storms and (ii) new shortest paths discovered should be shorter than those already available. The discovery step is implemented by state *flood-to-N* in Figure 1.

Analysis of Distributed K-SPH Algorithm: Having described the distributed K-SPH in the previous section, we now turn to its properties. We use a directed *request graph* to show the

```
# send update request to all children
for all fragment children do
    send fragment index, distance to closest fragment,
        and shortest path information to each child
end for

# query all nodes closer than closest fragment
for nodes nearer than closest fragment do
    send nodes query for component information
end for

wait for responses
update shortest path information information

# forward results
if not leader then
    send summary of shortest path information
        to parent
end if

if leader then
    $f \leftarrow$ closest fragment
```

Figure 5: Fragment leader pseudocode for the discovery step in distributed K-SPH.

relationship of fragments to one another during the execution of the algorithm. Each fragment in the network is represented by a node in the request graph and its current choice of preferred fragment by a directed edge. Figure 7 illustrates an example graph with three vertices representing fragments A, B, and C. In this example, the fragment pair A and B request each other, while a third, more distant fragment C requests fragment B. Fragments A and B will merge, creating a new fragment that will form a pair with fragment C and merge. A fragment is considered *stable* when it is in the states wait or request since its choice of preferred partner is unknown when the fragment is in states connect, flood-to-N, or init.

The request graph can be used to show that the distributed K-SPH algorithm does not deadlock. To prove that the algorithm will terminate, we need to only show that at any time during the execution of the algorithm, the shortest-path distances maintained by two of the fragments to each other will converge to the same value in a finite time (that is, a cycle of length 2 in the request graph). These two fragments will then merge to form a new fragment. Thus, by induction, the algorithm will terminate in finite time.

Lemma 1 *At any time during the execution of the algorithm, the shortest-path distances maintained by two stable fragments to each other will converge to the same value within a finite time.*

Proof: Initially, the shortest path between any two single-node multicast members is the shortest path between their fragments. This path is symmetric in the sense that both paths consist of

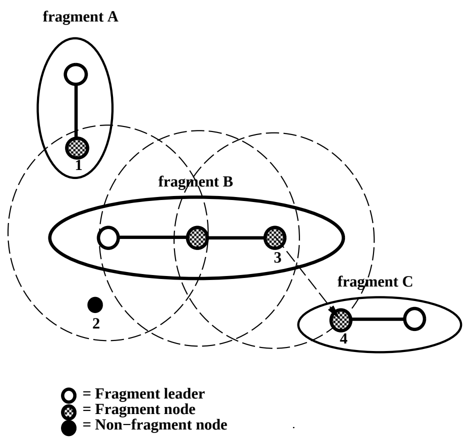


Figure 6: Example of the discovery phase in distributed K-SPH. Fragment B's leader believes that fragment C is the closest fragment. During fragment B's discovery step, it instructs fragment nodes to query those nodes closer than fragment C. This distance is the distance between node 3, the head of the path to fragment C, and node 4, its tail, and is marked by the dotted circles around each of fragment B's nodes. Since nodes 1 and 2 fall within one such circle, they receive queries and fragment B's leader discovers the closer fragment A.

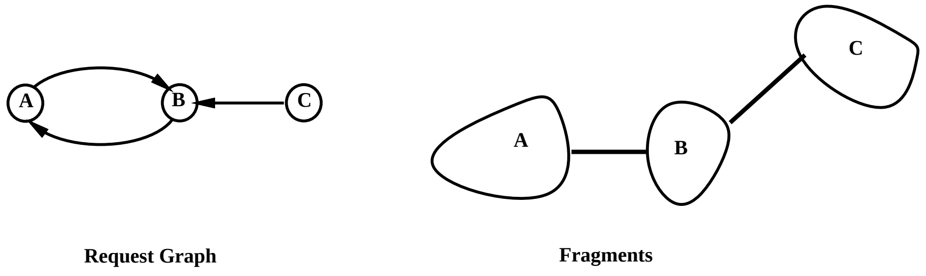


Figure 7: An example for request graph.

the same nodes and edges. This is because of the strict ordering of all shortest paths. Suppose that at some later time, the shortest paths between two fragments differ in distance as shown in Figure 8. Suppose further, that one fragment, in this case fragment B, has the longer path. The next time fragment B enters state flood-to-N, it will query every node in its neighborhood for fragment information. Any node in fragment A closer to B must fall within B's neighborhood and will become the tail of a new, shorter path to fragment A. The shortest of all such paths will become fragment B's new shortest path to fragment A. By a symmetrical argument, the paths between fragments A and B must converge on the same distance.

Inconsistencies may also occur when a path is blocked. Assume that only one of a pair of fragments finds the shortest path between them blocked. Assume fragment A has a shortest path to B, but B's shortest path to A is blocked. The shortest path between fragments can only be blocked by a node belonging to a third fragment C. In this case, fragment A will query the blocking node in fragment C the next time A enters the discovery phase (state flood-to-N). In addition, C will also update its distance to A during its recovery phase, resulting in consistent values for the distance

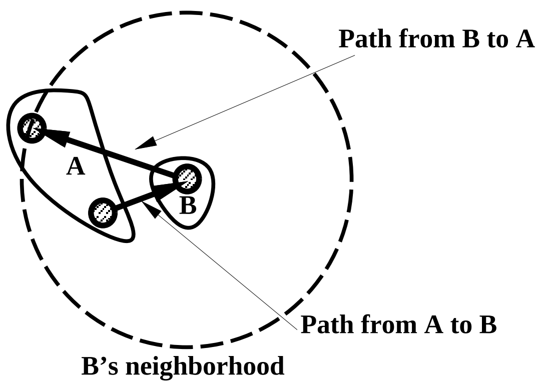


Figure 8: Two subtrees with different shortest paths.

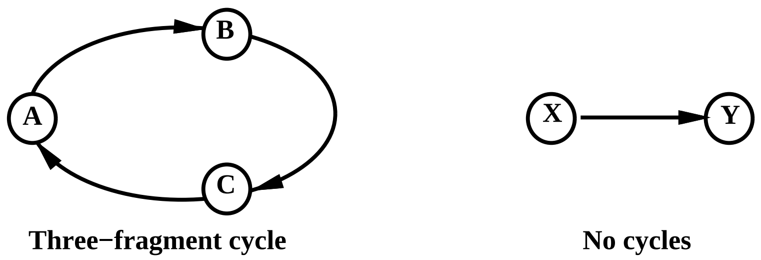


Figure 9: A request graph demonstrating deadlock.

between A and C.

A deadlock occurs when no two-fragment cycle exists in the request graph even when all fragments are stable. Figure 9 shows two such examples. In the first case, three fragments are locked in a cycle and in the other, one fragment has no outgoing edges and cannot merge with any other fragment. Either of these cases could mean that distributed K-SPH would never terminate. In the following, we show that such deadlocks cannot occur.

Lemma 2 *Distributed K-SPH does not deadlock.*

Proof: Let $d(I, J)$ represent the distance between fragments I and J. In Figure 9, stable fragments A, B and C are locked in a three-node cycle. Since each fragment prefers the closest fragment, the following inequalities must hold: $d(A, B) < d(A, C)$, $d(B, C) < d(B, A)$, and $d(C, A) < d(C, B)$. However, we know from Lemma 1 that at least two of the fragments, say A and B, must have equidistant shortest paths. This leads to a contradiction. A similar argument holds for any cycle of more than two fragments. Consider the case where no cycle exists as shown by fragments X and Y in Figure 9. Fragment Y has no outgoing edge, which indicates that it has no shortest path to any fragment. This means that Y's path to X must be blocked and by Lemma 1 will eventually be discovered. Distributed K-SPH terminates with an error when both X and Y have no outgoing edges.

Convergence Time and Number of Messages: We now derive some simple asymptotic bounds on the number of messages and convergence time of the distributed K-SPH algorithm.

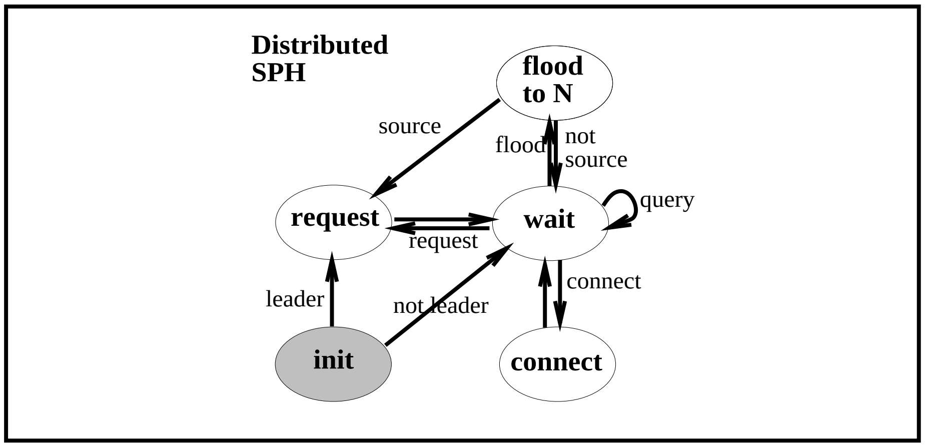


Figure 10: The finite state machine for each node performing distributed SPH.

Distributed K-SPH uses the least number of messages when the network has $Z = 2^i$ multicast nodes, any number of non-multicast nodes, and fragments always find a partner. Under these conditions, a total of $\frac{Z}{2^1} + \frac{Z}{2^2} + \cdots + \frac{Z}{2^i} = Z\frac{2^i-1}{2^i} = 2^i\frac{2^i-1}{2^i} = 2^i - 1$ merges occur during i rounds. Each fragment merges using a relatively small number of messages and the new fragment enters the discovery phase, In the discovery phase, each fragment node queries every child and neighbor for fragment and distance information. Assume that on average each fragment node queries a finite number of neighbors and children approximated by c. The total number of messages sent by multicast members during each round is $cZi = cZ\log Z = \Omega(Z\log Z)$. During each of the $\log Z$ rounds, the longest round-trip message time between leader and fragment root dominates. This round-trip time can be at worst twice the diameter of the graph, and at best a constant. Thus, the time to converge is lower-bounded by $c\log Z = \Omega(\log Z)$.

In the worst case, only one fragment finds a partner during any round. Thus, $Z-1$ rounds occur before a solution is found. If fragments are always relatively large then the number of messages would be the number of rounds times the number of nodes on all fragments, $c(Z-1)N = O(ZN)$. If the round-trip times during each of the $Z-1$ rounds is large and close to twice the graph diameter, $2D$, then the convergence time for this case is $2D(Z-1) = O(DZ)$.

These bounds are summarized in Tables 1 and 2, along with the bounds for the other algorithms considered in this paper. Our results from simulations of the algorithm show that the rates of increase of both the convergence time and the number of messages with the number of nodes fell within these bounds as shown in Section 3.

2.1.3 Distributed SPH

The distributed shortest path heuristic is a special case of distributed K-SPH described in section 2.1.1. In distributed SPH, any one of the multicast members may act as the source of the multicast, referred to here simply as the *source node*. In contrast to distributed K-SPH, only one fragment, the *source fragment* grows, connecting multicast members to itself until all the multicast members are part of the same fragment. The heuristic terminates when a single tree remains.

In SPH, the preferred fragment of every fragment is always the source fragment. The sole

Bound	Distributed K-SPH	Distributed SPH
Lower Bound	$Z \log Z$	Z^2
Upper Bound	ZN	ZN

Table 1: Messages bounds for distributed heuristics K-SPH and SPH.

Bound	Distributed K-SPH	Distributed SPH
Lower Bound	$\log Z$	DZ
Upper Bound	DZ	DZ

Table 2: Convergence-time bounds for distributed heuristics K-SPH and SPH.

exception, of course, is the source fragment itself which prefers its closest fragment. Note that all other fragments are trivial one-node subtrees containing one multicast member each. However, to maintain uniformity with our previous heuristic description we will continue to use the term fragment instead of multicast member. Using the same connection step outlined for heuristic K-SPH, the source fragment merges with its closest fragment. As the source fragment grows, it uses the same discovery step to determine the new, closest fragment. The source fragment never changes its index. This preserves the source fragment's original index so that non-source fragments never need to change their preferred fragment index. As a consequence, non-source fragments do not enter the discovery phase. In all other respects, distributed SPH is very similar to distributed K-SPH. Figure 10 shows the finite state machine used by each node.

Algorithm Analysis: Since distributed SPH is a special case of distributed K-SPH, its analysis proceeds similarly to that of distributed K-SPH. For example, it too will not deadlock as shown by Lemma 3.

Lemma 3 *Heuristic SPH does not deadlock.*

Proof: Consider Figure 9 again. A three-node cycle such as the one in Figure 9 cannot occur in distributed SPH because every fragment except the source prefers the source fragment. Thus, the longest request graph cycle in distributed SPH has length two: an edge from the source to a fragment and the return edge. A longer request cycle is an error. Likewise, a zero-node cycle indicates an error since every fragment except the source fragment always prefers the source fragment and the source fragment prefers its closest fragment.

Messages and Convergence Time: Like distributed K-SPH, distributed SPH uses the least number of messages when $Z = 2^i$ multicast members exist. Distributed SPH differs from distributed K-SPH in that only two fragments merge during a round. Assume that on average each fragment node queries a finite number of neighbors and children approximated by c. The number of messages in this case would be $c + 2c + 3c + \cdots + (Z-1)c = c\frac{(Z-1)^2-(Z-1)^2}{2} = O(Z^2)$. The round-trip time during each of the $Z-1$ rounds cannot be greater than twice the graph diameter and the convergence time is $c(2D)(Z-1) = \Omega(DZ)$.

In the worst case, assume that the source fragment grows quickly and the round-time distance for messages approaches twice the graph diameter, $2D$. The number of messages in this case is $c(Z-1)N = O(ZN)$. Likewise the convergence time becomes $2D(Z-1) = O(DZ)$.

The convergence-time and message bounds for the distributed heuristics are summarized in Tables 1 and 2, respectively.

3 Simulation Results

To evaluate the two distributed heuristics presented in the last section, we implemented the algorithms in a simulator and performed extensive simulations on randomly generated test networks. We choose the distributed MST algorithm due to Gallager, Humblet, and Spira [6] as our baseline algorithm to compare the results. This algorithm was used to produce a minimum spanning tree of the network graph, which was then pruned to obtain a Steiner tree. We chose this MST algorithm as our baseline algorithm because the majority of previous distributed algorithms reviewed find multicast trees are based on finding minimal spanning trees [4, 12]. This algorithm differs from our algorithms, distributed K-SPH and SPH, in the fact that all the network nodes must participate in the execution of the algorithm in the former, while only the multicast members and nodes in the vicinity of the multicast tree being set up execute the algorithm in the latter.

This section summarizes the simulation results and compares the algorithms in terms of their convergence time, competitiveness, and the number of messages exchanged.

3.1 Evaluation Methodology

3.1.1 Network model

Because our choice of existing network topologies and multicast applications was small, we chose to compare Steiner heuristics using randomly generated networks. Each algorithm was run on a total of 1000 test networks. Each of the 1000 networks is a sparse 200-node network. 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. Note that the number of edges in an n-node connected graph can vary from $n-1$ for a tree to $\binom{n}{2}$ for the complete graph on n nodes. For our test networks on 200 nodes, the number of edges must fall in the narrow range from 199 to 5% of maximum edges possible = 995. Figure 11 shows the distribution of number of edges for our test networks. We believe such a graph describes a plausible WAN because a large network is likely to be loosely interconnected. 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. For example, consider a video conference in a large corporate network. The conference is most likely to directly involve a minority of nodes in the network. The choice of 10% and 30% of the nodes was made since these figures represent more difficult cases of the Steiner problem. Later in Section 3.3 we discuss how these heuristics scale with increasing multicast membership size (10% to 90%) and network size (20 to 200 nodes).

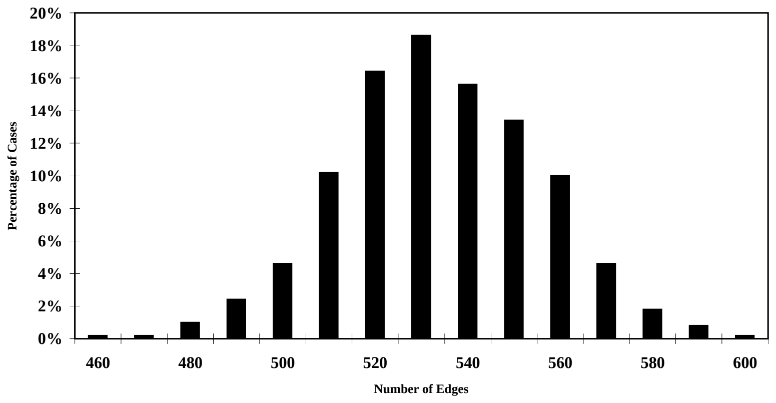


Figure 11: The histogram of the number of edges in the test graphs.

The 1000 networks were generated to resemble real networks in a manner similar to that of Doar [5]. Each of the 200 nodes is distributed across a Cartesian coordinate plane with minimum and maximum coordinates $(0, 0)$ and $(400, 400)$, creating a forest of 200 nodes spread across this plane. 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 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. Each edge's distance is its rectilinear distance plus a small constant. This distance is also the time it takes for a message to traverse this edge. 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 [5]. 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, we chose $\alpha = 0.10$ and $\beta = 0.20$ for generating the graphs used in this simulation. These values produced graphs of realistic density.

We performed two different simulations on each generated graph by varying the multicast group size in two ways; the number of multicast nodes was chosen as either 20 or 60 of the 200 nodes. Results are presented for both combinations for each graph. The nodes in a multicast group were chosen randomly in each case. The random numbers were chosen from a uniform distribution. To ensure fairness, each heuristic was run on the same 1000 networks.

3.2 Evaluation Metrics

The metrics we use for comparison are the competitiveness, convergence time, and messages passed. Competitiveness is the ratio of heuristic tree cost $C(T)$ to that of the best solution C_{best} found by any heuristic. To determine the best solution, we considered solutions produced by the two distributed heuristics described in this paper and the distributed MST algorithm, as well as the serial heuristics described in [1]. We use the best heuristic solution found for each test network rather than an optimal solution because explicit algorithms to find optimal solutions are prohibitively expensive on large networks. The convergence time was found by measuring the elapsed time in the simulated network from the start of simulation to the time at which the last message reaches its destination. Since message-passing delays are likely to dominate processing delays on the convergence time of the algorithm in a wide-area network, we considered only the former in computing the simulation time. We used the distance between two nodes as the delay to pass a message between them. Messages passed is the total number of messages passed between nodes before convergence.

3.3 Simulation Results

Having described the algorithms and the simulation environment, we now turn to the results of our simulations.

Figure 12 shows the competitiveness distribution for the centralized versions of SPH, K-SPH, and pruned MST algorithms. Each of the three plots shows the cumulative percentage of cases whose competitiveness is less than or equal to a given value. Figure 13 shows the same distributions for the distributed versions of the three algorithms. Note that the distributed versions of SPH and K-SPH may provide inferior solutions compared to their centralized versions because of the lack of global topology information in each node in the former. However, the degradation in the competitiveness was small in our test networks. In fact, the competitiveness produced by distributed K-SPH was often superior to that of centralized SPH.

When comparing the competitiveness, heuristics SPH and K-SPH consistently outperformed the pruned MST heuristic, in both centralized and distributed cases. This result is consistent with the known theoretical upper bounds on the heuristics. It has been shown that the cost of a solution produced by either SPH or K-SPH is within twice the cost of an optimal solution [20]. In contrast, the ratio between the cost of a solution produced by pruning a minimum spanning tree and that of an optimal solution can be as large as the number of non-multicast nodes [17]. In our case, the cost of pruned MST solutions was rarely worse than twice that of the best solution found, but was often significantly worse than that produced by shortest path heuristics. Figure 14 displays the complete cumulative distribution for the pruned minimum spanning tree algorithm. In Figure 13, 90% of the solutions produced by both distributed K-SPH and SPH were within 4% of the best in terms of their cost. In comparison, when the best 90% of the solutions produced by the pruned MST algorithm were considered, some of the solutions had costs as high as 50% more than that of the optimal algorithm. Thus, if competitiveness is the most important criterion in the choice of the algorithm, distributed K-SPH is the heuristic of choice.

Heuristics SPH and K-SPH also enjoy the advantage that neither requires the participation of all the nodes in the network. Only the nodes in the multicast tree and within its neighborhood need to participate in the execution of the algorithm. The pruned minimum spanning tree algorithm, on the other hand, requires participation from every node of the network, a condition difficult to satisfy in practice in a large wide-area network.

Viewed from the perspective of messages exchanged and convergence-time, however, the pruned MST heuristic enjoys an advantage over shortest path heuristics SPH and K-SPH. Figure 15 displays the cumulative percentage of networks solved within a given number of messages for the

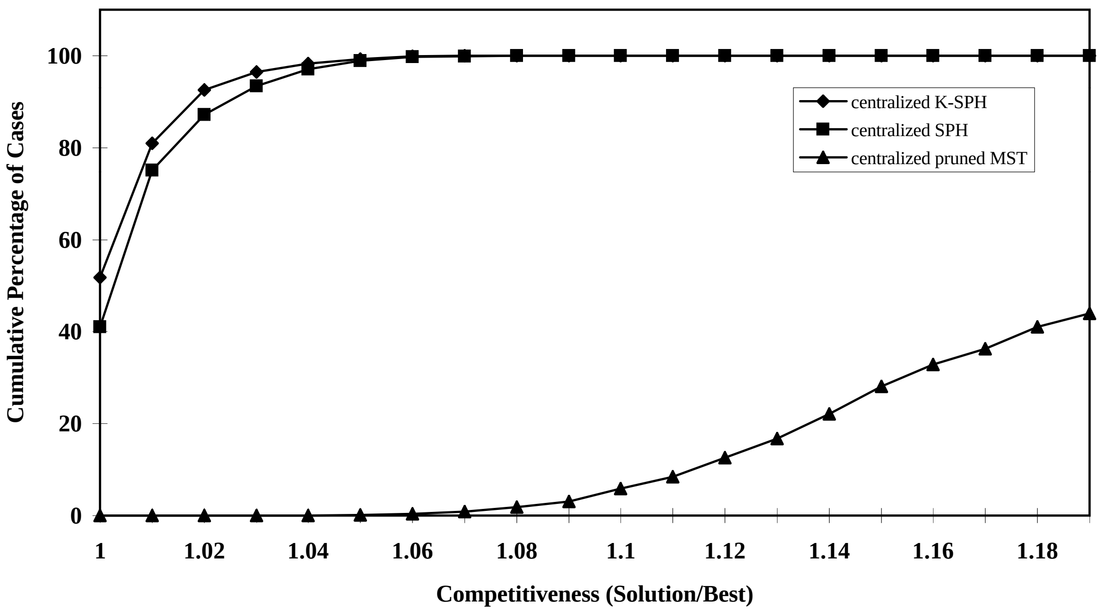


Figure 12: Competitiveness distribution for centralized Steiner heuristics.

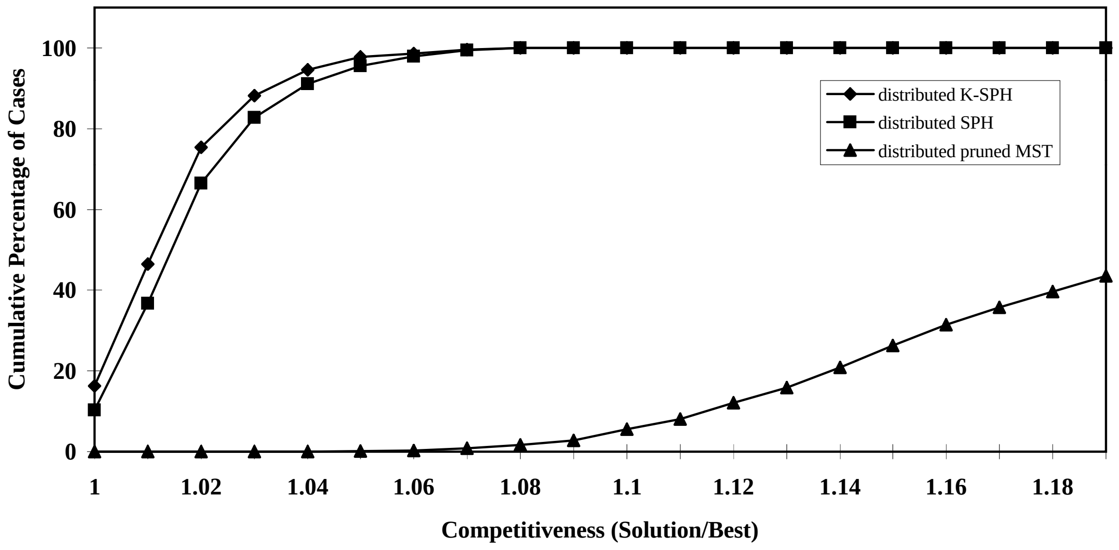


Figure 13: Competitiveness distribution for distributed Steiner heuristics.

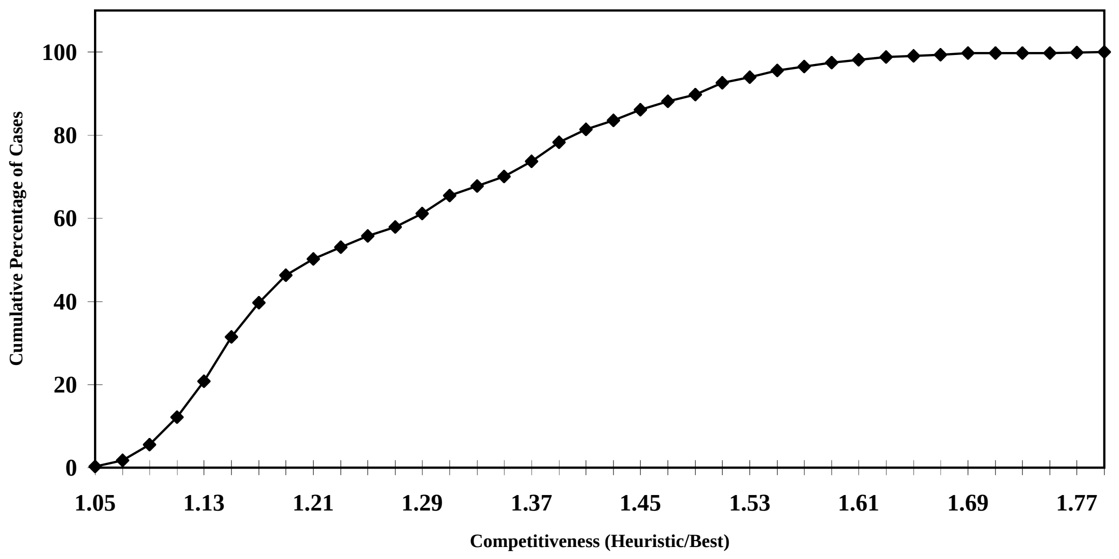


Figure 14: Competitiveness distribution for the distributed pruned minimum spanning tree heuristic.

three algorithms. Likewise, Figure 16 displays the cumulative percentage of networks solved within a given convergence time. Both the number of messages and the convergence time for the solutions produced by pruned MST algorithm fell well within a much narrower range as compared to the results for distributed K-SPH and SPH. This is again consistent with the known theoretical bounds on the number of messages generated by the heuristics. In the case of the pruned MST heuristic, the additional effort to prune the minimum spanning tree is inconsequential as compared to the effort required to find the minimum spanning tree. Thus, the theoretical upper bound on the number of messages in the pruned MST heuristic is $O(N \log_2 N + E)$ [6]. In comparison, the upper bound on the number of messages for both SPH and K-SPH is $O(ZN)$. Thus, when the number of multicast nodes is large in comparison to $\log N$, the pruned MST heuristic has a smaller upper bound on the number of messages.

On comparing the SPH and K-SPH algorithms, it is interesting to observe that the algorithms had the same level of communication complexity in terms of the number of messages generated, yet the range of convergence times produced by K-SPH was significantly tighter. This is primarily due to the disparate approaches used by the algorithms in growing the multicast tree. Distributed SPH grows the tree by adding one multicast member at a time to the source fragment, concentrating much of the work at the source, while distributed K-SPH allows multiple fragments of the tree to combine in parallel. This allows distributed K-SPH to provide lower convergence times without increasing the number of messages substantially.

To answer the question of how the distributed heuristics scale with multicast group size and network size, we performed 3600 additional simulations summarized by Figures 17 through 19.

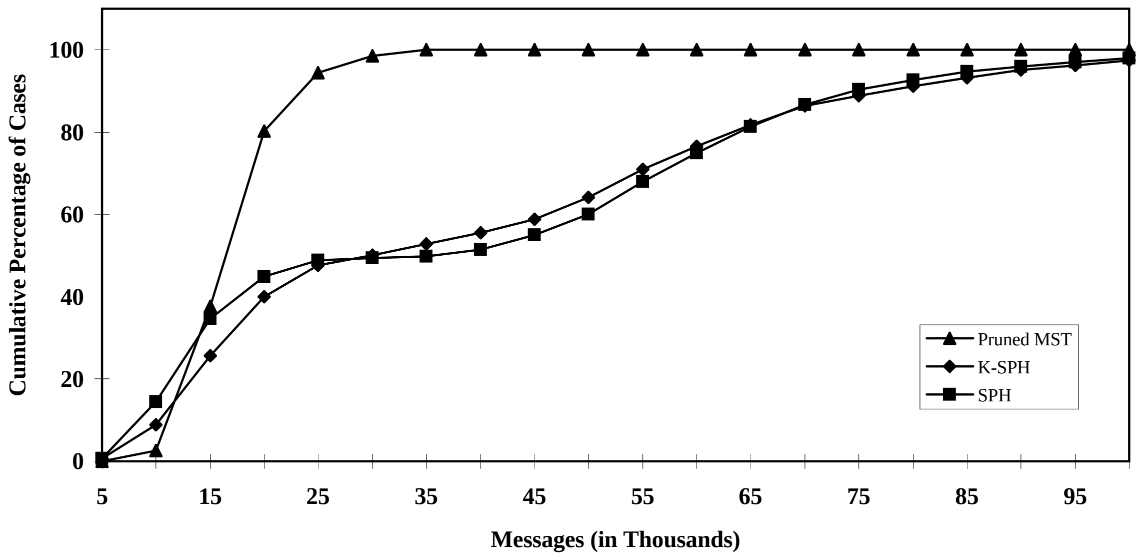


Figure 15: Cumulative distribution for number of messages transmitted by the three distributed Steiner heuristics.

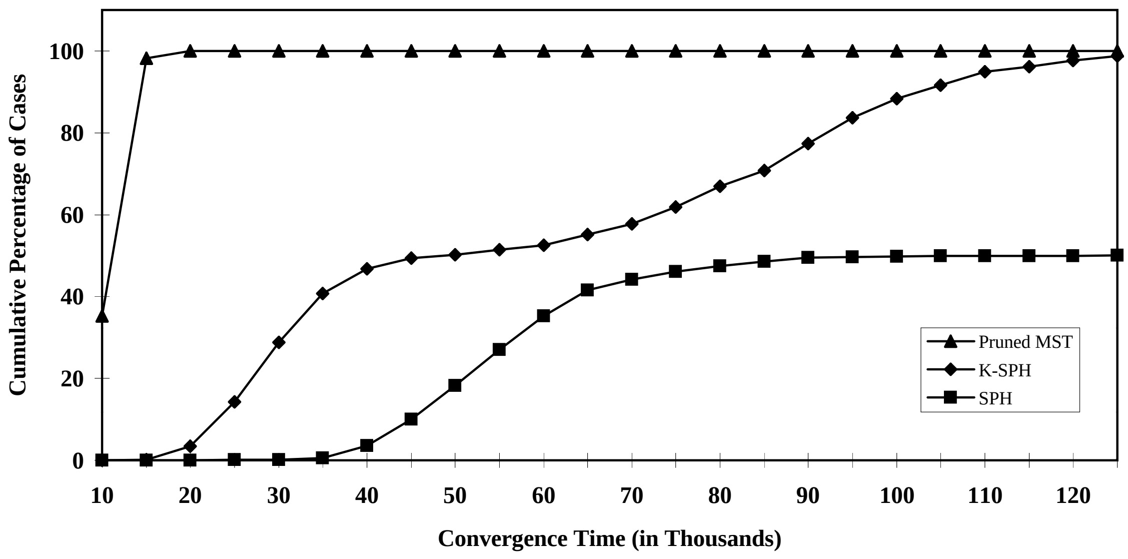


Figure 16: Convergence time distribution for distributed Steiner heuristics.

Figures 17 and 18 show the average convergence time passed for both SPH and K-SPH when either the multicast group size is varied from 10% to 90% in a 200-node graph or when network size is varied from 20 nodes to 200 nodes. Each graph point summarizes the average value for 200 test networks. Minimum and maximum values for each are displayed using error bars. Similarly, Figure 19 summarizes the average messages passed when membership size and graph size are varied. The equivalent graphs for distributed heuristic SPH are omitted since they mirror the results shown in the convergence time graphs. These graphs demonstrate that distributed heuristics SPH and K-SPH scale reasonably well for both multicast-group size and network size.

Even though the convergence times for distributed K-SPH in Figure 16 are higher than those of the pruned MST algorithm by as much as 10 times, we believe that the former can be brought down by modifying distributed K-SPH in the following ways:

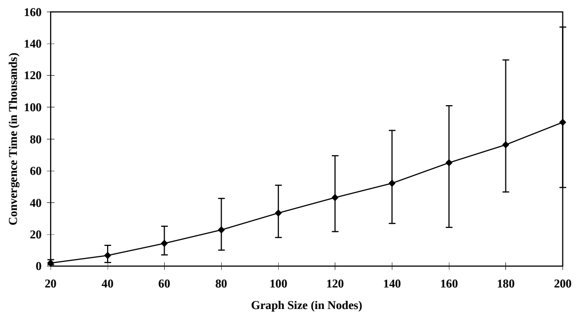
1. If a fragment receives a reject message because the preferred fragment has already merged with another, the fragment enters the discovery step and looks for the next closest fragment. If the rejecting fragment indicates its new fragment index, a fragment could skip the discovery step and send a merge request to the new, merged component. Preliminary tests indicate that this can reduce convergence time by 5%.

2. Inefficiencies result when a large fragment merges with a small fragment. This inefficiency is evident in distributed SPH because the largest fragment, the source fragment, always merges with a fragment containing a single multicast member. Although such merges cannot be avoided in distributed K-SPH, the duration of such a merge can be reduced by taking into account the fragment size when merging fragments as follows: if a small fragment is added to a large fragment, the discovery step can be shortened by performing a *partial* discovery step. A partial discovery step updates the fragment leader's knowledge of fragment distances using existing fragment distance information and new information gathered by propagating a multicast through the smaller fragment's nodes. Such a partial discovery step is significantly faster — as much as 35% in our tests. However, the new fragment may be unaware of fragment distance information it might otherwise have found through a full discovery step and hence the competitiveness may suffer. Thus, this approach trades off competitiveness for speed.

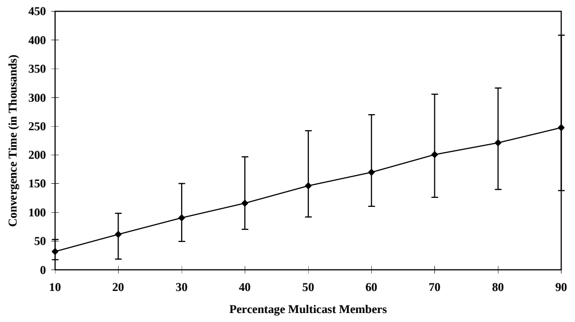
3. In our simulations, we observed that few connection attempts between fragments are blocked in practice. This allows the connection and discovery steps to be overlapped, reducing the time it takes for two fragments to complete a merge. The price paid is an occasional failed connection attempt requiring longer discovery steps in both the original fragments to restore the fragments to their original states.

4 Concluding Remarks

In this paper we introduced two distributed heuristics based on shortest path Steiner hueristics, and evaluated their performance relative to a baseline pruned minimum spanning-tree heuristic. The primary advantage of our distributed algorithms over previous algorithms is that they require participation from only the nodes in the multicast tree and within their neighborhood. Among the two algorithms studied, distributed K-SPH emerged as the clear winner; in comparison to distributed SPH, it has substantially lower convergence time and slightly better competitiveness.

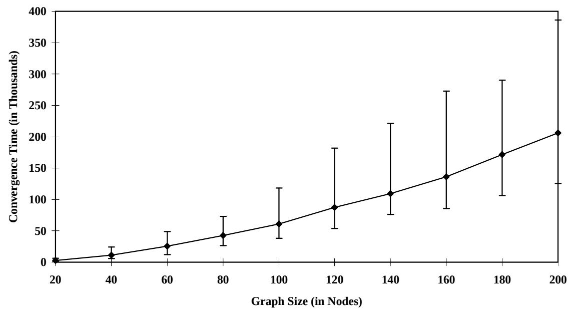


(a)

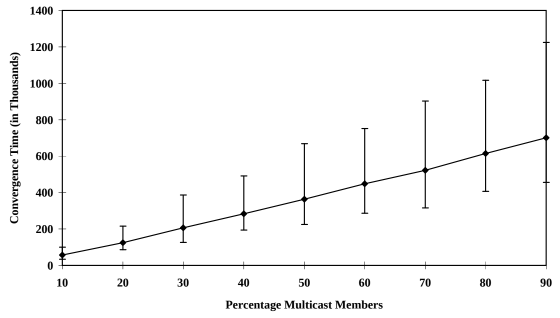


(b)

Figure 17: Convergence time of the distributed K-SPH algorithm as a function of network size (a) with 30% multicast membership, and (b) as a percentage of multicast membership in a 200-node network.

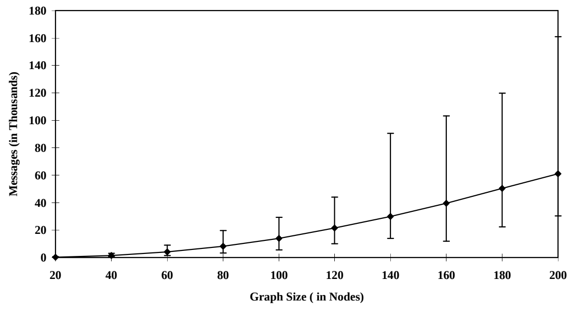


(a)

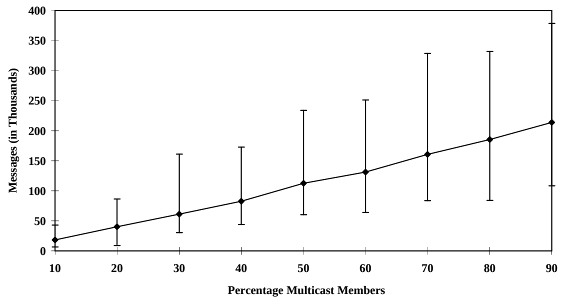


(b)

Figure 18: Convergence time of the distributed SPH algorithm as a function of network size (a) with 30% multicast membership, and (b) as a percentage of multicast membership in a 200-node network.



(a)



(b)

Figure 19: Number of messaged generated by the distributed K-SPH algorithm (a) as a function of network size with 30% membership, and (b) as a percentage of multicast membership in a 200-node network.

The heuristics developed are an improvement over existing distributed Steiner heuristics based on the minimum spanning tree [4, 12] for two reasons: they produce solutions of superior quality in most cases and requires the participation of only a subset of network nodes. Our results show that the competitiveness of the solutions produced by both of our algorithms were, on the average, at least 25 percent better in comparison to those produced by the pruned spanning-tree approach. In addition, the competitiveness found by our algorithms in almost all cases was within 10% of the best solution found by any of the Steiner heuristics considered, including both centralized and distributed algorithms.

Limiting the execution of the algorithm to a subset of the nodes in the network can result in an increase in convergence time over the pruned spanning-tree approach. Indeed, the convergence time of distributed K-SPH was as large as ten times that of pruned MST algorithm in many our test networks. However, we believe that the distributed K-SPH algorithm can be streamlined in several ways, as discussed in the last section, to narrow this gap.

Areas for future research include provisions for robustness of the algorithms in an environment where node and link failures occur. The heuristics as stated here assume reliable delivery of messages and a stable topology during their execution. If an environment is assumed where nodes and links do fail during execution, a combination of schemes to ensure convergence and correctness need to be applied. These would certainly include an adaptive all-paths distributed algorithm such as the one described by Humblet [7] and a timeout mechanism to detect rejected merger requests. Additional work is required to study the effectiveness and performance of these schemes.

References

[1] F. Bauer and A. Varma. "Degree-constrained multicasting in point-to-point networks," in *Proc. IEEE INFOCOM*, Boston, Apr. 1995, pp. 369–376.

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

[3] 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.

[4] G. Chen, M. Houle, and M. Kuo. "The Steiner problem in distributed computing systems," *Information Sciences*, vol. 74, no. 1-2, pp. 73–96, Oct. 1993.

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

[6] R. Gallager, P. Humblet, and P. Spira. "A distributed algorithm for minimum-weight spanning trees," *ACM Transactions on Programming Languages and Systems*, vol. 5, no. 1, pp. 66–77, Jan. 1983.

[7] P. Humblet. "Another adaptive distributed shortest path algorithm," *IEEE/ACM Transactions on Communications*, vol. 39, no. 6, pp. 995–1003, Jun. 1991.

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

[9] M. Imase and B. Waxman. "Dynamic Steiner tree problem," *SIAM J. Disc. Math.*, vol. 4, no. 3, pp. 369–384, Aug. 1991.

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

[11] 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.

[12] V. Kompella, J. Pasquale, and G. Polyzos. "Two distributed algorithms for the constrained Steiner tree problem," in *Proc. Comput. Commun. and Netw.*, San Diego, CA, Jun. 1993.

[13] 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.

[14] V. Rayward-Smith and A. Clare. "On finding Steiner vertices," *Networks*, vol. 16, pp. 283–294, 1986.

[15] 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.

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

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

[18] B. Waxman. "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.

[19] J. Westbrook and D. Yan. "Greedy algorithms for the on-line Steiner tree and generalized Steiner problems," in *Algorithms and data structures. Third Workshop, WADS '93.*, Montreal, Quebec, Canada, Aug. 1993, pp. 621–633.

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