ARIES: A Rearrangeable Inexpensive Edge-based On-line Steiner Algorithm

Fred Bauer
Anujan Varma

UCSC-CRL-95-36
July 18, 1995

Computer Engineering Department
University of California
Santa Cruz, CA 95064

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

Abstract

In this paper, we propose and evaluate ARIES, a heuristic for updating multicast trees dynamically in large point-to-point networks. The algorithm is based on monitoring the accumulated damage to the multicast tree within local regions of the tree as nodes are added and deleted, and triggering a rearrangement when the number of changes within a connected subtree crosses a set threshold. We derive an analytical upper-bound on the competitiveness of the algorithm. We also present simulation results to compare the average-case performance of the algorithm with two other known algorithms for the dynamic multicast problem, GREEDY and EBA (Edge-Bounded Algorithm). Our results show that ARIES provides the best balance among competitiveness, computational effort, and changes in the multicast tree after each update.

Keywords: multicast algorithms, on-line Steiner problem, rearrangeable multicast 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. A shortened version of this paper will appear in the proceedings of INFOCOM '96.

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. Multicasting is usually supported in a point-to-point packet network by setting up a multicast tree connecting the members of the multicast group. Many of the multicast applications also require the network to support dynamic multicast sessions, where the membership of the multicast group changes frequently. Supporting such applications efficiently requires the ability to alter an existing multicast tree to accommodate membership changes as nodes join and leave the multicast session. While much has been written on the subject of establishing a static multicast tree in point-to-point networks [1, 4, 10], algorithms to modify an existing multicast tree by adding and deleting members is a relatively unexplored area of research. Since many multicast applications are delay-sensitive, the efficiency of the algorithm used to maintain the multicast tree assumes special significance. This paper presents a new, efficient heuristic for updating the multicast tree for dynamic multicast groups.

Previous authors have established that determining an optimal multicast tree for a static multicast group may be modeled as the NP-complete *Steiner problem in networks* [3, 4, 5, 7, 16]. Consequently. its explicit solutions are prohibitively expensive. For example, two popular explicit algorithms, the *spanning tree enumeration algorithm* and the *dynamic programming algorithm* [16], 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 heuristics exist for the Steiner problem in networks and have been reviewed extensively elsewhere [1, 5, 7, 10, 11, 12, 13, 16].

This paper addresses the problem of modifying an existing multicast tree when new members enter or existing members leave the multicast group. The problem of updating the multicast tree after each addition and deletion is known as the *on-line multicast problem in networks.* This paper focuses on its Steiner equivalent, the *on-line Steiner problem in networks* [15]. If the m multicast members added or deleted are represented by a vector $R = \{r_1, r_2, \ldots r_m\}$, where each element of vector R is a request to add or delete a member, the on-line multicast problem is defined formally as follows.

GIVEN: A simple, undirected, connected graph $G = (V, E)$ with n nodes, non-negative edge costs, p multicast members $Z \subseteq V$, request vector $R = \{r_1, r_2, \ldots r_m\}$, and existing multicast tree $T = (V', E')$, $V' \subseteq V$ and $E' \subseteq E$.

FIND: Multicast trees $T_1, T_2, \ldots T_m$ such that each tree T_i's members are those of tree T modified by requests $r_1, r_2, \ldots r_i$ and T_i's cost (sum of edge weights) is minimum among all possible choices for tree T_i.

In the extreme case, a multicast tree may be completely rebuilt after each change using a static multicast heuristic. This approach, however, is prohibitively expensive if used for each addition and deletion. In addition, real-time multicast sessions cannot tolerate large changes in the multicast tree after each update. This is because data packets are constantly in flight within the multicast tree and large changes to the tree might cause an unacceptable disruption in the packet flow. An ideal multicast algorithm, therefore, must minimize both the number of changes between successive updates and the cost of the multicast tree formed after each update. We know, however, that the Steiner problem in graphs is NP-complete and no such ideal algorithm exists which runs in polynomial time.

The on-line multicast problem was first presented by Waxman [14] and has received little attention since [8, 9, 15]. We present a new heuristic for the on-line Steiner problem, balancing heuristic run-time against competitiveness, that is, the ratio between the cost of the heuristic tree and the cost of an optimal tree. The cost of the tree is taken as the sum of the weights of the edges in the tree. We derive analytical bounds on the competitiveness of the new heuristic. In addition, we also compare the heuristic with previous on-line heuristics by simulation on a large number of random test graphs representing sparse, point-to-point networks with low to medium multicast membership. 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 feasible 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 or distance education. The heuristics are compared on the basis of three criteria: competitiveness, CPU time, and differences between successive trees. Note that our results are not specific to any particular type of network such as the Internet or ATM networks, but apply to both connectionless and virtual-circuit-based networks.

Our results show that heuristic ARIES performs extremely well — often producing solutions close in quality to those generated by a good, static Steiner heuristic. In addition, it often outperformed two known heuristics — the *Edge-Bounded Algorithm* (EBA) and GREEDY. Further, the algorithm may be tuned to achieve the desired tradeoff between performance and cost. The difference between successive trees produced by ARIES is often small, reducing the disruption caused by rearrangements.

The remainder of this paper is organized as follows. Section 2 reviews previous heuristics for the on-line Steiner problem and summarizes their bounds. Section 3 describes our new on-line heuristic and Section 4 derives an upper bound on its competitiveness. Section 5 presents simulations results comparing our algorithm to previous on-line heuristics. Finally, Section 6 concludes the paper with a discussion of the results.

2 Previous Algorithms for the On-Line Steiner Problem in Networks

This section summarizes previous algorithms to solve the on-line Steiner problem. We use the following basic definitions and notations in the paper. Z is the set of multicast nodes, 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 length of path $P_{i,j}$. Because multicast nodes enter and leave the multicast tree over time, the node sets Z and S also vary over time. The *cost* of a tree is the sum of its edge weights. Multicast nodes are referred to as *members* and non-multicast nodes as *non-members*. The distance between two nodes is defined to be the distance of the shortest path between them. Likewise, the distance between a node and a tree is the minimum among the shortest paths between the node and every 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.

In his original paper on the on-line multicast problem (referred to as the *dynamic multipoint problem*), Waxman divides on-line heuristics into two types: those that allow rearrangement of the tree and those that do not [14]. In this first paper and a subsequent one, Waxman and Imase describe a heuristic of each type [8, 14]. We summarize both heuristics below for the reader's convenience.

2.1 Heuristic GREEDY

The non-rearrangement on-line heuristic, GREEDY [8, 14], perturbs the existing tree as little as possible. For each add request, it connects the new member to the nearest tree node using the shortest path. For each delete request, GREEDY deletes only leaf nodes. If this deletion creates a non-member leaf, GREEDY also deletes the new leaf. This continues until no non-member leaves remain.

2.2 The Edge Bounded Algorithm

The rearrangement heuristic EBA (the edge-bounded algorithm) [8] enforces bounds on the distance between nodes in the tree after each change. In addition, every tree T_i must also be an *extension tree* for its set of multicast members. An extension tree is one that contains all multicast members and for which the degree of every non-member is greater than two. EBA starts by converting the original graph to a complete distance graph. In a distance graph every edge represents the shortest path between nodes. The original tree is also converted to its distance graph equivalent. For each add request, EBA connects the new member by its shortest edge to the existing tree. This shortest edge represents the shortest path between the new member and the closest tree node in the original graph. EBA then tests the length of each path in the tree from the new member to every node in the tree. If the length of the path consisting entirely of tree edges between the new member and a node in the tree does not fall within constant k times the length of the shortest path in the original graph, EBA modifies the tree as follows: The maximum-weight tree edge in the path between the new node and the offending tree node is deleted, breaking the tree into two components. The two components are then reconnected by adding the edge between the two nodes. This edge represents their shortest path. If the new tree now contains non-members of degree less than three, these non-member nodes are deleted as described in the next paragraph. Since Imase and Waxman use constant $k = 2$ in [8], we also use $k = 2$ in our simulations.

For each delete request, EBA's actions are related to the deleted node's degree with respect to the multicast tree. If the node has degree three or more, no action is taken. If the node has degree one, it is deleted just as in GREEDY. If the node has degree two, it and its adjacent edges are deleted, splitting the tree into two components. These two components are reconnected by the path between the node's neighbors that has the smallest maximum weight edge. If this deletion results in a non-member edge of degree two, the non-member is deleted and the process repeats until all non-member nodes in the tree have degree three or more.

2.3 The Geographic-Spread Dynamic Multicast Heuristic

A second rearrangement heuristic is Kadirire's GDSM, the *geographic-spread dynamic multicast routing algorithm* [9]. When adding nodes, this heuristic relies on exploring four explicit connection configurations between nodes in the tree and the new member. For each add request, it identifies the closest tree node to the new member and the closest members in the tree on either side of the closest tree node. The heuristic then chooses one of four ways to connect the new member through the three identified nodes. If more than one cheapest alternative exists, GDSM chooses the one with greatest *geographic spread* [9]. For each delete request, GDSM behaves exactly as GREEDY does.

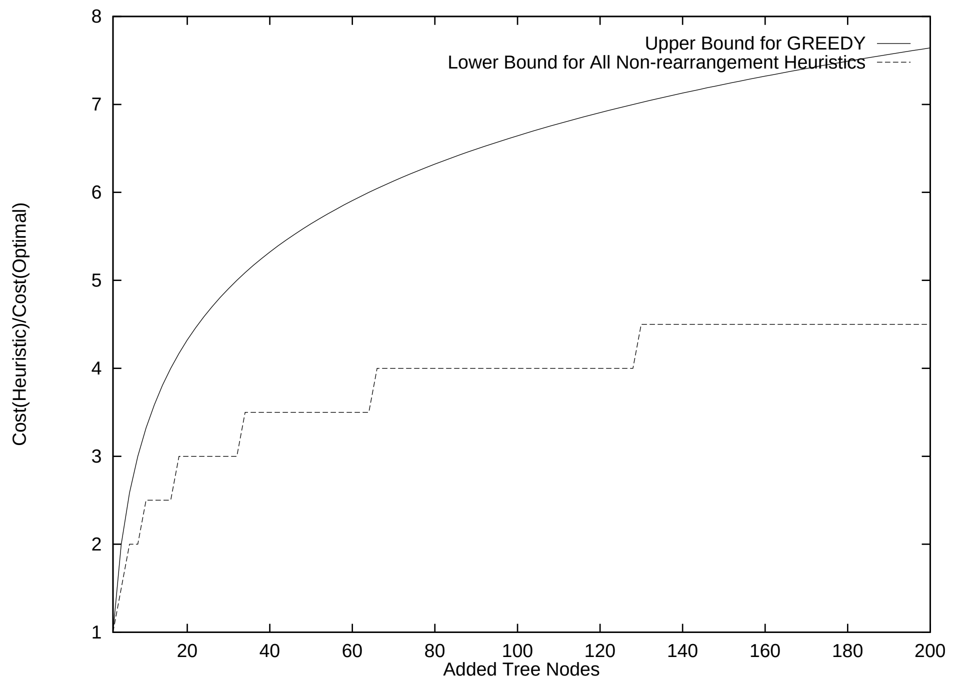

Figure 1: Bounds on non-rearrangement heuristics' competitiveness for add requests.

2.4 Bounds for On-Line Heuristics

Waxman and Imase [8] provide bounds for the *competitiveness* of all non-rearrangement heuristics in general and GREEDY in specific when only add requests are honored. Here competitiveness is defined as the ratio between the cost of a multicast tree found by the heuristic and that of an optimal tree. The lower bound for all non-rearrangement algorithms is

$$1 + \frac{1}{2}\lfloor \log_2(n_i - 1) \rfloor,$$

and the upper bound for GREEDY is

$$\log_2(n_i),$$

where n_i is the number of nodes added to the tree while processing the requests $r_1, r_2, \ldots r_i$. These bounds are shown in Figure 1. Westbrook [15] provides a tighter upper bound for heuristic GREEDY when only add requests are honored. This upper bound is

$$O(\log_2(\frac{d_{max}}{C_{opt}} n_i)),$$

where d_{max} is the maximum distance between members along the tree and C_{opt} is the cost of an optimal tree for the modified multicast group. Imase and Waxman have shown that *no* such finite bound exists if delete requests are also honored [8].

A rearrangeable heuristic, however, can have a finite bound for competitiveness for both add and delete requests. Heuristic EBA's upper bound is 4δ where δ is EBA's constant as described in Section 2.2. In both our simulations and Imase and Waxman's paper $\delta = 2$ and the upper bound is therefore 8 [8].

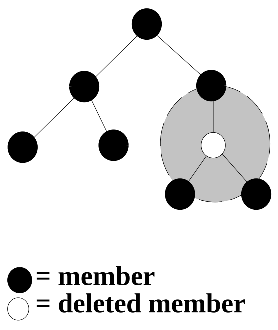

Figure 2: A tree with one deleted member.

No similar bounds are given for GDSM. However, since GDSM is a non-rearrangement heuristic similar to GREEDY, we assume its bounds to be similar to GREEDY's.

3 Heuristic ARIES

While our simulation results show GREEDY to be superior to EBA in terms of competitiveness on our random test graphs, the former still suffers from the disadvantage that it may perform poorly for delete requests. A superior rearrangeable heuristic would be one that combines GREEDY's light computation requirements with the ability to force a rearrangement when the solution tree's competitiveness has degraded beyond a certain threshold. This was our motivation for creating heuristic ARIES.

Like GREEDY, ARIES does the minimum necessary modifications for each add and delete request. For each add request, ARIES joins the new member to the existing tree by the shortest path. For each delete request, ARIES deletes the node only if it is a leaf. The difference lies in ARIES's rearrangement mechanism. When the accumulated *damage* to a part of the tree is judged to be too high, ARIES rearranges that region of the tree as described in the next paragraph. Formally, damage to the entire tree is measured by the tree's *degradation factor* $C(i) - C_{opt}(i)$ where $C(i)$ is the cost of the modified tree after request r_i and $C_{opt}(i)$ is the cost of an optimal Steiner tree after this same request r_i. As the tree is modified, this degradation factor will tend to increase.

Informally, ARIES monitors the degradation factor of the heuristic tree and rearranges portions of the tree as necessary to reduce the degradation factor. In the example given in Figure 2, the grey shaded region of the tree is that portion of the tree most affected by the deletion of the white node. The "deleted" node is not removed from the tree, but merely marked as a former member of the multicast group. If ARIES judges that the damage to the tree introduced by the white node exceeds a given threshold, only the grey shaded portion of the tree would be rearranged, sparing the majority of the tree from disruption during the rearrangement. In this way, the effort required to maintain a low cost tree is performed only on the most likely region of the tree.

Damage to the tree is monitored by a number of counters within multicast member nodes that register the number of changes (additions or deletions) in their immediate vicinity. Each member node has one counter corresponding to each of its edges that form part of the tree. When a member node is deleted or a new node added, the node sends a message to all its multicast neighbors to increase their counters by one. Propagation of the counter-update messages is confined to regions within the tree bordered by multicast members, so that rearrangements can be confined to these regions as well. A rearrangement is triggered when the value of a counter in a multicast node exceeds a chosen threshold. We will later formalize the concept of the "region" used to confine the rearrangements.

3.1 Definitions and Notations

Having outlined the basic concepts, we can now formalize our description of ARIES. We begin by introducing the following assumptions and definitions. W assume that additions and deletions occur one at a time with adequate time between events to allow each node's edge counters to settle before the next addition or deletion event. We further assume that a reliable protocol is used to communicate counter updates within the network.

Let $G(t)$ denote the set of nodes belonging to the multicast session at time t. Let $t = 0$ represent the time the multicast session was first set up. In general, the multicast tree for $G(t)$ may contain four different types of nodes at any time $t > 0$:

1. Active multicast nodes: These are nodes currently belonging to the multicast group $G(t)$. A node $i \in G(t)$ is defined as an active multicast node if it satisfies one of the following conditions:

    (a) Node i has remained a member of the multicast group since the group was first set up, that is, $i \in G(\tau)$ for all τ in the interval $0 \leq \tau \leq t$;

    (b) A part of the multicast tree containing i was rearranged since the most recent addition of i to the multicast group.

    We refer to the set of active multicast nodes at time t as $Z(t)$, or simply as Z.

2. Deleted nodes: These are nodes that were once part of the multicast group, but were since deleted. A node $i \notin G(t)$ is defined as a deleted node if it satisfies the following condition: Let $\tau < t$ be the last time i was a member of the multicast group. Then, no rearrangements have occurred in any part of the multicast tree containing i during the interval (τ, t).

    We use $D(t)$, or simply D, to denote the set of deleted nodes at time t.

3. Appended nodes: An appended node is one that was added to the group after it was formed and has not been involved in a rearrangement since it was added. Formally, a node $i \in G(t)$ is defined as an appended node if the following condition is satisfied: Let $\tau > 0$ be the time at which node i was last added to the multicast group G. Then no rearrangements have occurred in any part of the multicast tree containing i during the interval (τ, t).

    We use $A(t)$, or simply A, to denote the set of appended nodes at time t.

4. Non-multicast nodes: These are nodes currently not belonging to any of the previous three categories. A node belongs to this set if it satisfies one of the following conditions.

    (a) Node j was never part of the multicast group G during the interval $(0, t)$, or

    (b) node j was last deleted from G at time $\tau < t$, and a rearrangement occurred in a part of the multicast tree containing j after time τ.

    We refer to this set of non-multicast nodes as S-nodes.

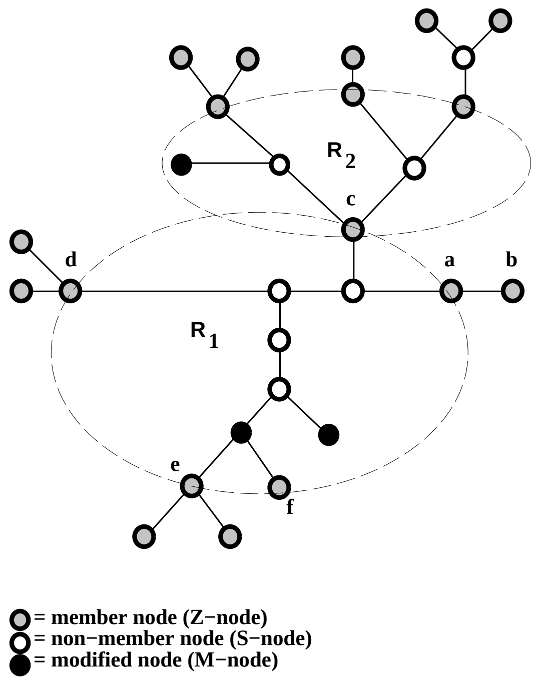

Figure 3: A multicast tree containing two modified regions R_1 and R_2.

Thus, the damage to the multicast tree is in the areas surrounding the deleted and appended nodes. We use *modified node*, or M-node, as a common term to refer to both these types of nodes. That is, the set of modified nodes is defined by

$$M(t) = D(t) \cup A(t).$$

Since the area of damage to the multicast tree is confined to the modified nodes and the active multicast neighbors surrounding them, it becomes necessary to formalize the concept of *multicast neighbor*.

Definition 1: Two nodes $i, j \in Z(t) \cup M(t)$ are multicast neighbors in T if and only if either (i) i and j are directly connected through an edge in T, or (ii) the path between i and j in T passes through only nodes in the set $S(t)$.

For example, in Figure 3, the multicast neighbors of node a are b, c, d, e, and f.

If rearrangements are performed locally, they should be applied to areas where the damage is likely to be maximum. The following definition allows grouping of M-nodes into regions so that rearrangements can be confined within one or more of the regions where changes have occurred.

Definition 2: A *modified region* R_i of a multicast tree T is a subtree of T defined as follows:

1. R_i contains at least one modified node in the set M.
2. If a modified node j is in R_i, then every multicast neighbor of j in T is also in R_i.
3. If two nodes j and k belong to R_i, then all the edges in the path between j and k in T are included in R_j.

Thus, for a given M-node j, the modified region containing j is the maximal connected subtree of T containing only S-nodes or M-nodes as its internal nodes. A leaf node of R_i may be either a Z-node or a pendant

M-node in T, while an internal node must be an M-node.

We refer to a modified region simply as *region* for convenience. When a local rearrangement is performed on T, the edges in the region are the candidates for rearrangement. For example, the modified nodes in the example tree of Figure 3 form two modified regions R_1 and R_2.

3.2 Algorithm Description

Having introduced the concept of a region, we can now describe the details of the algorithm. The number of modified nodes in each region is monitored by a set of counters, one within each Z-node belonging to the region. Since a Z-node can be part of as many distinct regions as its degree, the maximum number of counters needed in a Z-node is equal to its degree. When the multicast tree is initially set up, the counters in each node are reset to zero. When a Z-node is deleted or a new node added, each Z-node in that region increment its counter corresponding to the region. This can be achieved in a distributed system by the modified node broadcasting a counter-update message to all the Z-nodes within the region along the multicast tree.

The pseudocode of ARIES is shown in Figure 4. Given a Steiner tree T_{i-1} and an update request r_i, the objective of the algorithm is to determine a Steiner tree T_i for the modified multicast group after the update. Note that r_i can either be an add request or a delete request; these two cases are handled separately.

If the request is an add, the new node v is connected to the existing tree T_{i-1} via the shortest path from v to T_{i-1}. The only exception to this is the trivial case when v is an S-node in the existing tree. The new node is then marked an an A-node. If all of v's multicast neighbors are currently Z-nodes, then a new region is formed containing node v and its multicast neighbors. On the other hand, if v is already part of a region, at least one of the multicast neighbors of v must be an M-node. In either case, let R_j denote the region containing v. Every Z-node u within region R_j has a counter $c_{u,j}$ whose value is equal to the number of M-nodes within R_i; each of these counters must be increased by 1 when v is added to the multicast tree. This is accomplished by broadcasting a counter-update message from v to all Z-nodes within the region.

With delete requests, the situation can be more complex. First, if the node being deleted is a leaf node of the multicast tree, it is simply marked as an M-node and a counter-update message is transmitted to all the Z-nodes in the region as in the case of the add operation explained in the previous paragraph. Since paths to deleted leaf nodes can be removed from the multicast tree with little effort, an optional pruning algorithm is used to remove any inactive subtrees in T_i left behind by the deleted node. This step is optional because such subtrees will automatically be removed when the region undergoes a rearrangement.

When the node being deleted, v, has a degree of 2 or more, it may belong to more than one region. Thus, when v is deleted, all regions containing v must be merged into a single region. This is achieved by v sending counter-update messages to all Z-nodes within regions it is currently part of. The counter increment sent to a Z-node in region R_k is one plus the sum of all counters in v excluding that representing region R_k. Thus, when the counters are updated by adding the increment, all counters corresponding to the merged region are now set to the total number of M-nodes within the region.

After processing each add or delete request, the algorithm checks to see if the quality of the tree has degraded beyond a threshold. This checking is accomplished simply by comparing the counter values for the region enclosing the newly modified node against a chosen threshold. If the counter values reach the set threshold, a rearrangement is triggered. Although any static Steiner heuristic can be used to perform the rearrangement, we use the *Kruskal shortest-path heuristic* (KSPH) for a number of reasons. First, it is naturally suited to constructing multicast trees by combining fragments of the tree [1, 11, 16]. Second, the algorithm lends itself to distributed, asynchronous implementation [2]. Finally, K-SPH has been shown to

```
# Given existing tree $T_{i-1}$, find new tree $T_i$ taking into account request $r_i$
case $r_i$ of
    add request for node $v$:
        if $v \notin T_{i-1}$ then
            # Join node $v$ to $T_{i-1}$ by shortest path to closest node $u$
            $T_i \leftarrow T_{i-1} \cup P_{u,v}$
            # Node $v$ joins region $j$
            $R_j \leftarrow R_j \cup v$
        end if
        mark node $v$ as an $A$-node
        # Update edge counters in all $Z$-nodes of $R_j$
        # Let $c_{k,j}$ = node $k$'s edge counter for $R_j$
        for all Z-nodes, $u \in R_j$ do
            $c_{u,j} \leftarrow c_{u,j} + 1$
        if $c_{u,j} \geq threshold$, $u \in R_j$ then
            rearrange $R_j$
    delete request for node $v \in R_j$:
        mark node $v$ as a $D$-node
        if $v$'s degree = 1 then
            for all active nodes $u \in R_j$ do
                $c_{u,j} \leftarrow c_{u,j} + 1$
            perform optional pruning algorithm (Figure 6)
        end if
        else if $v$'s degree > 1 then
            # Need to merge regions containing node $v$ into one
            for all $R_k$ with $v \in R_k$, do
                # Update counters in each region containing $v$
                for all Z-nodes $u \in R_k$ do
                    $c_{u,k} \leftarrow c_{u,k} + 1 + \sum_{\substack{m \\ m \neq k}} c_{v,m}$
            if $c_{u,j} \geq threshold$, $u \in R_j$ then
                rearrange merged region
        end if
end case
```

Figure 4: Pseudocode for heuristic ARIES.

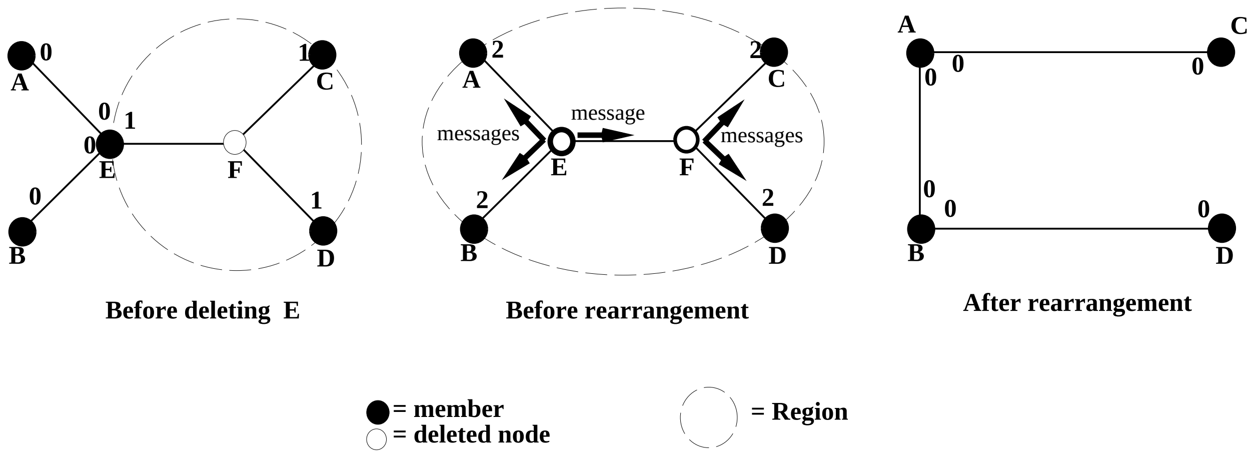

Figure 5: Operation of ARIES on a delete request.

be among the best of Steiner heuristics in terms of the cost of the multicast trees produced in our previous evaluations [1, 2].

Using K-SPH, the rearrangement algorithm proceeds as follows. If R_j is the region in which the rearrangement was triggered, the algorithm removes all the D-nodes and edges in R_j. This fragments left behind are then combined into a new multicast tree by the K-SPH algorithm. Note that each of the A-nodes in R_j forms a fragment by itself. K-SPH combines the fragments pairwise, starting with two that have the shortest distance between them, and repeating the procedure until a single fragment remains. The counters corresponding to the region that triggered the rearrangement are reset to zero once the rearrangement is completed.

Figure 5 illustrates the operation of the algorithm with respect to a delete request. The numbers in the figure denote counter values within nodes. Initially there is a single modified region in the multicast tree containing the deleted node F; the counters in C, D, and E corresponding to this region have a value of 1. When node E is deleted, the region expands to include nodes A and B, and the counters in A, B, C, D corresponding to this region are updated to 2. If the threshold is set as 2, a rearrangement is now triggered. The rearrangement algorithm removes the nodes E and F and all the edges within the region, leaving behind four disconnected nodes A, B, C, D. The nodes are reconnected using K-SPH to obtain the new tree.

The choice of the threshold to trigger a rearrangement affects the behavior of the algorithm significantly. A larger threshold reduces the frequency of rearrangements, but may cause large variations in the cost of the multicast tree in comparison to that of an optimal tree. In addition, if rearrangements are performed infrequently, more work may be needed each time the rearrangement algorithm is invoked. In the next section, we derive an analytical upper bound on the competitiveness of the algorithm as a function of the threshold value selected.

3.3 Pruning Algorithm

After each delete operation, the multicast tree may be further pruned by removing subtrees consisting of only deleted nodes. The optimization is initiated by each D-node that is currently a leaf of the multicast tree. The procedure is performed iteratively until no D-nodes remain as leaves of the tree. Each D-node that is currently a leaf of the multicast tree sends a message to its neighbor in the tree. Any S-nodes in the tree with a degree of 2 receiving such a message simply relay it along the tree. The message terminates when it reaches a Z-node, an M-node, or an S-node with more than two incident edges in the tree. Depending on the type of node where the message terminates, the following actions are performed by the terminating node:

1. If the message reaches a Z-node, say x, the entire path from x to the leaf node that originated the message is removed from the tree. The counter in x corresponding to the path is reset to zero.

2. If the message terminates at an M-node, say y, the path from y to the originating leaf node is removed as in the previous case. In addition, y broadcasts a counter update message to all the nodes in its region. This decrements their counters by one.

3. Finally, if the terminating node is an S-node, say s, the path from s to the originating node is removed as in the above cases. In addition, s is now marked as a D-node. Since the number of modified nodes in the region enclosing the originating leaf node remains unchanged, no counter-update messages are generated in this case.

It is easy to see that, when performed iteratively, the above distributed algorithm removes all inactive subtrees from the multicast tree. Pseudocode for this pruning algorithm is shown in Figure 6.

3.4 A Distributed Version of ARIES

Heuristic ARIES was described in this section as a centralized algorithm for clarity. To be practical, however, the algorithm must have a distributed implementation. The rearrangement heuristic we have chosen, heuristic K-SPH, already has a published distributed implementation [2]. The remaining portion of ARIES may be implemented as shown by the finite state machine in Figure 7. Each tree node would run the finite state machine shown.

When a node joins the multicast tree, it enters the state *wait*. Inactive nodes leave this state only when directed to by an active node to participate in the execution of the distributed algorithm. Active nodes leave the wait state and enter state *update counter* when they receive an update message from a modified node. If the updated edge counter meets or exceeds the threshold value, it then enters the *rearrange* state. While in this state, the node initiates and participates in a distributed Steiner heuristic such as distributed K-SPH [2] to rearrange the nodes and edges in the modified region. At the conclusion of a rearrangement, participating nodes return to state wait.

4 Algorithm Analysis

We now turn to an analysis of heuristic ARIES to evaluate its worst-case behavior. Our primary objective is to derive an upper bound on its competitiveness between rearrangements. Between rearrangements, ARIES behaves the same as GREEDY and shares its bounds. From Imase and Waxman [8], we know that the upper bound considering only add requests is $\log_2(n_i)$, where n_i is the number of nodes in the tree after request r_i. However, no such finite bounds exist for GREEDY with respect to delete requests. In this section, we derive an upper bound for the competitiveness of ARIES considering both adds and deletes. We show that, starting from an optimal solution, the ratio of the cost of the multicast tree produced by ARIES to that of

```
while D-nodes with degree 1 remain in $T_i$ do
    $u \leftarrow$ D-node with degree 1
    $c \leftarrow 0$
    while $u$ an S- or D-node of degree 2 do
        if $u \in D$ then
            $c \leftarrow c - 1$
        delete $u$ and its adjacent edge
        $u \leftarrow$ neighbor($u$)
    end while

    $R_i \leftarrow u$'s region
    case $u$ of
        type $Z$:
            $c_{u,i} \leftarrow 0$
        type $D$ or $A$:
            if $c < 0$ then
                broadcast counter increment of $c$ to all Z-nodes in $R_i$
        type $S$:
            Modify $u$'s type to type $D$
            $c \leftarrow c + 1$
            if $c < 0$ then
                broadcast counter increment of $c$ to all Z-nodes in $R_i$
    end case
end while
```

Figure 6: Pseudocode for the pruning algorithm.

an optimal Steiner tree at any time before a rearrangement is triggered is given by

$$\frac{C(t)}{C_{opt}(t)} \leq c_{th},$$

where c_{th} is the counter threshold that triggers a rearrangement.

We first prove that, in steady state, the counters in Z-nodes corresponding to a modified region have all the same value, the number of modified nodes in the region.

THEOREM 1: Let R_i be a modified region in the multicast tree T containing n_i modified nodes, and let j be a Z-node within R_i. Let $c_{j,i}$ be the value of the counter in node j corresponding to the path connecting it to the rest of the nodes in R_i. When all the messages have converged since the last change to the multicast group, $c_{j,i}$ is equal to n_i, the number of modified nodes in R_i.

PROOF: The proof is by induction on the number of modified nodes in the region, n_i.

Base step, $n_i = 1$: In this case, all multicast neighbors of the modified node x are Z-nodes. Consider the counter in each of these neighbors corresponding to the edge connecting them to x. Each of these counters must be zero before x was modified because there are no other multicast neighbors in modified state that are accessible through this edge. In addition, all the counters in x must also be zero before the change occurred. Since all multicast neighbors of x increment their counters by 1 in response to the change, the theorem is proved for $n_i = 1$.

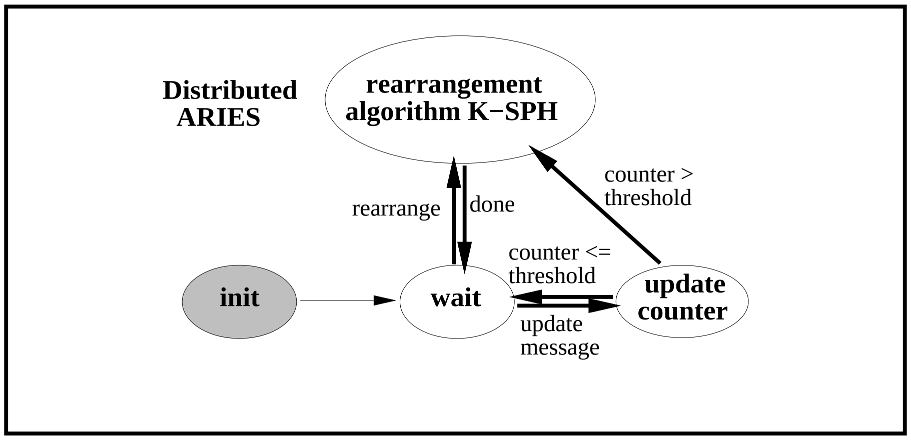

Figure 7: The finite state machine in member nodes corresponding to a distributed implementation of ARIES.

Inductive step: Now assume that a state change increases n_i by one. This may be the result of

1. Adding an A-node that is not currently part of the multicast tree,
2. converting an S-node currently in the multicast tree to an A-node, or
3. deleting a Z-node from the tree.

Let x be the node undergoing the state change. If all of x's multicast neighbors are Z-nodes, the base step applies. Therefore, assume at least one of the multicast neighbors is a Z-node.

Consider case 1 first: adding an A-node that is not currently part of the multicast tree. In this case, x will be connected to the existing multicast tree T along its shortest path to T. Assume this path terminates on T at a node y along an edge e incident to y. Since $n_i > 0$, y must be either an S-node or an M-node, and is already part of a region R_i. Furthermore, node y is not an active node and forwards counter update messages on to the rest of the region. Thus, it follows that a counter increment of $+1$ will be propagated from x to all the active multicast nodes in R_i along the tree T.

Now consider case 2: converting an S-node currently in the multicast tree to an A-node. Since $n_i > 0$, at least one of x's multicast neighbors must belong to a region R_i. In addition, all of x's multicast neighbors that are M-nodes must belong to the same region R_i, since they were multicast neighbors before x was added to the multicast group. Since x was an S-node, all of its counters were zero before the state change occurred. Hence, a counter increment of $+1$ will reach every active multicast node in R_i.

Finally, consider case 3: deleting a Z-node from the tree. Since x is an active multicast node and $n_i > 0$, x is already part of at least one region R_i. In the general case, x may have two types of incident edges that are part of the tree T — edges connecting x to its modified multicast neighbors, and those connecting it to its active multicast neighbors. Each of the former type of edges must belong to a distinct region. Let $e_1, e_2, \ldots, e_k$ be edges of the former type incident to x, and let $e_{k+1}, e_{k+2}, \ldots, e_m$ edges of the latter type. Let $R_1, R_2, \ldots, R_k$ be the regions containing $e_1, e_2, \ldots, e_k$, respectively, and let $n_1, n_2, \ldots, n_k$ be the corresponding number of M-nodes in the regions. On deleting x, the regions $R_1, R_2, \ldots, R_k$ are merged into a single region.

Let $c_{x,j}$, $1 \leq j \leq m$, be the counter value of node x corresponding to its edge in region R_j before the state of x changed. Then, by induction hypothesis,

$$c_{x,j} = n_j, \quad 1 \leq j \leq k,$$

and $c_{x,j} = 0$ for $k+1 \leq j \leq m$. When node x is marked for deletion, every active multicast node in region j receives a counter increment of $\sum_{i=1}^{k} c_{x,i} - c_{x,j} + 1$, and its new counter value becomes

$$n_j + \sum_{i=1}^{k} c_{x,i} - c_{x,j} + 1 = \sum_{i=1}^{k} n_i + 1.$$

Since this is the total number of modified nodes in the new region created by merging $R_1, R_2, \ldots, R_k$ and x, the theorem is proved for case 3.

This concludes the proof of Theorem 1. It is easy to see that the theorem holds even when the optional pruning step of the algorithm is applied, since any change in the number of M-nodes in the region is broadcast to all the counters in the Z-nodes at the periphery of the region.

Each update to the multicast group can cause the cost of the multicast tree grown incrementally by the algorithm to diverge from that of an optimal Steiner tree. The following theorem provides an upper bound on the cost of the multicast tree after a sequence of updates relative to an optimal Steiner tree for the active multicast nodes at that time. We will later derive a different bound based on the sizes of the individual regions within the incremental multicast tree.

Theorem 2: Let $T(0)$ be an optimal Steiner tree for a network at time 0 for a given multicast group $G(0)$, and let $T(t)$ be the corresponding tree at $t > 0$ after a sequence of additions and deletions. Let $C(t)$ be the solution cost of $T(t)$ and $C_{opt}(t)$ be the corresponding cost of an optimal Steiner tree for $G(t)$ at time t. If no rearrangements have occurred in the interval $(0, t)$,

$$C_{opt}(t) \geq C_(t) - m(t) d_{max}(t), \tag{1}$$

where $m(t)$ is the total number of modified nodes in $T(t)$ and $d_{max}(t)$ is the maximum distance between two multicast neighbors in T reached during the interval $(0, t)$.

Proof: We will first prove the theorem ignoring the pruning step of the algorithm and later extend the result to cover the pruning step.

Let $t_1, t_2, \ldots, t_k$ be the instants in the interval $(0, t)$ at which updates to G occurred, with $0 < t_1 < t_2 < \cdots < t_k < t$. Let $C(t_i)$ and $C_{opt}(t_i)$ represent the values of C and C_{opt}, respectively, after the update at the time t_i. We will show inductively that, if Eq. (1) is satisfied at time t_i, then it is also satisfied at time t_{i+1}. Since, in the basic algorithm, a modified node is never removed from the tree until a rearrangement occurs, the number of modified nodes can never decrease with an update to the multicast group. Thus,

$$m(t_{i+1}) \geq m(t_i), \quad \text{for every } 0 \leq i \leq k, \text{ with } t_0 = 0.$$

Eq. (1) is trivially satisfied at time $t_0 = 0$. Consider the update at time $t_i, i > 0$. By induction hypothesis,

$$C_{opt}(t_{i-1}) \geq C(t_{i-1}) - m(t_{i-1}) d_{max}(t_{i-1}), \tag{2}$$

The update at time t_i may be either an add or a delete operation. We will consider these two cases separately.

If the operation is an add, let x be the new member node that was added to $G(t)$. If x was not part of $T(t_{i-1})$, it will now be connected to $T(t_{i-1})$ along its shortest path. Let d_x be the distance of this shortest path. Then, $d_x \leq d_{max}(t_i)$.

$$\begin{aligned} C(t_i) &= C(t_{i-1}) + d_x \\ &\leq C(t_{i-1}) + d_{max}(t_i) \end{aligned} \quad (3)$$

Since adding a node to the multicast group can, in no case, decrease the cost of an optimal Steiner tree,

$$C_{opt}(t_i) \geq C_{opt}(t_{i-1}).$$

Therefore, from Eq. (2),

$$C_{opt}(t_i) \geq C(t_{i-1}) - m(t_{i-1})d_{max}(t_{i-1}).$$

Substituting for $C(t_{i-1})$ from Eq. (3), this becomes

$$C_{opt}(t_i) \geq C(t_i) - d_{max}(t_i) - m(t_{i-1})d_{max}(t_{i-1}).$$

Since $m(t_i) = m(t_{i-1}) + 1$ and $d_{max}(t_i) \geq d_{max}(t_i)$, it follows that

$$C_{opt}(t_i) \geq C(t_i) - m(t_i)d_{max}(t_i). \quad (4)$$

Now if the appended node x was part of the tree at time (t_{i-1}), that is, if $x \in S(t_{i-1}) \cup D(t_{i-1})$, then $C(t_i) = C(t_{i-1})$ and $C_{opt}(t_i) \geq C_{opt}(t_{i-1})$. Thus, Eq. (4) is again satisfied.

Next, consider the case when the update at time t_i is a delete operation. Then $m(t_i) = m(t_{i-1}) + 1$ and $C(t_i) = C(t_{i-1})$. Let x be the node that was deleted from $G(t_{i-1})$. Assume if possible that an optimal Steiner tree at time t_i has cost $C_{opt}(t_i) < C(t_i) - m(t_i)d_{max}(t_i)$. In this case, we can append x to such a tree along a path that has a cost of at most $d_{max}(t_{i-1})$, resulting in a Steiner tree for $G(t_{i-1})$ with cost less than

$$C(t_i) - m(t_i)d_{max}(t_i) + d_{max}(t_{i-1}) = C(t_{i-1}) - m(t_{i-1})d_{max}(t_{i-1}).$$

Since this contradicts the induction hypothesis, we conclude that

$$C_{opt}(t_i) \geq C(t_i) - m(t_i)d_{max}(t_i).$$

This concludes the proof of Theorem 2.

We can now extend Theorem 2 to the case when the optimization procedure to prune inactive subtrees is added to the algorithm. Note that it is easy to extend the proof if the subtree that was removed did not consist of any of the multicast nodes in the original Steiner tree at time 0. For example, if a node x was added to the tree at time t_i and later deleted at time $t_j > t_i$, it is easy to see that at any time $t > t_j$, both $C(t)$ and $C_{opt}(t)$ are not affected by these updates. However, it is non-trivial to show that Eq. (1) holds even when a node initially part of the multicast group $G(0)$ is removed from the tree as part of the optimization step.

The effect of the optimization step is to remove one or more D-nodes in a region of the multicast tree. Since the regions in T are edge-disjoint, it is sufficient to prove the result considering the effect of the pruning procedure in a single region.

LEMMA 1: Let $T(0)$ be an optimal Steiner tree for a network at time 0 for a given multicast group $G(0)$, and let $T(t)$ be the corresponding tree at $t > 0$ after a sequence of additions and deletions that did not trigger the optimization step. Let $T(t+)$ be the tree obtained by pruning the inactive edges in a region R of $T(t)$, with cost $C(t+)$; and $C_{opt}(t+)$ the the corresponding cost of an optimal Steiner tree for $G(t+)$ at time $t+$. If no rearrangements have occurred in the interval $(0, t)$,

$$C_{opt}(t+) \geq C(t+) - m(t+)d_{max}(t+), \tag{5}$$

where $m(t+)$ is the total number of M-nodes in $T(t+)$ and $d_{max}(t+)$ is the maximum distance between two multicast neighbors in T during the interval $(0, t+)$.

PROOF: The proof is by contradiction. Consider the set of M-nodes that were removed from $T(t)$ as a result of the pruning. This set, in general, consists of nodes that were added to the multicast group during the interval $(0, t)$, as well as nodes in the original multicast tree $T(0)$ that remained in G throughout the interval $(0, t)$. Of the nodes in the second category, we can further partition this set into two subsets — the first containing those nodes that caused the conversion of an S-node into an M-node and the second containing those that didn't. Let us denote these subsets by P and Q respectively. Note that each node in Q appears on a subtree in the original tree $T(0)$ that is completely removed during the pruning at time t. Let α denote the cardinality of P and E_Q the total cost of the edges that were removed when nodes in Q were pruned.

We can convert the sequence of updates in the interval $(0, t)$ into an equivalent sequence in which no updates of the pruned nodes occur during the interval $(0, \tau)$ and only the pruned nodes are updated during the interval (τ, t). Thus, the pruning at time $t+$ removes all the nodes that were newly added to M during the interval (τ, t), and converts α of the S-nodes into M-nodes. Thus,

$$m(t+) = m(\tau) + \alpha. \tag{6}$$

All newly-introduced edges in T during the interval (τ, t) are removed during the pruning at time t. In addition, edges with a total cost of E_Q are removed as a result of pruning nodes that were part of $G(t)$. Hence, the cost of the tree left behind after pruning is given by

$$C(t+) = C(\tau) - E_Q. \tag{7}$$

Assume, if possible, that an optimal Steiner tree $\hat{T}$ can be constructed for the nodes in $G(t+)$ with cost

$$\hat{C} < C(t+) - m(t+)d_{max}(t+).$$

Substituting for $C(t+)$ and $m(t+)$ from equations (7) and (6), respectively, this becomes

$$\hat{C} < C(\tau) - E_Q - (m(\tau) + \alpha)d_{max}(t+). \tag{8}$$

Since $C(\tau) \leq C(0) + m(\tau)d_{max}(\tau)$, Eq. (8) can be re-written as

$$\hat{C} < C(0) - E_Q - \alpha d_{max}(t+). \tag{9}$$

We can now construct an optimal Steiner tree for the initial multicast group $G(0)$ as follows: $\hat{T}$ contains all nodes in $G(0)$ except those in $P \cup Q$. Therefore, starting from $\hat{T}$, first add the edges that were removed from $T(t)$ when nodes in the set Q were pruned. When these edges are added, every node in Q would be connected to a Z-node in $\hat{T}$, and the cost of the tree increases by E_Q. We can now connect each

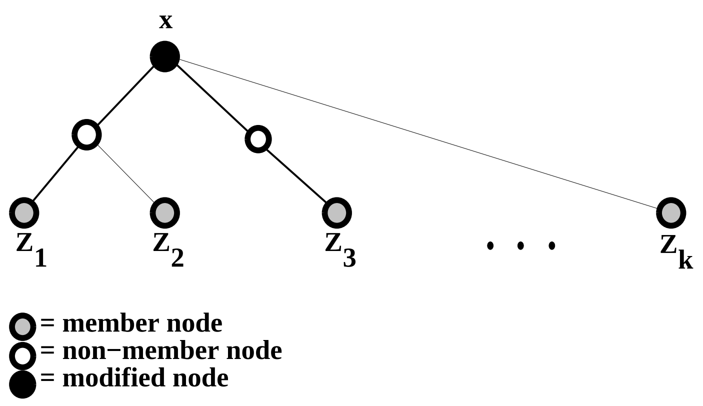

Figure 8: Example in the proof of Lemma 2.

of the nodes in P along its shortest path to the tree. This increases the cost by at most $d_{max}(t)$ for each node in P. Thus, the cost of the Steiner tree obtained is less than

$$\hat{C} + E_Q + \alpha d_{max}(t).$$

By Eq. (9), this is less than $C(0)$. Since $C(0)$ is the cost of an optimal Steiner tree for $G(0)$, this results in a contradiction. This concludes the proof of Lemma 1.

The bound in Theorem 2 did not take into account the locations of the updates. Partitioning the M-nodes into regions allows us to derive our main result, a bound based on the number of M-nodes in individual regions.

THEOREM 3: Let $T(0)$ be an optimal Steiner tree for a network at time 0 for a given multicast group $G(0)$, and let $T(t)$ be the corresponding tree at $t > 0$ after a sequence of additions and deletions. Let $C(t)$ be the solution cost of $T(t)$ and $C_{opt}(t)$ be the corresponding cost of an optimal Steiner tree for $G(t)$ at time t. If no rearrangements have occurred in the interval $(0, t)$,

$$\frac{C(t)}{C_{opt}(t)} \leq c_{th}, \tag{10}$$

where c_{th} is the counter threshold to trigger a rearrangement.

We will prove this theorem considering only the delete operations in the interval $(0, t)$. Since the effect of add operations on the competitiveness is less than that of deletes, the bound also holds for arbitrary update sequences consisting of both add and delete operations. We assume that the optimization step of the algorithm has been applied, with the result that no deleted nodes appear as leaves in the tree $T(t)$.

Before we prove the theorem, we first prove the following lemma.

LEMMA 2: Let R_i be a region in the multicast tree $T(t)$ at time t after a sequence of delete operations in the interval $(0, t)$. Let $Z_i = \{z_1, z_2, \ldots, z_k\}$ be the set of Z-nodes in R_i. Then, any Steiner tree in $T(t)$ for the nodes in Z_i must have a minimum cost of

$$\frac{C_i}{n_i + 1}, \tag{11}$$

where C_i the total cost of edges in R_i and n_i the number of M-nodes in R_i.

PROOF: Since the proof of this lemma for the general case is complex, we start with a simple proof for the case $n_i = 1$. In this case, the region R_i consists of one M-node, say x, and k Z-nodes $z_1, z_2, \ldots, z_k$, as illustrated in Figure 8. Assume that a Steiner tree $\hat{T}$ has been found in the network connecting the nodes $z_1, z_2, \ldots, z_k$ with cost less than $C_i/2$. Since x is not a leaf node in the tree $T(t)$, there is a path from x to one of the Z-nodes in the region, say z_i, with cost $\leq C_i/2$. We can append this path to the Steiner tree $\hat{T}$ to obtain a subtree connecting nodes $x, z_1, z_2, \ldots, z_k$, with total cost $< C_i$. This shows that the initial solution $T(0)$ was not an optimal Steiner tree for $G(0)$, resulting in a contradiction. Note that the result is true even if inactive subtrees were pruned off the node x during the interval $(0, t)$.

Now consider the general case with $n_i \geq 1$. Assume, again, that a Steiner tree $\hat{T}$ has been found connecting the nodes $z_1, z_2, \ldots, z_k$ with cost $\hat{C} < C_i/(n_i + 1)$.

We can partition the edges in region R_i into two subsets as follows: Let $E_{i,1}$ be the set of edges belonging to segments in R_i that connect two M-nodes; and $E_{i,2}$ the set containing the remaining edges in R_i. The edges in $E_{i,1}$ form a subtree of $T(t)$ connecting the set of M-nodes in R_i. The edges in $E_{i,2}$ connect this subtree to the Z-nodes in the region. Let $C_{i,1}$ and $C_{i,2}$ be the total costs of the edges in $E_{i,1}$ and $E_{i,2}$, respectively, with $C_{i,1} + C_{i,2} = C_i$.

It is easy to see that any path from a Z-node in R_i to the subtree consisting of edges in $E_{i,1}$ must have distance no more than $\hat{C}$. If not, we can remove this path from R_i and add the edges in $\hat{T}$ to obtain a connected graph containing all the Z- and M-nodes in R_i with a cost lower than C_i.

Similarly, it can be shown that $C_{i,1} \geq C_i - 2\hat{C}$. Otherwise we can add to $E_{i,1}$, the Steiner tree $\hat{T}$ and a path from the subtree formed by $E_{i,1}$, resulting in Steiner tree for all the Z- and M-nodes in R_i with a cost lower than C_i. Thus,

$$C_{i,1} \geq C_i - 2\hat{C}. \tag{12}$$

Since the edges in $E_{i,1}$ form a subtree of T with at least n_i nodes, one of the paths in this subtree between two closest M-nodes must have distance greater than or equal to

$$\frac{C_{i,1}}{n_i - 1}.$$

Removing the edges on this path disconnects the subtree into two or more fragments. We can re-combine these fragments into a Steiner tree connecting the M- and Z-nodes of R_i by adding to these fragments (i) all edges in $E_{i,2}$ and (ii) edges in $\hat{T}$, and pruning off unnecessary edges. The cost of this tree is less than or equal to

$$C_i - \frac{C_{i,1}}{n_i - 1} + \hat{C}. \tag{13}$$

Since the cost of the Steiner tree so constructed must be no more than C_i, we have

$$C_i - \frac{C_{i,1}}{n_i - 1} + \hat{C} \geq C_i. \tag{14}$$

Using the inequality in Eq.(12) and simplifying, this becomes

$$\hat{C} \geq \frac{C_i}{n_i + 1}. \tag{15}$$

This concludes the proof of Lemma 2. Note that the upper-bound of Eq.(15) can actually be reached. For example, Figure 9 shows a region that is a linear chain consisting of two Z-nodes at the ends and n_i internal M-nodes. If each segment of the chain has a weight of 1 unit. The total cost of the chain is $n_i + 1$. Assume there is a network edge between the two Z-nodes with cost $(1 + \epsilon)$. Then,

$$\hat{C} \geq \frac{1 + \epsilon}{n_i + 1}. \tag{16}$$

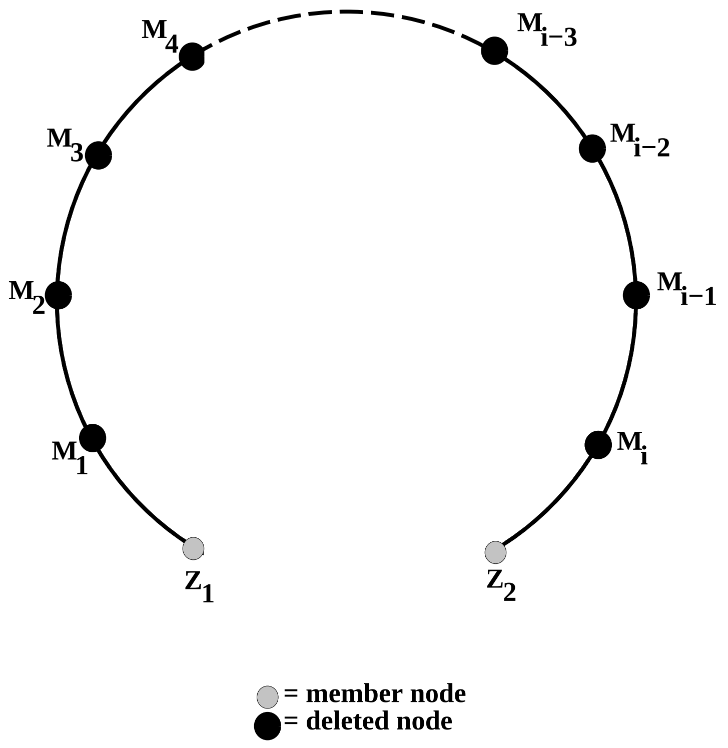

Figure 9: An example where the lower bound of Eq. (15) is actually reached.

We can now prove Theorem 3.

PROOF OF THEOREM 3:

Let $R_i, 1 \leq i \leq k$ be the regions in $T(t)$ and n_i be the number of M-nodes in R_i. Denote by C_i the cost of the edges in R_i, and by Z_i the set of Z-nodes in R_i. Let

$$n_{max} = \max_{1 \leq i \leq k} n_i.$$

Since no rearrangements occurred during the interval $(0, t)$, $n_{max} \leq c_{th} + 1$. Hence, we need to show that

$$\frac{C(t)}{C_{opt}(t)} \leq n_{max} + 1,$$

where $C_{opt}(t)$ is the cost of an optimal multicast tree $T_{opt}(t)$ for the multicast group $G(t)$. Note that the edges in the regions are disjoint from each other. In addition, the edges in $R_1 \cup R_2 \cdots \cup R_k$ form a subtree connecting all the Z-nodes in the k regions, as well as the M-nodes.

Let C_Z be the cost of an optimal Steiner tree $\hat{T}$ for the nodes in $Z_1 \cup Z_2 \cdots \cup Z_k$. Consider any optimal Steiner $\hat{T}$ tree for the multicast group $G(t)$ at time t, with cost $\hat{C}$. Assume that we add to $\hat{T}$ the edges in each of R_i. Since the nodes in $Z_1 \cup Z_2 \cdots \cup Z_k$ are connected in both $\hat{T}$ and the edges in $R_1 \cup R_2 \cdots \cup R_k$, we must have

$$C_{opt}(t) + \sum_{i=1}^{k} C_i - C_Z \geq C(t). \tag{17}$$

That is, when the cost of a minimal Steiner tree is subtracted from $C_{opt}(t) + \sum_{i=1}^{k} C_i$, the result must be greater than or equal to the cost of the tree at time t.

Dividing the terms in Eq. (17) by $C_{opt}(t)$ and rearranging, we get

$$\frac{C(t)}{C_{opt}(t)} \leq \frac{1}{1 - \frac{\sum_{i=1}^{k} C_i - C_Z}{C(t)}}. \tag{18}$$

Consider the term $\frac{\sum_{i=1}^{k} C_i - C_Z}{C(t)}$. Since $\sum_{i=1}^{k} C_i \leq C$, this can be written as

$$\begin{aligned} \frac{\sum_{i=1}^{k} C_i - C_Z}{C(t)} &\leq \frac{\sum_{i=1}^{k} C_i - C_Z}{\sum_{i=1}^{k} C_i - C_Z} \\ &\leq 1 - \frac{C_Z}{\sum_{i=1}^{k} C_i} \end{aligned} \tag{19}$$

By Lemma 3, $C_Z \geq \sum_{i=1}^{k} \frac{C_i}{n_i + 1}$. Therefore, Eq. (19) can be written as

$$\begin{aligned} \frac{\sum_{i=1}^{k} C_i - C_Z}{C(t)} &\leq 1 - \left(\sum_{i=1}^{k} \frac{C_i}{n_i + 1} \right) / \sum_{i=1}^{k} C_i \\ &\leq 1 - \left(\sum_{i=1}^{k} \frac{C_i}{n_{max} + 1} \right) / \sum_{i=1}^{k} C_i \\ &\leq \frac{n_i}{n_i + 1}. \end{aligned} \tag{20}$$

Substituting in Eq. (18), we get

$$\frac{C(t)}{C_{opt}(t)} \leq n_i + 1. \tag{21}$$

This concludes the proof of Theorem 3. Note that the theorem provides us a way to limit the disparity between the incremental solution and the optimal solution by setting a threshold for the counters for triggering a rearrangement. If the rearrangement algorithm leaves behind an optimal Steiner tree, the bound in Eq. (10) holds at any point in time for a given counter threshold. In practice, however, the rearrangements must be done using a heuristic algorithm, and the bound may not hold at all times. Nevertheless, the theorem provides valuable insight into the behavior of the algorithm.

5 Performance Evaluation

Having proved an analytical upper bound on the competitiveness of solutions produced by ARIES, we now turn to its average behavior. Since the analytical upper bound provides little insight into the algorithm's average behavior, we simulated ARIES on a large number of random test networks. We found that in practice the competitiveness of the solutions produced by ARIES were much better than the upper bound. In fact, the majority of the solutions found were within 10% of the best solution found by a static Steiner heuristic. The details of our simulations follow.

5.1 Evaluation Methodology

We simulated our heuristic on 50 randomly generated, sparse, 200-node test networks, each with 60 multicast members. We also simulated heuristics GREEDY and EBA, and include their results for comparison. We chose to use random networks because our choice of suitable existing networks was very limited. Instead, we generated random networks as described below. We consider our graphs to be sparse because each has less than 5% of the possible $\binom{200}{2}$ edges present in a 200-node complete graph. We chose 60 multicast members for two reasons: (i) we believe it likely that a multicast group will consist of a minority of graph nodes and (ii) 60 multicast members in a 200-node graph presents a difficult problem for Steiner heuristics.

Each heuristic received 100 requests to add or delete a multicast member for each test network. The probability of an add request is related to t, the number of nodes in the tree, by the function [14]:

$$\frac{\gamma(200-t)}{\gamma(200-t)+(1-\gamma)t}.$$

The value of γ determines the equilibrium point at which the probability of an add or delete is equally likely. In our simulations γ was set to 0.3, the fraction of multicast members. As a result, each test network received approximately the same number of addition and deletion requests. In our simulations, each request was presented to the network only after the previous request was completely serviced.

The 50 test graphs were generated to resemble real networks in a manner similar to that of Doar [6]. 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 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. Each edge's distance is its rectilinear distance. We used the probability function

$$P(x,y) = \beta e^{\frac{-d_{x,y}}{400\alpha}},$$

where $d_{x,y}$ is the rectilinear distance between nodes x and y [6]. 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 and degree-distribution.

Each heuristic was implemented on top of our 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.

5.2 Simulation Results

In this section we present our simulation results for heuristic ARIES. For comparison, we also include simulation results for heuristics GREEDY and EBA. The metrics we use are competitiveness, CPU time, and the number of edges that are different between successive multicast trees. Competitiveness is defined to be the ratio between the heuristic tree cost and the cost of the static heuristic K-SPH for each test case. Ideally, we should be comparing heuristic results to the optimal tree for each case, but this is impractical.

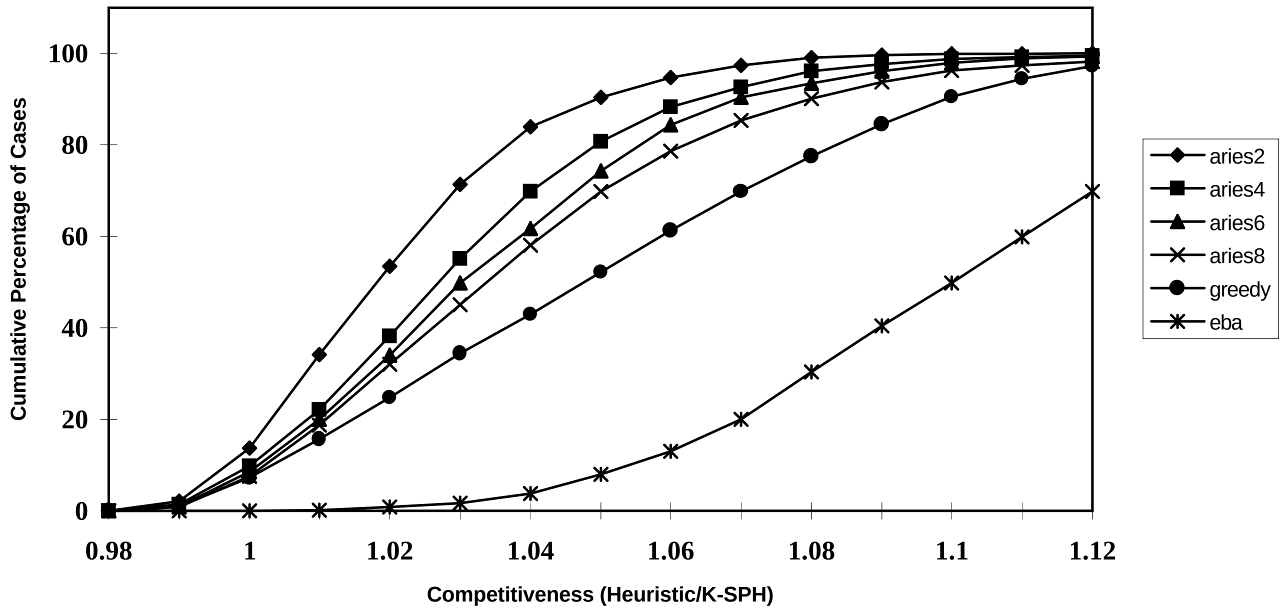

Figure 10: The cumulative distribution of competitiveness for each heuristic.

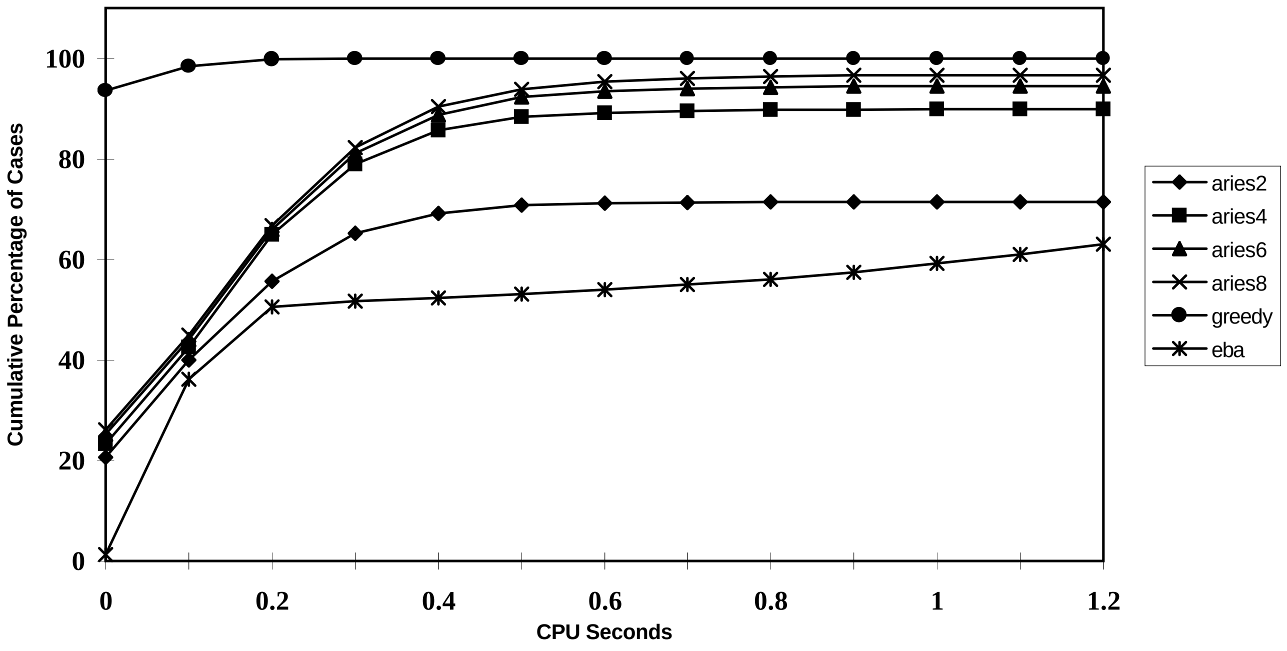

Figure 11: The cumulative distribution of CPU time for each heuristic.

Instead, we use static heuristic K-SPH as our benchmark. Our second criterion, CPU time, is measured by the CPU seconds on an IBM RS/6000 for a serial implementation of the algorithms. Our third criterion, the difference between successive trees, is measured by the number of edges that are different between successive trees.

Figure 10 presents the cumulative distribution for the competitiveness for four variants of heuristic ARIES as well as heuristics GREEDY and EBA. Each variant of ARIES uses a different threshold value: 2, 4, 6, and 8. For example, in ARIES6, each region of the tree may have at most five modified nodes without triggering a rearrangement. As seen in Figure 10, all of ARIES's variants occupy the favorable upper half. As rearrangements ares postponed longer, ARIES's cumulative distribution becomes closer to that of GREEDY. It is noteworthy that EBA has the worst overall performance of any heuristic. This is because the heuristic runs on a complete distance graph. This means that each Steiner tree resembles a merged shortest path tree when translated back to its real-world equivalent and we know merged shortest-path trees to be sub-optimal [13]. This disadvantage is exaggerated because EBA requires internal, non-member nodes to have degree greater than two.

Figure 11 complements Figure 10 by presenting the cumulative distributions for CPU seconds. Heuristic GREEDY has the lightest computation requirements and finishes all cases well within the 1.2 CPU seconds shown. By comparison, within the same CPU time ARIES's least expensive variant, ARIES8, completed 97% of its cases; while the most expensive variant, ARIES2, completed only 71% of its cases. However, heuristic EBA completed just 63% of its cases. Again, heuristic EBA displays the worst behavior. This is for at least two reasons: (i) EBA requires up-to-date distance information for both the graph and the tree, and (ii) EBA frequently rearranges the tree. Although distance information for the original graph may be pre-computed, distance information for the current tree requires additional computational effort for each request. Because EBA rearranges the tree in roughly half its cases, its computational effort per request becomes significant. EBA requires similar computational effort for processing each request. By contrast, most of the total CPU time for ARIES is concentrated in its rearrangements.

Figure 12 completes the comparison of heuristics by displaying the number of edges that are different between successive trees for all fifty test networks. This criterion would be important to applications seeking the smallest possible change between multicast trees such as video conferencing and real-time data broadcasts. Heuristic GREEDY changes successive trees the least, accomplishing each change using no more than four edges. The ARIES variants appear next followed by heuristic EBA. All of the ARIES variants change successive trees by four or less edges for over 90% of the test cases. EBA changes only 50% of its cases using four or less edges. This is so because of the edge changes necessary to make the tree an extension tree after each request.

For our fifty test networks, the two most promising variants of ARIES were ARIES4 and ARIES6. Both cases strike a middle ground between frequent rearrangements and degradation in tree quality. Their competitiveness is roughly equivalent even though ARIES6 rearranges the graph less frequently than ARIES4. For the rest of this paper, we use ARIES6 as our representative ARIES variant. We choose this variant because it represents the best balance among competitiveness, computational effort and edges changed for our simulations. The best threshold value in general will, of course, depend on the problem instance.

The next two figures, Figures 13 and 14, present tree cost and CPU time as a function of each addition or deletion request for one example network of the fifty test networks used in our simulations. In Figure 13 static heuristic K-SPH is the lowest curve and represents the baseline. ARIES closely follows this curve. Heuristic GREEDY deviates moderately from K-SPH's curve while heuristic EBA shows the largest

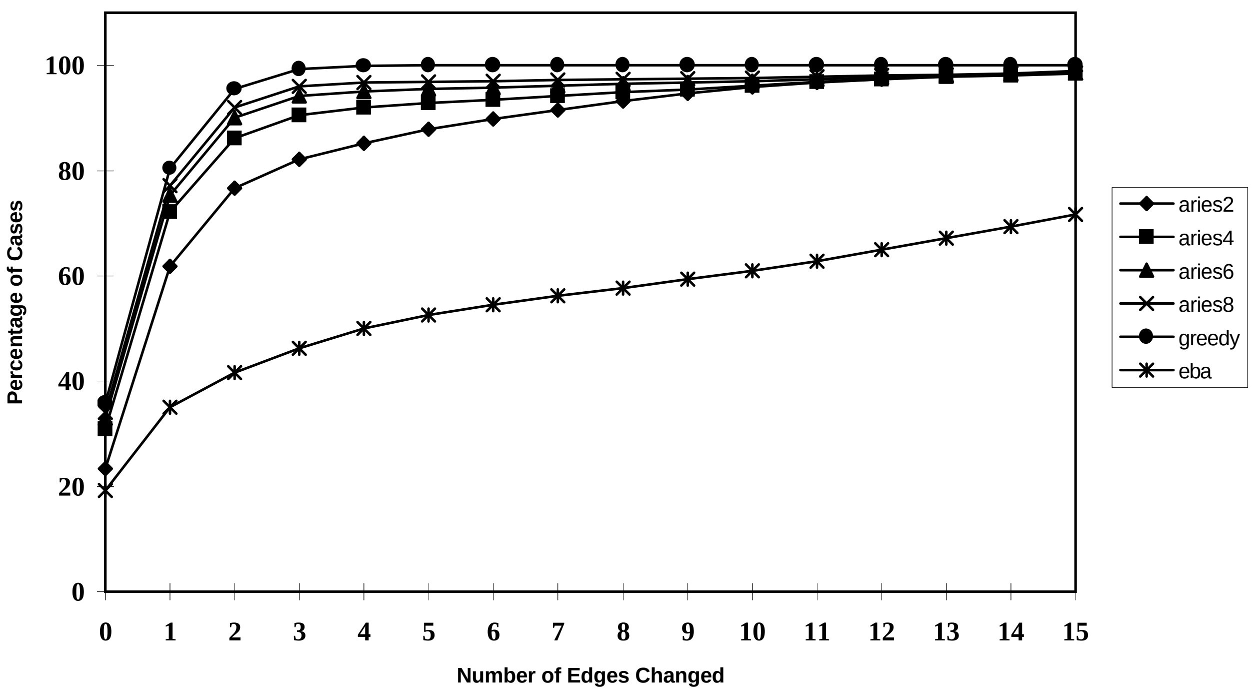

Figure 12: The cumulative distribution of edges changed for each heuristic.

variation.

In Figure 14 static heuristic K-SPH is the upper curve. This upper curve plots the CPU time necessary to recompute each case using the static heuristic K-SPH. As seen in the figure, the computational requirement of ARIES is light except for the occasional rearrangements shown by spikes in the curve. The number and size of these spikes are dependent on the counter threshold chosen. Heuristic EBA, by contrast, performs a more uniform amount of work for most requests. This difference can be more clearly seen in Figure 15. Here, heuristic EBA is contrasted with heuristic ARIES6. While each of ARIES6' 7 spikes exceed 5 CPU seconds, the remaining cases fall well below 0.5 CPU seconds. EBA, in contrast, often uses five times as much CPU time as ARIES6 for the same request. This same difference is also reflected in Table 1.

In Table 1 we summarize the number of rearrangements, average number of edges changes and total CPU time spent for each of the ARIES variants, EBA, and GREEDY. Heuristic EBA rearranges the tree most times, 47, with the highest average number of edges different between successive trees, 9.4. However, this disadvantage is balanced by its modest CPU requirement of 91.6 CPU seconds. As observed above, this CPU time is distributed more evenly over the 100 requests as compared to the ARIES variants. ARIES variants perform between 5 and 31 rearrangements and have a smaller value for the average number of edge changes between successive trees. This is primarily the result of confining rearrangements within modified regions. ARIES variants require the most CPU seconds overall, although most of this time is spent during rearrangements. Even the most expensive variant of ARIES, ARIES2, requires less than 5% of the CPU time of static K-SPH. We find this to be an acceptable tradeoff for increased competitiveness. In addition, the advantage of EBA in terms of its lower CPU time is unlikely to hold in a distributed implementation, since the algorithm cannot be extended to the distributed case in a straightforward manner. Heuristic GREEDY, of course, performs no rearrangements, has the minimum value in terms of the average number of edges

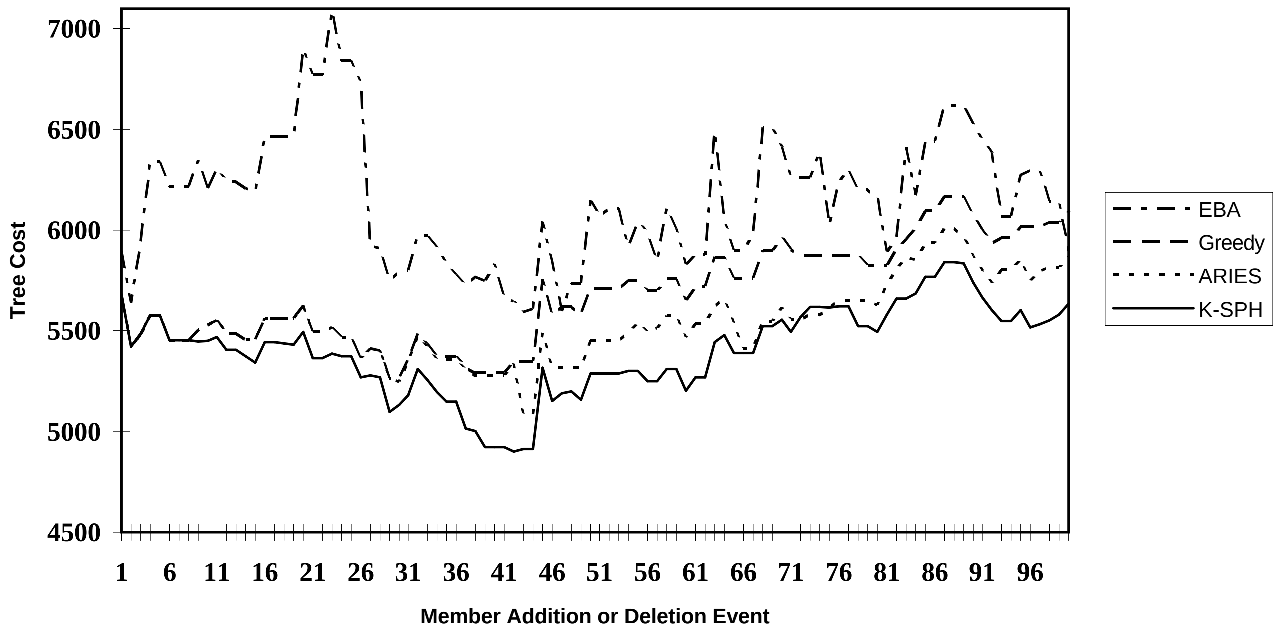

Figure 13: Tree cost for one example graph over all requests.

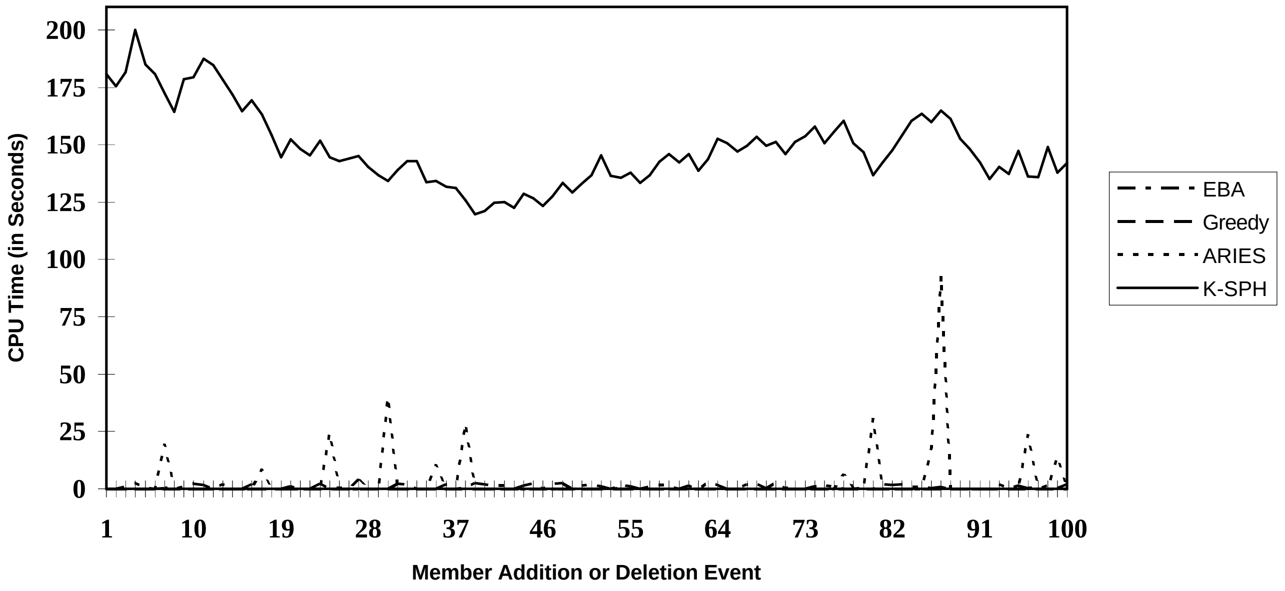

Figure 14: CPU time for one example graph over all requests.

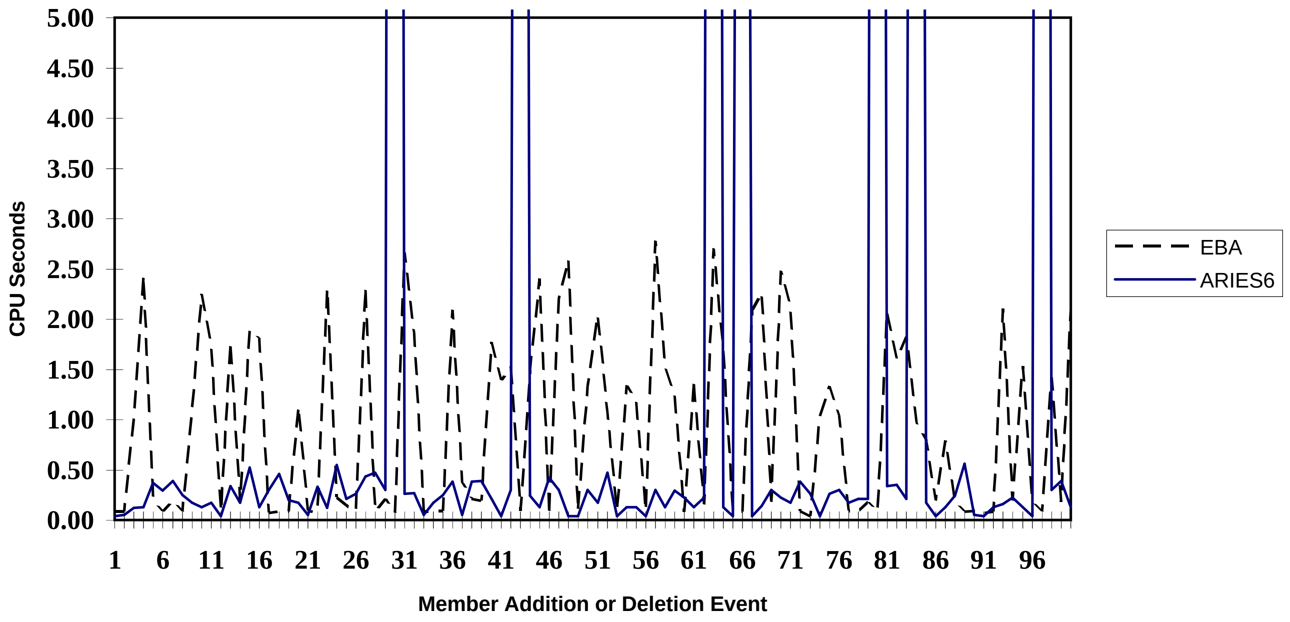

Figure 15: Comparison of CPU time of heuristics EBA and ARIES6 for one example graph over all requests.

changed between successive trees, and requires the least CPU time.

6 Concluding Remarks

In this paper we proposed and evaluated a new, rearrangeable heuristic, ARIES, for the on-line multicast problem. The algorithm is based on monitoring the accumulated damage to the multicast tree within local regions of the tree as nodes are added and deleted, and triggering a rearrangement when the number of changes within a connected subtree crosses a set threshold. We presented analytical upper bounds on the competitiveness of the algorithm and presented simulation results using competitiveness, CPU time, and differences between successive trees as the metrics. The results were compared with those from two previous heuristics, GREEDY and EBA.

Of the heuristics evaluated, heuristic ARIES performed closest in competitiveness to the baseline solutions produced by the static Steiner heuristic K-SPH. It also lends itself to tuning by varying its counter threshold that triggers a rearrangement. The quality of solutions produced by ARIES on our test networks, on average, was considerably better than the worst-case analytical bound. In addition, successive trees were often very close, reducing the disruption caused by large rearrangements. Among the variants of ARIES, ARIES6 emerged as the most promising for our 50 test networks. The best-performing variant for a particular case will, of course, depend on the network and the pattern of updates to the multicast tree.

In addition to our description of heuristic ARIES as a centralized heuristic, we have also outlined how this heuristic could be implemented as a distributed, asynchronous algorithm. This distributed version would likely have the greatest practical use.

At least two open topics remain: First, the upper bound for competitiveness we have shown assumes

Heuristic	Number of Rearrangements	Average Number of Edges Changed	Total CPU Seconds
Static K-SPH			14834.0
EBA	47	9.4	91.6
ARIES2	31	2.3	670.5
ARIES4	11	1.8	312.6
ARIES6	7	1.5	286.8
ARIES8	5	1.3	210.1
GREEDY	0	0.9	4.4

Table 1: The message and convergence bounds for distributed K-SPH.

that the multicast tree is restored to an optimal tree after each rearrangement. A more difficult, but interesting problem would be to bound the competitiveness across more than one rearrangement, taking into account the sub-optimal solutions produced by the rearrangement algorithm.

Second, we have only sketched a distributed implementation of ARIES. Admittedly, a distributed implementation in our ideal assumed environment would be straightforward. In a real environment, however, network nodes may enter and leave the multicast group simultaneously. Multiple simultaneous modifications to the multicast membership could result in race conditions under which the distributed algorithm does not converge. Mechanisms must be devised to coordinate the updates in a consistent manner. In addition, counter updates need to be communicated to tree nodes within a local region reliably. These difficulties, however, are not unique to ARIES, but arise in the design of many distributed algorithms.

ARIES represents a balance between the conflicting demands of multicast tree quality, computational requirements and distributed implementation complexities. Of particular interest are its finite theoretic bounds for multicast member addition *and* deletion. This coupled with its good average case behavior make ARIES a suitable choice for real-world on-line multicast routing applications.

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] F. Bauer and A. Varma. "Distributed algorithms for multicast path setup in data networks," in *Proc. IEEE GLOBECOM*, Singapore, Nov. 1995, to appear.

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

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

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

[9] J. Kadirire. “Comparison of dynamic multicast routing algorithms for wide-area packet switched (Asynchronous Transfer Mode) networks,” in *Proc. IEEE INFOCOM*, Boston, Apr. 1995, pp. 212–219.

[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] 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.

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

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

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

[15] 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.

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