Corrections, Improvements, Simulations and optiMSTic Algorithms for the Distributed Minimum Spanning Tree Problem

by

Michalis Faloutsos

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

In this document, we present several new techniques for the problem of finding Minimum Spanning Trees in undirected graphs using distributed algorithms. First, we review related algorithms and the techniques they use to increase the efficiency. Some problems were found and corrections were made for an algorithm due to Awerbuch [Awe 87]. We show that as described in his paper, the algorithm does not work properly. We then present several new techniques that are based on distributing summary information about the fragment among its nodes. We discuss the impact of our techniques on the performance. In addition, we offer an overview and a grouping of all techniques with respect to optimal performance. The discussion concludes in a pattern for creating optimal algorithms. Simulations prove the feasibility of our techniques and demonstrate the trade-off between time and communication complexity. The results show an increase in efficiency that exceeded our expectations.

Contents

List of Figures

Chapter 1

Background and Related Work

1.1 Background

We assume that the reader is familiar with the elementary definitions and properties of graphs, paths, cycles and trees. Given a graph G(N, E) and weights assigned to each edge, we want to find a spanning tree that would be of minimum total weight, denoted MST for the rest of this document. The total weight is defined as the sum of the weights of all the tree edges. We define a **fragment** to be a a connected subset of the MST. An outgoing edge is an edge that connects nodes of different fragments, as opposed to an internal edge. We will use the abbreviation **MOE**[1] for minimum outgoing edge and the abbreviation **local MOE** of a node for the MOE that is adjacent to that node.

We can assume that our graph G(N, E) has uniquely labeled nodes, or edges with distinct weights. It is easy to see that these two assumptions are equivalent and that if neither of them holds, we can not identify uniquely the subtrees. In the case of unique node identities, we can extend the comparison of equal weight edges to comparison of the pair of the node ids - we can assume that multiple edges of equal weight between nodes is not a problem, since it can be detected easily by the nodes. Similarly, in the case of distinct weights, we can name each node with the concatenation of the weights

[1]This abbreviation is going to be heavily used for the rest of this document. We apologise to our reader for the inconvenience this may create.

of its edges. Having this in mind, we can consider both distinct weights and node identities for the rest of this document. A more detailed discussion on such issues, can be found in [GHS 83, p 73].

The problem is stated as follows: nodes know initially the weights of their edges and they exchange messages with their neighbors. Finally the MST is established and all nodes know which of their edges belong in the MST and a unique node (the root) is also aware of its leading role.

We can note the following important property of the MST problem: it can be solved by a greedy algorithm. Thus, an algorithm can start choosing minimum weight edges for the MST, without having to backtrack and relabel. In evidence of this, we can state (for the proofs see [GHS 83]):

Property 1. Given a fragment of an MST with e its MOE, we can join the adjacent fragments along this edge and have another fragment of the MST.

Property 2. If all the edges of a connected graph have different weights, then the MST is unique.

We will assume that the edges are bidirectional error-free channels and that they deliver the messages in sequence. We also assume that all nodes are originally "awake", i.e., the algorithm is initiated in all of them in parallel (see [GHS 83] for more details).

It is known that, for general graphs, this problem requires at least $\Omega(E+N\cdot log(N))$ messages and $\Omega(N)$ time [Awe 87] where $N = |\mathrm{N}|$ and $E = |\mathrm{E}|$. Time is equal to the actual execution time assuming that the delivery of every message across one hop (message delivery between adjacent nodes) requires one time unit. Using the notation from complexity theory, Ω means lower bound.The symbols O and Θ which will be used later will denote the upper bound and tight bound respectively.

1.2 Motivation

Minimum Spanning Trees (MST) are fundamental structures in networks. Finding non-distributed algorithms is also considerably easier than finding distributed MST algorithms and the problem has been studied extensively. However, the distributed problem can arise in practice in several distributed environments.

Spanning trees are essential in distributed computation. They can enforce synchronization on a network and thus enable centralised applications to run on asynchronous networks. They are needed in synchronization protocols and distributed algorithms, such as breadth-first search [AG 85]. We can also think of problems in distributed databases as deadlock resolution [AM 86], or the replacement of a malfunctioning central lock-coordinator [MMP 81]. In all these cases, instead of using an arbitrary spanning tree for the broadcast of messages, we would prefer an MST that minimizes some cost function.

Among distributed algorithms for solving the MST problem, one would prefer the most efficient algorithm. Naturally, as happens in most cases, there is a trade-off that one has to consider: time efficiency (elapsed running time) versus communication efficiency (total number of messages sent).

On the one hand, time efficiency can be of primary interest for some applications. For example in a distributed database problem, it may be more expensive to delay all transactions in order to ensure that the lock-coordinator resumed its role.

On the other hand, communication cost can also be important. First, the transmission of a message has a resource-use cost that may vary in nature and price depending on the case. Second, flooding a network with messages can degrade the Quality of Service for other applications in terms of delay, de-synchronization or message loss. Note that this degradation can cause avalanche effects, meaning that the applications, in their effort to increase their own Quality of Service, can release more messages that will just make things worse for everyone. Third, this type of message congestion can degrade the time performance of the distributed MST algorithm. As we will see in our time analysis we consider queuing delay and processing time to be negligible. However, excessive message circulation can inflate the theoretically expected time performance.

To conclude, we think that it is important to follow a balanced approach considering both running time and message complexity. Having studied both, one can then choose the algorithm that suits more the given conditions for a practical problem.

In addition to that, while implementing the algorithms, we had the chance to fully explore all different cases that can appear. Both this and the experimental results, was a second more detailed check for correctness and completeness of the algorithms. This close study also inspired us with different ideas to improve the efficiency in practice.

1.3 Related Work

Several distributed algorithms have been presented for the MST problem. Among them we can distinguish the following algorithms that are based on common general concepts:

[GHS 83]: Gallager, Humble and Spira did a pioneering, simple and elegant work. Although optimal with respect to communication, it requires $\Theta(N \cdot log(N))$ time. It is a brilliant and very well structured paper that we recommend to readers who want to obtain a basic feeling for the algorithms that followed. It was also a milestone for this kind of algorithm.

[CT 85], [Gaf 85]: Continuing in the same direction as the previous algorithm, work by Chin and Ting and independently by Gafni decreased the time complexity to $\Theta(N \cdot G(N))$, where $G(N)$ is the number of times that the *log* function must be applied to N to get a result less than or equal to 1. Note that because of the similarity of the work we will reference only the algorithm of [CT 85] in the rest of this document.

[Awe 87]: Awerbuch proposed what in the bibliography must be the first optimal algorithm in both aspects. It is ingenious, but a bit more complicated. Two phases are required: a Counting phase in which the number of nodes is calculated followed by the actual MST finding phase. The number of nodes is used to achieve optimality.

We can also mention that in our literature research, we saw other algorithms, whose time complexity was defined by the diameter of the network [AZ 89] [GKP 93]. In the second one, it is argued that the diameter of the network D is a more indicative parameter. The first one, is based on echo algorithms and achieves a time complexity of $O(D \cdot log(N))$ which in the worst case could be $O(N \cdot log(N))$. However, in the second, the time complexity achieved is $O(D + N^{0.614})$. Though more advanced techniques are applied, such as graph decomposition and edge elimination, still the first of the three phases of the algorithm is based on [GHS 83]. Note that no analysis of communication was offered in the second paper and the authors explicitly said that they were not interested in it.

The main aim of our work is to show that using what we call the Distributed Information idea can be beneficial in the process of finding an MST. The idea is as simple as storing some types of information that in previously algorithms was used and discarded instantly. The Distributed Information approach led to the creation of a variety of techniques and algorithms that take advantage of the stored information. It is worth noticing that some of them can be incorporated most previous algorithms, thereby increasing their efficiency. An interesting way to see our approach is as an addition of memory and intelligence to the nodes.

It is worth noticing that the storage demands of our approach stay within the same order of magnitude of the previous storage requirements. In more detail, our most demanding request is that every node has as many counters as its number of incident edges (i.e. the degree of the node). This is realistic to expect, if we think that nodes must have anyway enough storage space to buffer incoming messages which, at some point of time, may be arriving simultaneously from all its edges.

In our study of the previous algorithms, we discovered some flaws in [Awe 87] concerning both its correctness and complexity analysis. Solutions are suggested for both of the problems.

In our effort to compare the different techniques, we grouped them and their corresponding algorithms according to their complexity characteristics. Using the multi phase framework of [Awe 87] and different combinations of algorithms we can create several optimal algorithms. Combining algorithms of our approach we suggest an optimal one that we call optiMST+.

The second goal of our work was to compare the algorithms from a practical point of view. The absence of experimental results in the bibliography was disappointing. We created a simulator and ran several experiments trying different families of graphs. The results confirmed the theoretical expectations of our approach in a flattering way.

The rest of this document is organised as follows. In Chapter 2, we summarize the previous work that we identified above. In Chapter 3 and 4, the issues of correctness and complexity are described respectively. We also give some corrections on one of the algorithms ([Awe 87]). In Chapter 5, we present the idea of Distributed Information and how this can give rise to efficient techniques and algorithms. In Chapter 6, we show the experimental results that were obtained.

Chapter 2

Overview of Previous Work

2.1 Basic Concepts

The pioneering algorithm of [GHS 83] forms the backbone for the other previous algorithms [CT 85] [Awe 87]. Thus they are all implementations of a generic algorithm that has the following concepts:

1. *Fragments.* As mentioned before, the algorithms operate on fragments, each of which is a connected subgraph of the MST. Each node is initially a trivial fragment, and larger fragments are formed by joining adjacent fragments along their common edge.

2. *Edge Labels.* There are three possible labels for edges. Initially, all edges are labeled **Unvisited**. Upon termination of the algorithm, all edges will be relabeled to either **Branch** (if they are included in the MST) or **Rejected** (otherwise). Note that according to the Greedy Joining policy (see below), each edge is relabeled only once, at which point it is either added to some fragment or rejected.

3. *Level concept.* In addition to a unique identifier (F_{id}), each fragment is characterized by a level L. All nodes have zero level in the beginning. When comparing one fragment to another, we will use **smaller**, **equal** and **greater** to indicate the relative value of their level. The level increases when equal fragments join. We will refer to this kind of joining as **equi-join**, to distinguish

it from **submission** that refers to the absorption of a small fragment into a greater one.

4. *Greedy Joining Policy.* Each fragment tries to find its MOE and joins along it with another fragment. That edge is labeled as Branch of the new fragment and thus an edge of the MST. Smaller fragments can submit anytime to greater ones, but greater fragments must wait on their MOE until the other fragment submits or becomes of equal or greater level. Notice that no backtracking (i.e. edge relabeling) is required.

Within each fragment, these algorithms follow a centralised, and thus, synchronous procedure. The root of each fragment instructs all nodes to search for their local MOE, gathers all the data, and decides which join should be made. In particular, the selection of the next MOE, is always done by the root, i.e., a place known from the beginning of the search. As we can see, the information flows from root to leaves and back through the network in a wave-like fashion. First we have the out wave towards the leaves that initiates a new search for the MOE and then the in wave that gathers all the responses and forwards the best choice towards the root.

2.2 The pioneering work - [GHS 83]

One of the major innovations of this paper, which is regarded as classic not only for the MST problem but for distributed algorithms in general, was the concept of the **level**. Levels characterize fragments and enforce a hierarchy, that breaks the symmetry problem in the behavior of fragments during the joining procedure. The underlying idea is that levels are an estimate of the size of the fragment. Since each level increase requires an equi-join, the maximum possible level is $log(N)$, which will help us estimate the complexity of the algorithm.

Within each fragment, one node is the root[1] of the fragment. The way the root

[1][GHS 83] uses the idea of the core, which is an edge and the adjacent nodes that act like a root, but the functionality is the same, and thus we avoid confusing the reader. The other algorithms use the idea of the root as we describe it here.

is determined will be explained later in this section. We use distance from the root to define a hierarchy (typical "father-child" hierarchy for trees). The nodes know if a message, sent along the Branches, travels from or towards the root. Naturally, the father of a node is the neighbor node towards the root. We will see that, as the root changes the father relations may change.

Nodes can be roots, leaders or simple nodes.

- root. The root is the coordinator and decision maker of its fragment. Its responsibilities are to oversee a Finding procedure and, after the completion of the Reporting procedure, to either nominate a new node (named the leader, see below) to carry out the next join, or else end the algorithm.

- leader. It is the node that attempts to join its fragment with another adjacent fragment. Its responsibilities, after having completed those of a simple node, are to follow the Joining policy and join correctly with the other fragment (this will become clear later). When the leader receives an *initiate* message, it becomes a simple node.

- simple node. None of the above. Its responsibilities are to participate in both the Finding procedure (find its local MOE which as we said is a MOE adjacent to this node) and the Report procedure, i.e., report the best MOE among all those reports from nodes below it in the "father-child" hierarchy.

It is better to examine the algorithm in all its various cases, through the explanation of the messages exchanged between nodes. We can distinguish two procedures.

Finding Procedure.

This is the procedure by which the fragment looks for its next MOE. When a node becomes a root it starts this procedure by sending a copy of the following message over each of its Branch edges and broadcasting the following message(s).

- *initiate message:* The root of the fragment has mandated a search for the MOE of its fragment. The message must be forwarded along each outbound branch in the "father-child" hierarchy, thereby reaching all nodes of the fragment. It also carries the information of the new fragment identity and level.

Once a node a of fragment F_a has received such an *initiate* message, it must choose from its edges, the minimum outgoing one, which we will refer to as its **local MOE**. It picks its minimum weight Unvisited edge and carries out the following dialog to determine where it leads to. Assume that it connects to node b of F_b.

- *test message:* node a asks "Is this an outgoing edge, to a greater or equal level fragment?".

- *reject message:* node b can reply "No, we are in the same fragment and I have already rejected this edge myself". It is not difficult to see that we can reject an edge using only two messages - even two *test* messages can be enough.

  Node a will repeat the same procedure with its next minimum Unvisited edge, until an *accept* message is received (see below) or it runs out of Unvisited edges. In this case, infinity is considered to be the weight of the local MOE.

- *accept message:* node b can reply "Yes, it is outgoing, and my level is greater or equal to your level". This edge is the local MOE of the node, and the search stops. However, if $L_b < L_a$, node b puts the test message aside and carries on with the procedures initiated within its own fragment. (This is case 1 where a non-trivial delay can occur in responding to a message in this algorithm.) Eventually, node b will receive an *initiate* message with $L_b \geq L_a$, in which case it can send a delayed *accept* message back to node a. The only exception is if node b receives a *changeRoot* message in order to join along the edge (a, b) while $L_b < L_a$. In this case, node b first submits to node a and then, after the joining procedure is complete, sends a delayed *reject* message back to node a. In addition, since F_b joined F_a before node a reported its local MOE, node a will extend the Finding procedure into F_b.

- *report message:* it is the answer to the *initiate* message. Every node reports the weight of its MOE to its parent. Every parent compares the incoming *report* messages and its local MOE and reports only the minimum of them all.

Joining Procedure.

The root accumulates the report messages, decides which one is the MOE for the whole fragment, and then informs the adjacent node that it will be the new leader

- *changeRoot message:* it is sent along the path from the root to the node adjacent to the fragment MOE inviting that node to become the **leader** of the fragment. On the way the message, reverses the "father" relation of adjacent nodes and so when it finally reaches the leader, the Branches of the fragment are rooted towards the leader.

Let us assume that the old root chose a as leader. Having received a *changeRoot* message node a knows that it can proceed in joining along the edge (a, b), which is both the local MOE of a and the MOE of the whole fragment.

- *connect message:* node a says "My fragment and I would like to join with you". The joining could be either equi-join or submission. Node a, waits for a *connect* message (case 2 of non trivial delay) that will establish the joining in terms of equi-join ($L_a = L_b$) or submission ($L_a < L_b$). Note that a *connect* message will always lead to a join, since the only way for rejection of the edge would be if F_b submits to F_a, but this is impossible because the *accept* message, previously sent by b, guarantees that $L_b \geq L_a$. Although L_b may have increased, it could never decrease and thus the Joining Policy is not violated. This way cycles are avoided (see the correctness section).

We can now discuss the way the root is determined. In the beginning, every node is a trivial fragment and thus a root. In an equi-join, the root can be one of the two nodes adjacent to the MOE by say smaller node id number. In a submission, the root of F_b remains and node a will have b as its father.

The algorithm terminates when the final root can not find an outgoing edge, i.e., the reported MOE is equal to infinity.

2.3 The node counting improvement - [CT 85]

The major innovation of the algorithm is that it tries to keep the fragment level a better estimate of the fragment size. It is obvious that any fragment of level L must have at least 2^L nodes. However, this is just a lower bound of its size: the fragment may have many more nodes than 2^L if it has accepted a lot of submissions. The modified algorithm demands that

$$2^L \leq Size(F) < 2^{L+1}$$

Tracking the fragment size can be achieved by having the root count the *report* messages it receives. More accurately, each *report* message has a counter that is increased at each hop of the message. Each node adds the counters of all the messages that it receives. At the root, the level of the fragment is compared with the size. If $Size(F) \geq 2^{L+1}$ then the level is increased until it satisfies the previous double inequality. Then an *initiate* message is broadcasted and the procedure of finding the MOE is repeated. We will call this procedure **Root Level Increase**.

A complementary idea suggested in this algorithm concerns a change in the Joining policy in order to exploit better the counting procedure. When a fragment (F_1, L_1) with v_1 as new root, submits to (F_2, L_2) at node v_2 with $L_1 < L_2$, F_1 will either be included in the size counting procedure of F_2 or it will begin a size counting procedure for itself. Node v_1 will calculate the new possible size according to the previous formula. If the new level of F_1 is greater than L_2 then the submission is cancelled and a new Finding procedure is initiated in F_1.

This procedure increases the efficiency of the algorithm to $\Theta(N \cdot G(N))$ where $G()$ is the function explained in the introduction. Intuitively, we can observe that the mentality of the Joining Policy expects small size fragments to submit to bigger ones. By keeping in fragments, the level closely related to the size, we make the enforcement of this policy easier [CT 85, p 258].

A pathological example provided in [CT 85] proves that the bound is tight (see complexity section for an intuitive explanation).

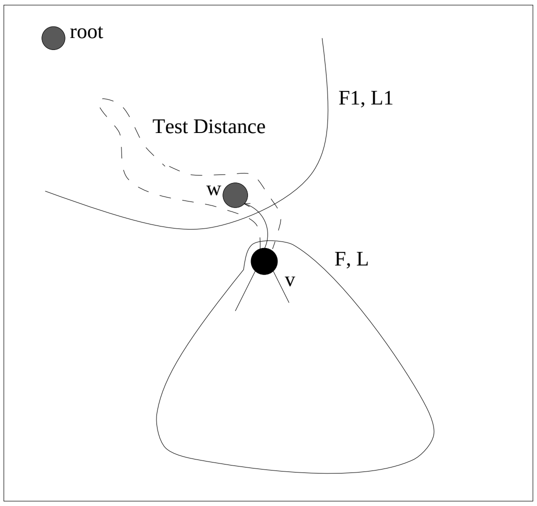

Figure 2.1: Test Distance Procedure.

2.4 The optimal - [Awe 87]

This is the first algorithm that achieved the optimal bounds for both communication and time. The price paid is the loss of simplicity; two phases are required and the second one is further subdivided in two parts.

Two new procedures are introduced : the **Root Update** procedure and the **Test Distance** procedure.

The Root Update resembles the Root Level Increase procedure of the previous algorithm. The difference is that instead of counting the number of report messages, the existence of "long" paths is detected. The *initiate* message has a counter and counts the number of nodes it visits. The counter is initialised to 2^{L+1} and is decreased at each hop. When the counter becomes negative (we will say that the message *expired*), a message is sent back to the root. The level is increased by one and a new *initiate* message is issued, restarting the Find MOE procedure.

The Test Distance procedure applies to fragments that have just submitted. The fragment tests to see if its distance from the new root is big enough to justify a level increase. Thus instead of staying idle, it manages to have its level increased in time related to the level.

Assume a fragment F, L with root v that has submitted to F_1, L_1 at w (see Fig.2.1), and that v has not received an *initiate* message from the "new" root, i.e., F has not yet been "officially" recognised as part of the great tree. Node v sends a $testDistance$ message towards the new root. As in the $initiate$ message, the message has a counter initialised to 2^L. Decreasing at each hop, it may become negative, in which case we say that the procedure *succeeded* and the message returns to v that it can now increase its level by one. A $testDistance$ message will be sent again, with the increased level, until the new root is reached or an $initiate$ message arrives from the new root.

Note that the role of this procedure is to update the level of the submitted fragment as soon as possible, so that adjacent fragments, that are connected with their MOE to it, will get to submit sooner (see complexity section).

The algorithm is divided into phases and parts as follows:

1. *Counting Nodes.* In this auxiliary phase, we want to count the nodes of the network and thus we try to find a Spanning Tree that will help us do that correctly. Weights are neglected and the joining policy changes as follows: each fragment joins along the edge that leads to the greater fragment. Therefore, we achieve fast level increase and the communication and time complexity of this phase are $O(E + N \cdot log(N))$ and $O(N)$ respectively (see [Awe 87] for details).

   Having a Spanning Tree, the number of nodes in the network can be counted.

2. *MST phase.* This is the main part of the algorithm, where an MST is found. This phase is divided into two parts.

   (a) *Fragments' Size: 0 to* $\frac{N}{log(N)}$. In this part, the algorithm behaves exactly the same as the first algorithm we examined. The complexity of this part is optimal (see [Awe 87] for details), because the phase ends when the sizes of the fragments become $\frac{N}{log(N)}$. Intuitively, we can observe that the first level increases are very fast compared to the later ones.

   (b) *Fragments' Size:* $\frac{N}{log(N)}$ *to* N. After the size of F becomes $Size(F) \geq \frac{N}{log(N)}$, the two new procedures are brought into action. Having fragments of this size makes sure that we have fewer than $log(N)$ fragments in this phase. We are going to use this in the complexity analysis.

Some comments and corrections are offered in the section about the correctness and the complexity of the algorithm. Our comments concern correctness, the efficiency of Test Distance and important details in the Joining policy. In our opinion, the algorithm as presented does not work correctly.

Chapter 3

Correctness of Previous Algorithms

3.1 General Results

In order to prove that the algorithms are correct, we have to prove that they terminate and find the MST.

For termination, the following theorem holds for the three algorithms and proves that they terminate.

Theorem 3.1 *The algorithms [GHS 83], [CT 85] and [Awe 87] are deadlock free.*

A slightly different proof than the one in [GHS 83] will be given when we talk about our approach.

To prove that the algorithms find the minimum of the spanning trees we can recall two properties that we mentioned in the introduction. In other words it is sufficient to verify that the algorithms find at each step the MOE of each fragment. The previous description must have left no doubt that the edge selected by the root as MOE is indeed of minimum weight and outgoing for all the nodes that received the *initate* message and participated in the Finding procedure.

In addition, we must prove that it actually finds a tree, i.e., that it does not create a cycle.

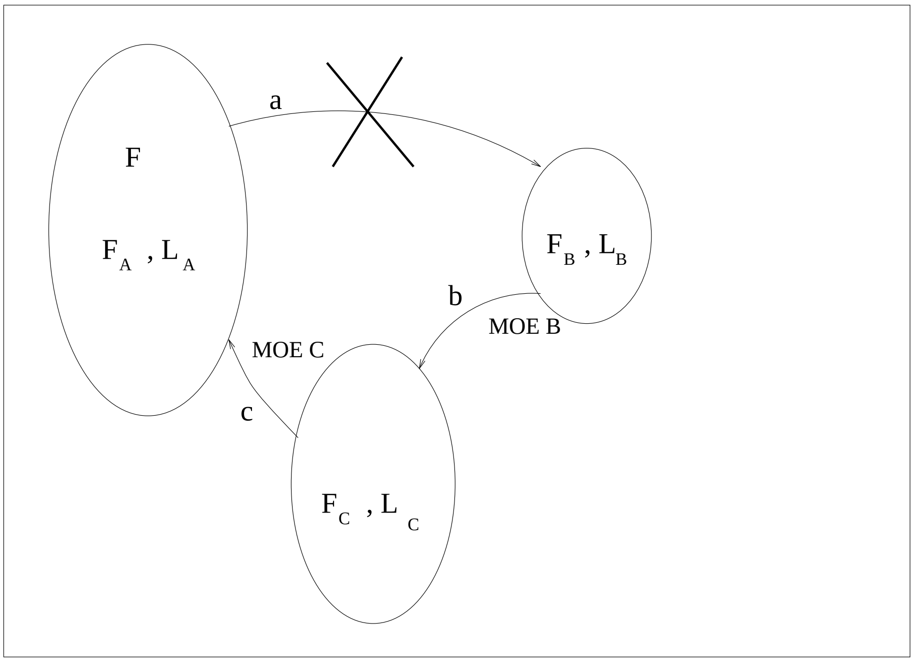

Figure 3.1: Cycles are avoided

The following theorem holds for [GHS 83], [CT 85], while algorithm [Awe 87] is discussed separately.

Theorem 3.2 *The algorithms [GHS 83] and [CT 85] do not create cycles.*

PROOF. Cycles are avoided because of the ordering of edges. The cycle of Fig.3.1 can't exist since $c < b < a$ where a, b, c are the weights of the edges and therefore, F_A can not submit on a, since $c < a$.

Also, because the decisions are taken within a fragment in a centralised way, it is not possible to have a cycle, i.e., a fragment decides to join to one fragment at a time and then participates in a new Finding procedure and re-examine their previous outgoing edges. This centralization and the use of levels in the joining policy are clear in the following discussion.

Assume a cycle of MOEs of fragments (see Fig.3.1), considering fragments as "generalised" nodes.

It is obvious that such a cycle would have to include a submission. A cycle of equi-joins is not possible, since equi-joins are only done along common MOEs for both fragments and so we can have only one for each *initiate* message of each fragment.

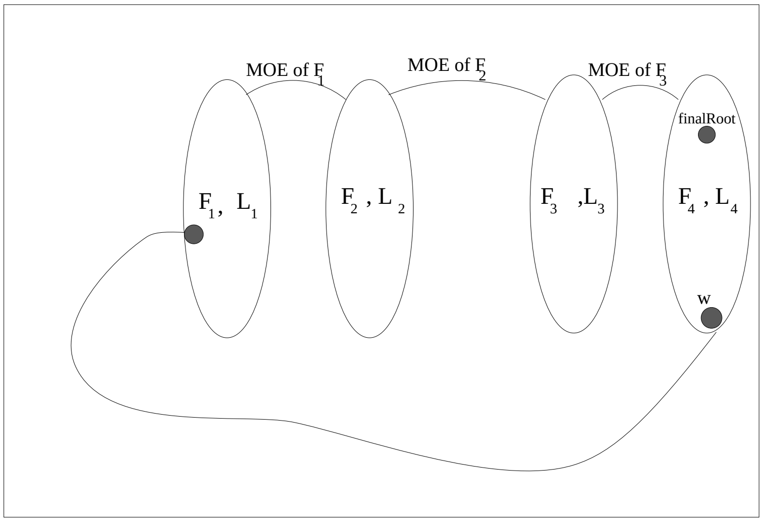

Figure 3.2: A bad case example.

After the new *initiate* message arrives, the level is increased.

Assuming a submission, say F_C to F_A, we have to assume a difference in levels $L_A > L_C$. Assume that F_B wants to join with F_C, then it must be $L_B \leq L_C$. Obviously, $L_A > L_B$ and thus F_A can neither submit nor equi-join with F_B. This way, cycles can not be created.•

3.2 A problem in [Awe 87]

The case that may arise in [Awe 87] but not in [GHS 83] and [CT 85] has to do with nodes of a fragment raising their level independently even after they have submitted, i.e., subfragments raising their level using the Test Distance procedure. We can have a fragment that maintains its ID temporarily, but increases its level to a level that could be equal to or greater than that of the **current** level of the root. It is not difficult to imagine a group of four equal level L fragments submitting (see Fig.3.2). The root of the final fragment (whose ID prevails), say F_4, originally will be at $L+1$ and a distant fragment, say F_1 may be at $L+1$ or more because of the Test Distance procedure. In this case, we can easily see that a node, say w near the final root may receive the $initiate(L+1)$ message from the root. If the local MOE of w goes to F_1, the answer to the *test* message will be positive, and w will report the edge as its local

MOE. There is no reason why this edge can't be minimum, although its not outgoing and therefore F_4 could decide to join along that edge creating a cycle.

In the way things are described in the algorithm, there is no way out of this. Nothing can guarantee that the "bad" edge will not be chosen. Notice that for a given maximum distance D between the root and any node, we can have $2^{L+1} < D < 2^{L+2}$ meaning that both procedures can justify a level increase to $L+1$ but not to $L+2$.

Fortunately, we have found a way to break the symmetry of the two procedures. We demand that Test Distance increases the level of a submitted fragment to $L+1$ only when it detects the distance to the root to be at least 2^{L+2}. This way it will be guaranteed that if Test Distance succeeded and made it to level $L+k$ then Root Update will ultimately increase the level of the final fragment to $L+k+1$. Thus we guarantee that eventually the fragment F_4 will submit to a fragment of greater than the level of any of its subfragments ever was. In the section for the correctness of our approach, we talk about this idea extensively.

Chapter 4

Complexity of Previous Algorithms

This section will offer a rough description of complexity issues of the previous algorithms, in order to show the trade-off between time and message complexity. We also present a necessary modification of the time complexity proof of [Awe 87]. We will insist on the intuitive feeling that one obtains after reading the papers, so as not to bore the reader who can refer to the original papers for more details.

Before starting, it is worth mentioning that communication complexity is defined as the total number of times that messages are exchanged between adjacent nodes, and not, the number of distinct messages issued, i.e., broadcasting an *initiate* message in a fragment of k nodes, is counted as $k-1$ message exchanges. As we saw, every node receives one copy of the message and forwards it to each child, duplicating if necessary.

Time complexity corresponds to the actual time required for the termination of the algorithm, if the transmission delay at each edge is one time unit. Processing time for each message within a node and queuing delays are considered negligible.

The length of messages is $O(log(N))$, i.e., capable of representing N node identifiers. It is optimal and it won't be discussed further (see [GHS 83]).

We can notice that the MST problem in a general graph requires at least $O(E + N \cdot \log(N))$ messages, where E is the number of edges and N the number of nodes.

Every edge must be tested with at least one message or else there may be a node at the center of that edge that will never be detected. The term $N \cdot \log(N)$ is not so straightforward to explain. We can roughly see that if at each step the number of fragments halves, then we have $\log(N)$ such steps and at each step we have N messages. The termination also requires at least $O(N)$ time.

4.1 The pioneering work - [GHS 83]

From a communication point of view, we can easily see that each edge is rejected only once (if at all) and only two messages (two *test* messages or a *test* and a *reject* message) are required. Thus, edge rejection uses $O(E)$ messages. Notice that *test* messages not leading to rejection will be counted in the sequel.

For the rest of the messages we can see that, at each level, the number of messages that a node receives or transmits, is bounded by a constant. In more detail, a node can receive at most one *initiate* message and one *accept* message. It can transmit at most one successful *test* message, one *report* message and one *changeRoot* or *connect* message. As we already said, the maximum level is $log(N)$ and thus the total number of the other messages is $O(N \cdot log(N))$. Therefore in total, the communication complexity is $O(E + N \cdot log(N))$.

As for time complexity, it is easy to see that in the worst case termination time is bounded by $O(N \cdot log(N))$. At each of the $log(N)$ levels, we may have at most N maximum delay, i.e., the messages visit all the nodes sequentially. The bound is tight since there can be a graph that achieves this complexity [GHS 83, pp 74-75].

4.2 The improvement - [CT 85]

The communication complexity does not change and its optimality can be proven with the same arguments.

As for the time complexity, we will not reproduce the example that achieves $O(N \cdot G(N))$, where as already said $G(N)$ is the number of times that the *log* function

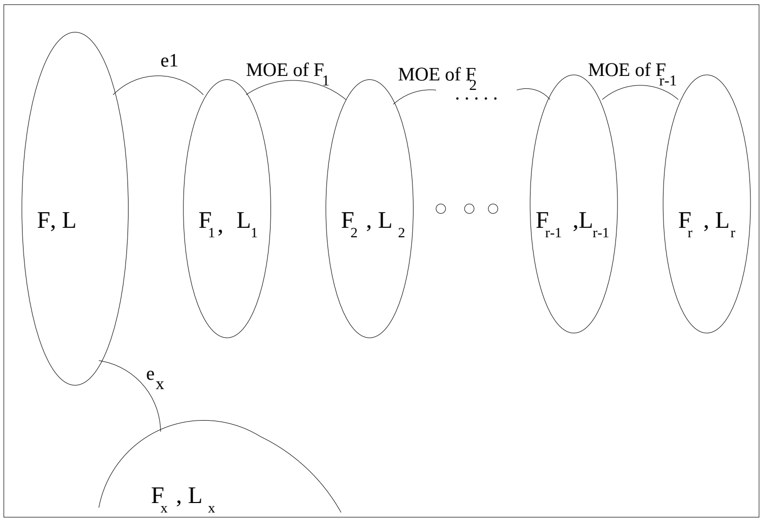

Figure 4.1: Bad case example.

must be applied to N to get a result less than or equal to 1. We find it more interesting to describe a typical case that introduces non-linear time.

A fundamental problem in complexity: Suppose that an outgoing edge of fragment F goes to a smaller or equal level fragment F_1. In addition, F_1 also connects with its MOE to another fragment F_2 of smaller or equal level compared to F_1. We have, that is, a chain of fragments of levels $L \leq L_1 \leq L_2 \leq ... \leq L_r$, where $r >> 1$ (see Fig.4.1). Furthermore, these fragments are connected with their MOE edges in such a way that the joins start at the end of the chain, in other words: $L_{i-2} \rightarrow L_{i-1}$, $L_{i-1} \rightarrow L_i$, and so on.

Since F_1 is smaller, it delays its answer to the *test* message of F. Thus F waits for a "very long" time (i.e., until all nodes of the chain of fragments are traversed) before it can complete its Finding procedure. If F joined finally with the chain, then it would get a great level increase and it would be compensated for the delay. However, this is not guaranteed, since F could end up joining with F_x and get a very small level increase.

4.3 The optimal - [Awe 87]

For this algorithm, we will first examine its time complexity for reasons that will be apparent at the end of this section.

The complexity of this algorithm is the sum of the complexities of its different phases and sub phases. It is comparatively easy and considerably less exciting to see that the Counting phase and the first part of the MST phase are optimal with respect to time (see [Awe 87]). The Counting phase adjacent fragments join in a way similar to that of [GHS 83], but since it is trying to find just a Spanning Tree it is relatively easy to guarantee that fragments almost never wait for other trees. In the first part of the MST phase, the size of the fragments has at most $\frac{N}{\log(N)}$ nodes and the maximum level is bounded by $\log(N) - \log(log(N)) \leq \log(N)$. We can assume that for each of these subfragments we run independently a [GHS 83] algorithm for a a graph of size $\frac{N}{\log(N)}$ and thus the termination time which is the product of the maximum level and the size of the graph is $O(N)$.

For the second part of the MST phase, we will provide an overview of the proof.

CLAIM: For the second part of the MST phase, The length of the time period during which the minimum level is L is bounded by $O(2^L)$ time units.

PROOF - sketch. (Considering $Size(F) = O(2^L)$ could facilitate understanding).

Let A_L denote the first time at which L is the minimum level in the network.

We can distinguish two phases. In each one of them, one of the new procedures.

Phase 1. Root Update Procedure : Consider the minimum level fragments after A_L. In some of them, an *initiate* message may visit 2^{L+1} nodes corresponding to 2^{L+1} time units and the Root Update procedure will increase their level in $O(2^L)$ time.

In the rest of the fragments, after $c_1 \cdot 2^L$ time where c_1 is some constant, they will have found their MOE. Since the fragments are of minimum level, they will be able to join in constant time, i.e., case 2 of non-trivial delay is also bounded. Note that this claim is not true the way things are described in the original paper, but can be made true with a small modification (see the following section).

Therefore all minimum level fragments, after $c_1 \cdot 2^L$ time units, will have increased their level or have submitted. It is this case that we have to examine now.

Phase 2. Test Distance: if a fragment (F, L) has submitted, and received an *initiate* message from the greater fragment it is an acknowledged part of the latter and the previous phase takes care of its level upgrades.

In case where the *initiate* message delays much more than $c_2 \cdot 2^L$ time where c_2 some constant, the Test Distance procedure will cause a level increase.

Notice that in both phases, the time required for the level increase to propagate to all nodes of the fragment is bounded by $c \cdot 2^L$ time units where $c = c_1 + c_2$.

Consequently, after time $O(2^L)$ all nodes will have level greater than L.•

It is not difficult now to prove that the time complexity is linear, by adding the time intervals during which each level is the minimum level.

$$Time = \sum_{i=0}^{log(N)} c \cdot 2^i \leq 2 \cdot c \cdot 2^{log(N)} = 2 \cdot c \cdot N$$

We can now go back to Fig.4.1 and see that L_1 will be updated using the Test Distance procedure, and will reach L "faster" (i.e. $c_2 \cdot 2^L$ time), and thus F will have an answer to its *test* message sooner and thus avoid getting stuck at an edge that may not be its MOE, say e_1 in Fig.4.1.

As for the communication complexity, it is really interesting to notice that the Test Distance procedure could increase the cost above the optimal, i.e., we can think of N/2 submissions each with N/2 *testDistance* messages adding a $O(N^2)$ term in the communication complexity. This is the reason why this procedure is activated after the number of fragments is reduced to $log(N)$. We can't have more than $log(N)$ submissions and thus $O(N \cdot log(N))$ *testDistance* messages. The communication complexity remains optimal.

The communication complexity of the other phases can be proven optimal [Awe 87].

Finally, we can see the logical steps that the creation of the algorithm seems to follow. In order to achieve $O(N)$ time, the Time Distance procedure was needed. This increased the communication cost and in order to keep it optimal we had to demand fewer than $\log(N)$ submissions of fragments that would be using this Test Distance technique. Ideally we would like something like a global observer that would keep track of the number of non-submitted fragments and when this would become less or equal than $\log(N)$, then we would allow the use of Test Distance. Since we can't have such a global observer, Awerbuch thought that if we require each fragment to reach threshold size $S_{Threshold}$ such that before using the Test Distance procedure we would have the total number of fragments bounded by $\frac{N}{S_{Threshold}}$. Since we want the upper bound to be no greater than $\log N$, we let $S_{Threshold} = \frac{N}{\log(N)}$. Each fragment should know this critical size and therefore be aware of N, which makes the Counting phase necessary.

4.4 Correcting [Awe 87]

The complexity analysis of the algorithm seems to have problems in two points. First, when a fragment wants to submit to an equal level fragment, the delay until an answer from the other node arrives may be great enough to destroy time optimality. Second, the Test Distance procedure fails to work as expected in some cases.

4.4.1 Fast Joining Demand

As we said in the previous section, the time complexity proof implies that after minimum level fragments find their MOE, they can submit or equi-join in constant time. However, the implementation details as presented in the paper can not support either claim. For lack of the detailed description, we fill in the gaps with facts from [GHS 83] that the author seems to use as the basis for his algorithm.

First of all we can state the following :

Observation 1. The *changeRoot* message is sent only to the node that has the MOE and thus the majority of the nodes are not aware of the new join and they still wait for a *changeRoot* message or an *initiate* message.

Observation 2. The node that receives the *changeRoot* message, say node v belonging to F, has to wait for the other node, say w of F_1, to express the intention of

F_1 to equi-join or accept the submission of F.

Using these two observations we can think of the following bad case. Assume a long chain of minimum level fragments (Fig.4.1 is a good illustration). We can see that fragment F may wait for time more than $O(2^L)$ before its level is increased. The claim of [Awe 87] time complexity proof doesn't hold.

We have two ways of solving the problem that are connected with the two observation above.

Solution 1. A straightforward way out of this problem is to have each root broadcast a "New root found and you are not it" message, that we can call $notYou$ message, to all the other nodes of the fragment (that participated in the last Report procedure). Thus in $O(2^L)$ time, node w can answer back to v that F_1 will increase its level with another edge and so v must submit.

Note that we will have one $notYou$ message per node per level, i.e., $O(N \cdot log(N))$ messages and thus communication remains optimal.

Solution 2. We can modify the protocol of the joins as follows. When a node receives a *connect* message, then if it has also sent a *connect* message message along the same edge; the node with the greater id becomes the root. If it has not sent a *connect* message, (the bad case is included here) the node accepts the other fragment as submitted, so that the other node can start the Test Distance procedure. The election of the new root is in constant time and unambiguous in both cases.

The second solution requires fewer messages than the first one. In any case we believe that this faster joining policy is more efficient.

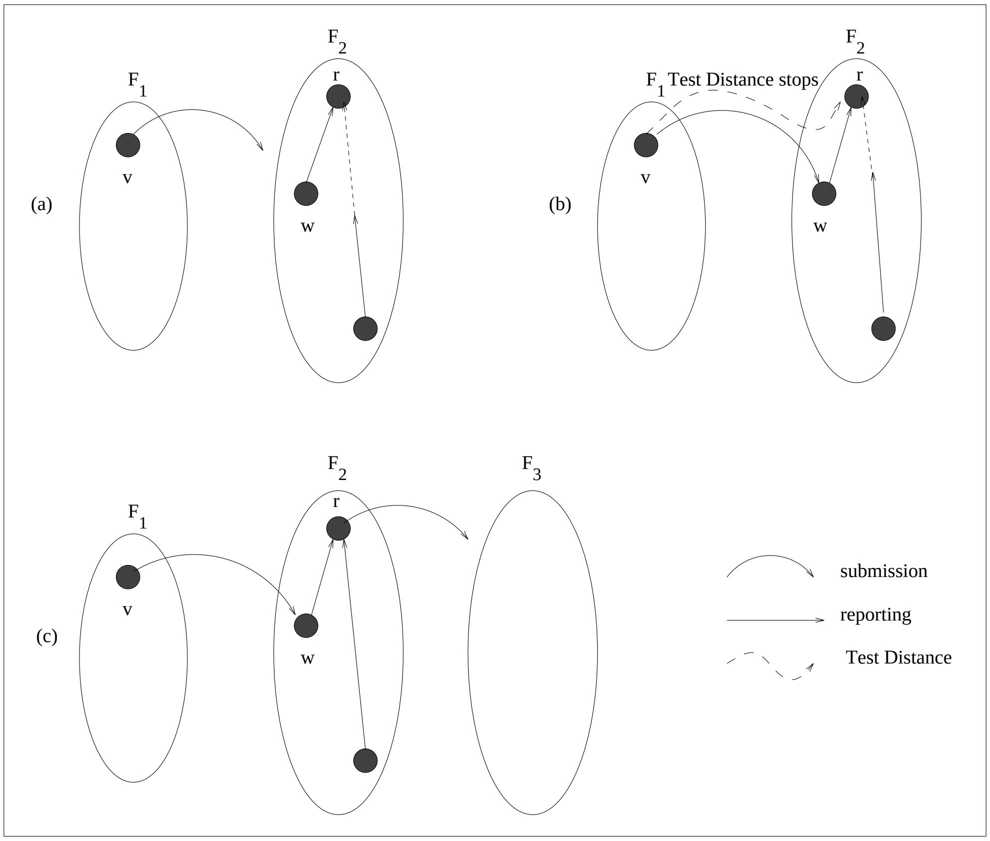

Figure 4.2: Test Distance failure

4.4.2 Test Distance Failure

In this section we will show that there are situations where the Test Distance procedure may fail to fulfill its purpose. As already said, the purpose of the procedure is to keep increasing the level of a subfragment that is very far from the final root (the one that eventually will send the next *initiate* message to that subfragment). So we could say that the Test Distance procedure is needed to keep measuring the distance to the root, until an *initiate* message arrives.

However, we can think of the following situation (see Fig.4.2). A fragment (F_1, L_1) with v as leader submits to node w of fragment (F_2, L_2). We can assume that w has already reported to the root r of F_2, but the Reporting procedure of F_2 has not been completed. Fragment F_1 will activate the Test Distance procedure, but it is possible for the $testDistance$ message to arrive at r before the Reporting procedure is completed, so the message "dies" and the Test Distance procedure of F_1 stops.

We can imagine now that F_2 completes its Reporting procedure and submits to a fragment F_3. It is not difficult to see that the distance of F_1 from the final root (that will send the next *initiate* message to v) may be arbitrarily great. However, the Test Distance procedure has already stopped and it won't detect this distance.

The failure of the Test Distance procedure in this case is apparent. Given this, the proof of time optimality of [Awe 87] does not hold.

A solution to this problem would be to have roots store *testDistance* messages and forward them to the new root after they submit. Note that this idea was suggested by the author of the algorithm in a brief discussion we had with him. In more detail, a *testDistance* never "dies" at the root as originally said in the paper, but it resides there until the decision for the join is taken and it gets forwarded towards the new root. *testDistance* messages "die" when they "cross" with an *initiate* message at an edge.

The only implication of this solution is that each node should be able to store at least *Sub* *testDistance* messages, where *Sub* is the number of subfragments that may submit at some time. Since this trick will be active only in the second sub-phase of the MST phase of [Awe 87], we can have $\log(N)$ subfragments and thus may have to store up to $O(\log(N))$ such messages. Note that this requirement may exceed the default memory that is defined by the number of incident edges a node has. Thus **this approach sets a memory requirement for each node related to the size of the graph**.

We strongly believe that this memory requirement is only a worst case requirement and in most practical cases the absence of this memory won't lead to problems. However, from a theoretical point of view, we can't allow such a possibility of failure and thus we must demand this memory capacity.

As we will see in our approach, we can replace this trick with another that does not require this increased memory requirement.

Chapter 5

Distributed Information

It was surprising to notice that the information in the previous algorithms was never stored. We show here how algorithms can exchange and store information concerning other distant parts of the network and benefit from it. The only cost that we pay is an increase in the length of the message which in any case stays within the same order of magnitude (the new message length is less than double the original message length) and the storage requirements at each node which are also trivial.

In more detail, the information that is stored concerns the previously reported MOE or the distance from the root. Thus the first kind of information can facilitate the Reporting procedure, while the second one assists the level increase of subfragments far away from the root.

We prove that the gain from using distributed information is great both theoretically and experimentally. We present three different ways to make use of this mentality.

5.1 Fast Report using Pruning

The first step would be for each node to store the MOE that exists towards each of its edges. We will call this **edge-MOE**. Every time a message is sent it carries also the information of the MOE that exists -to the best of its current knowledge-from its side of the edge. For example, a node that receives a *report* message will store the reported MOE as the edge-MOE of that edge. Similarly an *initiate* message sent from v to a child w, will carry the minimum edge-MOE of all the other edges including that of the parent edge. Other messages like *connect* can also carry this information.

This could be used in reducing the waiting time for the Report procedure, since in some cases it can be guaranteed that the reports we are waiting from some of our children won't be better than the ones we already had. In such a case, we already know which is the MOE of our subtree and thus we can report it and save time.

Furthermore, it can be accompanied with a new way to see the problem altogether. We can see that what we are really doing in the Finding and Reporting procedure in the wave-out wave-in movement of messages is equivalent to searching a kind of game tree:

> what is the next "move" I should do in order to minimize my accumulated penalty function?

It is important to notice that under the "standard" Joining policy introduced in [GHS 83], the weight of the MOE reachable via a branch can only increase over time. In other words, **an edge-MOE is a lower bound of the actual MOE**, and also **edge-MOEs are non-decreasing in time**. These two observations are fundamental for the correctness of our approach.

We list the ways we can arrange to update and use those lower bounds with the flow of the "waves".

1. in Reporting, each node takes the min of the MOE from each incoming branch, so the node can just store a copy for later use. The idea is that nodes don't wait for a report from an edge whose edge-MOE is greater than the currently reported Best Weight.

   In other words, we apply "search tree pruning" rules to speed up the procedure.

   For example, if branch "a" has reported an MOE of 5, and branch "b" has not reported its MOE yet but its lower bound is greater than 5, then we can report 5 as the MOE from our subtree immediately.

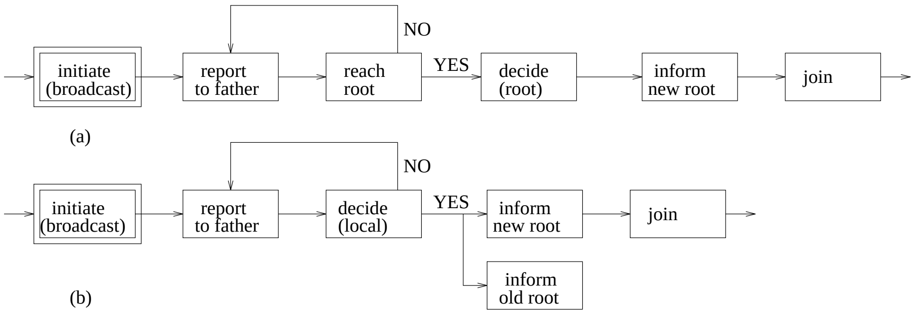

Figure 5.1: The difference in procedure: (a) [GHS 83] and (b) Local Decision to Join.

2. in Finding, the nodes can update their edge-MOEs according to the results of the last Finding procedure.

3. we can eliminate some sub-trees whose last report indicated a MOE of infinite weight. It is clear that there is no need to visit these nodes in the Finding procedure.

The important aspect of this technique is that it can be incorporated in [GHS 83] and gives rise to the algorithm that we will call the **Fast Report algorithm**. In addition, it is “compatible” with all the other techniques that we have seen so far and thus it can be incorporated in the other algorithms directly. This way we can increase the performance of several algorithms by substituting [GHS 83] with Fast Report in all algorithms that use [GHS 83] in one of their phases, or by incorporating the Fast Report technique in their algorithm (or both). We believe that this mentality of storing and using previous information could be prove to be beneficial to other distributed algorithms as well.

The experimental results were impressive, reducing both time and number of messages (see the chapter of Experimental Results).

5.2 Local Decision to Join - AutonoMST

The idea is that nodes can decide locally whether the MOE of the fragment has been found in their subtree. In this case they issue a *changeRoot* for the node to join and also inform the root of the decision (see Fig.5.1). It is obvious how this can speed up the algorithm.

We can note that the decision is based on local and partial information, when this is sufficient for a decision to be made. In other words, nodes have the ability to take advantage of a "good" configuration. We can regard this as an optiMSTic attitude which is a characteristic of our approach. In addition, nodes can function autonomously up to a degree and thus we gave the name **autonoMST** to the algorithm that uses both this and the Fast Report technique. Note that the decision is taken wherever this is possible, and at the worst case at the root.

A brief presentation of the new concept of autonoMST follows:

Local Decision to Join: Each node stores for each of its Branches the edge-MOE which is, as we said, the weight of the Minimum Outgoing Edge that exists in the subfragment that is reachable by that edge. Notice that the two nodes adjacent to a Branch edge store an edge-MOE for that edge, whose value refers to the subfragment on the opposite side of the edge. At some point, an edge-MOE can be the weight of a previous MOE, and therefore smaller than that of the current MOE. Here we rely on the basic claims.

The conceptual differences with respect to the mentality of the other algorithms are: first the nodes can function asynchronously, i.e., part of the fragment may be still in Reporting while a join has already been established; second the place of the decision is not known beforehand, and the nodes themselves check their capability of deciding.

Another way to see this difference is to observe that this algorithm achieves a second degree of distribution - but not of parallelism - within each fragment. As a fragment's decision can be based on "local" data with respect to the graph, a node's decision can now be based on "local" data with respect to its fragment.

It is important to stress that this technique is not "compatible" with the others that demand a centralised decision, i.e., if we expect the root to increase the level and repeat the Finding procedure, we can't allow Local Decisions.

In conclusion we can say that in non-favourable configurations this algorithm will perform as well as [GHS 83], while in favourable configurations it will perform better.

5.3 The Tentative approach

We can think of another way to take advantage of the edge-MOE information.

We can change the Testing policy and have nodes always answer immediately to a test message independently of the relative level difference, with a "look who I am message" including its level and fragment Id. However if the level of the other fragment is less than ours, then the MOE will be considered as **Tentative**. If the level of the adjacent fragment is equal to or greater than our level, the MOE is a **Real** one. We will call **Fake** MOE a Tentative MOE that eventually becomes an internal edge in the same Finding procedure. *Observation:* this may happen to a fragment F of level L, when the other fragment adjacent to the Tentative MOE submits to F at a level lower than L.

The Report procedure has three phases as follows:

- Tentative phase: nodes report fast answers including tentative MOEs.
- Decision phase: The root finds the MOE. If no outgoing edge can be found then the algorithm ends. If this is a Real one the procedure follows as before. If it is Tentative, then the root delays and passes to the next phase.
- Update phase: Nodes that reported a tentative answer will report again, whenever they receive a *reject* or *accept* message, updating the edge-MOEs. We will call this kind of messages *update* messages. The *update* message on its way to the root will select the next MOE of the subtree. The fragment MOE can be rejected or become Real in which case we go to the Decision phase.

We can use also the Root Update technique described in [Awe 87]. As a reminder, this technique enables a root of a fragment to increase its level whenever an *initiate* message visits more than 2^{L+1} nodes, where L is the current level of the root. In this section we will rename the technique to Root Test Distance (RTD).

We will call the algorithm that has both the Tentative approach and the RTD procedure, **Tentative algorithm**. We will prove that it is optimal with respect to time but not with respect to communication.

Communication. The side effect of the Tentative technique is that communication complexity can become non-optimal. We can think of the following example; at a given point we may have a fragment of $N/2$ nodes and $N/2$ fragments of single nodes. The bad case, although completely unnatural, is to have single nodes delay so much that the big fragment will complete its Tentative phase before the nodes submit and that every time a node submits, an *update* message goes back to the root. Thus the nodes that end up submitting to the big fragment will cause $O(N^2)$ *update* messages.

Time. A fragment never gets stuck waiting at an outgoing edge that is not its Real MOE. As we saw this is a major cause of delay that ruins the linearity of time.

We will prove the time optimality in detail.

Definition 5.1 *We call $T_r(L)$ the first time that L is the minimum level of the roots.*

We will prove the following theorem:

Theorem 5.1 *For all possible L, we have $T_r(L) - T_r(L-1) \leq c \cdot 2^L$ time units, for some constant c.*

PROOF. What we want to prove is equivalent to proving that all roots of level L either increase their level or submit after $c \cdot 2^L + T_r(L)$. Assume a fragment F at level L. We can examine the time delay in the different phases of the algorithm.

- The Tentative phase: If RTD succeeds then in $2 \cdot 2^{L+1}$ time units the level is increased. If RTD does not succeed then the distance of any node from the root is less than 2^{L+1} time and we pass to the next phase. In the rest of the phases we will assume that RTD didn't succeed at level L.

- The Decision phase: If the MOE is not Tentative, the root ceases to be the root. Notice that it takes at most 2^{L+1} time units for the Report to complete itself (immediate answer to all test messages). If the MOE is tentative then we have the next case.

- Update phase: If the fragment can not find an outgoing edge then this is the final root and it won't take more than 2^{L+1} time units with L being the final level.

  Assume now that there is a Real MOE found finally. As we said in the observation above, Fake MOEs will submit at a level $L' < L$. Thus, by $T_r(L)$ all the L' level fragments have found their MOE and it takes at most $2^{L'+1} \leq 2^L$ time units for the leader to assume its role and submit. In other words, by $2^L + T_r(L)$ time the fragments of all Fake MOEs have submitted and it takes at most 2^{L+1} time units for the *report* message to reach the root of F.

  Thus in adding the delays of Finding, Reporting and the submission of lower levels fragments we see that by

  $2^{L+1} + 2^{L+1} + 2^L + T_r(L) = 5 \cdot 2^L + T_r(L)$

  all the level L roots have submitted or have increased their level. Therefore there exist values of c that satisfy the inequality of the theorem. •

Theorem 5.2 *The termination of the algorithm takes $O(N)$ time units.*

PROOF. All roots of level 0 know their MOE as soon as the algorithm starts. Trivially using Theorem 5.1,

$$\sum_{L=1}^{\log(N)} T_r(L) - T_r(L-1) \leq \sum_{L=1}^{\log(N)} c \cdot 2^L \leq 2 \cdot c \cdot 2^{\log(N)} \leq 2 \cdot c \cdot N$$

where c is the constant of the previous theorem. After that the Finding procedure of the final root takes at most $2 \cdot N$ time units. •

5.4 Distributed distance from the root - optiMST

As we already saw in the complexity analysis of [CT 85], the Test Distance procedure of [Awe 87], addresses the following problem: if all level increases take place at the root, a small fragment of a level $L+1$ say, waits for a long time to receive an answer

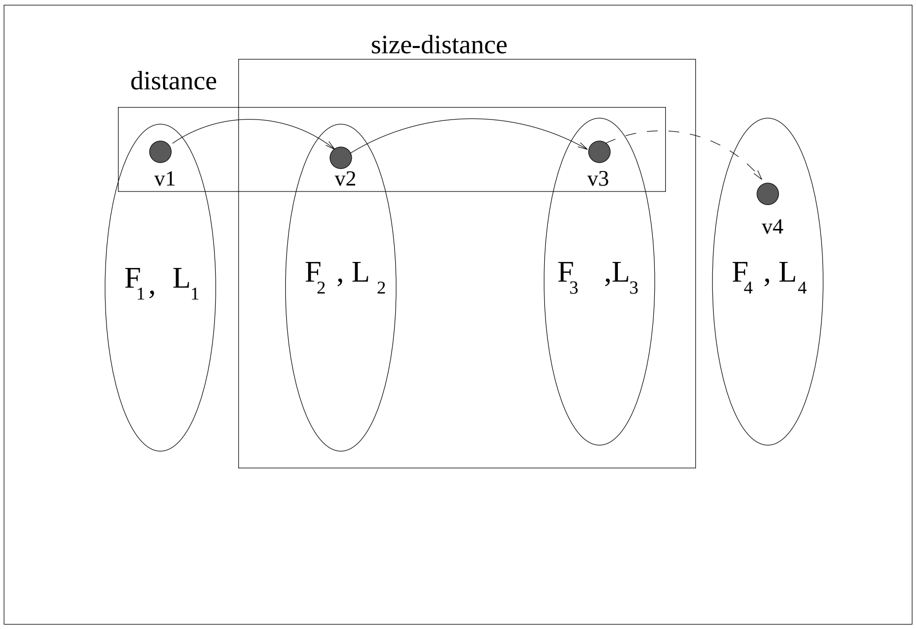

Figure 5.2: Size-distance and simple distance.

from the tail of a long chain of L level fragments. Since, it may finally join along another edge, it is not certain that it is going to be compensated for this big delay by a proportionally big level increase.

Extending the distributed information mentality, we propose a technique to detect the distance from the root by storing distance information at the nodes. Each fragment, that submits to say the end of a long chain, learns the distance from the root. On the other hand when the current root submits, messages are broadcasted and the distance of the already submitted fragments increases.

Furthermore, instead of detecting the distance (number of edges along the path), we count the 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, since the size is an upper bound of the distance (string of nodes) and we will refer to it as **size-distance**. For example in Fig.5.2, we can assume that at a given time v_1 still thinks that v_3 is the root, so its estimate of the size-distance must be equal to $Size(F_2) + Size(F_3)$, where $Size()$ indicates the number of nodes of a fragment. On the other hand, for the simple distance $d(v_1, v_3) = 2$.

The reasons we prefer size-distance to distance are:

- size-distance only increases, while distance can decrease too.
- size-distance is greater or equal to distance and thus we may have faster level increases.

In more detail (see 5.2), consider a fragment F_1, L_1 (denoting the fragment id and the level) that decides to join with F_2, L_2 along (v_1, v_2). Whenever the root changes, a *changeRoot* message reaches the node with the fragment MOE, v_1 in our example. We can claim that this is done in $O(2^L)$ time units. Then, v_1 will send a *connect* message on its MOE and according to the Fast Joining policy, an answer will arrive with no delay (two hops). Then node v_1 issues a *newRoot* message (what we referred to as Solution 1 when discussing [Awe 87]) that will inform every node of its fragment of the possibly new level and will offer an estimate of the size-distance from the root. All nodes will broadcast the *newRoot* message to all subfragments that submitted to them, i.e., when F_2 submits to F_3 the *newRoot* message issued by v_2 will be also sent to v_1. Thus recursively, we can see that information about the size-distance from the root propagates. This information can be used to increase the level of the nodes as the Test Distance technique of [Awe 87] did.

Therefore, as we will prove in the time complexity section, fragments achieve a level increase in time related to their current level. In other words, **the level of the subfragment is increased dynamically and in the worst case the delay for an increase is linear to the size-distance**. It follows that this technique achieves at least the same rate of increase as the Test Distance procedure of [Awe 87].

We could regard the techniques employed in this algorithm as ways to exploit as much as possible "favourable" configurations. We can regard this as an optiMSTic attitude. This is the reason we named the algorithm **optiMST**.

5.4.1 The overview of optiMST

We integrated the previous characteristic to an algorithm. The basis is always [GHS 83] but we had to use some other techniques to make it correct and fast.

This algorithm is optimal with respect to time, but the communication complexity of the newRoot messages is non-optimal.

A brief description of the algorithm (optiMST) follows, assuming the [GHS 83] backbone:

1. Fast Joining policy: **The response to a *connect* message is immediate.** When v of F_1 sends a *connect* message to w of F_2 both the same level, then if w has not sent a *connect* message to v, it will accept F_1 as submitted. Notice that if fragment F_2 finally decides that it wanted to connect to F_1 (common MOE) then w will receive a *changeRoot* message and it will be considered as an equi-join.

   If w has sent a *connect* message before it receives the one from v, then the root will be the node with the smallest (or greatest) id number.

2. Fast Report: We have already seen this technique. Though not indispensable for asymptotic complexity, it is included for improving the practical efficiency in both time and communication.

3. *newRoot* message: When the root of a fragment (F, L) finds the fragment MOE, the node adjacent to that edge becomes the leader (the old root could also become the leader) sends this message to notify every node of the change and to increase their local size counters by the size of the fragment they submit to. For this purpose the message is provided with two fields, FragmentLevel and AddedNodes both of which refer to the level and size of the fragment that accepted the submission. Note that the AddedNodes is at least $2^{L\_other\_Fragment}$. Note that in case an equi-join takes place and the leader becomes the root of the final fragment, it will issue an *initiate* message instead of a *newRoot*. It is also important to see that *newRoot* message will be broadcasted to all fragments that have already submitted. submitted.

4. Root Test Size procedure (Root Level Increase in [CT 85]). When the Finding procedure has been completed, the root checks whether the size of the fragment is big enough to justify a level increase.

The procedure is important because it guarantees the following:

In case of a long chain where the Subfragment Test Size procedure may have succeeded (see below), when the decision for the next MOE is taken, **the join is done with a fragment of level greater than the level of the tail ever was**. This way cycles are avoided.

5. Root Test Distance procedure (Root Update in [Awe 87]). This guarantees that in $O(2^L)$ time, a fragment of minimum level, say L, has either found its MOE or increased its root to $L+1$. Note that for communication optimality, nodes forward only one expired message towards the root $expInit$ at a given level.

6. Subfragment Test Size procedure (Distributed Information - equivalent to Test Distance of [Awe 87]). Intuitively, it gives freedom to the tail of a long chain to increase its level when it sees that it is very far from the root, as not to waste time waiting for an *initiate* message.

5.4.2 Extended Description

We will present first some definitions and terms.

Definition. We will let "subfragment" denote the fragment that has submitted but has not received yet an *initiate* message that increases its level. The root of a subfragment will be called "leader".

Glossary - Notation.

MOE: Minimum Outgoing Edge.

local MOE of a node: the MOE adjacent to that node.

RTD: Root Test Distance procedure.

RTS: Root Test Size procedure.

STS: Subfragment Test Size procedure. In this case size is the number of nodes of the fragments between subfragment and the current (to the best of our knowledge) root.

$message(L)$: $message$ broadcasted from a node at level L.

Fid: abbreviation for fragment identifier.

fragment (F, L): fragment of Fid equal to F and of level L.

Important Points

1. Roles. Nodes within a fragment can take on different roles, which we identify by the following:

   - root. The coordinator and decision maker of the fragment. Like in [GHS 83], its responsibilities, added to that of the simple node, are to establish a Finding procedure and after completing the Report procedure, to nominate the new node to join (leader) or to end the algorithm.
   - leader. It is a node that is adjacent to the fragment MOE and assumes this role when it receives a *changeRoot*. Its responsibilities, added to that of the simple node, are to attempt to join its fragment with the adjacent fragment at the other side of the fragment MOE. Unlike [GHS 83], the leader may have invoke the STS procedure if necessary. If this is the case, it issues *newRoot* to inform the nodes of the fragment of level increases or updates the size-distance from the root. It also filters and dumps *initiate* messages of lower than its current level. The "leadership" ends when it receives an *initiate* message that increases its level.

     Note that a node may receive a *changeRoot* message and become the root right away or after sending a *connect* message. We can call this state tempo-leader, but we can easily see that with the Fast Joining Policy this is a temporary state. Thus we are not going bother about tempo-leader.
   - simple nodes. None of the above. Its responsibilities are to participate in both the Finding procedure (find its local MOE) and Report procedure (report MOE of the subtree).

   Independent of their role in the fragment nodes can be in the following states. First the local MOE can be: **Found** or **Not Found**. Second considering the progress in responding to the most recent *initiate* message we say that the node may be: **Reported** or **Not Reported**. The components described above (role, Found, Reported) determine all the different states for a node.

Here is a list of the different roles and states that a node can be found at. We will start with the situation where root initiates a Finding procedure its fragment:

(a) root enters state (root, Not Found, Not Reported) and broadcasts an *initiate* message to all other nodes. If RTD succeeds, the Finding procedure is restarted (new level).

(b) all other nodes are in state (simple node, Found, Reported) but upon receiving the *initiate* message, enter the state (simple node, Not Found, Not Reported) and pass on the *initiate* message to their children.

(c) all nodes switch from Not Found to Found when they find their local MOE.

(d) simple nodes in state Found that received reports from all children, send their report to their parent and switch to Reported.

(e) root in state Found that has received reports from all children calculates the fragment MOE. If RTS succeeds the Finding procedure is restarted and the fragment MOE recalculated (as we will see in the Correctness section, this is crucial). If RTS does not succeed, the root sends a *changeRoot* message to the node adjacent to the fragment MOE and enters the state (simple node, Found, Reported).

(f) simple node receiving a *changeRoot* forwards it towards the node adjacent to the local MOE. If the latter receives the *changeRoot*, it becomes a leader and invokes the Fast Joining policy (this is the tempo-leader state that we discussed above).

(g) according to the Fast Joining policy, the leader may become the new root, in state (root, Not Found, Not Reported) or become a (non-temporary) leader and invoke the STS procedure until an *initiate* message with greater level than its own current level arrives causing it to enter the state (simple node, Not Found, Not Reported) and forward the message to its children.

2. *Initiate* messages propagate even after the RTD counter expires. The principle that the level never decreases is preserved. This way we take full advantage of the Fast Report technique, we must let *initiate* messages continue as far as they can, because then nodes invoke a Report procedure and the edge-MOEs are being updated. However, leaders ignore *initiate* messages that have smaller level of the leader (this may have happened if STS succeeds)and thus the fundamental principal that the level of a node never decreases is preserved.

3. Level increases in the following cases:

   - root: where RTD or RTS has succeeded. The corresponding messages are *exp_Init* and *report*.
   - node or leader: receives *newRoot* that either is of a greater level than their current level or that makes STS succeed (for the leader only).
   - node or leader: that receives *initiate* of greater level.

5.4.3 How optiMST works

Here are different functional phases of the algorithm. The basic idea has a lot of similarities with those in [GHS 83], so we will focus on the new points.

As we already said, each node in the beginning is the root of a single node fragment and all their edges are considered Unvisited. At the end, the MST edges are labeled Branches and the rest Rejected.

1. **Root takes control:**

   As soon as a node becomes the root of a fragment, say (F, L), it initiates a Finding procedure at that level. The root, and every node that receives an *initiate* message, broadcasts it to its children and then sets out to find its local MOE.

According to the Root Test Distance (RTD), if an *initiate* message visits more than 2^{L+1} nodes, we say that RTD succeeds and an *expInit* message is sent back to the root. The first time an $expInit(L+1)$ message arrives at the root, increases its current level to $L+1$ and restarts the Finding procedure at level $L+1$. Additional $expInit(L+1)$ messages are ignored.

2. **Testing**: The Testing policy is the same as in [GHS 83], i.e., when a node belonging to fragment (F, L) receives a *test* message from node w belonging in fragment (F_w, L_w), it does the following:

   (a) if $F = F_w$, it sends a *reject* message and both nodes label the edge Rejected.

   (b) else if $L \geq L_w$ it sends an *accept* message. The node this message from an edge will consider that edge as its local MOE in the current Finding procedure (but it will test it again each time the procedure is repeated).

   (c) else it must be that $L < L_w$, it delays its answer until its level increases. Note that if the small level fragment F later submits to the bigger F_w, then F will change its ID to F_w before sending its reply and thus the bigger fragment will never consider the edge as outgoing. This is a crucial point of the algorithm.

3. **Reporting**: A node finds its local MOE among its previously Unvisited edges, gathers all the *report* messages from its children and finds the minimum of all the outgoing edges that have been reported to it. Then the node forwards this minimum to its father with a *report* message. The minimum that is calculated at the root is the fragment MOE.

   To support the Root Test Size procedure, *report* messages also have a field that counts the number of nodes that have participated in the reporting procedure so far. I.e. nodes sum up these fields from the report messages of their children, and add one to account for themselves.

If there is not a fragment MOE, then the algorithm terminates. Otherwise, the Root Test Size (RTS) procedure takes over; if the total number of nodes that reported is greater or equal than 2^{L+k}, for some $k > 1$ then the level is increased to $L + k$ and a new Finding procedure begins. This procedure is equivalent to the Root Level increase in [CT 85]. The reason we use it in our algorithm is to guarantee the correctness as it will be explained in the section about correctness.

Assuming that RTS does not succeed, the root sends a *changeRoot* message to the node adjacent to the fragment MOE, appointing it as the leader. On its way, the *changeRoot* message reverses the father relations of the nodes along the path re-orienting the tree towards the leader.

4. **Leader joins:** We will present in detail the Fast Joining policy. Note that it guarantees constant delay (more accurately 2 time units), until the leader receives some answer from the adjacent fragment or becomes the root. With NEW we will indicate the innovations in the Joining policy compared to that of [GHS 83]. Assume that node v receives a *changeRoot* message and becomes the leader of fragment (F, L) and wants to join with w of (F_w, L_w).

   (a) If v has already received a *connect* message from the fragment on the other side of the fragment MOE. It will become the root (since our level is equal to theirs and they submitted first), increase its level and start the Finding procedure.

   (b) If v has not received a *connect* message from that edge, it sends a *connect* message. We have to consider the different states of node w which as we said is at the other side of the edge.

      i. $L_w > L$ and w is in the Not Reported state. Node w sends an *initiate* message to node v, and fragment F participates in the Finding procedure of F_2.

      ii. Node w has already sent a *connect* message to v, i.e., w is also a leader. In this case an old fashioned equi-join will take place; the leader with the minimum id will become the root and initiate a Finding procedure for the merged fragment.

iii. In any other case ($L_w = L$ or $L_w > L$ in Not Reported). Node w sends a *connAccept* message, acknowledging the connection as a submission to F_2. Node v invokes the STS procedure.

Now we will see how the STS procedure works. Upon reception of a *connAccept* message, v invokes the STS procedure, and broadcasts a *newRoot* message right away. Notice that the *connAccept* message carries an estimation of the size of the fragment that accepts the submission. Similarly every *newRoot* message has the field $AddedNodes$ that broadcasts the same information. This information is stored and accumulated at every node at the $SizeCounter$ buffer.

When the leader receives a *newRoot* message, it adds the $AddedNodes$ to its $SizeCounter$ and calculates the corresponding level as[1]

$\lfloor \log(SizeCounter) \rfloor - 1$

If the above calculation yields a greater value than the current level L, then an increase takes place and we say that STS succeeds. Note that this corresponds to the formula:

$2 \cdot 2^{L+k} \leq SizeCounter < 2 \cdot 2^{L+k+1}$

where k is the constant by which the level increases. The factor of 2 is a handicap that we enforce on the STS procedure compared to RTS. In brief, for the same level increase STS must detect twice as many nodes as RTS. This way, it is guaranteed that:

(STS succeeds from L to $L+1$) $\rightarrow$ (RTS succeeds from $L+1$ to $L+2$)

As we will see in the correctness section, this is a crucial property of the algorithm.

[1] $\lfloor x \rfloor$ is the integer k for which $k \leq x < k+1$

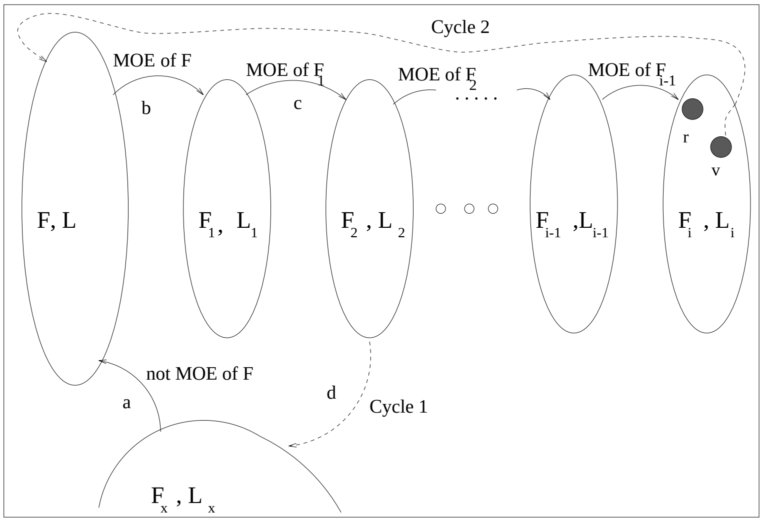

Figure 5.3: Cycles are avoided.

5.4.4 Correctness

For the proof of correctness, we won't repeat the same arguments as for the correctness of the previous algorithms. Instead, we will present a proof that all these algorithms do not arrive in a deadlock. This proof is slightly different from the one in [GHS 83].

To complete the correctness discussion, in Theorem 5.4 we explain how our algorithm avoids cycles.

Theorem 5.3 *The algorithm is deadlock free.*

PROOF. Assume at some time that the algorithm has not finished, i.e., there are at least two distinct fragments. It is sufficient to prove that, at least one join will take place. Consider the minimum unvisited edge from all the edges that don't connect nodes of the same fragment. Since all internal edges will be rejected in finite time, this edge is going to be the MOE for both its adjacent fragments. One of these fragments will be of equal or smaller level with respect to the other. Therefore, its messages will be answered in constant time and the fragments will proceed in a submission or an equi-join depending on whether the levels are different or equal.•

Theorem 5.4 *The algorithm does not create cycles.*

Cycles in general are avoided because of the ordering of edges; in Fig.5.3 Cycle 1 can't exist since $c < b < a < d$ where a, b, c, d are the weights of the edges and therefore, F_2 can not submit on d, since $c < d$. Note that we can assume that the edges have unique weights or that the nodes have unique IDs which is equivalent for our purpose.

We already saw an example (section A Problem in [Awe 87], Fig.3.2), where the algorithm of [Awe 87] can create a cycle. Recall that in that example, fragment F_1 has increased its level (Test Distance for [Awe 87], STS for optiMST) without receiving a message from the final root and thus it does not have the Fid of the final fragment. The Finding procedure of F_1 is already started and node v tests an edge that leads to F_1 and considers it as local MOE, although F_1 has already submitted and is part of the final fragment.

Even in optiMST, this edge will be reported as MOE. However, a circle will not be created because we can guarantee that the final root will have RTS procedure succeed and therefore the Finding procedure will be restarted. Notice that since F_1 had its STS increase the level to $L+1$, it detected a size-distance of $2 \cdot 2^{L+1}$ (factor 2, as already explained, is the handicap factor for STS compared to RTS). Therefore, RTS will detect at least the same number of nodes and increase its level to $L+2$ and restart the Finding procedure. In this new procedure, we can easily see that w will not get an *accept* answer from F_1, because while F_1 is unaware of the new identity (F_1) it is also unaware of the level increase, i.e., it is still in level $L+1$ and therefore delays its answer. When F_1 becomes $L+2$ by receiving the *initiate* message from the final root, it also changes its Fid to F_1 and therefore rejects the edge.

Generalizing the above (see Fig.5.3 Cycle 2), we can say that when a subfragment succeeds to increase its level to $L+k$, its STS procedure detects at least $2 \cdot 2^{L+k}$ nodes in the fragment. Therefore, the RTS procedure invoked by the final root will detect at least the same number of nodes and will be able to increase its level to $L+k+1$. Therefore, the final decision for submission will be taken at this level and it will be with a fragment of at least $L+k+1$, so this could never be a subfragment of the chain. In other words, **the join is done with a fragment of level greater than the level of any subfragment ever was**.

5.5 Complexity of the algorithm

Note that the maximum level that the fragments may ever reach is the floor of the logarithm of N and thus the number of different levels is $1 + \lfloor \log N \rfloor$. For the rest of the discussion we will assume that N is a power of 2 and omit the floor function.

5.5.1 Communication Complexity

The number of messages that are also used in [GHS 83] are bounded by $O(N \cdot \log(N))$ and so we will examine the new messages. As for the length of the messages, in [GHS 83] and the other algorithms the messages had enough bits to carry the information of the fragment ID (N distinct IDs) and level and the weight of an edge. In our algorithm(s) the length does not have to increase by more than two new fields that would carry the information of the size (limited by N) and or the weight of an edge to update the edge-labels.

A node can send at most one *expInit* message at each level as we already said and thus we can have at most $O(N \cdot \log(N))$ such messages.

The *connAccept* messages are sent only as a response to a *connect* message and thus there are as many as the number of branches, i.e., $O(N)$ and thus the total is still bounded by $O(N \cdot \log(N))$.

The complexity of the *newRoot* messages is not optimal. The broadcast of *newRoot* messages begins when a fragment, say F, submits to F_1 and messages of this kind are broadcasted to the nodes of F and also to the nodes of fragments that have submitted to F. Notice that if F_1 submits it will send *newRoot* messages to its nodes and the fragments that have submitted to it (i.e. F) which they will broadcast to submitted fragments and so on.

We can see that we can have $O(Sub \cdot N)$ *newRoot* messages, where Sub is the number of different fragments (at most $O(N)$) at a given moment, and therefore in the worst case we can have $O(N^2)$ *newRoot* messages which is non-optimal.

A pathological example is shown in (Fig.5.4). We can assume the graph to be a string of nodes with every node, except the first and last, connected to two others.

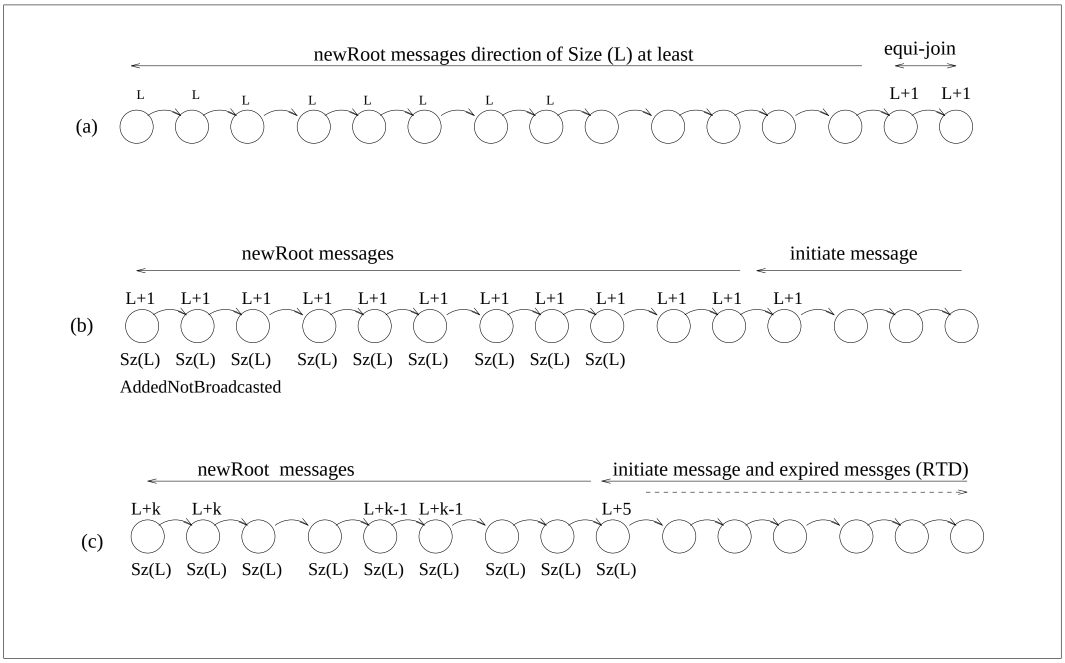

Figure 5.4: A bad example for communication complexity

The weights decrease from left to right, so every node will submit to each neighbor on the right. Note that other edges may exist, as long as they don't ruin the previous situation of submissions. However other edges are not of interest right now.

This way we end up having a huge line of submitted subfragments. It is very easy to see that the total number of *newRoot* messages is $O(N^2)$.

A solution to this problem would have been to filter and concentrate the *newRoot* messages. The leaders could concentrate the information of the arriving *newRoot* messages and broadcast such a message only when they cause a level increase or increase the $SizeCounter$ (counting the size-distance) of the nodes that receive this message. It is not difficult to see that this can reduce the complexity of the messages.

The difficulty lies in the fact that almost all nodes (except for a few at the right end, i.e., the final root and the nodes near it) at the same time have the same level and information ($AddedNotBroadcasted \neq 0$); they are identical from a functional point of view. So, if the protocol does not allow them to broadcast the information they have, none will broadcast information and nothing will happen. On the other hand if they all broadcast their information we will end up broadcasting all *newRoot* messages. In other words, we need a way to break this symmetry and have some nodes send *newRoot* messages and some wait and concentrate information.

We can go back to our example. In a high level view, the information about the distance from the root propagates in small chunks in a lot of messages. It seems reasonable to have the leaders concentrate the information of the *newRoot* messages they receive and broadcast the information only when the "chunk" of the information is "big" enough. (A reasonable rule is to broadcast *newRoot* messages if the information about the size-distance they carry is comparable to 2^L where L is the current level of the subfragment.)

However, we can easily imagine a situation where all nodes in a magical synchronized manner will pass a *newRoot* to the left and then receive (probably) a *newRoot* message from the right node.

Thus at each step each node will:

- send a *newRoot* (if eligible)
- receive a *newRoot* – increase level if eligible.
- repeat

Assume that the leaders broadcast all the *newRoot* messages they receive. We can see that nodes that increase their level with STS will do so synchronously. For example, in 4 time units all eligible nodes will have received four *newRoot* messages and increase their level to $L = 1$ and in 8 time units all eligible nodes will have received 8 *newRoot* in total, and increase to $L = 2$ and so on.

The symmetry is obvious and thus if the filtering procedure will enable one of the nodes not to broadcast a *newRoot* message it receives then they will all do and therefore no more *newRoot* messages will be broadcasted and so the nodes will stay idle waiting for a *initiate* message. The failure of STS is obvious in this case.

Further research can be done to explore the possibility of an efficient filtering policy.

5.5.2 Time Complexity

Following time complexity analysis for the previous algorithms, we will assume that any message exchanged between two adjacent nodes introduces delay of one time unit.

Definition 5.2 *We call $T(L)$ the first time where all the nodes are at least of level L.*

Lemma 5.1 *In any active Finding procedure after $T(L-1)$, the following statements hold for consecutive initiate messages (originating from the same root):*

a) a root issues a consecutive initiate$(L_1), L_1 > L-1$ after initiate$(L-1)$, in no more than $2 \cdot 2^L$ time units.

b) after an initiate$(L-1)$, a simple node receives a consecutive initiate$(L_1), L_1 > L-1$ in no more than $2 \cdot 2^L$ time units.

PROOF. a) There are two cases for a root to restart a Finding procedure: First, if the RTD procedure succeeds, then it takes the *initiate* message 2^L time units to expire and 2^L time units for the *expInit* to return to the root and for a total of $2 \cdot 2^L$ time units for the root to issue the consecutive *initiate* message.

Second, if RTD does not succeed and RTS does, then the distance from the root, r, to of any node v of the fragment satisfies

$d(r, v) < 2^L$

Note that since it is after $T(L-1)$, all *test* messages from a $L-1$ level node are answered without extra delay, adding just 2 time units. Therefore the total time delay until RTS succeeds and the next *initiate* messages are sent is determined by the most distant node w:

$2 \cdot d(r, w) + 2 \leq 2^L$

b) Since consecutive *initiate* messages from the same root follow the same path to the simple node, the proof of (b) follows directly from (a).•

Theorem 5.5 *The following claims hold for all L, $L = 1, ..., \log(N)$* [2] *and some constant numbers c_1, c_2, c_3:*

[2] we are not going to bother taking the floor. We can assume N to be a power of 2

Claim 1(L). *All root nodes reach at least level L or submit and the node to join becomes a leader in at most $c_1 \cdot 2^L$ time units after $T(L-1)$.*

Claim 2(L). *All leaders reach at least level L, or change roles to roots or simple nodes of level at least L in at most $c_2 \cdot 2^L$ time units after $T(L-1)$.*

Claim 3(L). *All simple nodes reach at least level L in at most $c_3 \cdot 2^L$ time units after $T(L-1)$.*

Note that we count the time after $T(L-1)$ in "2^L time chunks". We know that 2^L is the maximum size of a fragment that submits at $L-1$ level (because of RTS) and so the constants offer an indication of how many times a fragment may have to be traversed until an increase is achieved.

PROOF. For all three claims we will use simultaneous induction on L. We will use M to denote the dummy variable of the induction.

- For $M = 1$. This not difficult to prove since in the beginning all fragments are of 0 level, i.e., single nodes. Thus, every node simply selects its minimum edge as its MOE, takes the role of the leader and joins along that edge. If an equi-join takes place, then in 1 time unit all nodes that are roots at level 1 are aware of the fact and thus for Claim 1 we just have to demand:

  $1 \leq c_1 \cdot 2$

  Every node then is a root or a submitted leader. If a leader is further than 4 nodes from its final root (the root that will issue the next *initiate* message the leader will receive) then after 4 time units at most, it receives a *newRoot* and increases its level to 1. Thus

  $5 = 1 + 4 \leq c_2 \cdot 2$

  When the algorithm starts all nodes are leaders so if Claims 1 and 2 hold, so does Claim 3. □

- For $M = L - 1$. We assume that:

  Claim 1($L-1$): All roots reach at least level $L-1$ or submit and the leader is aware of its role no later than $c_1 \cdot 2^{L-1} + T(L-2)$.

Claim $2(L-1)$. All leaders reach at least level $L-1$ or become roots or simple nodes of level at least $L-1$ no later than $c_2 \cdot 2^{L-1} + T(L-2)$.

Claim $3(L-1)$. All simple nodes reach at least level $L-1$ no later than $c_3 \cdot 2^{L-1} + T(L-2)$.

- For $M = L$.

Proving Claim 1(L). We want to prove that all roots have reached a level of at least L by time $c_1 \cdot 2^L + T(L-1)$ or control of the fragment has been transfered to the leader.

Consider a root of level $L-1$ after time $T(L-1)$. Its first action as root is to send an *initiate*$(L-1)$, so it will send one now unless it has already done so. There are two cases:

1. Either the *initiate* message expires and Root Test Distance succeeds (in $2 \cdot 2^L$) or Root Test Size succeeds (complete report at $L-1$ without RTD succeed and *test* messages answered right away) and in both cases by at most $2 \cdot 2^L + T(L-1)$ the root is L.

2. Neither RTD nor RTS succeed. It takes at most 2^L for the *initiate* message to reach all nodes in the fragment, else RTD would have succeeded. All those nodes will send *test* messages, for which there is no delay in the answer because it is after $T(L-1)$ and all adjacent nodes are at least of $L-1$. Therefore by time $2 \cdot 2^L + 2 + T(L-1)$ at most, the report procedure has been completed and the root submits. It takes 2^L more time units for the *changeRoot* message to arrive to the appropriate node.

   Max Time: $2^L + 2 \cdot 2^L + 2 + T(L-1)$

We can see that the worst case from a time point of view is $3 \cdot 2^L + 2 + T(L-1)$

So the theorem holds if c_1 verifies the following formula:

$3 \cdot 2^L + 2 + T(L-1) \leq c_1 \cdot 2^L + T(L-1)$

which leads to :

$3 + 2^{-L+1} \leq c_1$

and for the minimum value of L which is 1, we get the max value of the left hand side:

$4 \leq c_1$. □

Proving Claim 2(L). We want to prove that all leaders are at least of level L, or changed roles to become roots or simple nodes of level at least L, by the time $c_2 \cdot 2^L + T(L-1)$.

Using Claim $1(L)$, we know that after $c_1 \cdot 2^L + T(L-1)$, all fragments that will submit at $L-1$ have done so, or in other words there are no "free" roots of level $L-1$ and all leaders at level $L-1$ will be aware of their role.

The picture of the graph at $c_1 \cdot 2^L + T(L-1)$: we have roots of level at least L, leaders of $L-1$ or more (with those of level $L-1$ aware of their role), and simple nodes of level $L-1$ or more. We will focus on the leaders of level $L-1$.

Let v be a leader after $c_1 \cdot 2^L + T(L-1)$. In constant time (2 time units) after realizing its new role, it will know whether it is the new root of the join or just a leader. We have two cases.

In the first (easy) case, the leader becomes the root in constant time according to the Fast joining policy.

In the second case, node v remains the leader (it receives a *newRoot*, i.e., its submission is accepted). Following the path towards the current root, after $2 + c_1 \cdot 2^L + T(L-1)$ we will certainly arrive at either the current root or a leader of level greater than or equal to L. Let us denote this root or leader by r.

We have the following cases:

1. $d(v,r) \leq 4 \cdot 2^{L-1}$, where $d(v,r)$ is the number of edges in the path between v and r.

   In both possible roles of r, there is a message of level L on its way to v, *initiate* (if r is a root) or *newRoot* (if r is a leader). This message will reach v in at most $4 \cdot 2^{L-1}$.

Max Time: $4 \cdot 2^{L-1} + 2 + c_1 \cdot 2^L + T(L-1)$

2. $d(v, r) > 4 \cdot 2^{L-1}$.

   In this case, there must be a node x for which $d(v, x) = 4 \cdot 2^{L-1}$. If x or another node on the path from v to x is of level greater or equal to L then the time delay until v increases its level, is as in the previous case.

   Therefore, we assume the worst case, i.e., that x and all nodes on the path from x to v, are at level $L-1$. We know that by $T(L-1)$ at the latest, x had its level increased to $L-1$. This may have happened with two messages:

   (a) x received a *newRoot*.

   We assume that leaders broadcast all *newRoot* messages they receive, i.e., there is no Filtering policy.

   The *newRoot* will travel towards v and will arrive at v having visited the $4 \cdot 2^{L-1}$ nodes of the path. As we saw the size-distance that v will estimate is an estimate of the distance. Thus v will have STS succeed and increase its level in $4 \cdot 2^{L-1} = 2 \cdot 2^L$ time units.

   Max Time: $4 \cdot 2^{L-1} + 2 + c_1 \cdot 2^L + T(L-1)$.

   (b) x received an *initiate*$(L-1)$.

   If v does not receive this *initiate*$(L-1)$ message, then v must have submitted after the Reporting was completed on the path from v to x and thus, v invokes STS. As we assumed the nodes from v to x are of level $L-$ and therefore submit by $c_1 \cdot 2^L + T(L-1)$, In at most $4 \cdot 2^{L-1}$ after that, v is informed of this size-distance and increases its level (STS).

   If v receives this *initiate*$(L-1)$, this will not require more than $4 \cdot 2^{L-1}$ time. It is obvious that this message will have expired. Using Lemma 5.1 (b), we know that v will receive an *initiate*(L) in $2 \cdot 2^L$ after receiving the previous message.

$$\text{Max Time: } 4 \cdot 2^{L-1} + 2 \cdot 2^{L} + 2 + c_1 \cdot 2^{L} + T(L-1)$$

We can see that the worst case is $4 \cdot 2^{L-1} + 2 \cdot 2^{L}$ after $2 + c_1 \cdot 2^{L} + T(L-1)$ and so, we have to demand:

$4 \cdot 2^{L-1} + 2 \cdot 2^{L} + 2 + c_1 \cdot 2^{L} + T(L-1) \leq c_2 \cdot 2^{L} + T(L-1)$

$4 \cdot 2^{L-1} + 2 \cdot 2^{L} + 2 + c_1 \cdot 2^{L} \leq c_2 \cdot 2^{L}$

which gives:

$2 + 2 + 2^{1-L} + c_1 \leq c_2$

or for $L = 1$:

$5 + c_1 \leq c_2$ □

Proving Claim 3(L). We want to prove that all nodes are at least of level L by time $c_3 \cdot 2^{L} + T(L-1)$.

If a node is a root or leader by $T(L-1)$ then we can use Claim 1(L) or Claim 2(L) with $c_1 \leq c_3$ and $c_1 \leq c_3$.

Since any simple node has the same Fid as its associated root or leader, say r, from a root or leader say r. We can intuitively see that the techniques RTD and RTS increase the level in such a way that a node is "near" their root or leader with respect to their level. More accurately, the word "near" means $d(r, v) \leq 2^{L_1+1}$ where L_1 is the current level of the root (in case r is a root) and the level the last Finding procedure was carried out (in case r is a leader). Therefore, using Claims 1 and 2 all nodes will be updated in $O(L)$ time units.

In more detail:

1. If r is a root then by $T(L-1)$ v has received an $initiate(L-1)$ from r. We have the following two cases. First, if the fragment submits at level $L-1$ then $d(r, v) \leq 2^{L}$ and the root becomes a leader. Using Claim 2, this leader is at least of level L by $c_2 \cdot 2^{L} + T(L-1)$ and thus it may take v at most $d(w, r) + c_2 \cdot 2^{L} + T(L-1) \leq 2^{L} + c_2 \cdot 2^{L} + T(L-1)$ time to increase its level.

Second, if RTD or RTS succeed then using Lemma 5.1 we know that v will receive its next *initiate* message of level at least L in at most $2 \cdot 2^L$ time units thus no later than $2 \cdot 2^L + T(L-1)$ time.

Max Time: $d(w,r) + c_2 \cdot 2^L + T(L-1)$ with $d(r,v) \leq 2^L$.

2. If r is a *leader* and L_1 the level at which the fragment submitted, we have the following cases. If $L_1 \geq L$, v is already at a level at least L. If $L_1 < L$, then $d(r,v) \leq 2^{L_1+1} \leq 2^L$. Therefore, when by $c_2 \cdot 2^L + T(L-1)$ r increases its level to at least L, v will do the same in at most $d(w,r) + c_2 \cdot 2^L + T(L-1)$ time.

   Max Time: $2^L + c_2 \cdot 2^L + T(L-1)$

We demand that:

$d(w,r) + c_2 \cdot 2^L + T(L-1) \leq c_3 \cdot 2^L + T(L-1)$

with $d(w,r) \leq 2^L$ and so:

$1 + c_2 \leq c_3 \square$

Listing all the constraints:

$2 \cdot c_1 \geq 1$, [from M=1 step]

$2 \cdot c_2 \geq 5$

$c_1 \geq 4$, [from M=L step]

$c_2 \geq 5 + c_1$,

$c_3 \geq c_2 + 1$

$(c_3 \leq c_1, c_3 \leq c_1$ are implied)

We can easily assign values to the constants that satisfy the above constraints.•

Corollary. The time until the algorithm terminates is $O(N)$.

PROOF. Using Claim 3, it is easy to see that $T(L) - T(L-1) \leq c_3 \cdot 2^L$ and so nodes reach the final level $L \leq \log(N)$ in at most:

$$\sum_{L=1}^{\log(N)} T(L) - T(L-1) \leq \sum_{L=1}^{\log(N)} c_3 \cdot 2^L \leq 2 \cdot c_3 \cdot 2^{\log(N)} \leq 2 \cdot c_3 \cdot N$$

time units.

Therefore after time $T(\log(N))$ at most, nodes have either the final fragment ID or are in a fragment that has submitted to the final fragment. Otherwise, we would have two different fragments of level $\log(N)$ and when they would equi-join the final level would be greater than the maximum possible. Therefore, it will take the final root at most $2 \cdot N$ to find out the termination of the algorithm and N more for all nodes to be informed.

Thus the algorithm is optimal with respect to time. •

5.6 Creating Optimal Algorithms

So far we have seen a number of different algorithms. All of them are based on [GHS 83], and they use different techniques. We can group these techniques and of course the corresponding algorithms according to whether they have optimal behavior.

Communication: The optimal number of messages is a $O(E + N \cdot \log(N))$.

1. the plain algorithm ([GHS 83]).
2. Node Counting ([CT 85]) - The RTS in optiMST.
3. Fast Report the algorithm and the trick.
4. Local Decision to Join (autonoMST).
5. Root Test Distance in optiMST - Root Update in [Awe 87].

Time: The optimal time is $O(N)$ time units (assuming that all edges have transmission delay equal to 1 time unit).

1. Tentative technique (Tentative algorithm).
2. Test Distance (third phase of [Awe 87]'s algorithm).
3. STS (optiMST).

Both: Combining the above performances.

1. The algorithm of [Awe 87].
2. Any algorithm with
   - *Counting Nodes Phase.* We can use the [Awe 87] Counting algorithm, or any algorithm for finding a Spanning Tree as long as it is optimal algorithm with respect to communication and time.
   - *MST Phase.*
     - *Fragments' Size: less than* $\frac{N}{log(N)}$. Any algorithm optimal with respect to communication that for the "small" size fragments is also optimal to time. It was proven that [GHS 83] has this property and so do the communication optimal we listed above, since they are based on it.
     - *Fragments' Size: greater than* $\frac{N}{log(N)}$. Any algorithm optimal with respect to time which for a total of less than $\log(N)$ independent fragments, is optimal with respect to communication (Awe's third phase algorithm, optiMST).

       Note: Every fragment that finds out that its size satisfies the above inequality, can activate communication expensive techniques.

5.7 optiMST+: the optimal extension

It is easy to see how we can have an optimal algorithm based on the Distributed Information mentality.

If the number of subfragments that invoke STS is less than $\frac{\log(N)}{N}$, then the number of *newRoot* messages is bounded by $O(N\log(N))$ even assuming that for every submission all nodes receive a *newRoot* message.

Thus using the "optimal pattern" we have:

1. Counting phase. In [Awe 87] we find such an algorithm with optimal characteristics.

2. MST phase.

(a) *Fragments' Size: less than* $\frac{N}{log(N)}$.

In this phase we can use Fast Report which is simple or autonoMST to achieve maximum speed. The complexity of this part is within the time and communication optimal bounds, since the [GHS 83] is optimal (see [Awe 87]) and Fast Report is just a faster version using the same number of messages.

(b) *Fragments' Size: greater than* $\frac{N}{log(N)}$.

We can use optiMST. As we said in the communication complexity section, having $\log(N)$ fragments, we have $O(N \cdot \log(N))$ *newRoot* messages which were the only messages with non-optimal complexity. The time complexity of optiMST is optimal as we proved in the time complexity section.

Therefore this part is also within the optimal bounds and therefore optiMST+ is optimal.

Chapter 6

Experimental Results

We created a simulator using SMURPH ([GR 91], [Gbu 93]), a package for simulating low level communication protocols. The package is implemented in C++ and appears to the end user as a low level network protocol extension of the language. It was especially helpful to have the SMURPH code take care of all the time related issues. However, we had to define a few higher level classes that were missing such as bidirectional edges, nodes and messages.

The correctness of the solutions was tested with a non-distributed MST algorithm known as Prim's algorithm [Pri 57], and described in [AfP 90].

The algorithms that we decided to test are:

- the basic [GHS 83] algorithm, which is non-optimal time.
- Fast Report ([GHS 83] + Fast Report technique), which is also non-optimal in time.
- optiMST - non-optimal communication.
- the second sub-phase of the (MST) phase of [Awe 87], i.e., optimal time, non-optimal communication. Note that this algorithm has the same role in [Awe 87] that optiMST has in optiMST+. Notice that we didn't implement the forwarding of the *newRoot* messages (see subsection of Test Distance Failure), but we implemented the necessary for correctness handicap of the Test Distance

procedure with respect to the Root Update. The Test Distance failure is partly responsible for the longer termination time this algorithm has compared to optiMST. We believe that having the Test Distance work correctly, the time performance of these algorithms will be similar to that of optiMST.

- the second (MST) phase of optiMST+, i.e., the algorithm originally behaves like Fast Report and after the level of fragments reach $\frac{N}{log(N)}$ it behaves like optiMST. Notice that the Counting phase of optiMST+ would increase the calculated delay and the number of messages. Notice that still the total is optimal in both respects.

Notice that the algorithms terminate when the final root is aware of the end, i.e., the report procedure indicates no outgoing edge. However, the nodes are not aware of the termination and/or the final fragment identifier and level, since for example with the Fast Report procedure, some *initiate* messages are not broadcasted to all nodes. One can argue that a final *terminate* message may not be necessary, since the root may right after termination start sending some kind of useful message for the application that requires the MST.

The following sections present different groups of experiments. We chose families of graphs with different topological properties (see Fig.6.1). Family A and family B are random graphs, meaning that every pair of nodes has equal probability to be interconnected with an edge. The third group is a family of graphs is a family of paths that represents a pathological case for these algorithms. We also employed different policies of assigning transmission delay to edges, because it is necessary for weights to represent the actual transmission delay.

We compare the previous algorithms by presenting their time and communication behavior with respect to edges and nodes. Notice that time is the actual termination time as measured by SMURPH, i.e., the delivery of a message across an edge that we consider, to have a transmission delay of T time units, would require T time units of measured time. Finally we plot the behavior of the algorithms from a level point of view.

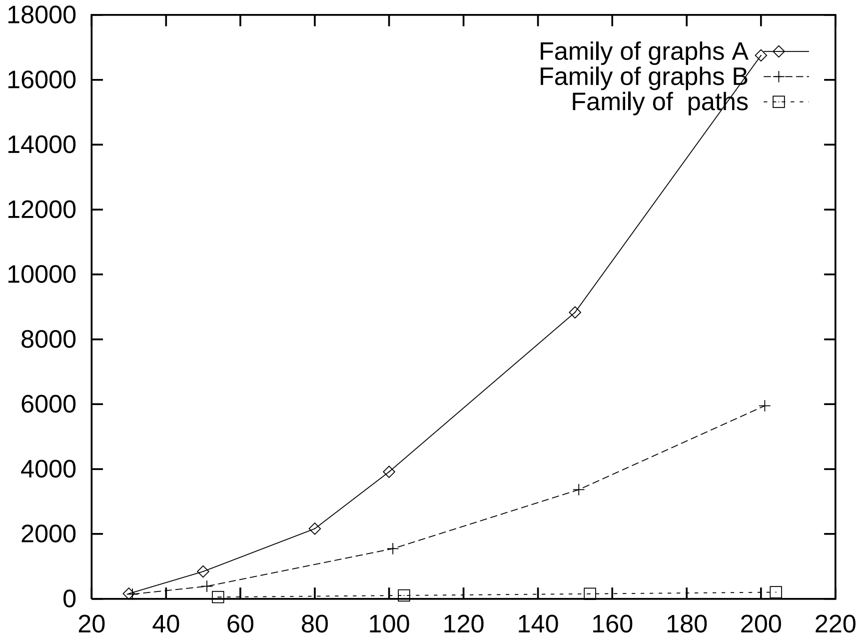

Figure 6.1: Families of graphs. Edges vs Nodes

We would like to stress the importance of the topological properties in the performance. The experiments showed that different graphs would produce different results even if they had they same topological characteristics and number of nodes. It is also clear that the number of nodes and edges is not enough to characterize and classify graphs with respect to their behavior in distributed algorithms. However, what we finally achieve by this is to mainly show the inadequacy of such a simple approach to describe anything. We believe that there is a lot of work that can be done in characterizing graphs with properties that would be "interesting" to distributed algorithms.

6.1 Dense Random Graphs (Family A)

In this family of graphs, that we call A, we created random graphs. Every node had a constant probability to connect with any other node. In more detail, for each graph we set a threshold and then for every pair of nodes we take a random number from a uniform distribution. If the number is above the threshold then we create an edge between the two nodes and assign another random number as its weight.

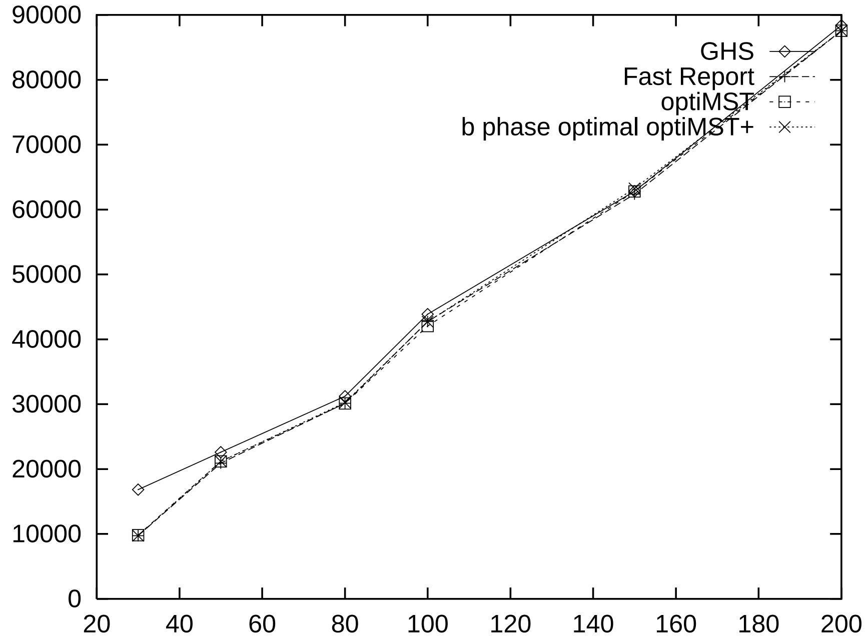

Figure 6.2: Time versus nodes for A family graphs

The relation of the edges and the nodes is plotted in the figure 6.1.

6.1.1 Time Performance

As we can notice (Fig.6.2, Fig.6.3), the efficiency of the algorithms is very similar for this family. It is worth saying that from the techniques used only the Fast Report technique and the RTS have an active role in shaping the results. The RTD procedure appears only in the very beginning, say level 1 and 2 out of 7. We must notice that STS doesn't appear at all. The success of each of the procedures was traced in a log file during the execution of the program.

6.1.2 Number of Messages

We can recall that the number of messages for communication optimal algorithms is $O(E + N \cdot \log N)$. Since the graphs are quite dense $E >> N$ and thus it is the edges that finally define the message complexity. The non-linear relation of the messages with respect to nodes is demonstrated in Fig.6.4.

We were really happy to see the linear relationship between the edges and the

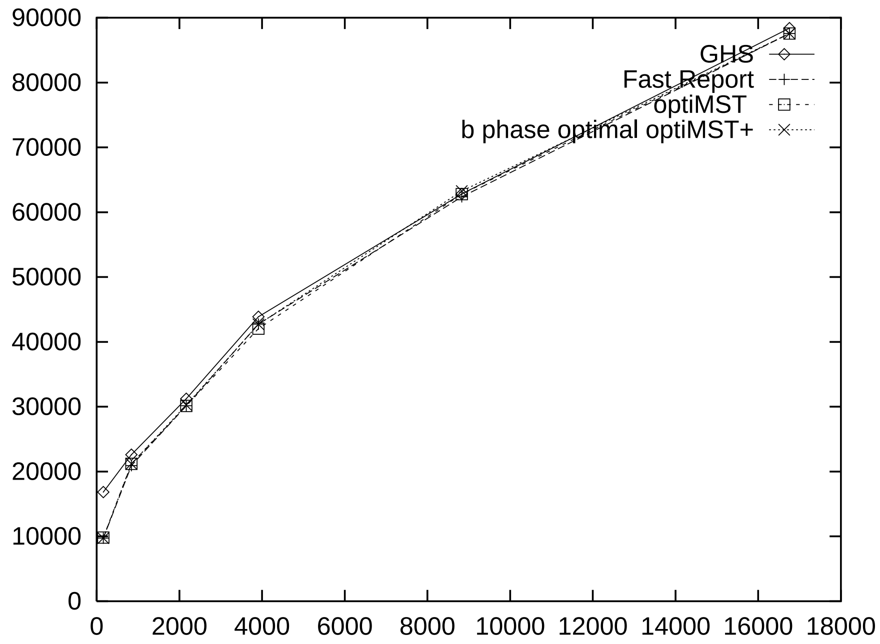

Figure 6.3: Time versus edges for A family graphs

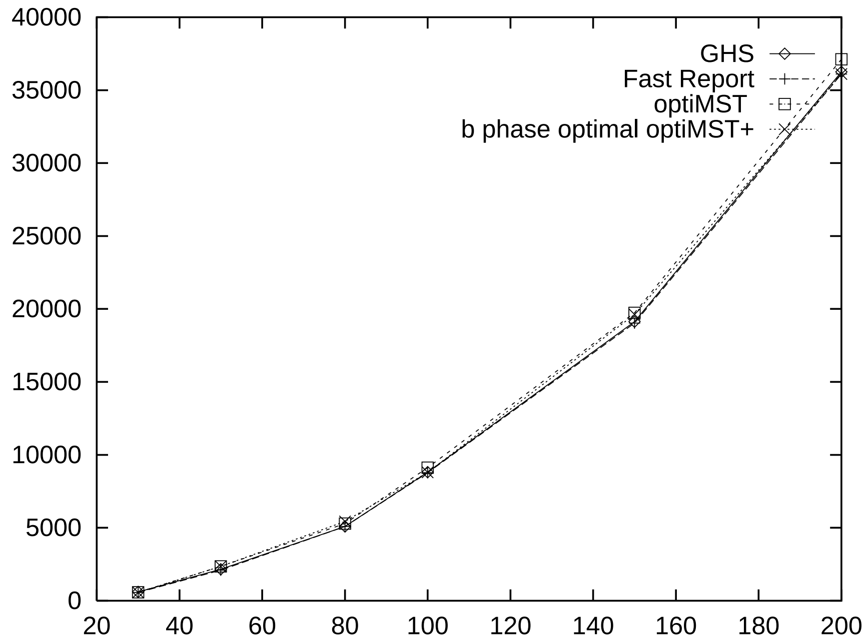

Figure 6.4: Messages versus nodes for A family graphs

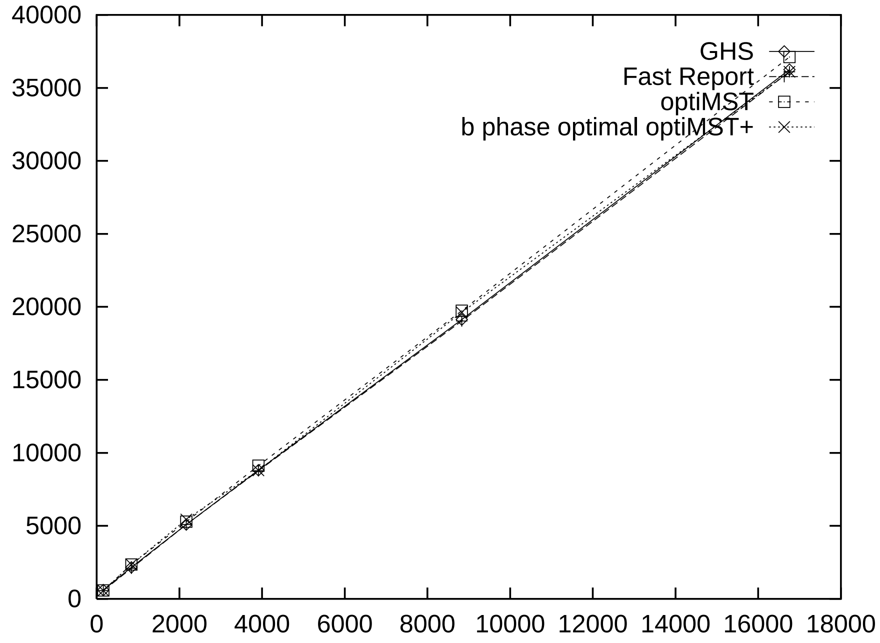

Figure 6.5: Messages versus edges for A family graphs

number of messages (Fig.6.5).

We can also notice that optiMST, although not optimal in communication, exhibits only small differences in its message complexity compared to that of the communication optimal algorithms as the size of the graph increases (Fig.6.4, Fig.6.5).

The conclusion is that for random graphs, all algorithms have comparable performance.

6.2 Paths

Recall that in our discussion of complexity, we used as a pathological example a graph that consists of a line of nodes with increasing weights of edges between them. The only exception is the weights of four nodes at one end of the line that have such weights that they form a fragment. The rest of the nodes submit to the neighbor that does not submits to them, and thus create a huge line of submitted fragments. The 4-node fragment has to wait for the tail of the chain to increase to at least the four-node level (L=2), before it could submit. As we will see next, STS makes a difference in this case.

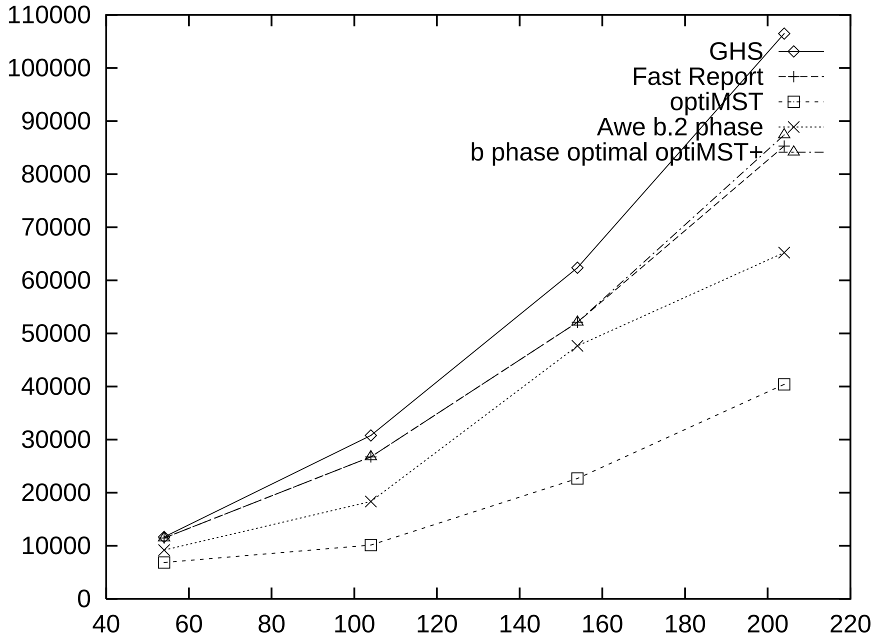

Figure 6.6: Time versus nodes for paths

Notice that the edges connecting the neighboring nodes were the only edges. However, we could add more edges if their weights are such that they don't ruin the above order of submissions.

6.2.1 Time Performance

The gain in time performance is obvious (Fig.6.6, Fig.6.7). We can see that optiMST outperforms the rest. What is surprising though is that the Fast Report algorithm achieved the performance of the MST phase of optiMST+. It is very interesting to note that in the optiMST+, the STS procedure never succeeded, i.e., even this configuration was not "bad" enough to demand the activation of STS. This for us, is an indication of the great effect of the Fast Report technique.

The other surprising observation, in both these figures, is that [GHS 83] performance stays within a reasonable distance from that of the optimal optiMST+. Especially if we think that we will add the Counting Phase termination time to the time of the b phase of optiMST+ (for the complete run-time of the optimal optiMST+) we can expect the optimal and [GHS 83] performances to come really close. Note that

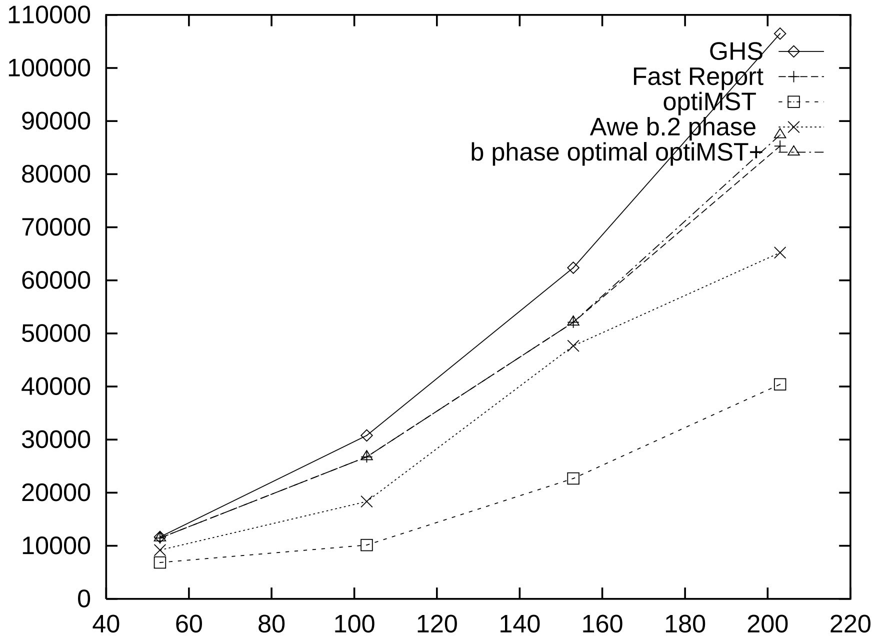

Figure 6.7: Time versus edges for paths

the difference between [GHS 83] and optiMST+ is much smaller than that between optiMST and optiMST+ which are both optimal. Furthermore, the behavior of all algorithms seems to differ by just a constant (see Fig.6.14, Fig6.14 in the section with the section Paths and Constant Delay).

Notice that there is a relative increase in the time in the second sub-phase of [Awe 87] for the 154 node graph. This irregularity seems to be due to the failure Test Distance that we have seen theoretically (see chapter of Correctness of the previous algorithms).

Finally we can observe that the second sub-phase of the MST phase of [Awe 87] is a little bit slower than the optiMST algorithm.

6.2.2 Number of Messages

In Fig.6.8 and Fig.6.9, we can see the trade off between the time and communication performance. The algorithm that achieved best running time, paid for it with an increasing communication complexity. This increased complexity follows the square of N: $O(N^2)$. In practice it actually turns out to be:

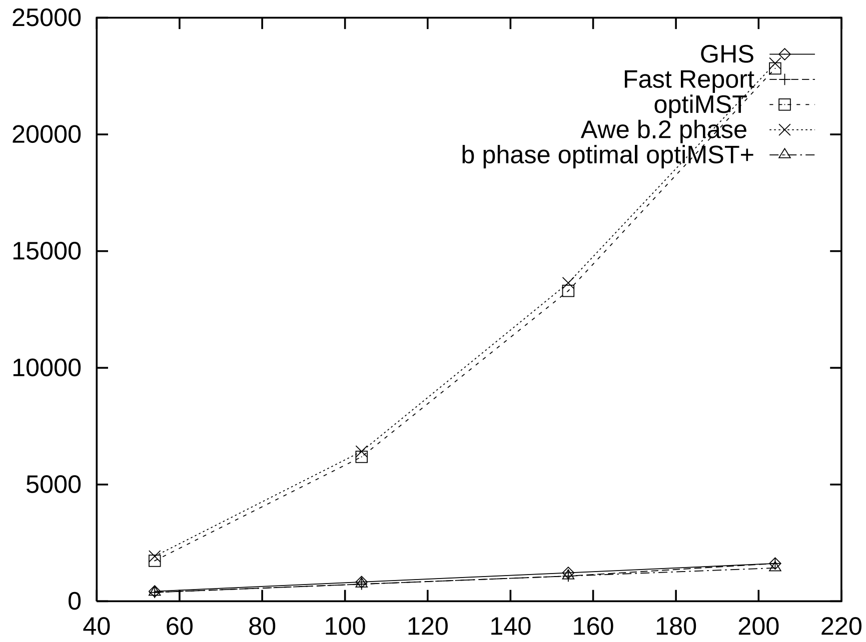

Figure 6.8: Messages versus nodes for paths

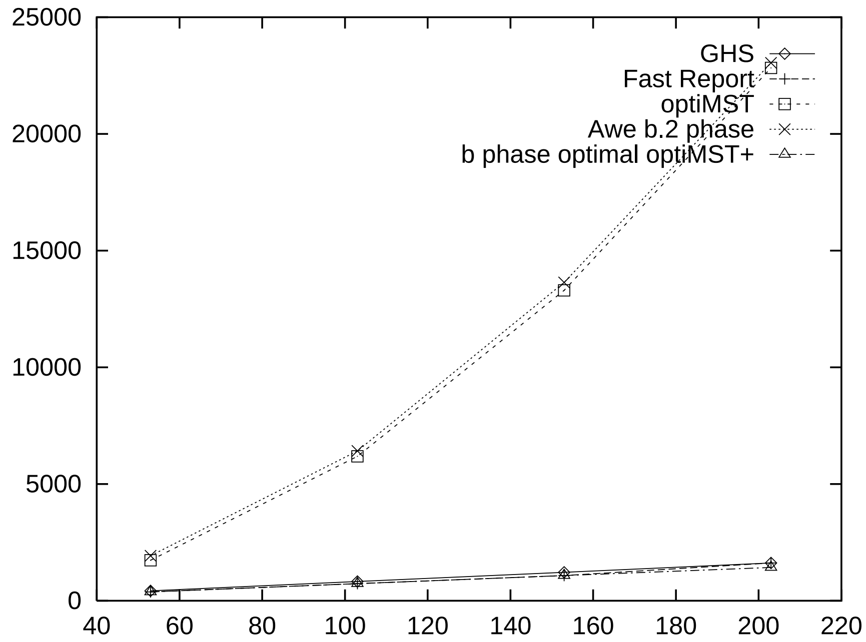

Figure 6.9: Messages versus edges for paths

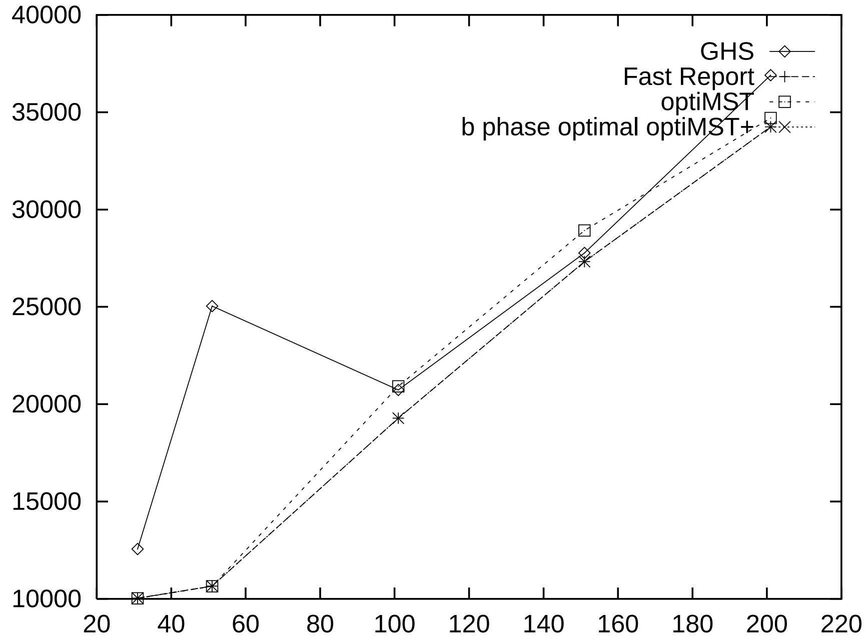

Figure 6.10: Time versus nodes for B family graphs

$messages = \frac{1}{2} \cdot N^2$

The linear relationship between the messages of the other algorithms and the nodes is not a surprise because as we said for this family of graphs the number of nodes is almost equal to the number of edges.

The linearity between messages and edges is theoretically expected as we said.

6.3 Sparse Random Graphs (Family B)

This family has a constant 0.3 probability of having two nodes connected. The procedure to create these graphs is similar to that of the family A graphs. In Fig.6.1 we present the relation of the edges and the nodes of the families of graphs we have seen so far.

6.3.1 Time Performance

From a time performance point of view, the behavior of all algorithms is quite similar (Fig.6.10, Fig.6.11). Fast Report, and the b phase of optiMST+ seem to do a little better than the others. What is really worth noticing is the "unstable" behavior of

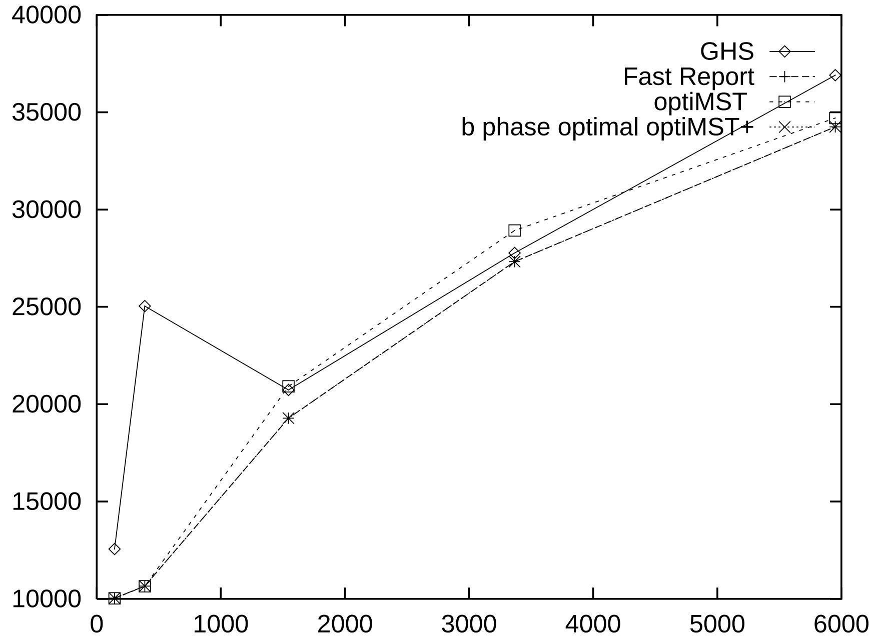

Figure 6.11: Time versus edges for B family graphs

[GHS 83] algorithm. Our experience from the experiments also indicates that the techniques employed by the different algorithms on top of the [GHS 83] backbone, seem to have a stabilizing effect in the behavior. Note that optiMST seems to be in most of the cases slower than Fast Report, which shows that in this family some of the RTS, RTD techniques delay the procedure.

6.3.2 Number of Messages

In Fig.6.12 and Fig.6.13, we see again that optiMST uses more messages, although in this case the difference is negligible. Notice that the rest of the algorithms have almost identical message complexity.

6.4 Paths and Constant Delay

For this family of experiments, we made the time delay on every edge constant.

This expresses the idea that the actual time delay of a message transmission is not necessarily connected with the weight of the edge. Although, this is a very natural approach, the weight can very well represent the cost in money or in some other

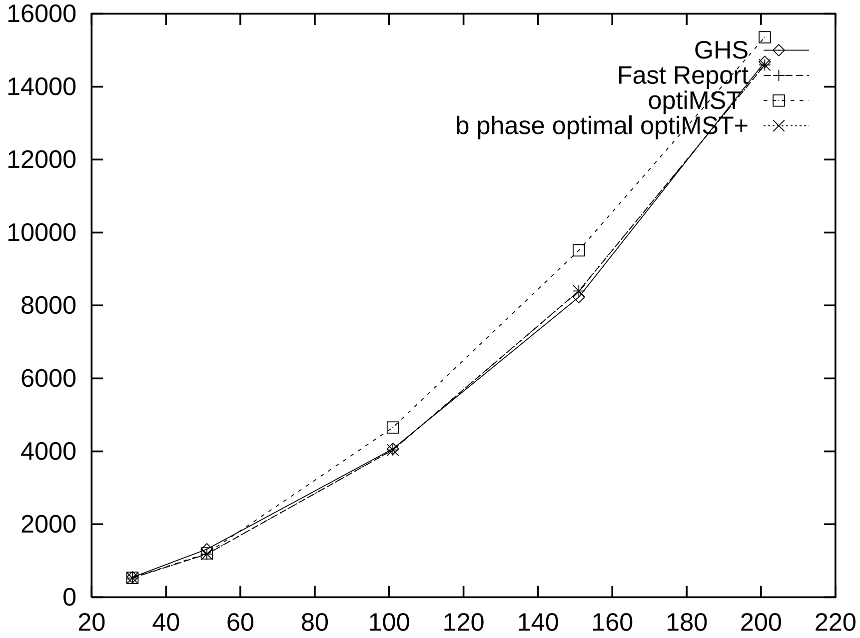

Figure 6.12: Messages versus nodes for B family graphs

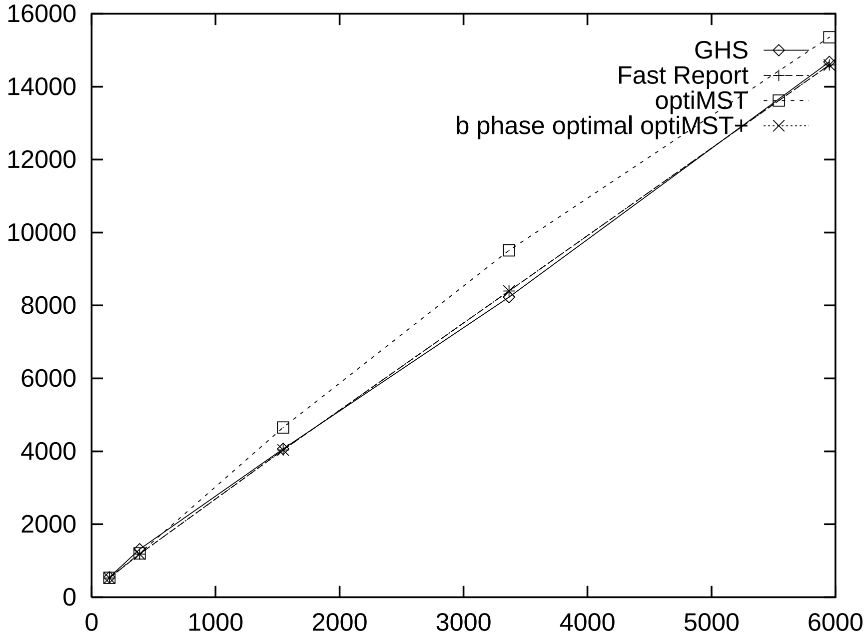

Figure 6.13: Messages versus edges for B family graphs

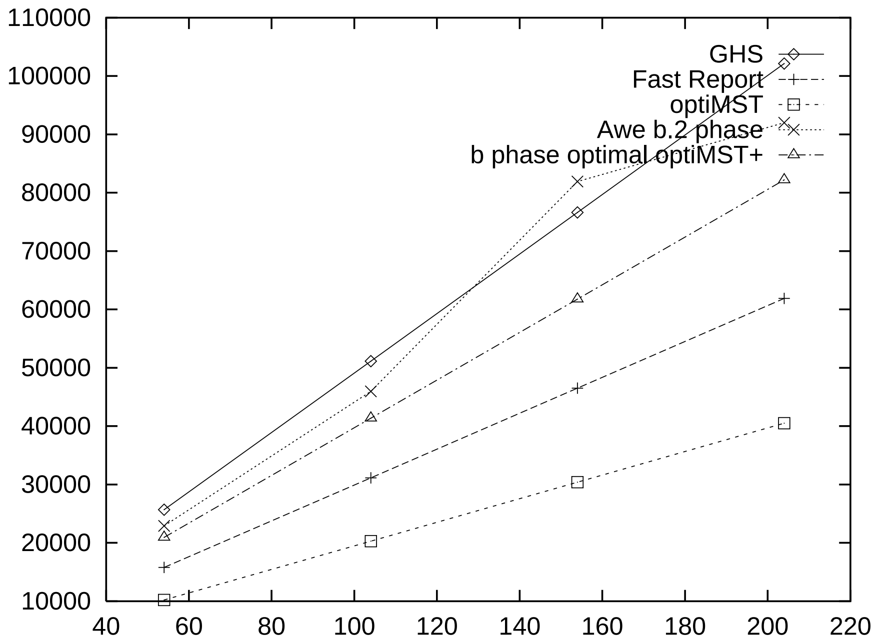

Figure 6.14: Time versus nodes for paths with constant delay

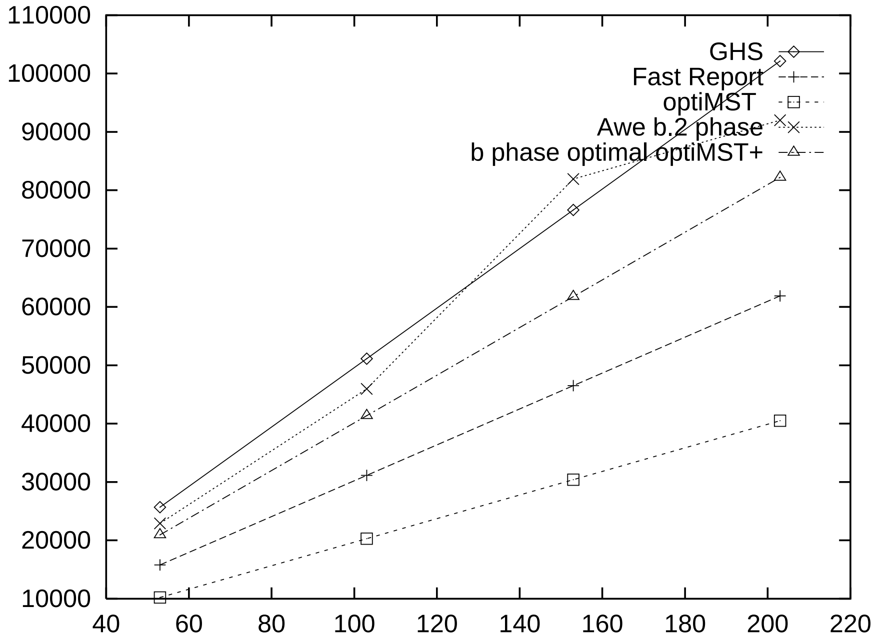

Figure 6.15: Time versus edges for paths with constant delay

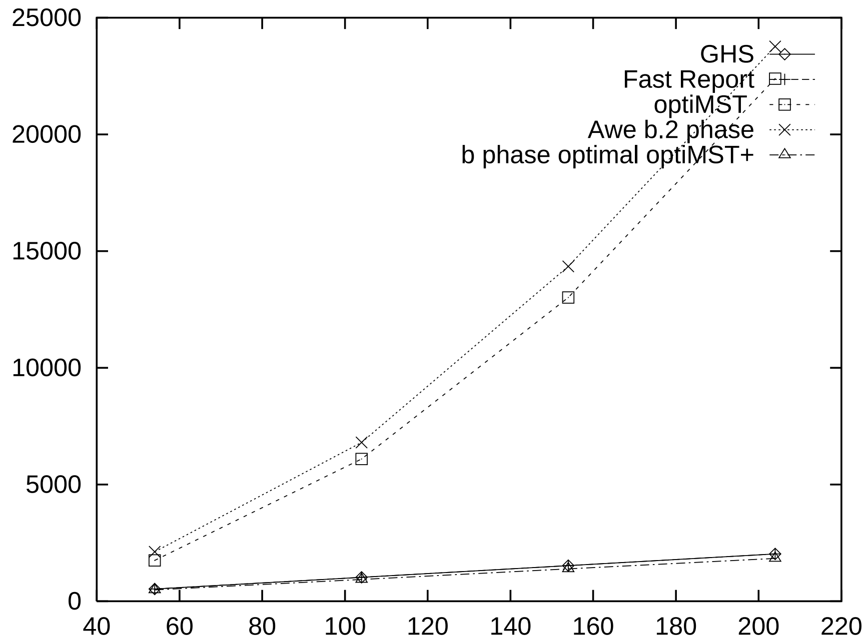

Figure 6.16: Messages versus nodes for paths with constant delay

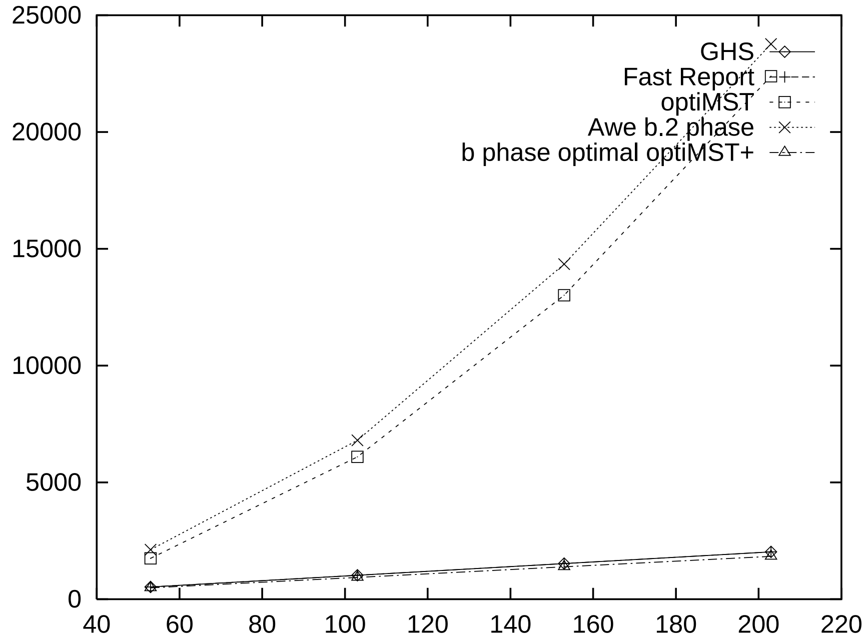

Figure 6.17: Messages versus edges for paths with constant delay

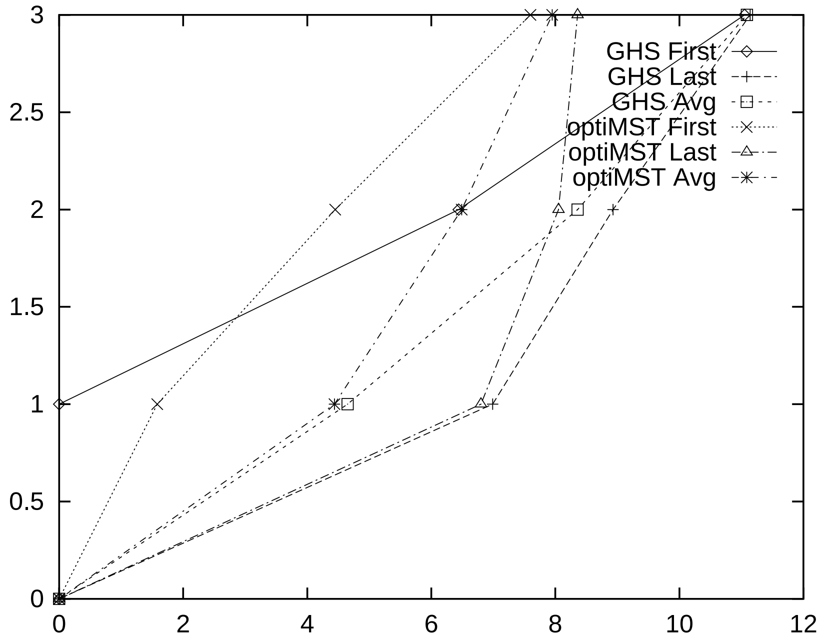

Figure 6.18: The 100 node A graph. Level versus the logarithm (base 2) of time.

resource.

Notice that optiMST once again outperforms the rest with respect to termination time (Fig.6.14 and Fig.6.15) but has a dramatic ($O(N^2)$) increase in the number of messages (Fig.6.16 and Fig.6.17). What is worth noting is that Fast Report outperforms the optimal optiMST+ (b phase).

A very nice achievement of this experiment is that it shows the linearity of time. This happens because of the constant weights, since in the time analysis we also assumed equal delay in the delivery of messages.

Note once more the irregularity for the 154-node example for the second sub phase of [Awe 87]. The explanation for it, must be again the Test Distance failure.

6.5 Level Statistics

In our effort to understand the way the algorithms progress, we plotted the time at which the different levels are reached (Fig.6.18, Fig.6.19). We thought that apart from the first and last node that reach a given level, it would be of interest to plot the average time of reaching a level.

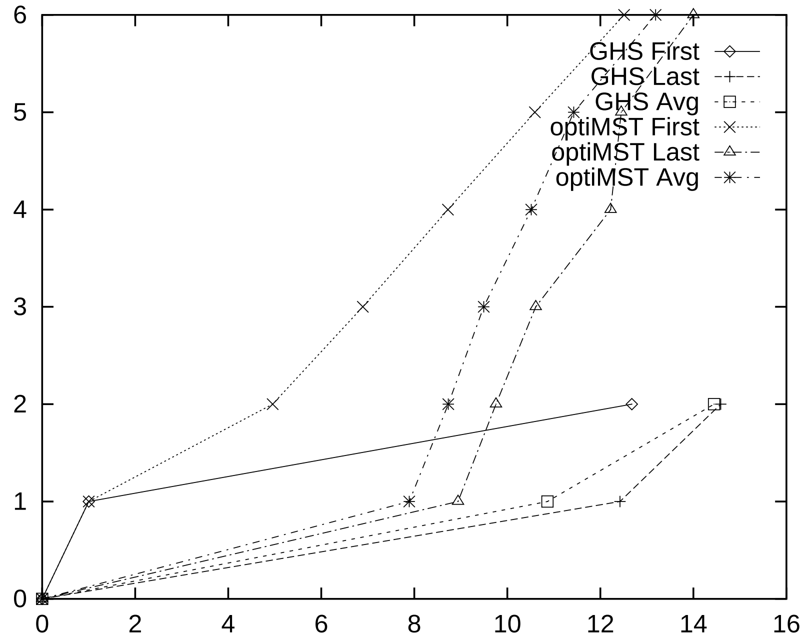

Figure 6.19: The 104 node path graph. Level versus the logarithm (base 2) of time.

In order to have all the nodes reach the same level, we introduced a terminating procedure, during which the root broadcasts a special *terminate* message and makes all nodes aware of the final fragment identifier and level. Notice that this is not necessary, because one could argue that this terminate message could be some kind of useful message for the application that requires the MST. Thus it is not irrational to have the algorithm terminate when the root becomes aware of the termination, as we had before.

The figures (Fig.6.18, Fig.6.19) relate the first, last and average time (measured in time units) that each level was reached with the level. The x-axis in the figures is the logarithm of the measured time, and the y-axis is the levels the nodes reach. The best way to read the map, is to compare (horizontally) the different times at each level.

In both algorithms the latest level increases are much slower than the first ones. This is something that we expected, since our complexity proof bounded the time spent with L as the minimum level to $O(2^L)$.

We can also notice that in [GHS 83], it takes more time to reach the same level, since this algorithm does level increases only in equi-joins and makes no effort to have L track the size of the fragment. Notice also that in both examples the level behavior of [GHS 83] is worse. However, in the A graph, this does not affect the final time performance as we have already seen. The reason for this is that the graph is so dense that the termination delay is determined by the rejection of a great number of heavy weight edges in the last Finding of the algorithm. All nodes arrive in the final level at a relatively early (2200 units for [GHS 83] and 500 units for optiMST) and it takes approximately $40,000$ time units for the algorithm to terminate. Notice that this delay can be roughly analyzed as follows: there are 40 edges to be rejected per node, and each rejection requires two messages and each message delivery takes 500 time units in average (since it is a uniform distribution from $1-1000$).

More specifically for the path family example, the absence of RTD or RTS does not allow the update of the level of the root as the size of fragment (time) increases. This has the effect that the root of a huge fragment has very small level, smaller than the group of four nodes at the end of the chain. This way, it is the big size fragment that ends up submitting to the small size fragment, and the delay of the propagation of this message is a major factor for the final termination difference.

In addition, especially in the path graph, the figure of optiMST shows a linear nature which comes to agree with the argument we used in the optimality proof.

6.6 Observations

First, the Fast Report algorithm was very successful in the sense that it achieved optimal behavior for both the A graphs and paths. This of course does not mean that it is optimal, but just that the Fast Report trick can really make a great difference in practical situations. However, we consider this as a triumph of our distributed information approach.

Second, all algorithms perform equally well on the random graphs we used. In the family of paths also, the termination times of the algorithms differed only by a constant. In the random graphs, the fact that STS didn't have to succeed is an indication that these graphs didn't have configurations that would lead to long chains of fragments. Therefore there were no situations were the time optimality was threatened and thus communication optimal algorithms had termination times equal to those of the time optimal algorithms. The complementary observation to that is that when STS does not become active enough to succeed, time optimal algorithms used numbers of messages comparable with those of the communication optimal algorithms. Another way to see this is that the techniques are there in case we come across some arbitrarily "bad" configuration, but they may not be necessary for some categories of graphs.

Third, it is not clear to us that the time performance of [GHS 83] in would be non-linear in practice. In (Fig.6.6, Fig.6.7), we see that the differences of the times of all algorithms (optiMST and optiMST+ are optimal with respect to time) an within a constant. Theoretically, there may exist a graph that may demand non linear time (see [CT 85]). In other words it is not clear to us how we can implement a bad configuration where we would observe non linear termination times. However, we think that the difficulty of defining and constructing such a bad configuration could indicate that it is not something that practical applications should worry about. Note also that bad cases usually assume arbitrary propagation or processing delays of some messages, which are not usually met in practice. This is an idea that we think should be examined further. It could lead to something as important as a better approach towards time performance for distributed algorithms, or it could just point out the difference between the theoretical analysis and the performance in practice.

Fourth, the level behavior of the algorithms seems as expected, in the plots we were able to make.

Chapter 7

Epilogue

In this thesis, we presented a new approach to distributed algorithms for MST. The innovation is that information percolates and resides in nodes. We suggest two kinds of distributed information: first a lower bound of the MOE that can be achieved and second, the size-distance from the root. We presented several ways in which we can take advantage of this information to either avoid unnecessary delay in the Report procedure (Fast Report, Tentative), in deciding for the next node to join (Local Decision to Join). We also proposed a way subfragments can increase their level using the distributed size-distance information and incorporated this and a few other techniques in a time optimal algorithm that we called optiMST.

Apart from reviewing previous algorithms, we corrected and clarified several points in [Awe 87]. We showed that the algorithm could not work, as described, and we proposed solutions.

We finally generalised the structure of [Awe 87] and show how one can make optimal algorithms using non-optimal ones. We also proposed an optimal algorithm optiMST+ that uses this structure and algorithms of our approach in the MST phase.

We also provided experimental results that showed the performance of different algorithms, basically the ones we introduce. For random graphs, the performance was very similar for all algorithms. The surprise was that the Fast Report algorithm had exceptional behavior in the families of graphs we tried. It is worth noting that one of the families was a long chain configuration.

At this point, we have to make the following observation that occured to us suddenly after the long struggle for the optimal time. From a practical point of view, a time complexity of $O(N \cdot log(N))$ is not so much worse than $O(N)$ that one would not stop to consider aspects of simplicity of implementation and debugging. For a network with as many as $1024 = 2^{10}$ nodes, the time difference is ten times greater. We should also mention that this is only a worst case comparison, while in practice the common network configurations may change things dramatically in either direction. For example as we saw in our results, the performance for a whole family of graphs was identical. For a practical application and prior to any implementation, the most probable graph configurations must be considered, since implementing the theoretically optimal algorithm may prove to be less efficient. As we said in the introduction, the relative cost between time and communication should also be considered.

Future Work. *THEORY.* We expect that the Distributed Information mentality could be used in other distributed algorithms. It would not be a surprise if it could reduce the complexity of such algorithms.

Our ambition was, and still is, to find a technique to have an optimal algorithm without three phases, i.e., without having to follow the optimal structure algorithm we proposed.

PRACTICE. Many more experiments can be performed. We strongly believe that there is a lot we can learn through such simulations about both the way these algorithms work and about the nature of the problem. We also expect that the results will continue to surprise us.

A problem that we faced trying to simulate the algorithms was the creation of graphs that represent the salient feature of realistic network configurations. It seems to be of great importance, not only for this case, but for distributed algorithms and computer networks, in general.

References

[AfP 90]: F. Afrati, G. Papageorgiou, "Graph Algorithms", National Technical University of Athens Press, 1990.

[AG 85]: B. Awerbuch, R. Gallager, "Distributed Breadth-First-Search Algorithms", IEEE Symposium on Foundations of Conference Science, October 1985, Portland, Oregon.

[AM 86]: B. Awerbuch, S. Michali, "Dynamic Deadlock Resolution Protocols", Proceedings of 1986 FOCS Conference, Toronto, Ontario, October 1985.

[Awe 87]: B. Awerbuch, "Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and related problems", Proc. 19th Symp. on Theory of Computing, May 1987, pp. 230-240.

[AZ 89]: Mohan Ahuja and Yahui Zhu, "Distributed algorithm for minimum weight spanning trees based on echo algorithms", 9th International Conference on Distributed Computing Systems, publ. by IEEE, Conf. No. 12464, 1989.

[CT 85]: F. Chin and H.F. Ting, "An almost linear time and $O(V logV + E)$ messages Distributed Algorithm for Minimum Weight Spanning Trees", Proceedings of 1985 FOCS Conference, Portland, Oregon, October 1985.

[Gaf 85]: Eli Gafni, "Improvements in the time complexity of two message-optimal election algorithms", Proceedings of 1985 PODC Conference, Minacki, Ontario, August 1985.

[Gbu 93]: P. Gburnyski "The SMURPH Protocol Modeling Environmnet, ver. 1.71", 1993 (obtained by anonymous ftp from the University of Alberta).

[GHS 83]: R.G. Gallager, P.A. Humblet , P.M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees", ACM Trans. on Programming Languages and Systems, Vol.5, No.1 January 1983, pp 66-77.

[GKP 93]: J.A. Garay, S. Kutten, D. Peleg, "A Sub-Linear Time Distributed Algorithm for Minimum-Weight Spanning Trees", 34, FOCS 1993.

[GR 91]: P. Gburnyski, P. Rudnicki, "The SMURPH Protocol Modeling Environmnet, ver. 1.12", 1991 (obtained by anonymous ftp from the University of Alberta).

[MMP 81]: D.A. Menasce, R. Muntz, J. Popek "A locking Protocol for Resource Coordinator in Distributed Databases", Proceedings in ACM SIGMOD, June 1981.

[Pri 57]: R. C. Prim, "Shortest connection networks and some generalisations", Bell Syst. Tech. J. 36, 1957.