A Sub-Linear Time Distributed Algorithm for Minimum-Weight Spanning Trees*

Juan A. Garay † Shay Kutten ‡ David Peleg §

November 26, 1996

Abstract

This paper considers the question of identifying the parameters governing the behavior of fundamental *global* network problems. Many papers on distributed network algorithms consider the task of optimizing the running time successful when an $O(n)$ bound is achieved on an n-vertex network. We propose that a more sensitive parameter is the network's diameter $Diam$. This is demonstrated in the paper by providing a distributed Minimum-weight Spanning Tree algorithm whose time complexity is sub-linear in n, but linear in $Diam$ (specifically, $O(Diam(G) + n^{\varepsilon} \cdot \log^* n)$ for $\varepsilon = \frac{\ln 3}{\ln 6} = 0.6131..$). Our result is achieved through the application of graph decomposition and edge- elimination- by- pipelining techniques that may be of independent interest.

*An abridged version of this paper appeared in *Proc. 34th IEEE Symp. on the Foundations of Computer Science*, pp. 659-668 , Palo Alto, CA, November 1993.

†I.B.M. T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598, USA. garay@watson.ibm.com. Part of the work was done while visiting the Weizmann Institute of Science, Rehovot, Israel.

‡I.B.M. T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598, USA. kutten@watson.ibm.com.

§Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 76100 Israel. peleg@wisdom.weizmann.ac.il. Supported in part by a Walter and Elise Haas Career Development Award and by a grant from the Basic Research Foundation. Part of the work was done while visiting IBM T.J. Watson Research Center.

1 Introduction

1.1 Motivation

In many papers on distributed network algorithms, the task of optimizing the running time is considered successful when an $O(n)$ bound is achieved on an n-vertex network. Typically, the justification is that there exist n-vertex graphs for which this bound is the best possible. The sequence of solutions to the Leader Election problem (LE) exemplify the reasoning above. Following the $O(n \log n)$ running time, and a first improvement by Chin and Ting [CT] and Gafni [G] (with an $O(n \log^* n)$ running time), Awerbuch gave an "optimal" $O(n)$-time solution to the problem [A1]. Again, this solution is optimal in the sense that there exist networks for which this is the best possible.

This type of optimality may be thought of as "existential" optimality. Namely, there are points in the class of input instances under consideration, for which the algorithm is optimal. A stronger type of optimality, which we may analogously call "universal" optimality, is when the proposed algorithm solves the problem optimally on *every* instance.

An interesting "side-effect" of universal optimality is that a universally-optimal algorithm precisely identifies the parameters of the problem that are inherently responsible for its complexity. For example, returning to the LE problem, a more careful look reveals that the inherent parameter is the network's diameter $Diam$. Indeed, it was observed in [P] that it is possible to give a trivial $O(Diam)$-time distributed LE algorithm (although it should be noted that the solutions mentioned above were also message-optimal, whereas the algorithm of [P] is not).

The interesting question that arises is therefore whether it is possible to identify the inherent graph parameters associated with the distributed complexity of various fundamental network problems, and develop universally-optimal algorithms for them.

A closely related question has been dealt with before in the context of studying the role of *locality* in distributed computing. Various problems were shown to be essentially local, and hence amenable to a localized algorithm with very fast (e.g., polylogarithmic) running times. Notable examples include computing maximal independent sets and graph coloring [GPS, L, AGLP, PS2]. Locality-based techniques were developed for reducing communication and time complexities also for other problems, whose local nature is less apparent [AP1, AP2, AKP].

In contrast, we are interested here with problems that are essentially *global*, i.e., ones that do not admit localized solutions, but rather always require the algorithm to "traverse" the network. Problems of this type still raise the interesting (if more modest) question of deciding whether $\Omega(n)$ time is essential, or is the network's *diameter* the inherent parameter. In the latter case, it would be desirable to devise algorithms for these network problems that have a better complexity for the case of graphs with low diameter.

In this paper, we tackle the classical Minimum-weight Spanning Tree (MST) problem. This problem has been studied before as a canonical example for a graph-algorithmic problem whose communication-efficient distributed solution poses some surprisingly nontrivial subtleties [GHS]. The time complexity of the algorithm of [GHS] is $O(n \log n)$, which was later improved to the (existentially) "optimal" $O(n)$ in [A1]. As with other problems, such as the above LE example, it is natural to ask whether $O(n)$ is universally optimal, or it can be improved.

Once again, the MST problem proves to be a worthy candidate for this type of study. In other tree constructions, such as the Breadth-First-Search (BFS) tree (which is closely related to the LE problem), it is intuitively clear that the true time bound should be related to the network's diameter $Diam$, since the depth of the constructed tree is proportional to $Diam$. In contrast, the MST of a given network may be considerably deeper than $Diam$, and in fact, may be as high as $\Omega(n)$. Hence construction methods based on communication on the tree structure itself are doomed to require $\Omega(n)$ time, and the problem of breaking the $\Omega(n)$ barrier seems intrinsically harder.

In this paper we get closer to identifying the inherent parameters governing the behavior of distributed MST construction, by presenting a distributed MST algorithm whose time complexity is sub-linear in n, and linear in $Diam$ (specifically, $O(Diam(G) + n^{\varepsilon} \cdot \log^* n)$ for $\varepsilon = \frac{\ln 3}{\ln 6} = 0.6131..$), thus breaking the $O(n)$ barrier. This result is achieved through the application of graph decomposition and edge elimination techniques that may be interesting in their own right.

1.2 Model and Definitions

In this paper we focus on the problem of devising a time-efficient distributed MST algorithm. The statement of the problem is as follows. The network is represented by an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of links between them. (Henceforth we use the terms graph and network interchangeably, when no ambiguity arises.) The graph is given with a weight function $\omega : E \rightarrow R^+$ on the edges, such that each node in V is associated with its own processor, and processors are able to communicate with each other via the edges in E. The goal is to have the nodes (processors) cooperate to construct a tree covering the nodes in V whose total edge weight is no greater than any other spanning tree for G.

We assume that nodes have unique identifiers, and that each edge $e \in E$ is associated with a distinct weight $\omega(e)$, known to the adjacent nodes. The usefulness of having distinct edge weights stems from the fact that this property guarantees that the MST is unique. Clearly, having distinct weights is not an essential requirement, since one can always "create" them by appending the adjacent node's numbers to them. However, it is known that if the graph has neither distinct edge weights nor distinct node identifiers, then no distributed algorithm exists for computing an MST with a bounded number of messages [GHS].

For every subgraph F of the network, let $Diam(F)$ denote the diameter of F, i.e., the maximum distance between any two vertices of F, where distance is measured in the unweighted sense, i.e., in number of hops.

In order to be able to concentrate on the central issue of time complexity, we shall follow the common trend of stripping away inessential complications. In particular, we ignore the communication cost of our algorithm, i.e., the number of messages it uses. (We comment that while this number is not optimized in any way in our solution as presented in the current paper, it is not very large. In particular, it is still considerably smaller than the number of messages needed for every node to learn the topology of the entire network, as is used in a number of existing routing schemes, e.g., Internet routing [MRR].)

We also assume that the computation performed by the network is *synchronous*. Namely, computation proceeds in rounds, governed by a global clock, with each round taking one time unit. In each round each processor can examine the messages sent to it by its neighbors (if any), compute, and send messages to any subset of its neighbors. Such messages, if sent, are available to their recipients in the next round. The *time complexity* of a synchronous algorithm is the number of time units elapsing until the termination of the algorithm. This assumption is quite common in the literature. Note, however, that here it is not essential, since our decision to ignore communication costs allows us to freely use a synchronizer of our choice; for example, synchronizer α [A2] enables an asynchronous network to run any protocol that was designed for synchronous networks, with the same time complexity, at the cost of some increase in the message complexity.

Still, we shall not adopt the extreme model employed in previous studies of locality issues (cf. [L]), in which messages of arbitrary size are allowed to be transmitted in a single time unit, since in this model, the refined distinctions we focus on here disappear. Clearly, if unbounded-size messages are allowed, then the problem can be trivially solved in time $O(Diam(G))$ by collecting the entire graph's topology into a central node, computing an MST locally and broadcasting the result throughout the network.

Consequently, we will assume the more realistic (and rather common) model in which messages have size $O(\log n)$, and a node may send at most one message on each edge at each time unit.

We will also make the assumption that edge weights are polynomial in n, so an edge weight can be sent in a single message. (This assumption is required for the time analysis of all previous algorithms as well [GHS, A1].)

1.3 Our Results

The original distributed MST algorithm of Gallager *et al.* [GHS] has a time complexity of $O(n \log n)$. This was later improved by Awerbuch, who gave an $O(n)$-time algorithm [A1]. In this paper, we present a distributed Minimum-weight Spanning Tree algorithm with time complexity $O(Diam(G) + n^{\varepsilon} \cdot \log^* n)$ for $\varepsilon = \frac{\ln 3}{\ln 6} = 0.6131...$

2 Overview of the MST Algorithm

Our algorithm is based on a careful combination of two distinct approaches for the (distributed) construction of an MST. Thus, before explaining our algorithm, it is instrumental to review the two approaches, and try to understand their shortcomings (when used individually).

2.1 Distributed growth approach

The first approach is the construction method that is at the basis of the MST algorithm of [GHS] (referred to, from now on, as the GHS algorithm). This algorithm does not utilize a central controller, or "center of activity," but rather allows a large number of processes to proceed simultaneously and independently in the network. These processes gradually *grow* the MST from scratch.

The GHS algorithm operates by growing so-called *fragments* in a distributed manner, with each fragment consisting of a portion of the final MST. In each iteration of the algorithm, the nodes of each fragment explore the immediate neighborhood of the fragment, and collectively decide on a neighboring fragment to merge with (by adding the connecting edge to the tree), thus creating a larger fragment of the final MST. This is a distributed version of the "blue" rule for MST construction (cf. [T], p. 71).

It is crucial to understand why algorithms based on this approach cannot guarantee a time complexity proportional to $Diam(G)$. The inherent difficulty lies in the fact that the communication necessary for making the merging decisions for each fragment is done *on the fragment itself.* This is a problem since the MST is not guaranteed to have depth proportional to $Diam(G)$, and neither are any of its fragments. In fact, it is easy to come up with examples for n-vertex graphs with diameter 1 whose MST has depth $n-1$. Thus any approach based on communicating over the MST itself will have time complexity proportional to n in the worst case on some graphs.

Note, however, that in the initial stages of the GHS algorithm, the fragments are still small, and therefore communication on them is not as expensive. The idea on which our algorithm is based is thus to start by running the GHS algorithm up to an appropriately chosen point, and then switch to a different algorithm. In order for this idea to work, it is essential to have a version of the GHS algorithm that controls the rate of growth of the different fragments, preventing some fragments from growing too large while other are still very small. (The original algorithm of [GHS] allows uncontrolled growth of fragments.)

2.2 Coordinated elimination approach

The second approach to the distributed construction of an MST is based on synchronous, coordinated, centralized operation. One extreme example for an algorithm operating in this way is the centralized algorithm mentioned in the introduction, in which the entire graph's topology is collected into a central node, which then computes an MST locally and broadcasts the result throughout the network. This algorithm is slowed down considerably by the heavy communication bottlenecks that are bound to be created along the paths leading to the center node.

The crucial idea at the basis of the second approach is that some of this communication burden can be reduced by delegating the work of the center node to other nodes along the paths leading to the center. This can be achieved if instead of attempting to *build* the MST, we concentrate on *eliminating candidate edges*, using the so-called "red rule" for MST construction (cf. [T], p. 71).

In order to control the complexity of the edge-elimination process, we employ a new efficient pipelining technique, aimed at overcoming congestion. This new technique deserves some discussion, since it is different and considerably simplified compared to the one presented in earlier versions of this paper [GKP]. The new technique is motivated by the following complication in the solution of [GKP]. In the end of the first stage of the algorithm, nodes are informed about candidate edges in the graph (those that connect clusters). In the third stage of the [GKP] algorithm, the nodes forward the description of those edges to their parents on a spanning tree, towards the root. The problem is that there might be too many such edges, congesting the accumulation process on the tree. The algorithm of [GKP] overcomes this difficulty by introducing an intermediate stage employing a rather complex method for eliminating short cycles, and thus reducing the number of remaining edges for the third stage.

The algorithm presented here has only two stages. Both the second (cycle elimination) and the third stages of [GKP] are replaced by a single stage, similar to the third stage of [GKP], in which the nodes forward descriptions of candidate edges to their parents on a Breadth First Search spanning tree. However, we incorporate a new technique of cycle elimination into this stage. This new pipelining technique is very simple, and the main novelty lies in its analysis. As in the third stage of the algorithm in [GKP], nodes on the spanning tree forward the description of their candidate edges to their parents towards the root. However, here a node avoids forwarding the description of *cycle-heavy* edges. That is, a node forwards the description of edges according to their weight, and does not forward the description of an edge that closes a cycle with edges whose description has already been forwarded. Let us now give a hint regarding the difficulty in analyzing this stage of the protocol. It is not hard to prove that all non-MST edges get eliminated by this process. The difficulty lies in analyzing the time complexity. Note that the description above (i.e. that a node does not forward a description of a cycle- heavy edge) seems to suggest that at certain points in time a node might be forced to wait, due to not having an edge description that is eligible for forwarding. Thus it may seem that the convergecast may not be fully pipelined, and hence may take a long time. We prove that this simple convergecast is fully pipelined, and thus its running time is the one required, eliminating the need for the complex cycle elimination stage of [GKP]. This pipelining proof may be of interest in itself.

2.3 Combined approach

Let us now outline the structure of our algorithm. The algorithm consists of two parts, as follows.

`Algorithm Sublinear-MST`

Part I: `Controlled-GHS`;

Part II: Edge elimination.

As its name indicates, `Controlled-GHS` (Part I) is a modified variant of the original algorithm of [GHS]. The purpose of the modification is to produce a balanced outcome in terms of number and diameter of the resulting fragments. This is achieved by computing in each phase a small dominating set on the fragment forest, and merging fragments accordingly. This, in turn, is achieved by invoking the distributed maximal independent set (MIS) algorithm of [GPS]. At the end of this phase, we are left with a "small" number of fragments, all of which have a "small" diameter.

Finally, Part II performs the elimination of most of the remaining edges, leaving only a tree connecting the fragments, and thus yielding the final MST. This elimination process is carried out in a distributed manner, and requires nodes to forward the description of certain edges to a central node, while eliminating the description of certain other edges. (The forwarding is done on a superimposed breadth-first tree, and the center is the root of this tree.) The central node does the final elimination, which yields the output MST tree.

The details of each part are given in the remainder of the paper, followed by the analysis of the total complexity.

3 Part I: Controlled GHS

In this section, we provide a modified, *controlled* version of the Gallager-Humblet-Spira algorithm for MST, that is suitable for our purposes. We first provide a brief overview of the original algorithm of [GHS].

3.1 Brief Description of the `GHS` Algorithm

In the original distributed `GHS` algorithm of [GHS], nodes form themselves into *fragments* of increasing size. Initially, all nodes are in singleton fragments. Nodes in each fragment F are connected by edges that form a *rooted* MST, $T(F)$, for the fragment. (Initially, the sole node comprising a fragment is also its root). Each node (other than the root) in a fragment has a pointer to one of its neighbors, which is the next node on the path over the tree to the root; moreover, each node "knows" the root Id. (In the sequel we loosely use the word "fragment" to mean both the collection of nodes F and the corresponding tree $T(F)$.)

The events in the algorithm are divided into *phases*. Each phase takes as its input the fragment structure output by the previous one, and outputs larger fragments. (The first phase takes as an input the singleton fragments, one per node.)

Let us now describe one phase. We present a simplified version of the `GHS` algorithm compared to the original algorithm of [GHS]. (The original version is more complex because of the desire to save messages, and because of the asynchronous nature of the networks for which it was designed.)

Within each fragment F, nodes cooperate to find the minimum weight *outgoing* edge in the entire fragment (an outgoing edge of a fragment F is an edge with one endpoint in F and another at a node outside it). The strategy for identifying this edge involves broadcasting over the fragment's tree $T(F)$, asking each node separately for its own minimum weight outgoing edge. These edges are then sent upwards on the tree $T(F)$, towards the root. Each intermediate node first collects this information from all its children in the tree, and then passes up only the lowest-weight edge it has seen (which is therefore the lowest-weight edge in its subtree). The minimum weight outgoing edge is selected by the root to be included in the final MST.

Once a fragment's minimum weight outgoing edge is found, a message is sent out over that edge to the *chosen* fragment on the other side. The two fragments then combine, possibly along with several other fragments, into a new, larger fragment. If the other fragment chose the same edge, then the two fragments agree at that point to combine, and the edge they both chose is termed the *core edge*. Otherwise, following the route from a fragment F_1, to its chosen fragment F_2, to F_2's chosen fragment F_3, and so on, [GHS] show that one must eventually reach two fragments that agree to combine over a core edge. The combined fragment includes these two, as well as all those other fragments that have such a route to them.

The combined fragment has a root; it is the node with the higher Id of the two endpoints of the core edge. (This can be simulated in the case that nodes do not have distinct Id's but edges have distinct weights.)

This concludes the description of a single phase. In the next phase, the new fragment finds its own minimum weight outgoing edge, and the entire process is repeated until all the nodes in the graph have combined themselves into one single fragment. Each fragment (of size 2 or greater) is identified by the fragment's *core edge* and root.

We remark that in the original `GHS` algorithm, all nodes operate asynchronously. This creates the risk of undesirable "growth patterns" of fragments, resulting in excessive communication costs (measured in number of messages). This problem is handled by using special rules for merging fragments, designed to prevent these complications. These rules are based on a "balanced data structure" approach. A *phase number* is associated with each fragment. If $phase(F) = l$ for a given fragment F, then the number of nodes in F is greater than or equal to 2^l. Initially, all fragments (singleton nodes) are at phase 0. When two fragments at phase l are combined together, the resulting new fragment has phase $l + 1$. Thus, the total number of messages is kept to $O(n \log n)$ (although some more complex rules are needed to allow merges between fragments of unequal phases). Similarly, it is not hard to show by induction on the phase numbers that the time complexity of the algorithm is $O(n \log n)$ time units.

We refer the reader to [GHS] for further details.

3.2 Computing a Small Dominating Set on a Tree

In this subsection we present a procedure `Small-Dom-Set` for computing a small dominating set on a given tree. The procedure makes use of a sub-procedure for computing a maximal independent set in the tree. (A set M of vertices in a tree T is said to *dominate* the tree if every vertex outside M has a neighbor in M.) A distributed version of Procedure `Small-Dom-Set` will later be used as a component in our `Controlled-GHS` algorithm, which is a modification of `GHS`.

Our goal is as follows. Given a rooted tree T with a vertex set $V(T)$, find a set of vertices $M \subseteq V(T)$ such that:

1. M dominates $V(T)$, and
2. $|M| \leq \frac{|V(T)|}{2}$.

Furthermore, we would like this procedure to be amenable to a fast distributed implementation.

The procedure is based on the following. For a vertex $v \in V(T)$, let $\mathtt{Child}(v)$ denote the set of v's children in T. We use a *level* function $\check{L}(v)$ on the nodes, defined as follows:

$$\check{L}(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf;} \\ 1 + \min_{u \in \mathtt{Child}(v)}(\check{L}(u)) & \text{otherwise.} \end{cases}$$

We denote by $\check{L}_i$ the set of tree nodes at level i,

$$\check{L}_i = \{v \mid \check{L}(v) = i\}.$$

Procedure `Small-Dom-Set` for computing a dominating set M on a tree T is presented next.

`Algorithm Small-Dom-Set`

1. Mark the nodes of T with level numbers $\check{L}(v) = 0, 1, 2$;
2. Select an MIS, Q, in the set R of unmarked nodes;
3. $M \leftarrow Q \cup \check{L}_1$.

A pictorial example is given in Figure 1.

For the distributed implementation discussed in the next subsection, it is important to note that although the level numbers $\check{L}(v)$ are defined for every vertex v in the tree, only the vertices belonging to the first three levels, $\check{L}_0$, $\check{L}_1$ and $\check{L}_2$, are actually marked.

The fact that Procedure `Small-Dom-Set` produces a dominating set is established by the following lemma.

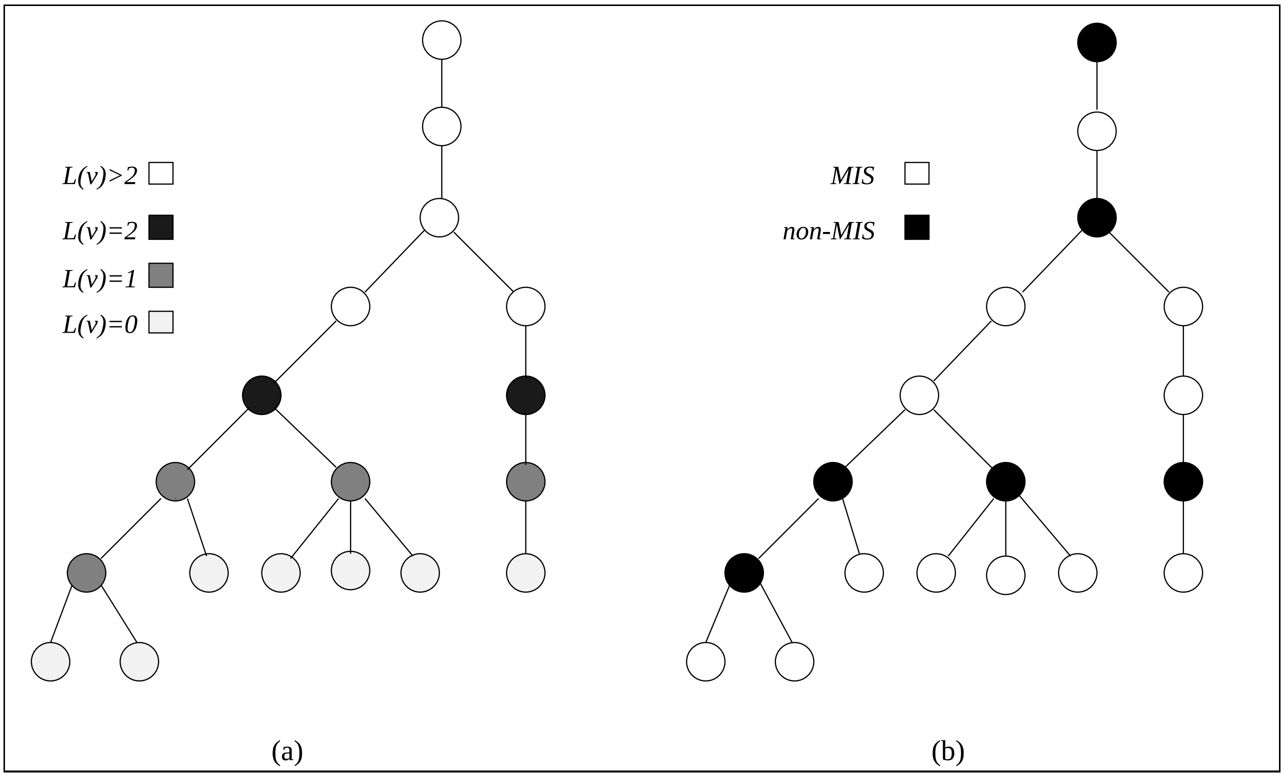

Figure 1: (a) The level numbers marked by the algorithm on a given tree T.
(b) A small dominating set M on the tree T.

Lemma 3.1 *Let M be the outcome of Procedure* `Small-Dom-Set` *on the tree T. Then for every node $v \notin M$ there exists an adjacent node $v' \in M$.*

Proof Partition $V(T)$ into $\check{L}_0 \cup \check{L}_1 \cup \check{L}_2 \cup R$. The set M output by Procedure `Small-Dom-Set` is composed of $Q \cup \check{L}_1$. Now, by choice of Q, it dominates each node of $R \setminus Q$. Also, each node of $\check{L}_0 \cup \check{L}_2$ has a neighbor in $\check{L}_1$. ∎

The "smallness" of the resulting dominating set is guaranteed by the following lemma.

Lemma 3.2 $|M| \leq \frac{|V(T)|}{2}$.

Proof By construction, $M = \check{L}_1 \cup Q$. It is clear that $|\check{L}_1| \leq |\check{L}_0|$, and hence

$$|\check{L}_1| \leq \frac{|\check{L}_0 \cup \check{L}_1|}{2} \ . \tag{1}$$

We now claim that

$$|Q| \leq \frac{|R \cup \check{L}_2|}{2} \ . \tag{2}$$

This can be proved by selecting for every $v \in Q$ a *distinct* match $\omega(v) \in (R \cup \check{L}_2) - Q$, thus establishing that $|R \cup \check{L}_2| \geq 2 \cdot |Q|$. The matching is done as follows: Pick $\omega(v)$ to be an arbitrary child of v (by definition, vertices in Q always have children in $R \cup \check{L}_2$). Distinctness is guaranteed by the fact that each node in the tree has a unique parent.

It now follows from (1) and (2) that

$$|M| = |Q \cup \check{L}_1| \leq \frac{|R \cup \check{L}_2|}{2} + \frac{|\check{L}_0 \cup \check{L}_1|}{2} = \frac{|V(T)|}{2} \quad . \quad \blacksquare$$

3.3 Controlled-GHS

3.3.1 Description of the `Controlled-GHS` Procedure

In this subsection, we provide the modified, *controlled* version of `GHS` (named `Controlled-GHS`) that is able to achieve the following:

1. Upon termination, the number of fragments is bounded from above by N (for N to be specified later).
2. Throughout the execution, the diameter of every fragment F satisfies $Diam(F) \leq d$ (for d to be specified later).

Intuitively, since we focus on a synchronous algorithm, and we do not care about communication complexity, our version of `GHS` is simpler than the original algorithm. In particular, we are oblivious to balancing fragment sizes, and we do not need to use the phase rules used in the original algorithm, since phases are imposed by phase synchronization (using the assumed global clock).

More specifically, `Controlled-GHS` starts with singleton fragments, just like `GHS` (see Subsection 3.1), and executes a total of I phases. Each phase outputs a collection of fragments, that serve as the input for the next phase. A fragment is a rooted tree, where each node (except for the root) has a pointer to the edge that leads to its parent in the tree. Each phase of `Controlled-GHS` consists of the following two stages.

Stage 1: Consider the fragments that are input to the phase. Execute a phase of `GHS` up to a point where each such input fragment F has chosen its minimum-weight outgoing edge, i.e., has decided which other fragment in the current fragment collection it wants to merge with.

This decision induces a "forest" structure on the fragment collection (possibly with length-2 loops at the tree roots). Henceforth we refer to this structure as the *fragment forest*, denoted FF.

Stage 2: Break the resulting trees into "small" ($O(1)$ depth) trees, and merge only these small trees.

This process is depicted in Figure 2. To accomplish stage 2, the algorithm first computes a dominating set $M_{FF}(\tilde{T})$ on each tree $\tilde{T}$ of the fragment forest FF. Note that the nodes of this tree are input fragments of the current phase of the algorithm. Let M_{FF} be the union over the trees $\tilde{T}$ (of the FF forest), of $M_{FF}(\tilde{T})$. The algorithm then lets each fragment $F \notin M_{FF}$ pick one neighboring fragment $F' \in M_{FF}$ and merge with it. This causes the actual merges performed in a phase of `Controlled-GHS` to have the form of "stars" in the fragment forest FF, and prevents merges along long chains, hence bounding the diameter of the resulting fragments.

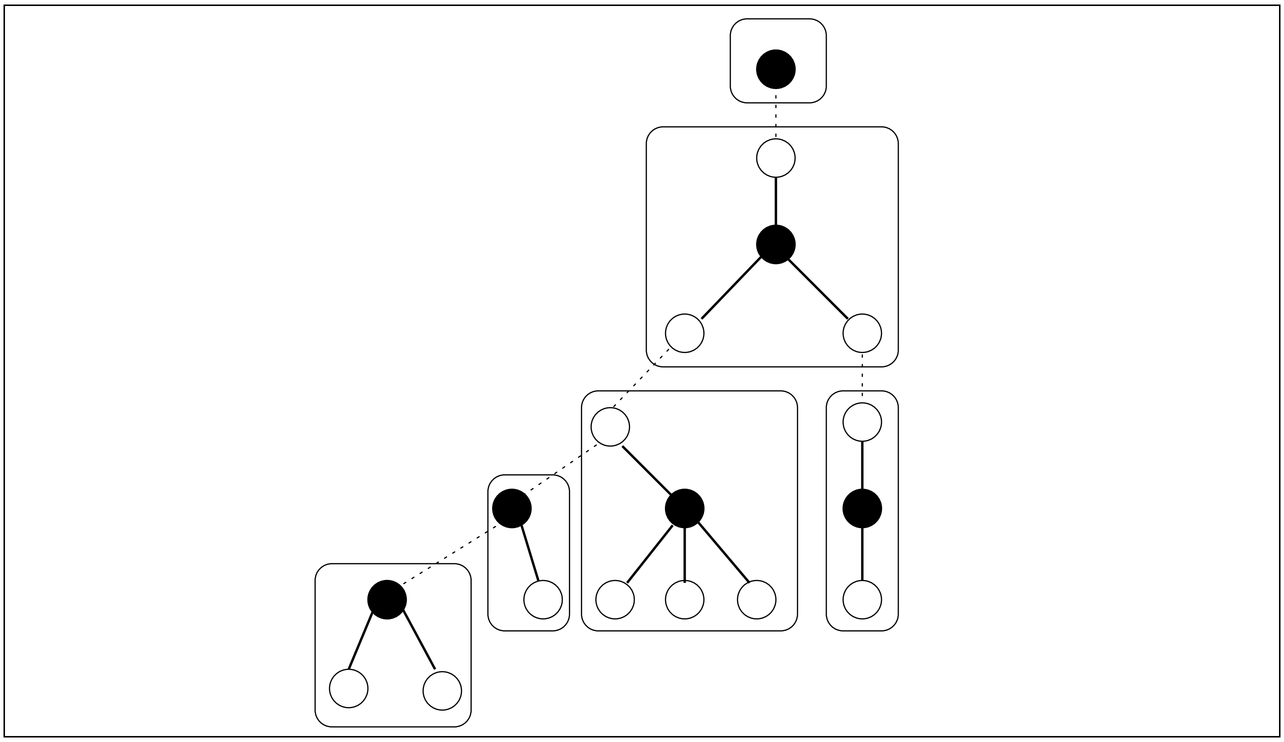

Figure 2: A Phase of `Controlled-GHS`: the tree T of Figure 1(b) is broken into "small" trees (represented by the solid edges) according to the MIS.

The dominating sets are computed using a distributed implementation of Procedure `Small-Dom-Set`, denoted `Dist-SDS`, applied separately to each tree $\tilde{T}$ in the fragment forest FF. A key aspect in this computation is the use of a distributed algorithm for computing a maximal independent set. Most distributed MIS algorithms in the literature (e.g., [AGLP, PS2]) can be used for our purposes. In fact, our solution only requires to compute an MIS on a tree, so we can use a distributed version of the algorithm of [GPS], which is optimal for trees. (This algorithm makes use of $O(\log n)$-bit messages, and therefore can be used within our model.)

The algorithm of [GPS] (henceforth referred to as the `GPS` algorithm) has the following properties.

Lemma 3.3 [GPS] *Applied in a given n-vertex tree G, Algorithm* `GPS` *computes (in a synchronous manner) a maximal independent set (MIS) of G using $O(\log n)$-bit messages, and its time complexity is $O(\log^* n)$.*

Note that although Procedure `Dist-SDS` is applied to the trees of the fragment forest FF, it is actually executed on the original network itself. Hence the procedure operates by simulating each fragment by a single representative, say, its root.

3.3.2 Analysis of Controlled-GHS

The bounds on the number and diameter of fragments are established by the next lemma. Given a fragment forest FF, let $Diam(FF) = \max_{F \in FF}\{Diam(F)\}$. Let FF_i denote the fragment forest produced by the algorithm `Controlled-GHS` at the end of the ith phase. (FF_0 is the initial set of vertices.)

Lemma 3.4 *In each phase i of* `Controlled-GHS`*,*

1. *the number of fragments at least halves, i.e., $|FF_i| \leq |FF_{i-1}|/2$, and*
2. *the maximum diameter of a fragment increases by a factor of at most 3 (possibly plus 2), i.e., $Diam(FF_i) \leq 3Diam(FF_{i-1}) + 2$.*

Proof The number of new fragments in each tree $T = T(F)$ of the fragment forest FF at the end of a phase is equal to $|M(T)|$. By Lemma 3.2, $|M(T)| \leq |V(T)|/2$. Hence the same holds for the entire fragment forest, and Claim 1 of the lemma follows. Claim 2 is readily satisfied, since the merges are star-shaped. ∎

Corollary 3.5 *After running* `Controlled-GHS` *for I phases,*

1. *the number of fragments in FF_I is at most $N(I) = \frac{n}{2^I}$, and*
2. *$Diam(FF_I) \leq d(I) = 3^I - 1$, for every fragment F.* ∎

Let us now turn to analyzing the time complexity of `Controlled-GHS`. We first examine the performance of Procedure `Dist-SDS`.

Lemma 3.6 *When executed on a fragment forest FF, Procedure* `Dist-SDS` *takes time $O(\log^* n) \cdot Diam(FF)$.*

Proof It is easy to see that if a fragment F is a leaf in the fragment forest FF, then it identifies itself as such, and subsequently marks itself $\check{L}_0$, in time $O(Diam(F))$. Similarly, fragments mark themselves $\check{L}_1$ and $\check{L}_2$ in time $O(Diam(F))$ as well. Another part that affects the time complexity is the MIS computation. By Lemma 3.3, the time complexity of the GPS algorithm is $O(\log^* n)$. Note that since the procedure is executed on the original network itself, it is slowed down by a factor of $O(Diam(FF))$. Hence the implementation of the GPS algorithm on the fragments of FF is slowed down to $O(\log^* n) \cdot Diam(FF)$ in our case. ∎

The properties of the Controlled-GHS algorithm are now summarized by the following *graph decomposition* lemma.

Lemma 3.7 *When Algorithm* Controlled-GHS *is activated for I phases, it takes $O(3^I \cdot \log^* n)$ time, and yields a fragment forest FF_I of up to $N(I) = n/2^I$ fragments, of diameter $Diam(FF_I) \leq d(I) = 3^I - 1$. Each fragment in the forest is a fragment of an MST of the graph G.*

Proof By Lemma 3.6, each phase i, $1 \leq i \leq I$, of Controlled-GHS takes time at most $Diam(FF_{i-1}) \cdot O(\log^* n)$. By Cor. 3.5, this is at most $3^{i-1} \cdot O(\log^* n)$. Thus, the total time is given by

$$\sum_{i \leq I} (3^{i-1} \cdot O(\log^* n)) \leq 3^I \cdot O(\log^* n) \ .$$

The size and diameter properties of the resulting fragment forest follow directly from Corollary 3.5. ∎

The rather even graph decomposition obtained by algorithm Controlled-GHS may conceivably be useful for other purposes as well.

4 Part II: Edge Elimination

The second part of our algorithm starts at the point where we are given a fragment graph $\tilde{F}_l$, whose vertex set $V(\tilde{F}_l)$ is a fragment forest FF_I containing $N \leq N(I)$ fragments of the MST, and whose edge set $E(\tilde{F}_l)$ is a collection of inter-fragment edges, which are the remaining candidates for joining the MST. We now proceed to reduce the total number of remaining inter-fragment edges (to the necessary $N - 1$).

4.1 The Pipeline Algorithm

We need the following technical definition. For a set of edges Q and a cycle-free subset $U \subseteq Q$, define $Cyc(U, Q)$ as the set of all edges $e \in Q \setminus U$ such that $U \cup \{e\}$ contains a cycle.

We are now ready to describe our pipelined procedure, given in Figure 3.

4.2 Analysis

Our analysis hinges on two main properties of the procedure. First, the edges reported by each intermediate node to its parent in the tree are sent in nondecreasing weight order.

1. Build a *breath-first search* tree B on G, with a root $r(B)$.

2. Throughout the execution, each node v on the tree maintains a set Q of all the inter-fragment edges it knows of (either directly or by "learning" about from its children in the previous rounds), ordered by non-decreasing edge weights. Initially this set contains only the inter-fragment edges adjacent to it. It also maintains a set U (initially empty) of all the edges it has already sent up to its parent in the tree.

3. A leaf v starts sending edges upwards at pulse 0. An intermediate node v starts sending at the first pulse after it has received messages from all its children.

4. At each pulse i, the node sends up to its parent in the tree the lightest edge e in Q that has not been sent up in the previous rounds up to round $i-1$ (i.e., $e \notin U$), and which does not close a cycle with the edges of U. If no such edge exists then terminate participation in the pipelining part of the procedure (i.e., go to step 6).

   More formally, the node computes the set of *remaining candidates*

   $$RC = Q \setminus (U \cup Cyc(U, Q)).$$

   If $RC = \emptyset$ then v sends a "terminating" message to its parent in the tree, and terminates its participation in the pipelining part of the protocol. Else, it sends up to its parent the lightest edge e in RC.

5. The root $r(B)$ computes locally the set S of the $N-1$ edges participating in the MST of the fragment graph $\tilde{F}$, from among the edges it hears of from its children.

6. The root $r(B)$ then broadcasts (over the tree B) the resulting set of $N-1$ inter-fragment edges (constituting the MST for the fragment graph $\tilde{F}$) to all nodes in G.

Figure 3: Procedure `Pipeline`.

Second, each intermediate node transmits edges upwards in the tree *continuously*, until it exhausts all the reportable edges from its subtree. Namely, once the set of candidates RC is empty, the node will learn of no more reportable edges.

Let us first make the following straightforward but crucial observation.

Lemma 4.1 *The edges reported by each intermediate node to its parent in the tree form a forest.*

Proof This follows immediately from the rule used by the procedure to select the next edge to be transmitted upwards. ∎

Lemma 4.2 *Every node v starts sending messages upwards at pulse $\hat{H}(v)$, where $\hat{H}(v)$ is the* height *function defined as follows:*

$$\hat{H}(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf;} \\ 1 + \max_{u \in \mathtt{Child}(v)}(\hat{H}(u)) & \text{otherwise.} \end{cases}$$

Proof By straightforward induction on the tree structure, from the leaves upwards. ∎

Our main technical lemma concerns the properties of a node on the tree in some round of the algorithm. Consider an intermediate node v at height H, that has still not terminated its participation in the algorithm, at round t, for some $t \geq H$. Note that each of the children of v in the tree is of height $H-1$ or lower, hence, by Lemma 4.2, all of them started transmission at round $H-1$ or earlier. Call a child *active* if it has not terminated yet (i.e., it has upcast an edge to v on round $t-1$). Let

$$A_t(v) = \{v_1, \ldots, v_p\}$$

be the set of v's active children at round t.

Lemma 4.3 *(a) At the beginning of round t, the candidate set RC examined by v contains at least one candidate edge upcast by each of its active children from $A_t(v)$.*

(b) If v upcasts an edge of weight ω at round t, then all of the edges v was informed of at round $t-1$ by its active children were of weight ω or greater.

(c) If v upcasts an edge of weight ω at round t, then any later edge it will learn of is of weight ω or greater.

(d) Node v upcasts edges in nondecreasing weight order.

Proof We prove the lemma by induction on the height of the tree, starting from the leaves upwards.

A *leaf* v has no (active or other) children, and therefore Claims (a), (b) and (c) hold vacuously. Claim (d) follows trivially from the rules of the procedure.

Let us now consider an intermediate node v, and assume that the claims hold for each of its children. We need to prove the four claims for v. We start with Claim (a).

Let U be the set of m edges upcast by v during the first $m = t - H$ rounds it has participated in (namely, rounds $H, \ldots, t-1$ if $t > H$). By Lemma 4.1, U forms a forest in G. Consequently, break U into the trees $U_1, \ldots, U_\ell$ in G, with $x_i = |U_i|$, where each such tree U_i has a vertex set $V(U_i)$ of exactly $x_i + 1$ vertices, and

$$\sum_i x_i = |U| = m. \tag{3}$$

Consider an active child u of v. Denote by D the set of edges upcast by u so far (up to and including round $t-1$). Since u was still active on round $t-1$, it has transmitted continuously to v since round $\hat{H}(u)$, which, as discussed before, is at most $H-1$. Therefore, we conclude that

$$|D| \geq m+1. \tag{4}$$

Suppose, for the sake of contradiction, that none of the edges upcast by u is a candidate in round t. In other words, for each edge $e \in D$, either e was upcast by v earlier (namely, $e \in U$), or e closes a cycle with the edges of U (namely, $e \in Cyc(U)$).

Thus, every such edge e has both of its endpoints in U. Furthermore, e cannot possibly connect endpoints that belong to two different trees U_i and U_j (since in that case, e would have been in neither U nor $Cyc(U)$).

This implies that the set D can be partitioned into sets $D_1, \ldots, D_\ell$ such that all the edges of D_i are restricted to vertices $V(U_i)$ of the tree U_i, for $1 \leq i \leq \ell$. Moreover, notice that each such set D_i is a forest, since the entire set D is a forest by Lemma 4.1.

The last two facts combined imply that

$$|D_i| \leq |V(U_i)| - 1 = x_i. \tag{5}$$

Combining (3), (4) and (5) together, we get

$$m+1 \leq |D| = \sum_i |D_i| \leq \sum_i x_i = m,$$

which is a contradiction. Hence Claim (a) must hold.

Next, we prove Claim (b) as follows. Consider any active child u of v. Let e be the edge upcast by u on round $t-1$. (Note that e does not necessarily have to be in RC.) Let e' be some edge that was upcast by u at some round $t' \leq t-1$ and is still in the candidate set RC on round t (such an edge must exist by Claim (a)). By the inductive hypothesis of Claim (d), $\omega(e) \geq \omega(e')$. By the edge selection rule of node v, $\omega(e') \geq \omega$, and Claim (b) follows.

Next, we note that Claim (c) follows trivially from Claim (b). Finally, Claim (d) follows trivially from Claim (c) and the edge selection rule of the procedure. ∎

Finally, we need to argue that nodes do not terminate the algorithm prematurely.

Lemma 4.4 *After a node v has terminated its participation in the algorithm, it will learn of no more reportable edges.*

Proof We need to argue that once the set RC becomes empty, no new candidate edges will become known to v. We prove this fact by induction on the structure of the tree, starting from the leaves upwards. The inductive step follows directly from Claim (a) of Lemma 4.3, which guarantees that if RC is empty on round t, none of v's children have upcast it an edge in round $t-1$, and hence all of them have already terminated. ∎

Lemma 4.5 *The running time of Procedure* `Pipeline` *is bounded by* $O(N + Diam(G))$*, and its output is an MST for* G*.*

Proof The fact that the resulting tree is an MST follows from the fact that the trees constructed in the first stage were fragments of the MST, and from the correctness of the "red rule" employed for edge elimination in the procedure (cf. [T], p. 71). (Essentially the red rule says that an edge that is the heaviest on any cycle is not a part of any MST.)

As for the running time, the bound is derived from the following facts. First, the root of the tree receives at most N edges from each of its children. Second, the children send these edges to the root in a fully pipelined fashion (namely, without stopping until exhausting all the edges they know of). Finally, the root starts getting such messages at time $Diam(G)$ at the latest. ∎

5 The Complexity of the Combined Algorithm

Combining the two parts, we get the following distributed algorithm for MST.

1. Perform Algorithm `Controlled-GHS` for I phases.
2. Perform Algorithm `Pipeline`.

Summarizing the results of the last two sections, we get:

Theorem 5.1 *There exists a distributed Minimum-weight Spanning Tree algorithm with time complexity* $O(Diam(G) + n^{\varepsilon} \cdot \log^* n)$ *for* $\varepsilon = \frac{\ln 3}{\ln 6} = 0.6131..$

Proof The complexities of the two parts of our algorithm are as follows, for the given parameter I specified for the first part:

Part I:	$3^I \cdot O(\log^* n)$
Part II:	$Diam(G) + \frac{n}{2^I}$

The total time complexity is thus optimized when choosing I such that $3^I = \frac{n}{2^I}$, namely, $I = \frac{\ln n}{\ln 6}$. For this choice of I, we get $3^I = \frac{n}{2^I} = n^{\varepsilon}$ for $\varepsilon = \frac{\ln 3}{\ln 6}$, which yields a total time complexity of $O(Diam(G) + n^{\varepsilon} \cdot \log^* n)$. ∎

Acknowledgements

It is a pleasure to thank Ambuj Singh and Jerry James for useful comments. Thanks are also due to two anonymous referees for their comments, which helped to significantly improve the readability of the paper.

References

[ADDJ] I. Althöfer, G. Das, D. Dobkin, and D. Joseph, Generating sparse spanners for weighted graphs, *Proc. 2nd Scandianvian Workshop on Algorithm Theory*, pp. 26–37, July 1990.

[A1] B. Awerbuch, Optimal distributed algorithms for minimum-weight spanning tree, counting, leader election and related problems, *Proc. 19th Symp. on Theory of Computing*, pp. 230–240, May 1987.

[A2] B. Awerbuch, Complexity of Network Synchronization, *J. of the ACM*, 32(4):804-823, October 1985.

[AGLP] B. Awerbuch, A. Goldberg, M. Luby and S. Plotkin, Network decomposition and locality in distributed computation, *Proc. 30th Symp. on Foundations of Computer Science*, pp. 364–375, October 1989.

[AGPV] Baruch Awerbuch, Oded Goldreich, David Peleg, and Ronen Vainish, A tradeoff between information and communication in broadcast protocols, *J. of the ACM*, 37(2):238–256, April 1990.

[AKP] B. Awerbuch, S. Kutten, and D. Peleg, Competitive Distributed Job Load Balancing, *Proc. 24th ACM Symp. on Theory of Computing*, pp. 571–580, 1992.

[AP1] B. Awerbuch and D. Peleg, Network synchronization with polylogarithmic overhead, 31^{st} *IEEE Symp. on Foundations of Computer Science*, pp. 514–522, October 1990.

[AP2] B. Awerbuch and D. Peleg, Concurrent online tracking of mobile users, *Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols*, pp. 221–233, September 1991.

[B] B. Bollobás. *Extremal Graph Theory*. Academic Press, 1978.

[CT] F. Chin and H. F. Ting, An Almost Linear Time and $O(n\log(n)+e)$ Messages Distributed Algorithm for Minimum-Weight Spanning Trees, *26th Symp. on Foundations of Computer Science*, Oct. 1985, pages 257–266.

[G] E. Gafni, Improvements in the time complexity of two message-optimal election algorithms, *Proc. 4th Symp. on Principles of Distributed Computing*, pp. 175–185, August 1985.

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

[GKP] J. Garay, S. Kutten and D. Peleg, A Sub-Linear Time Distributed Algorithm for Minimum-Weight Spanning Trees, 34^{th} *IEEE Symp. on Foundations of Computer Science*, pp. 659–668, November 1993.

[GPS] A. V. Goldberg, S. Plotkin, and G. Shannon, Parallel symmetry breaking in sparse graphs, *Proc. 19th ACM Symp. on Theory of Computing*, pp. 315–324, May 1987.

[L] N. Linial, Distributive graph algorithms - global solutions from local data, *Proc. 28th IEEE Symp. on Foundations of Computer Science*, pp. 331–335, October 1987.

[MRR] J.M. McQuillan, I. Richer and E.C. Rosen, The New Routing Algorithm for the ARPANET, *IEEE Trans. on Communications*, Vol. COM-28, No. 5, pp. 711-719, May 1980.

[P] D. Peleg, Time-optimal leader election in general networks, *Journal of Parallel and Distributed Computing*, Vol. 8, pp. 96–99, 1990.

[PS1] David Peleg and Alejandro A. Schäffer, Graph spanners, *J. of Graph Theory*, 13:99–116, 1989.

[PS2] A. Panconesi and A. Srinivasan, Improved distributed algorithms for coloring and network decomposition problems, *Proc. 24th Symp. on Theory of Computing*, pp. 581–592, 1992.

[T] R.E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, 1983.