# Creating Optimal Distributed Algorithms for Minimum Spanning Trees

Michalis Faloutsos and Mart Molle
University of Toronto

May 9, 1995

**Abstract**

This paper examines the complexity of distributed algorithms for finding a Minimum Spanning Tree in undirected graphs; the goal is to create algorithms optimal with respect to both communication $O(E + N \log N)$ and time $O(N)$, where $E, N$ is the number of edges and nodes respectively. A fundamental bad case that leads to non-optimal performance and the proposed techniques to overcome this problem are presented. We introduce new techniques based on the the idea that we call Distributed Information; nodes store information that summarizes properties of groups of nodes. The techniques (and the corresponding algorithms) are classified in communication optimal and time optimal ones. Finally, the structure of the algorithm proposed in [Awe87] and the above classification can lead to a pattern for creating optimal algorithms. In addition, a simple $O(E)$ messages and $O(N)$ time algorithm for counting the nodes of the network is introduced; It can be used as part of the optimal algorithm or it may be of independent interest.

## 1 Introduction

Given a weighted undirected graph $G$ with $N$ nodes and $E$ edges, we want to find a spanning tree for which the combined weight of all its edges is minimized, denoted an **MST** for the rest of this paper. Furthermore, we want to use a distributed algorithm to find this MST by placing a processor at each node and treating each edge as a bidirectional and error-free communication channel, over which the nodes can exchange messages. We assume that initially, all nodes are "awake" but that none has any special status (nor are any aware of the network topology except for their adjacent edges). Thus, we cannot simply send all the information about $G$ to a distinguished node that solves the problem and broadcasts the answer to the rest of the graph. Fortunately, our task is made easier because it is well known that the MST problem can be solved by a "greedy" algorithm, which can generate an optimal solution without backtracking.

For the general graph, the distributed MST problem requires at least $\Omega^1(E + N\log(N))$ messages, where we count the transmission of one message across one edge as the unit of "cost". It is known that the time complexity is at least $\Omega(N)$ where we assume that each message delivery along an edge takes one time unit. Note also that the length of the messages has to be at least $O(\log N)$ bits in order to carry the identifiers of nodes.

[1] With $\Omega$ we denote the lower bound of the asymptotic complexity.

## 1.1 Our Contribution and Other Approaches

The main goal of this paper is to study the difficulty of creating optimal algorithms for the MST problem and present new algorithms and techniques to solve the problem.

All the techniques and the algorithms discussed here are based on the algorithm of Gallager, Humblet and Spira [GHS83] which we refer to as the **basic algorithm**. The difficulty of achieving an optimal algorithm is studied through a bad case example. We examine the way the techniques of different algorithms behave in this example and we express the complexity in terms of number of distinct fragments and their maximum size; these can be considered as more accurate measures of the state of progress of the algorithm. The discussion concludes in a pattern introduced in [Awe87], for creating optimal algorithms using non-optimal ones.

An idea, that we call Distributed Information, is introduced in which nodes exchange and store information that summarizes properties of parts of the network. This information can be used to increase the efficiency of the algorithms. It is noteworthy that the only cost is a trivial increase first in the length of the messages, and second, in the storage requirements at each node.

We also present an algorithm that abandons a fundamental principle of the basic algorithm. Although its communication complexity can be as bad as $O(EN)$, the algorithm is time optimal, and simple. We believe that it could be quite competitive in practice, especially if communication complexity is not of primary importance.

A simple node counting algorithm is described which has $O(E)$ communication and $O(N)$ time complexity with very small constants. The algorithm constructs a spanning tree and thus slightly modified it can solve the leader election problem with similar complexity.

Simulations and experiments were conducted with different families of graphs and with variable network size. The results showed an increase in the efficiency that often exceeded our expectations (see [Fal95]). However, due to space limitations, it seemed impossible to include any of this information in this paper.

It should be noted that algorithms with different approaches compared to [GHS83] exist; first we can mention [AZ89] where messages accumulate graph information ($O(N \log N)$ bits per message are required) and the algorithm achieves $O(E + N \log N)$ message complexity and $O(D \log N)$ time units with small constants ($D$ is the diameter of the network); second we can mention [GKP93] and its improved version [KP95], where communication complexity is ignored, dominating sets and edge elimination techniques are used, and the algorithms achieve $O(D + N^{0.614})$ and $O(N^{1/2}G(N) + D)^2$ respectively.

The structure of this paper is as follows. In the rest of this section, the basic [GHS83] algorithm and a bad case example are described. Section 2 lists the innovations introduced by previous algorithms. Section 3 presents new techniques and algorithms for the MST and counting problem. Finally in section 4, we group all the techniques and examine the structure that can create an optimal algorithm using non optimal ones.

## 1.2 Basic Algorithm of Gallager, Humblet and Spira

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

In this basic algorithm, each node is initially the **root** of its own **fragment** (a trivial connected subgraph of the MST) and all the edges are **Unlabeled**. Thereafter, adjacent fragments join to

[2] $G(N)$ is the number of times that the *log* function must be applied to $N$ to get a result less than or equal to 1.

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

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

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

Another noteworthy feature involves reducing the required number of messages by having a node *delay its response* to any *test* message arriving from a fragment with a higher level than its own, since that other fragment will not be allowed to initiate a join with a fragment lower level.

It can be shown (e.g., [GHS83]) that the message complexity of this algorithm is $O(E + N\log(N))$, and hence optimal. However, its time complexity is $O(N\log(N))$ and hence not optimal.

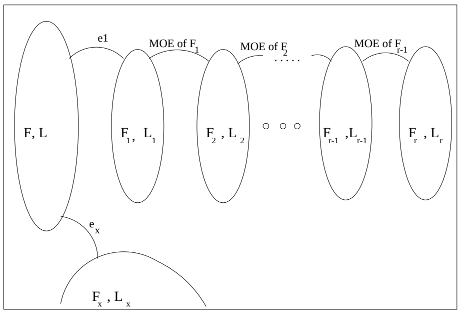


Figure 1: Bad case example.

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

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

# 2 Previous Techniques

Several techniques were proposed to improve the performance of the basic algorithm. These techniques will be presented without repeating features that do not change.

## 2.1 The Size Estimating Improvement

The technique which we will call **Root Size**[3] was introduced in [CT85] [Gaf85] and tries to make 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 the size of the fragment; it may have many more nodes than $2^L$, if it has accepted a lot of submissions. The modified algorithm demands that

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

The nodes can be counted trivially in the Reporting procedure, where the number of nodes is accumulated in a counter of the report message. 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. If not, the level is increased accordingly and the procedure of finding the MOE is repeated.

A complementary idea that accompanies the above, concerns a change in the Joining policy in order to make better use of the size estimating procedure. Assume that fragment $(F_1, L_1)$ submits to $(F_2, L_2)$. If $F_1$ does not participate in the current Finding procedure of $F_2$, then it will initiate a size counting procedure (Reporting without looking for a MOE). If $F_1$ finds that its size can justify a level higher than $L_2$ then it refuses its submission to $F_2$ and begins a new Finding procedure.

The termination time of the algorithm is reduced to $O(NG(N))$. Intuitively, we can observe that the Joining policy expects small size fragments to submit to bigger ones. Keeping the level of the fragment closely related to its size, helps in the enforcement of this policy ([CT85], p 258). Notice that the communication complexity remains $O(E + N \log N)$.

[3]Notice that all the two word names of the techniques follow the pattern: the first word denotes who is carrying out the procedure and the second one what it is estimating. We think that this convention makes things much clearer.

### 2.2 Estimating the Distance

Two techniques were introduced in [Awe87] and modified in its revised version [FM95]. The more complex structure of the two algorithms will be discussed in the next section. First, the **Root Distance** technique (called Root Update in [Awe87]) ensures that the root of a level $L$ fragment will never be blocked for longer than $O(2^{L+1})$, waiting for the Finding and Reporting procedure to complete. *Initiate* messages have a counter initialized to $2^{L+1}$, where $L$ is the level of the root. This counter is decreased at each hop, as if counting the distance from the root in reverse. When this counter becomes zero, a message is sent back to the root, which increases its level to $L+1$ and restarts the Finding procedure. This technique does not affect the communication complexity since the messages that return to the root are at most equal to the number of the *initiate* messages.

The second technique is the **Leader Distance** technique[4]. This technique helps a submitted fragment increase its level "fast enough", even if it is far away from the root. In more detail, the technique allows a submitted fragment currently at level $L$ to increase its level within time $O(2^{L+1})$, without waiting for a message from a possibly distant root. Analogously to the Root Distance technique, the leader of the submitted fragment sends a *testDistance* message, towards the root. If the message visits more than $2 \cdot 2^{L+1}$ nodes[5] or a node of greater level, say $L_{new}$, then a message is sent back to the leader which increases the level of its fragment to $L+1$ or $L_{new}$ respectively and repeats the procedure. When the *testDistance* message crosses with an *initiate* message, the *testDistance* message is discarded and the procedure terminates. In addition, nodes that receive a *testDistance* message of a higher level are also allowed to increase their level. Several issues have to be taken care of in order to guarantee efficiency and correctness (see [FM95] for details).

The symmetry of the two procedures is apparent; they explore the same path from different ends.

The second technique can increase the communication complexity to $O(E+N^2)$ in the following bad cases. First, assume that we have $I$ distinct fragments and the maximum fragment size is $MaxSize$, then we can have $I$ active procedures and each procedure can explore the greatest fragment using $O(MaxSize)$ exploration messages. The total number of messages is $O(IMaxSize)$, and for $N/2$ single nodes and a great fragment of $N/2$ nodes, this becomes $O(N^2)$. Second, recall our bad case example (see Fig.1) and assume that the number of fragments in the chain is $O(I)$. We have $O(I)$ submissions that lead to $O(I)$ activations of the procedure and each procedure can send messages along a path of length $O(I)$. This will lead to $O(I^2)$ messages and $I$ can be $O(N)$.

At the same time, the same technique improves the time performance. In our bad case example, fragment $F_1$ will be updated using the Leader Distance procedure, and will reach $L$ "faster" ($O(2^{L+1})$), and thus $F$ will have an answer to its *test* message sooner. The latter guarantees an upper bound for the delay of its Report procedure. In general we can guarantee that nodes reach $L$ in $O(2^L)$ time which leads to a termination time of $O(N)$.

## 3 Distributed Information and New Algorithms

Storing information can improve the efficiency of the algorithms and this was not considered in previous algorithms. In the rest of this section, we show techniques in which nodes exchange, store and use summary of information concerning other parts of the network and we examine their impact on complexity. Namely, we have two types of stored information. The first type of information concerns the previously reported MOEs and facilitates the Reporting procedure. The second type of

[4] the **Test Distance** procedure in [Awe87] differs in some crucial points

[5] For an explanation of factor 2 see [FM95]

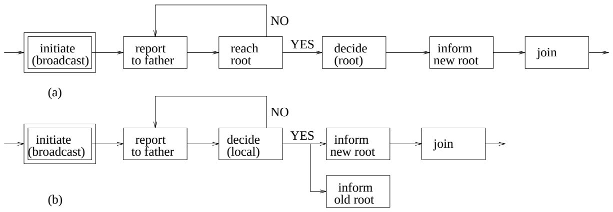


Figure 2: The difference in procedure: (a) [GHS83] and (b) Local Decision to Join.

information is some measure of the distance from the root and helps submitted fragments increase their level by giving them a measure of their distance from the root.

In addition, a simple node counting algorithm is described; its complexity is bounded by $2E$ messages and $2N$ time units.

### 3.1 Fast Report using Pruning

Each node can store the MOE that exists (or existed at some point) at the sub-tree towards each of its Branches and we will call this information **edge-MOE**. Notice that the two nodes adjacent to a Branch edge store an edge-MOE for that edge, whose value refers to the part of the fragment on the opposite side of the edge. Every message carries the information of the MOE that exists from the sender's side of the edge, to the best of its knowledge. This MOE information will update the edge-MOE of the corresponding edge of the receiver node. For example, a node that receives a *report* message will store the reported MOE as the edge-MOE of that edge.

It is important to notice that under the Joining policy introduced in [GHS83], the weight of the MOE reachable via a branch can only increase over time. In other words, **edge-MOEs are non-decreasing in time** and thus **an edge-MOE, possibly a previous MOE, is a lower bound of the current MOE**. These two observations are fundamental for the correctness of this approach.

The edge-MOE information could reduce the waiting time of the Report procedure, since in some cases it can be guaranteed that the MOEs to be reported to a node by some of its children will not be better than an already reported MOE from a child. For example, if branch "a" has reported an MOE of weight $W$, and branch "b" has not reported its MOE yet but its edge-MOE (lower bound) is greater than $W$, it is not necessary to wait for a report from "b", since this will be greater than $W$. In other words, we see the problem of MOE selection by each fragment as equivalent to searching some kind of game tree: "what is the next "move" I should select in order to minimize my accumulated penalty function?"

Furthermore, we can eliminate sub-trees whose last report indicated a MOE of infinite weight, since it is clear that there is no need to visit these nodes in the Finding procedure. Although this sounds trivial in practice, it speeds up the final time consuming Finding procedures and reduces the termination time considerably.

This technique which we call **Fast Report** can be incorporated in [GHS83] and give rise to the algorithm that we call the **Fast Report algorithm** with identical asymptotic complexity (see [Fal95] fro more details). Notice however, that this technique may affect the counting of the size of the fragment since it may complete the Reporting procedure without waiting for all the *report* messages.

### 3.2 Local Decision to Join - AutonoMST

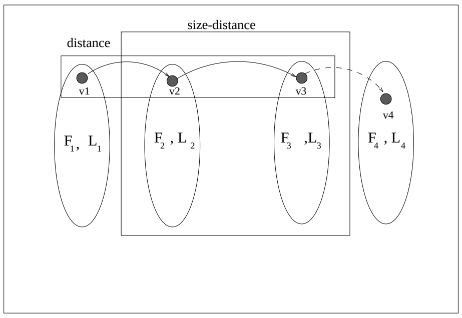


Figure 3: Size-distance and simple distance.

The edge-MOE information can be used even further. Nodes can decide locally whether the MOE of the fragment has been found in their subtree. A certain report at a node may indicate a MOE whose weight is lower than all the other edge-MOEs including that of the parent edge. Thus the node will know that this is the next MOE for the whole fragment. The idea is to give authority to that node to relieve the old root of its duties and to nominate the leader that will proceed to join. (see Fig.2).

Note that the decision is taken wherever this is possible, and at the worst case at the root. In other words, nodes have the ability to take advantage of a "good" configuration. Obviously, in non-favourable configurations this algorithm performs as [GHS83], while in favourable configurations it is faster. However, the worst case complexity analysis of the two algorithms is identical.

The conceptual differences of this algorithm compared to the [GHS83] algorithm are the following. First the nodes can function autonomously and asynchronously, i.e., we do not have the strict phases Find-Report-Join but rather a more aggressive "join when you can" approach. Second, the place of the decision is not known beforehand, and the nodes themselves examine the possibility of deciding. Also, this algorithm achieves a second degree of distribution - but not of parallelism - within each fragment. As each fragment's decision is independent and can be based on "local" data with respect to the graph, a node's decision is autonomous and can 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 centralised decisions, i.e., when we expect the root to increase the level and repeat the Finding procedure. We call **autonoMST** the algorithm that uses both this and the Fast Report technique.

### 3.3 Distributed distance from the root - optiMST

We already saw that it is important for submitted fragments to detect the distance from the root (Leader Distance procedure). Extending the Distributed Information idea, instead of having the leader issue exploration messages we make nodes keep track of their distance from the root. Every time a root changes, the new distance information is broadcasted. Note that this way, when a fragment submits, it becomes aware of an estimate of the distance right away.

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 and we will refer to it as **size-distance**. For example in Fig.3, after $F_1$ submits to $F_2$ and $F_2$ submits to $F_3$, the size-distance estimate of $v_1$ must be equal to $Size(F_2) + Size(F_3)$, where $Size()$ indicates the number of nodes of a fragment; however, the simple distance is just $d(v_1, v_3) = 2$.

The are two reasons why we prefer size-distance to simple distance. First, size-distance only increases[6], while distance can decrease too. Second, size-distance is an upper bound of the distance and thus we may have faster level increases.

We can examine the procedure through an example (see Fig.3). When fragment $F_1$ submits to $F_2$, the leader of $F_1$, $v_1$, will broadcast the size-distance from the root of $F_2$. If $F_2$ submits to $F_3$, $v_2$ will broadcast the change to all $F_2$ nodes and $v_1$. Then, as in Leader Distance, $v_1$ will decide whether the size-distance is large enough $(2 \cdot 2^{L+1})$ to justify a level increase for its fragment. In any case, $v_1$ will broadcast the new size-distance information and the possible level increase to the rest of $F_1$.

It is easy to see that this technique that we call **Leader Size**, is equivalent to the Leader Distance technique that we saw before, and leads to optimal time $(O(N))$ and non optimal communication $(O(E+N^2))$ (see the analysis in Leader Distance and [Fal95] for more details). The difference is that the size-distance is estimated by the fragment that accepts a submission and the submitted fragment can benefit from it right away.

We can combine the techniques Fast Report, Root Distance, Root Size and Leader Size in an algorithm that we call **optiMST**[7].

### 3.4 The Tentative approach

Let us recall the bad case example (see Fig.1) and focus again on the problem of $F$ waiting for an answer to its *test* message from $F_1$. As we saw, both Leader Distance and Leader Size techniques try to avoid the delay by increasing the level of for $F_1$ "fast enough" in order to offer an upper bound for the delay of the answer to the *test* message of $F$.

A different way to address the problem is to enable $F$ ignore the absence of response from $F_1$, if the concerned edge is not the MOE of the fragment. To implement this, we modify the Testing policy of the basic algorithm as we will see below, but we do not change the joining policy; a greater level fragment will still not be allowed to join with a lower level one.

In the modified Testing policy nodes answer immediately to a *test* message with a new type of message, ignoring any relative level difference. If the level of the adjacent fragment is less than the current level of the fragment in the Finding procedure the MOE is **Tentative**; if not, the MOE is **Real**. Nodes reporting Tentative MOEs, will report again with an *update* message as soon as the MOE becomes Real or rejected (as an internal edge). In the Reporting procedure, both Real and Tentative MOEs are reported. The only difference is at the root; if the MOE of the fragment is Real then the fragment proceeds to join; if not, the root waits for *update* messages. Notice that an *update* message starts from a node that reported a Tentative MOE and moves towards the root; every node receiving the message updates the corresponding edge-MOE and forwards the minimum of its edge-MOEs only if this is changed by the update. Note that because of the edge-MOEs, the fragment does not have to initiate a new Finding procedure at the same level.

We call the algorithm that uses both the Tentative and the Root Distance technique that we already saw, **Tentative algorithm**. This algorithm is optimal with respect to time but not with respect to communication.

**Communication.** The number of messages can become non-optimal. A node can issue an *update* message for each of its edges and the message may travel towards the root, i.e. up to $Size()$

[6]The size-distance can not decrease in one submission, i.e., in the period where the fragment has submitted and not received an *initiate* message from some root.

[7]The name comes from the fact that the techniques employed in this algorithm are able to exploit "favourable" configurations. Having an optiMSTic attitude, we can hope to come across such configurations.

messages for each edge. Therefore an upper bound for the *update* messages is $O(EN)$ thinking that $O(E)$ edges can initiate an *update* message that will visit $O(N)$ nodes on its way to the root. This bound is tight since we can imagine a $N/2$ size fragment and $N/2$ one-node fragments; the $N/2$ nodes can submit to the big fragment, with each submission initiating $O(N)$ *update* messages that travel towards up to $O(N)$ *update* message to go towards the root. In total $O(N^3)$ messages are required.

**Time.** As stated earlier, a fragment never gets "trapped" waiting at an outgoing edge that is not its Real MOE. In the bad case example (Fig.1), assume that $e_1 > e_x$ (denoting the weights of the edges) and recall that in [GHS83] fragment $(F, L)$ would have to wait for an answer from $(F_1, L_1)$, but this would happen only if $L_1 > L$, which may take a long time. However, in the Tentative algorithm, $F$ will join with $F_x$ relying on the immediate Tentative answer from $F_1$. Thus we don't mind if $F_1$ does not increase its level "fast". For completeness, we can say that, in case $e_1 < e_x$, $F$ would have to wait for $e_1$ to either become Real MOE or for $F_1$ to join with $F$. In the first case, $F$ is compensated for the delay by a proportionally big level increase. In the second case, the other fragment is at most of equal level and thus the delay of the answer is bounded.

In general, it can be proven that the time delay between level increases is linear to the size of the fragments involved and so the termination time is $O(N)$ (for a detailed proof see [Fal95]).

### 3.5 Counting the Nodes of the Network

We present here an algorithm that counts the nodes of the network by constructing a spanning tree. In [Awe87] there is an algorithm that creates a spanning tree with $O(E + N \log N)$ messages and $O(N)$ time units. We assume that we can originally activate one node, we can use the following algorithm. Note that this assumption seems to hold for real applications.

In the beginning, all the nodes are **Sleeping** except for one **Active** node which will be the root of the spanning tree. We have two phases that bear some similarities with the Finding and Reporting procedures of [GHS83]:

*Exploration phase.*The root broadcasts an *activate* message to all adjacent nodes. Every node that receives such a message marks itself Active, considers the sender of the message as parent and broadcasts the *activate* message to its children. Active nodes that receive *activate* messages, report their rejection to the sender node and ignore the *initiate* message.

*Reporting phase.* All nodes report to their parent the number of descendant nodes after receiving the reports from their children, while leaves report immediately. The final sum at the root is the size of the network.

The communication and time complexity of the counting of the nodes are bounded by $2E$ messages and $2N$ time units respectively. *Communication:* Every edge is traversed by one *activate* message and and one report or reject message. *Time:*the worst case is a line of nodes where we can have at most $N - 1$ consecutive (node activating) propagations of the *active* message and a possible N-th rejected propagation costs 2 time units. It takes in addition $N$ messages and at most $N$ time for the information to be broadcasted to all nodes.

The strong points of the algorithm that we call **Simple Count** are its simplicity and the low communication complexity.

Notice that by slightly modifying this algorithm, we can solve the leader election problem with the above complexity; in the reporting procedure nodes can also report the minimum ID of their subtree (instead of minimum it can be any criterion). The root finds the node with minimum ID and if required, notifies that node adding at most $N$ messages and $N$ time units to the complexity.

## 4 Creating Optimal Algorithms

So far we have seen a number of different techniques that can be incorporated in [GHS83] and give rise to new algorithms. We can now group these techniques according to whether they are optimal with respect to communication or time. Note the techniques will be denoted by the names given in this paper.

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

1. Root Size ([CT85] [Gaf85]).

2. Fast Report (Fast Report algorithm)

3. Local Decision to Join (autonoMST).

4. Root Distance ([FM95] [Awe87]).

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

**Time:** The optimal time is $O(N)$ time units.

1. Tentative (Tentative algorithm).

2. Leader Distance ( [FM95]).

3. Leader Size (optiMST).

The time-only optimal algorithms are: the algorithm of the third phase in [Awe87], optiMST and Tentative.

Note that the above time optimal techniques need the support of the communication optimal techniques. The former alone can not guarantee optimal time and they may also create cycles. Briefly, we can mention that the Leader Distance/Size techniques have to be combined with the Root Distance/Size technique respectively (for more details see [Fal95] [FM95]).

There is a tradeoff between communication and time; the techniques that achieve optimal time, require non-optimal number of messages. So far there does not exist a technique or a combination of techniques that could make [GHS83] optimal in both aspects; a more complicated solution to the problem is required and is presented next.

In [Awe87], Awerbuch suggested an innovative three-phase algorithm whose revised version [FM95] achieves optimal performance in terms of both message and time complexity. To elaborate, in the first **Counting** phase, the algorithm determines $N$. The two last phases represent a tradeoff between the demands of the initial part of the problem (involving large numbers of small fragments, where limiting the number of messages is most critical) and the final part of the problem (involving small numbers of large fragments, where limiting the execution time is most critical).

Generalizing the structure above, we can construct an optimal algorithm using a counting algorithm and two non optimal MST algorithms by guaranteeing that each phase uses $O(E + N\log N)$ messages and takes $O(N)$ time.

**I) Counting Phase.** The purpose of this phase is to let the nodes know the size of the network, i.e. $N$, that will be used as a switching criterion from phase II to phase III. If $N$ is known then this phase can be skipped. We can use the Counting algorithm of [Awe87] or the Simple Count that we previously saw.

**II) Small fragment MST Phase:** $Size < \frac{N}{log(N)}$. Small fragments join fast and therefore we do not need communication expensive techniques. We can use any algorithm optimal with respect to communication that for the "small" size fragments is also optimal to time. It was proven [Awe87] that [GHS83] has this property and so do the communication optimal algorithms we listed above, since they are based on it. Notice that every fragment that finds out that its size satisfies the above inequality, passes independently in the next phase.

**III) Large fragment MST phase:** $Size \geq \frac{N}{log(N)}$. For great fragments time delay becomes an issue and the need for time optimal techniques arises. The fragment that passes to this phase can use any algorithm optimal with respect to time as long as it is also optimal with respect to communication, for a total of less than $\log(N)$ independent fragments. Notice that from the time optimal algorithms listed, only Tentative does not have this property.

In creating an optimal algorithm, we can choose the algorithm from each category that, on top of optimality, will satisfy best a variety of criteria such as implementation simplicity, average case or experimental performance, and problem specific conditions; an extreme example is that if $N$ is known, we can skip the first phase.

Using a non-optimal algorithm could also be a acceptable in practice. For example, in sparse graphs or when the number of messages is not a constrain, the Tentative algorithm may be prefered for its simplicity since it guarantees time optimality.

## 5 Conclusions

This paper examined the problem of creating an optimal algorithm for finding an MST. Apart from the previous techniques that increase the efficiency of the basic algorithm [GHS83], new techniques were presented that use distributed information that resides at the nodes. The techniques can be classified according to their asymptotic behavior and using an optimal pattern, we can combine them to create optimal algorithms. This pattern also indicates the character of the problem by identifying three stages with different requirements.

*Future work.* Finding a technique or a combination of techniques that would make redundant the counting phase, i.e. a one phase algorithm, remains a challenge. Until this is found, we can try to determine among all the possible optimal algorithms the one that would offer more advantages with respect to different criteria such as simplicity of implementation, efficiency in practice or the specific families of graphs that we may have to deal with.

**Acknowledgments.** We would like to thank Vassos Hadzilacos for his valuable advice and support. Special thanks to Petros Faloutsos and Joanna Everitt for their keen comments. This research was supported by the University of Toronto, through the U. of T. Open and Connaught scholarship programs, and by the Natural Sciences and Engineering Research Council of Canada under grant #A5517.

## References

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

[AZ89] Mohan Ahuja and Yahui Zhu. Distributed algorithm for minimum weihgt spanning trees based on echo algorithms. *9th International Conference on Distributed Computing Systems, publ. by IEEE*, 1989.

[CT85] F. Chin and H.F. Ting. An almost linear time and $o(vlogv + e)$ messages distributed algorithm for minimum weight spanning trees. *Proceedings of 1985 FOCS Conference Portland, Oregon*, October 1985.

[Fal95] Michalis Faloutsos. Corrections, improvements, simulations and optimstic algorithms for the distributed minimum spanning tree problem. *Master's Thesis University of Toronto*, 1995.

[FM95] Michalis Faloutsos and Mart Molle. Optimal distributed algorithm for minimum spanning trees revisited. *to appear in PODC*, 1995.

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

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

[GKP93] J.A. Garay, S. Kutten, and D. Peleg. A sub-linear time distributed algorithm for minimum-weight spanning trees. *FOCS*, 1993. 34.

[KP95] S. Kutten and D. Peleg. Fast distributed construction of $k$-dominating sets ans applications. *to appear in PODC*, 1995.