What Features Really Make Distributed Minimum Spanning Tree Algorithms Efficient?

Michalis Faloutsos

University of Toronto
Department of Computer Science
mfalou@cs.toronto.edu

Mart Molle

University of California at Riverside
Department of Computer Science
mart@cs.ucr.edu

Abstract

Since Gallager, Humblet and Spira first introduced the distributed Minimum Spanning Tree problem, many authors have suggested ways to enhance their basic algorithm to improve its performance. Most of these improved algorithms have even been shown to be very efficient in terms of reducing their worst-case communications and/or time complexity measures. In this paper, however, we take a different approach, basing our comparisons on measurements of the actual running times, numbers of messages sent, etc., when various algorithms are run on large numbers of test networks. We also propose the Distributed Information idea, that yields several new techniques for performance improvement. Simulation results show that contrary to the theoretical analysis, some of the techniques in the literature degrade the performance. Moreover, a simple technique that we propose seems to achieve the best time and message complexity among several algorithms tested.

1 Introduction

Given a weighted undirected graph G, with N nodes and E edges, we define a **minimum spanning tree** (denoted by **MST** for the rest of this paper) as a connected subgraph of G for which the combined weight of all the included edges is minimized. We assume that each node acts as a processor and each edge as a bidirectional and error-free communication channel. In the distributed MST problem, we want to define an algorithm that involves only nearest-neighbor message exchanges between adjacent nodes, and eventually labels every edge in G as either a **branch** of the MST or **rejected**. Initially, all nodes are assumed to be "awake" and ready to begin executing the distributed algorithm. However, none of them has any special status nor is aware of any network topology except for its adjacent edges. Thus, we cannot simply send all the information to one node and solve a centralized problem, because this would involve a Leader Election problem which turns out to be comparably difficult [Awe87]. Fortunately, finding distributed MST algorithms is straightforward because the centralized MST problem can be easily solved by greedy algorithms. A distributed algorithm was proposed by Gallager et al [GHS83] requiring $O(E + N\log(N))$ messages and $O(N\log(N))$ time units. We refer to this algorithm as the **basic algorithm**, and we outline it in the next section. The interesting question is: how efficient can we make a distributed MST algorithm?

For the rest of this paper, we will overload the expression "efficient complexity" to mean $O(E + N\log(N))$ messages, and $O(N)$ time units. For the general graph, the distributed MST problem requires at least $\Omega(E + N\log(N))$ messages, where we count the transmission of one message across one edge as our unit of "cost" (Ω denotes the lower bound). So the basic algorithm is message efficient. For the time complexity, it was proven [SB95] that a tighter bound of the basic algorithm is $O((D_{MST} + d)\cdot\log(N))$ units, where D_{MST} is the diameter of the resulting MST and d is the maximum degree of the nodes. Following [GHS83], various authors have proposed enhancements to the basic algorithm to achieve efficient time and message complexity [CT85], [Gaf85], [Awe87], [FM95] and [Fal95]. As it will become apparent later, there is a trade-off between termination time and messages.

The first contribution of this paper is the introduction of further refinements to the basic algorithm, which are based on a novel approach called *Distributed Information*. The motivation for our approach is the observation that all nodes can easily keep track of certain types of information about the network by examining the contents of the exchanged messages. Namely, this information is a summary of the state of parts of the network. This way, the algorithm can make decisions by consulting this information without having to relay messages anew to these parts of the network. The gain in messages and time is apparent. Note that this additional information is simple, so the cost of these techniques is just a trivial increase in the message length, and in the storage requirements at each node.

The second contribution of this paper is an experimental study of the performance of the above algorithms on large numbers of sample graphs. For our simulations, we constructed a faithful representation of each test network at the interface between layers 2 and 3 of the OSI model using a special purpose network simulation package, with the code for the

MST algorithm implemented as an independent process executing at each node. Our experiments cover several families of graphs, each parameterized by different network sizes, and density. As we will see the results favour strongly algorithms with non-efficient worst case complexity, and especially an algorithm based on the Distributed Information approach.

The rest of this paper is organized as follows. In Section 2, we describe the basic algorithm of [GHS83], along with the path graph example that demonstrates its non optimal time complexity. We then list, in section 3, the innovations introduced by previous authors. Section 4 presents some new techniques and algorithms based on Distributed Information, and describes a methodology for creating optimal algorithm using non optimal ones. Finally, section 5 presents our experimental results.

2 The Basic Algorithm

In their pioneering paper [GHS83], Gallager, Humblet and Spira introduced the distributed MST problem and presented an algorithm that has formed the basis of much subsequent work in the area, including [CT85], [Gaf85], [Awe87], [Fal95], and [FM95].

In this basic algorithm, each node is initially the **root** of its own **fragment** (a trivial connected subgraph of the MST) and all the edges are **Unlabeled**. Thereafter, adjacent fragments join to form larger fragments by labeling their intermediate edge as a **Branch** of the MST. The new branch is chosen by the root of one (or possibly both) of the fragments, as the **minimum outgoing edge** (or **MOE**) for the entire fragment. This fragment MOE is determined by broadcasting an *initiate* message to all nodes in the fragment, asking them to send a *report* message with their local MOE to the root (**Finding procedure**). Each node determines its local MOE by testing (*test* message) its Unlabeled edges, minimum weight first, until it finds one that leads to another fragment (**Testing procedure**). Any edges that are found to connect to nodes in the same fragment are labeled as **Rejected**, and are subsequently ignored. Each node gathers the reports of his children and reports the minimum of all reported MOEs (**Reporting procedure**). Finally, the root selects the MOE of the fragment and sends a *changeRoot* message to the node adjacent to that MOE, appointing it as the **leader** of the fragment. The leader sends a *connect* message along that edge and joins with the other fragment. Note that the leader is a "temporary root"; it makes the decisions for its fragment and its role ends when an *initiate* message from a root arrives.

Even the basic algorithm presented in [GHS83] contains several subtleties. First, each fragment has a **level**, L, in addition to its unique fragment identifier F, denoted as a pair (F, L) for the rest of the paper. The fragment levels are used to make fragment joining less symmetric, so that certain types of "one-sided" joins can be permitted without the risk of forming cycles. If two adjacent fragments discover that they share a common MOE and wish to label that edge as a branch, then it is clear that the resulting "two-sided" join can be permitted because the combined fragment will still be a subgraph of the MST. However, since fragments operate asynchronously (and edge testing, MOE selection and joining are neither instantaneous nor atomic), "one-sided" joins create the risk of forming a cycle as one fragment tries to label its MOE as a Branch while the adjacent fragment tries to label another edge as a Branch, and so on.

Rather than reducing parallelism by delaying each join until it becomes "two-sided" (a situation we refer to as an **equi-join**), the algorithm allows small-level fragments to be absorbed by higher-level fragments (a situation we refer to as a **submission**). All fragment levels are initialized to zero, and thereafter at each join the higher level replaces the lower one in a submission while both sides increase their level by one in an equi-join. Thus, **the maximum level that can be reached is** $\log(N)$.

Another noteworthy feature involves reducing the required number of messages by having a low-level node *delay its response* to any *test* message arriving from a high-level fragment, since the high-level fragment can not equi-join with or submit to a low-level fragment (see [GHS83] for more details).

2.1 A Bad Case Example

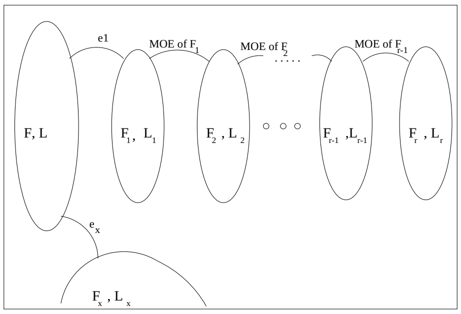

Figure 1: Bad case example.

An intuitive feeling of how the time complexity can "go bad" can be obtained by the following pathological test case that we refer to as the **bad case example**. Consider a chain of fragments $F_1, ..., F_r$ (as shown in Fig.1) with respective levels $L_1 \leq L_2 \leq ... \leq L_r$, and $r >> 1$. Furthermore, assume that these fragments are connected with their MOE edges in such a way that F_1 submits to F_2, F_2 submits to F_3 and so on without the previous (in line) fragment participating in the Finding procedure of the following one. We can assume that F_r is of a greater level and absorbs all the F_i fragments, or that it equi-joins with F_{r-1}. It is evident that it will take a long time for an *initiate* message from F_r to arrive at F_1.

Now suppose that an outgoing edge of a fragment (F, L) connects to the fragment (F_1, L_1), and that $L > L_1$. Because of this level difference, F_1 delays its answer to the *test* message from F. Thus, F waits for a "very long" time (in the worst case until all nodes of the chain of fragments are traversed) before it can complete its Finding procedure. If F joins finally with the chain, then it will get a great level increase and it will be compensated for the delay. However, this is not guaranteed, since F could end up joining with some other fragment F_x and get a very small level increase. In other words, the join of fragments F, F_x can be "greatly" delayed by F_1 and the chain that does not participate in the join.

3 Previous Techniques

Various enhancements have been proposed to improve the performance of the basic algorithm in situations like the previously described bad case example. In the subsequent sections, we will describe the new features introduced in the later algorithms, without repeating features that do not change.

3.1 Size Estimation

The technique that we will call **Root Size**[1] (or **RS**) was introduced in [CT85] [Gaf85]. The authors recognized that, although any fragment of level L in the basic algorithm obviously must have at least 2^L nodes, it may be much larger than that if it has accepted a lot of submissions. By modifying the algorithm to ensure that the fragment level is a better estimate of the true fragment size, they would enable other fragments waiting to submit to do so earlier. Thus, the RS modification demands that the size of fragment (F, L) is bounded: $2^L \leq Size(F) < 2^{L+1}$. The nodes can be counted in the Reporting procedure, by having each father report the number of each descendants. At the root, the level of the fragment is compared with the reported size of the fragment. If $Size(F) \geq 2^{L+1}$ then the fragment level is increased, and the Finding procedure is rerun at the new level (no joining takes place).

The RS technique can be further improved by changing the Joining policy, to make better use of the size estimating procedure. With these two changes, the time complexity is upper bounded by $O(N \log^*(N))$, where $\log^*(x)$ is the number of times that the log function must be applied to x before the result is no larger than unity. Intuitively, we can observe that the Joining policy expects small size fragments to submit to bigger ones. Keeping the level of the fragment closely related to its size, helps in the enforcement of this policy. Notice that the communication complexity remains $O(E + N \log N)$.

3.2 Estimating the Distance

The basic idea behind the RS technique — namely triggering level increases whenever the current fragment size exceeds the next power of 2 — was carried further in [Awe87], where two additional techniques were introduced[2]. These techniques were further refined in [FM95] after some problems were discovered in the original paper.

The **Root Distance** (or **RD**) technique (originally called Root Update in [Awe87]) dominates the RS technique described above; it ensures that the root of a level L fragment will never be blocked for longer than $O(2^{L+1})$, waiting for the Finding and Reporting procedures to complete. When an *initiate* message visits a node whose distance from the root is greater than 2^{L+1} edges, the message "returns" to the root. The root then increases the level of the fragment and performs a Finding procedure at the new level (see [Awe87] [FM95] for more details).

The other technique, which we call **Leader Distance** (or **LD**), works similarly, but is invoked by the leader of a submitted fragment. The leader sends a *testDistance* message towards the root, and if the distance is large enough (2^{L+2}), the leader increases the level of its fragment. The procedure is repeated until the leader receives a message from the root. With this technique, a submitted fragment is able to increase its level in a timely manner, even if it is far away from the new root. The symmetry of the concept of the two procedures is apparent; they explore the same path from different ends. However, several issues have to be taken care of in order to guarantee the efficiency and the correctness of the resulting algorithm (see [FM95] for details).

It is easy to see that the LD technique improves the time performance in our bad case example. Because of the LD procedure, fragment F_1 will reach L "faster", and thus fragment F will have an answer to its *test* message sooner. In particular, the LD procedure guarantees an upper bound on the delay of the Report procedures. Therefore, all nodes reach level L in $O(2^L)$ time, and hence a termination time of $O(N)$. Note that for the time analysis, the number of hops of a path corresponds to the same number of time units when traversed.

Unfortunately, the LD technique can increase the communication complexity to $O(E + N^2)$. The problem appears when a large number of distinct fragments submit and activate LD procedures. For a detailed explanation see [FM95] [Fal95].

3.3 Multiple Phase Operation

The basic algorithm, together with any of the enhancements except LD, is message efficient but not time efficient, whereas LD makes the algorithm time efficient but sacrifices the message complexity. This difficulty was overcome by Awerbuch [Awe87], who suggested an innovative three-phase algorithmic structure that switches from one sub-efficient version of the algorithm to another part way the execution, and thereby achieves efficiency in terms of both message and time complexity. To elaborate, in the first **Counting** phase, the algorithm simply determines N. The actual MST is constructed in the two remaining phases, which represent a tradeoff between the demands of the initial and the final part of the MST problem. The second **small fragment** phase involves large numbers of small fragments, where limiting the number of messages is most critical. The third **large fragment** phase involves small numbers of large fragments, where limiting the execution time is most critical. Each fragment switches from the small fragment MST phase to the large MST phase as soon as its size

[1] Notice that our two word names for the techniques follow a pattern where the first word denotes who is carrying out the procedure and the second one what it is estimating. We think that this convention makes things much clearer than simply preserving the arbitrary names chosen by the original authors.

[2] This algorithm also introduced a more complex multi-phase structure, which will be discussed in the next section.

reaches $\frac{N}{log(N)}$. For more details see [Awe87] [FM95] [Fal95].

4 Using Distributed Information

In this section, we introduce several techniques that use stored information at each node to improve the performance of the algorithms. That is, nodes are expected to exchange, store and use summary of information concerning other parts of the network to provide quicker answers to certain questions. Two types of stored information are considered. The first type of information concerns the previously reported MOEs, which is used to accelerate the Reporting procedure. The second type of information is some measure of the distance from the root, which is used to accelerate the level increases of submitted fragments. We call this approach *Distributed Information*, and the interested reader can find a more detailed analysis of the new techniques in [Fal95].

4.1 Storing Edge-MOE Information

As we saw, the main question for every fragment is to find its MOE. Thus, it makes sense to keep track of the MOE of previous Reporting procedures, and avoid repeating parts of the procedure when the result can be predicted. For this, we require each node to remember the *most recent edge-MOE value* for each of its branches. Namely, for each node, we want the **edge-MOE** for each of its edges to be the weight of the smallest outgoing edge reachable in that direction. For example, this value is equal to the weight of the edge, if the edge is currently Unlabeled.

These edge-MOEs are updated by including the most up-to-date information in every message. In more detail, each time a node *receives* any message from edge e_i, it overwrites the edge-MOE of e_i with the new value. Similarly, each time a node *transmits* any message on edge e_j, it sets the edge-MOE field in the message to the minimum known outgoing edge value over all its edges *except* e_j. It is important to notice that the edge-MOE values may not be completely up-to-date. However, under the Joining policy introduced in [GHS83], the weight of the current MOE reachable via a branch can only increase over time. In other words, *edge-MOEs are non-decreasing in time*, and thus *each edge-MOE is always a lower bound to the current MOE reachable in the same direction.* These two observations are fundamental for the correctness of this approach.

4.2 Fast Report using Pruning

The **Fast Report** technique (or **FR**) uses this edge-MOE information to accelerate the Report procedure. Namely, we see the problem of MOE selection by each fragment as equivalent to searching some kind of game tree, where we do not always need all the answers in order to make the correct decision. For this, FR allows the node to complete its Reporting procedure before all of its children have responded, if it can be guaranteed that the unreported MOEs cannot be better than the current candidate MOE. For example, assume that branch a reports an MOE of weight W, and the edge-MOE of b is greater than W. We know that the new MOE from b will be at least as large as W, and thus, we do not need to wait for a report from branch b. Recall that, in the basic algorithm, each node must receive the *report* messages from all its children before reporting to its parent node.

This idea can be further applied to the Finding procedure, where we can skip all sub-trees whose last report indicated a MOE of infinite weight, since it is clear that the fragment MOE will not be found in that direction. Although this observation sounds trivial, in practice it results in a substantial speed-up by shortening the final (and most time consuming!) executions of the Finding procedure.

The FR technique is compatible with the basic algorithm in [GHS83], and hence can be added to most of the algorithms described in the previous section. However, FR may interfere with the node counting procedure in RS, since FR may complete the Reporting procedure before the counting is completed. Obviously, adding FR to the basic algorithm does not affect its worst case time and message complexity.

4.3 The Fast Joining Improvement

The fundamental idea behind Fast Report can also be applied to the final selection of the fragment MOE by the root, producing an even greater speedup; sometimes a non-root node knows which is the fragment MOE. More precisely, if any node sees that the MOE value for its *report* message is less than the edge-MOE value for the edge towards its father, it is clear that its MOE must be the fragment MOE. The selection of the MOE can be made at that point, and we can avoid an unnecessary forwarding of the information to the root. Thus, under the **Fast Joining** technique (or **FJ**), we give the authority to the first node that can determine the fragment MOE to relieve the old root of its duties and to nominate the leader that will proceed to join.

Note that in the best case, the FJ allows the new leader to appoint itself, whereas in the worst case, the decision is not made until all the information reaches the current root. Thus, the FJ technique can take advantage of a "good" configuration, but in the worst case it does not do better than the basic algorithm.

Although the FJ technique is compatible with the basic algorithm in [GHS83], the addition of FJ results in a qualitatively different type of operation. The nodes have greater autonomy, which introduces some asynchronism in the way the algorithm handles a single fragment, i.e., the strict "scatter/gather" waves of the basic algorithm's Find-Report-Join cycle are replaced by a more aggressive "join when you can" approach. Note that the FJ technique is not compatible with those others that demand centralised decisions, such as RS and RD. However, FJ is compatible with FR, a combination that we call the **autonoMST** algorithm.

4.4 Distributed Distance Estimation

We already saw in the Leader Distance procedure that it is important for submitted fragments to detect their distance from the root. Extending the Distributed Information idea, in the **Leader Size** technique (or **LS**) we make each node store a distance metric instead of having the leader try to figure it out by

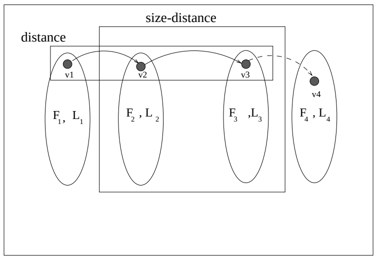

Figure 2: Size-distance and simple distance.

sending a probe message along the path to the root. The distance information is updated by the *initiate* messages, which are issued anyway whenever the identity of the root changes. This way, when a fragment submits, it becomes aware of the distance information towards the new root right away.

Moreover, instead of just measuring the length of the path to the root node, we can maintain an estimate of the total number of nodes that can be found in the direction of the path between a subfragment and the root. What we measure has a meaning between size and distance and we will refer to it as **size-distance**. For example in Fig.2, after F_1 submits to F_2 and F_2 submits to F_3, the size-distance estimate of v_1 must be equal to $Size(F_2) + Size(F_3)$, where $Size()$ indicates the number of nodes of a fragment; however, the simple distance is just $d(v_1, v_3) = 2$.

There are two reasons why we prefer size-distance to simple distance. First, size-distance only increases during the tenure of a given leader, while distance can decrease too. Second, size-distance is an upper bound of the distance, and thus we may have faster level increases.

We can examine the procedure through an example (see Fig.2). When fragment F_1 submits to F_2, the leader of F_1, v_1, will broadcast the size-distance from the root of F_2. If F_2 submits to F_3, v_2 will broadcast the change to all F_2 nodes and v_1. Then, as in Leader Distance, v_1 will decide whether the size-distance is large enough to justify a level increase for its submitted fragment.

It is easy to see that the LS technique is equivalent to the earlier Leader Distance technique that we saw before, and leads to efficient time ($O(N)$) and non efficient communication ($O(E + N^2)$) (see [Fal95] for more details). We find the combination of techniques Fast Report, Root Distance, Root Size and Leader Size in a single algorithm to be quite attractive, and denote it by the name **optiMST**.[3]

4.5 Grouping the Techniques

So far we have seen a number of different techniques that can be incorporated in [GHS83] and give rise to new algorithms. We can now group these techniques according to their performance characteristics.

[3] The name comes from the fact that these techniques are able to exploit "favourable" configurations. Having an optiMSTic attitude, we hope to come across such configurations.

The **Communication Efficient** techniques requiring $O(E + N\log(N))$ messages are:

1. Root Size ([CT85] [Gaf85])
2. Fast Report (Fast Report algorithm)
3. Local Decision to Join (autonoMST)
4. Root Distance ([FM95] [Awe87])

The communication efficient algorithms are: [GHS83], [CT85], [Gaf85], Fast Report and autonoMST.

The **Time Efficient** techniques requiring $O(N)$ time units are:

1. Leader Distance ([Awe87] with the corrections in [FM95])
2. Leader Size (optiMST)

The time efficient algorithms are: the algorithm of the third phase in [FM95], and optiMST.

Clearly, this plethora of choices raises the issue of which techniques should be used in practice. Attention is needed, since the above time efficient techniques need the support of the communication efficient ones. The former alone can not guarantee optimal time and they may also create cycles (or more details see [Fal95] [FM95]). Furthermore, we can create a number of efficient algorithms by combining the above techniques, and the multiple phase structure introduced by [Awe87]. For example, an algorithm that we will see in the next section, called **multiphase optiMST**, uses the two last phases of the multiphase approach; it starts off like the basic algorithm and then each fragment independently switches to optiMST, when it reaches a certain size.

5 Experimental Results

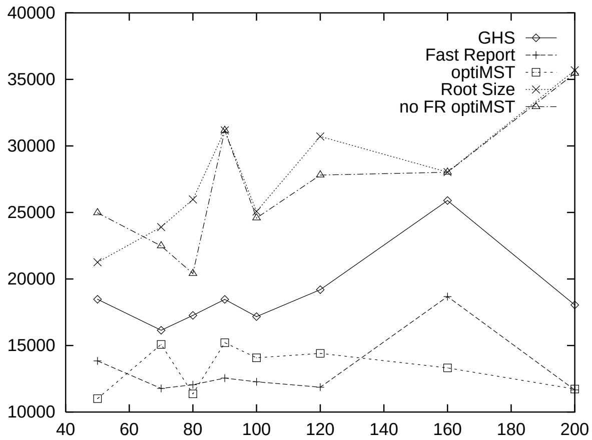

Figure 3: Sparse graphs: Time versus nodes

We found it important to measure the performance of the above algorithms and techniques, especially because we didn't find any previous experimental work. We created a simulator using SMURPH [Gbu95], a

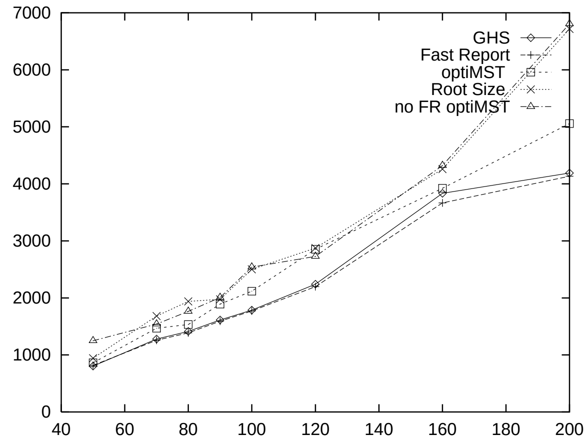

Figure 4: Sparse graphs: Messages versus nodes.

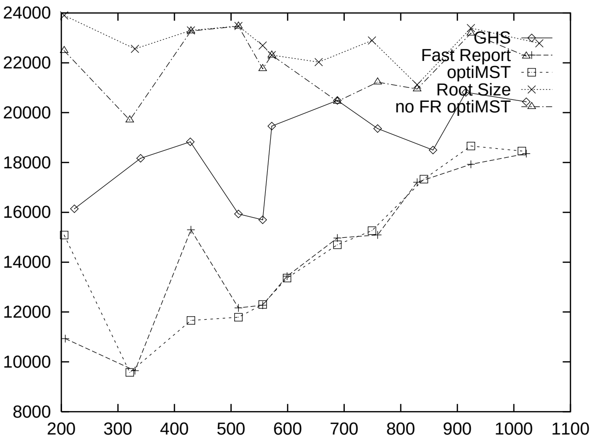

Figure 5: Graphs of 70 nodes: Time vs Edges.

package on top of C++ for simulating low level communication protocols. We found the package very useful though some extensions had to be made. The correctness of the solutions was tested with a non-distributed MST algorithm known as Prim's algorithm [Pri57], and described in [Afr90].

Given the wide variety of different algorithms that can be created, as stated already, we decided to test a representative group rather than exhaustively trying all combinations. We chose to test the following algorithms, each of which is identified by the nickname we gave it:

1. GHS: the basic [GHS83]
2. FR: Fast Report as described in a previous section
3. optiMST: optiMST as described in a previous section
4. Root Size: the basic plus the Root Size technique
5. no FR optiMST: optiMST without the Fast Report

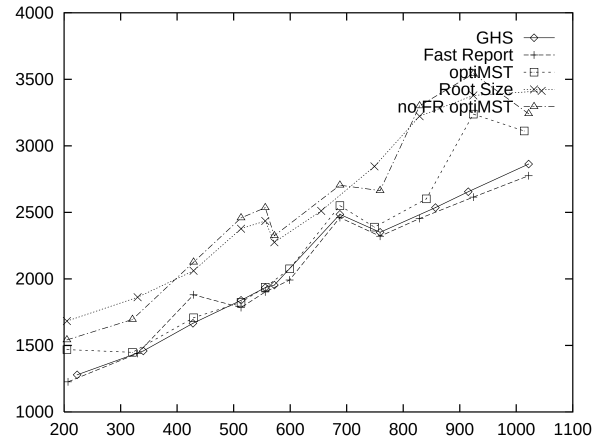

Figure 6: Graphs of 70 nodes: Messages vs Edges.

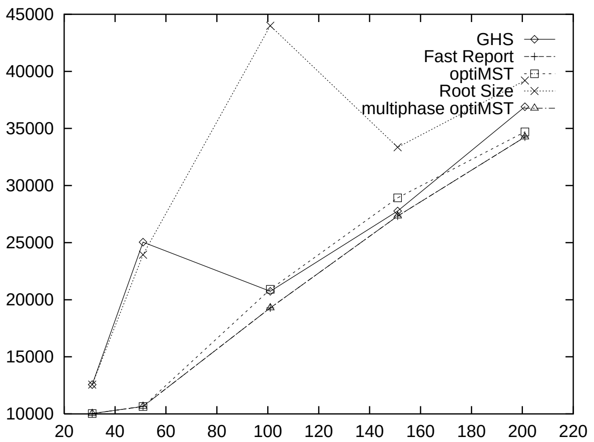

Figure 7: Dense graphs: Time versus nodes

6. Awe–FM: third phase of [Awe87] with the corrections of [FM95], i.e., the basic plus the Root Distance and the Leader Distance techniques technique.
7. multiphase optiMST: a two phase algorithm with I) the basic II) optiMST, as described in the previous section. Note our measurements do not include the time and message complexity of the Counting phase, which would increase the complexity further.

Comparison of Algorithms. Our first goal is to find the effect of the size, and the density of a network to the performance of the algorithms. For this, the following sets of graphs are used in the simulations (Fig.3–Fig.8). First, we use sparse random graphs with 50 up to 200 nodes with an average degree that increased from 3 to 5.5. We interpreted randomness as making all node pairs possible with equal probability. However, we ensure connectivity by creating each node with at least one appropriate edge. The weight of the edges was a uniform distribution (0, 1000), and

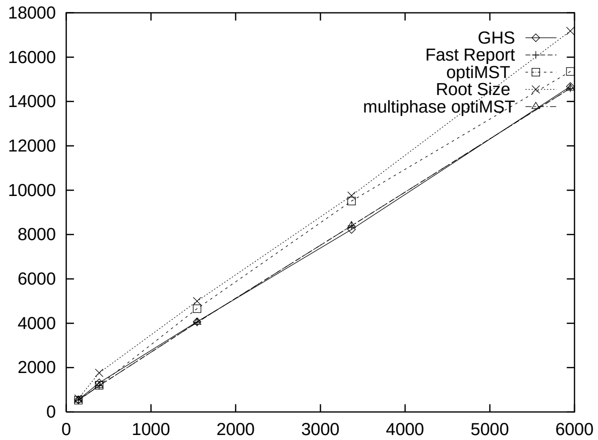

Figure 8: Dense graphs: Messages versus edges.

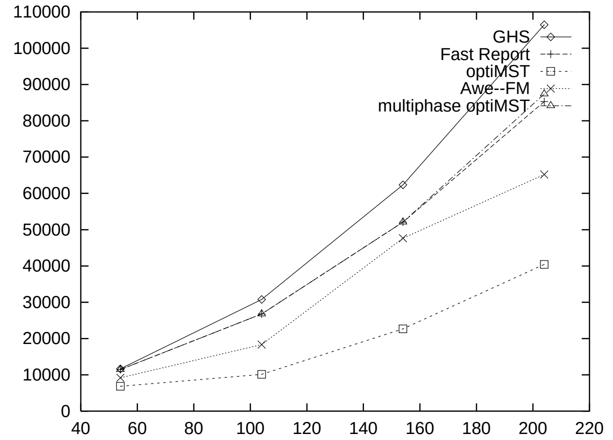

Figure 9: Path graphs: Time versus nodes.

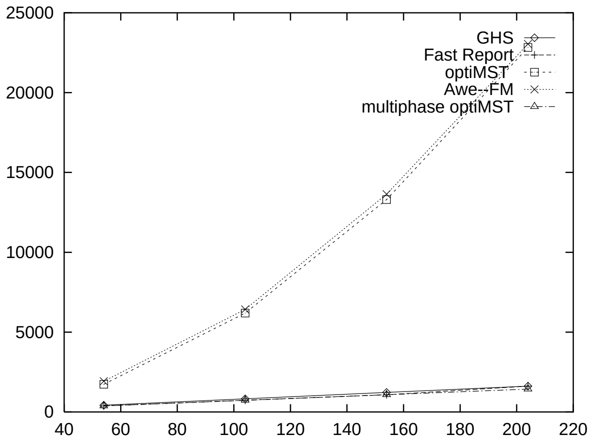

Figure 10: Path graphs: Messages versus nodes.

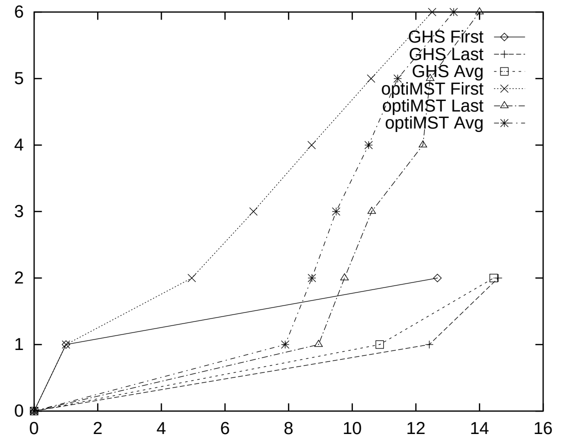

Figure 11: Level vs logarithm of time: Path graph with 104 nodes.

the time delay was equal to the weight. Second, we examine a family of 70-node graphs that was created with increasing average degree from 3 to 15. Third, we test a family of graphs, that we call dense, with a rapidly increasing average degree from 3 to 50. Notice that each measurement, is the median termination time of runs on three similar networks.

Our first major observation is that Fast Report outperforms the rest of the algorithms. In most experiments, it is the fastest algorithm, while its message complexity is also the lowest. The optiMST algorithm also performs well, but the removal of the Fast Report technique ("no FR optiMST") makes it very slow.

Considering the termination time, the algorithms are roughly grouped in fast ones (Fast Report, and optiMST), slow ones (optiMST without Fast Report, and Root Size) and the basic ("GHS") that is in between (see Fig.5 Fig.6). Note the very long termination time of Root Size in the dense graphs: increasing the level seems to degrade the performance. Note, also, that the termination time of the basic, Fast Report and optiMST seems to stay unaffected by the size of the network. To explain this we can recall that, as nodes function in parallel, the larger the size of the network the higher the parallelism. It is worth noticing that as the graphs become denser their time performances converge (Fig.5 and Fig.7) By observing the log file of the simulations, we saw that the algorithms reached their final levels in different times, but it was the final Finding procedure that delayed all of them enough to overwhelm these differences. Namely, the last Finding procedure requires the rejection of a great number of internal edges, and this rejection time was really proportional to the average degree of the network.

Considering the message complexity, the groups are changed to economical (Fast Report, and basic), expensive (Root Size, and optiMST without Fast Report), and optiMST who is in the middle. Note also (Fig.8), the linear relationship between the messages and the edges of the dense graphs, which indicates that the number of edges determines the message complexity, as expected ($E >> N\log(N)$).

Bad Case Comparison. We wanted to create a graph that would force the basic algorithm to

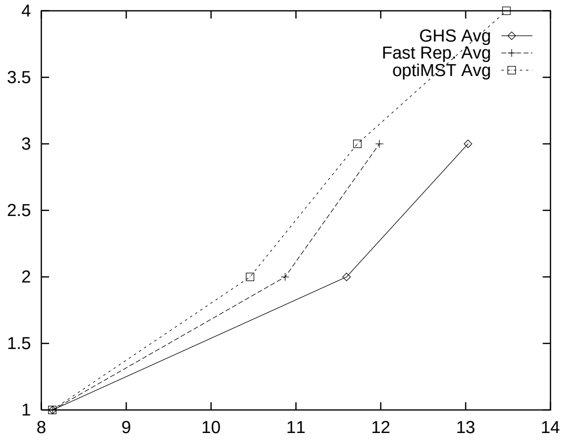

Figure 12: Level vs logarithm of time: Sparse graph with 70 nodes.

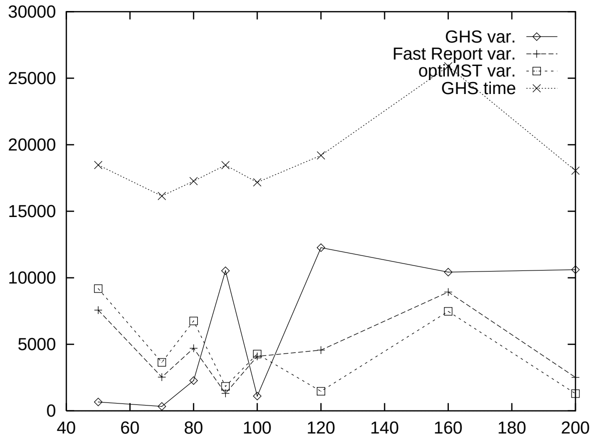

Figure 13: Maximum variation of termination time vs nodes: sparse graphs.

$O(N \log(N))$ time, but the implementation was not straightforward (see [CT85]). In an effort to approach a worst case scenario, we created a set of graphs borrowing ideas from our bad case example. These graphs consist mainly of a long line of nodes with increasing weights of edges between them. All nodes submit to their neighbor, say on the right, and form a huge line of submitted fragments. The only exception is four nodes at the end of the line with the higher weight: these nodes form a fragment of Level 2. The 4-node fragment has to wait for the tail of the chain to increase to at least the 4-node level, before it can submit to the long line. Obviously this example is tailored to demonstrate the positive effect of the Leader Distance, and Leader Size technique of Awe–FM, and optiMST respectively (Fig.9 and Fig.10). Moreover, it illustrates the time-message trade-off, since the communication complexity (Fig.10) of the "fast" algorithms increases proportionally with the square of number of nodes, as predicted theoretically. Note, though, that the time complexity is linear to the size of the graph, even for the "slow" algorithms. In addition, the time complexity of the multiphase optiMST is identical to that of Fast Report, which indicates that the multiphase structure passes on to its second "fast" phase too late to make a difference. Our conclusion from the above observations is that we can not achieve a great speed up with the time efficient techniques, unless we are prepared to sacrifice the communication complexity. Note, finally, that Awe-FM is slower than optiMST, though they have identical message complexity.

Comparison of Techniques. Another major observation is that, in some cases, increasing the level faster can increase the termination time. Theoretically, most of the previous techniques try to increase the level as fast as they can, since the complexity proofs rely on the speed with which the level increases (see [Awe87] [FM95]). In practice, however, the repetition of the Finding procedure at a new level can cause big delays. In support of that we trace the level of the nodes during the execution. Figures 11 and 12 show the first, last and average time (measured in time units) that nodes reach each level. The x-axis in the figures is the logarithm of the measured time, and the y-axis is the level the nodes reach. The best way to read the figures, is to compare for each level, horizontally, the different times that it is reached. Note that optiMST reaches the levels faster than both Fast Report and the basic, but ends up reaching more levels, which eventually reduces its advantage. In figure 11, optiMST keeps some advanatage untill the end, while in figure 12 it looses it completely. In that second case, Fast Report follows closely optiMST, but avoids the last level increase, and terminates much faster. In the same example, the basic terminates after optiMST eventually (see Fig.3 for 70 nodes), due to increased delay in the final Reporting procedure.

The Reliability of the Results. We wanted to see the extend to which we can generalize our results. For this, we examined the variation of the three runs that we did for each measurement. For the sparse graphs, Figure 13 shows the maximum difference among the three measurements, while the median termination time of GHS is provided as a reference point. The average of the maximum differences is $5,000$ time units or 25% of the termination time of GHS. It is worth mentioning that great variations for the same graph usually reflected a different final level, which strengthens our observation that increasing the level in practice is costly in time. Clearly, the discussion of this paragraph can serve only as an indication, while an accurate calculation of intervals of confidence lies beyond the scope of this paper.

6 Conclusions

This paper takes a practical look at the efficiency of distributed MST algorithms. The first contribution is several new algorithms based on the Distributed Information approach, that distributes information and decision making across all the nodes. The second contribution is various sets of experiments, where we measure the termination time and the number messages for several algorithms on various families of graphs.

Our primary conclusion is that in practice simple techniques usually perform better than the techniques used for theoretical reasons. On the one hand, the theoretically suggested techniques usually degrade the performance of the basic algorithm in practice. Such techniques increase the number of executions of algorithmic procedures by a constant, but the procedures can require the traversal of large parts of the graph. On the other hand, techniques that have no influence on the asymptotic complexity improve the performance of the basic algorithm in practice. Furthermore, it was surprisingly difficult to construct a test case that fully demonstrates the gain from the theoretically oriented techniques.

Our second conclusion was that Fast Report, a simple algorithm based on the Distributed Information strategy, outperforms the rest of the algorithms striking a balance between time and communication complexity. In addition, the simplicity of its implementation makes it a very attractive candidate for real applications. For completeness, we mention that Fast Report has the worst case complexity of the basic algorithm, namely, $O(E + N \log N)$ messages, and $O(N \log N)$ or $O((D_{MST} + d) \log N)$ time units, as explained in the introduction. In addition, the Fast Report technique when used in other algorithms improved their performance greatly.

Future work. These experimental results are only the first step of a comparison of algorithms for the distributed MST problem. More experiments should be conducted on different families of graphs, and with various assumptions. In addition, we believe that there is further room for improvement in devising techniques to speed up distributed MST algorithms in practice.

Acknowledgments. We would like to thank Vassos Hadzilacos for his valuable advice and support. Special thanks to Petros Faloutsos, and Joanna Everitt for their comments.

References

[Afr90] F. Afrati. *Graph Algorithms.* National Technical University of Athens Press, 1990.

[Awe87] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems. *Proc. 19th Symp. on Theory of Computing*, pages 230–240, May 1987.

[CT85] F. Chin and H.F. Ting. An almost linear time and $O(VlogV + E)$ messages distributed algorithm for minimum weight spanning trees. *Proceedings of Foundations Of Computer Science (FOCS) Conference Portland, Oregon*, October 1985.

[Fal95] Michalis Faloutsos. Corrections, improvements, simulations and optiMSTic algorithms for the distributed minimum spanning tree problem. Master's thesis, University of Toronto, Computer Science, 1995. Technical Report CSRI-316.

[FM95] Michalis Faloutsos and Mart Molle. Optimal distributed algorithm for minimum spanning trees revisited. *Proceedings of Principles Of Distributed Computing (PODC)*, 1995.

[Gaf85] Eli Gafni. Improvements in the time complexity of two message-optimal election algorithms. *Proceedings of 1985 Principles Of Distributed Computing (PODC)*, Ontario, August, 1985.

[Gbu95] Pawel Gburzynski. *Protocol design for local and metropolitan area networks.* Prentice-Hall, 1995. (author's e-mail: pawel@cs.ualberta.ca).

[GHS83] R.G. Gallager, P.A. Humblet, and P.M. Spira. A distributed algorithm for minimum weight spanning trees. *ACM Trans. on Programming Languages and Systems*, 5(1):66–77, January 1983.

[Pri57] R. C. Prim. Shortest connection networks and some generalizations. *Bell Syst. Tech. J. 36*, 1957.

[SB95] Gurdip Singh and Arthur J. Bernstein. A highly asynchronous minimum spanning tree protocol. *Distributed Computing*, Springer Verlag 8(3), 1995.