Tight Bounds For Distributed MST Verification

Liah Kor[1], Amos Korman[2], and David Peleg[1]

1 **Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel.**
{liah.kor,david.peleg}@weizmann.ac.il
Supported by a grant from the United States-Israel Binational Science Foundation (BSF).
2 **CNRS and LIAFA, Univ. Paris 7, Paris, France.**
amos.korman@liafa.jussieu.fr
Supported in part by the ANR projects ALADDIN and PROSE, and by the INRIA project GANG.

Abstract

This paper establishes tight bounds for the Minimum-weight Spanning Tree (MST) verification problem in the distributed setting. Specifically, we provide an MST verification algorithm that achieves *simultaneously* $\tilde{O}(|E|)$ messages and $\tilde{O}(\sqrt{n}+D)$ time, where $|E|$ is the number of edges in the given graph G and D is G's diameter. On the negative side, we show that any MST verification algorithm must send $\Omega(|E|)$ messages and incur $\tilde{\Omega}(\sqrt{n}+D)$ time in worst case.

Our upper bound result appears to indicate that the verification of an MST may be easier than its construction, since for MST construction, both lower bounds of $\Omega(|E|)$ messages and $\Omega(\sqrt{n}+D)$ time hold, but at the moment there is no known distributed algorithm that constructs an MST and achieves *simultaneously* $\tilde{O}(|E|)$ messages and $\tilde{O}(\sqrt{n}+D)$ time. Specifically, the best known time-optimal algorithm (using $\tilde{O}(\sqrt{n}+D)$ time) requires $O(|E|+n^{3/2})$ messages, and the best known message-optimal algorithm (using $\tilde{O}(|E|)$ messages) requires $O(n)$ time. On the other hand, our lower bound results indicate that the verification of an MST is not significantly easier than its construction.

General Terms: Algorithms, Reliability, Theory, Verification.
Categories & Subject Descriptors:
C.2.2 [Network Protocols]: Protocol verification;
C.2.4 [Distributed Systems];
G.2.2[Discrete Mathematics]: Graph Theory: Graph algorithms, Graph labeling, Trees;
B.8.1[Performance and Reliability]: Reliability, Testing, and Fault-Tolerance;

keywords: Distributed algorithms, distributed verification, labeling schemes, minimum-weight spanning tree.

1 Introduction

1.1 Background and Motivation

The problem of efficiently computing a Minimum-weight Spanning Tree (MST) of a given weighted graph has been studied extensively in the centralized, parallel and distributed settings. Reviews on the problem in the centralized setting can be found, e.g., in the survey paper by Graham and Hell [17] or in the book by Tarjan [33] (Chapter 6). The fastest known algorithm for finding an MST is that of Pettie and Ramachandran [30], which runs in $O(|E| \cdot \alpha(|E|,n))$ time, where α is the inverse Ackermann function, n is the number of vertices and $|E|$ is the number of edges in the graph. Unfortunately, a linear (in the number

of edges) time algorithm for computing an MST is known only in certain cases, or by using randomization [14, 19].

The separation between computation and verification, and specifically, the question of whether verification is easier than computation, is a central issue of profound impact on the theory of computer science. In the context of MST, the verification problem (introduced by Tarjan [32]) is the following: given a weighted graph, together with a subgraph, it is required to decide whether this subgraph forms an MST of the graph. At the time it was published, the running time of the MST verification algorithm of [32] was indeed superior to the best known bound on the computational problem. Improved verification algorithms in different centralized models were then given by Harel [18], Komlòs [23], and Dixon, Rauch, and Tarjan [9], and parallel algorithms were presented by Dixon and Tarjan [10] and by King, Poon, Ramachandran, and Sinha [22]. Though it is not known whether there exists a deterministic algorithm that computes an MST in $O(|E|)$ time, the verification algorithm of [9] is in fact linear, i.e., runs in time $O(|E|)$ (the same result with a simpler algorithm was later presented by King [21] and by Buchsbaum [5]). For the centralized setting, this may indicate that the verification of an MST is indeed easier than its computation.

The problem of computing an MST received considerable attention in the distributed setting as well. Constructing such a tree distributively requires a collaborative computational effort by all the network vertices, and involves sending messages to remote vertices and waiting for their replies. The main measures considered for evaluating a distributed MST protocol are the *message* complexity, namely, the maximum number of messages sent in the worst case scenario, and the *time* complexity, namely, the maximum number of communication rounds required for the protocol's execution in the worst case scenario. The line of research on the distributed MST computation problem was initiated by the seminal work of Gallager, Humblet, and Spira [15] and culminated in the $O(n)$ time and $O(|E| + n \log n)$ messages algorithm by Awerbuch [2]. As pointed out in [2], the results of [4, 6, 13] establish an $\Omega(|E| + n \log n)$ lower bound on the number of messages required to construct a MST. Thus, the algorithm of [2] is essentially optimal.

This was the state of affairs until the mid-nineties when Garay, Kutten, and Peleg [16] initiated the analysis of the time complexity of MST construction as a function of additional parameters (other than n), and gave the first sublinear time distributed algorithm for the MST problem, running in time $O(D+n^{0.614})$, where D is the diameter of the network. This result was later improved to $O(D + \sqrt{n}\log^* n)$ by Kutten and Peleg [27]. The tightness of this latter bound was shown by Peleg and Rubinovich [29] who proved that $\tilde{\Omega}(\sqrt{n})$ is essentially[1] a lower bound on the time for constructing MST on graphs with diameter $\Omega(\log n)$. This result was complemented by the work of Lotker, Patt-Shamir and Peleg [28] that showed an $\tilde{\Omega}(\sqrt[3]{n})$ lower bound on the time required for MST construction on graphs with small diameter. Note, however, that the time-efficient algorithms of [16, 27] are not message-optimal, i.e., they take asymptotically much more than $O(|E| + n \log n)$ messages. For example, the time-optimal protocol of [27] requires sending $O(|E|+n^{3/2})$ messages. The question of whether there exists an optimal distributed algorithm for MST construction that achieves *simultaneously* $\tilde{O}(|E|)$ messages and $\tilde{O}(\sqrt{n} + D)$ time remains open.

This paper addresses the MST verification problem in the distributed setting. Here, a subgraph is given in a distributed manner, namely, some of the edges incident to every vertex are locally marked, and the collection of marked edges at all the vertices defines the subgraph; see, e.g., [7, 15, 24, 25]. The verification task requires checking distributively

[1] $\tilde{\Omega}$ (respectively, $\tilde{O}$) is a relaxed variant of the Ω (rep., O) notation that ignores polylog factors.

whether the marked subgraph is indeed an MST of the given graph. Similarly to the centralized setting, one of our major motivations is to investigate the relationships between the MST construction problem and the related verification problem. Moreover, from an applicative point of view, since faults are much more likely to occur in the distributed setting, the verification task in the distributed setting is more practically significant than in the centralized setting. Finally, we note that generally, investigating the "simpler" verification problem may lead to breakthroughs in the efforts for solving the corresponding construction problem. This was indeed useful in the centralized setting, where the verification algorithm of [9], that determines for each edge whether it "improves" a given MST candidate, was used as a subroutine for the subsequent MST construction algorithms of [19, 30].

In this paper, we present an MST verification algorithm that achieves simultaneously $\tilde{O}(|E|)$ messages and $\tilde{O}(\sqrt{n}+D)$ time. This result appears to indicate that MST verification may be easier than MST construction. Conversely, we show that the verification problem is not much easier, by proving that the known lower bounds for MST construction also hold for the verification problem. Specifically, we show that $\Omega(|E|)$ messages must be sent in worst case by any MST verification algorithm, and that $\tilde{\Omega}(\sqrt{n}+D)$ communication rounds are also required.

Our $\Omega(|E|)$ lower bound on the number of messages is fairly straightforward. The $\tilde{\Omega}(\sqrt{n}+D)$ time lower bound is achieved by a (somewhat involved) modification of the corresponding lower bound for the computational task [29]. Our verification algorithm builds upon techniques taken from the time optimal MST construction paper [27], and from the papers on labeling schemes [20, 24][2]. More specifically, after collecting some topological structure at some central vertex (in a way inspired by [27]), the algorithm avoids spreading all this information to all vertices, in order to save on messages. Instead, it carefully divides this information into pieces using the flow labeling scheme of [20, 24] and sends one piece of information to each vertex. This distribution of information is done in such a way that allows the verification to carry on using these pieces of information only.

1.2 The Model

A point-to-point communication network is modeled as an undirected graph $G(V, E)$, where the vertices in V represent the network processors and the edges in E represent the communication links connecting them. The length of a path in G is the number of edges it contains. The *distance* between two vertices u and v is the length of the shortest path connecting them. The *diameter* of G, denoted D, is the maximum distance between any two vertices of G.

Vertices are assumed to have unique identifiers, and each vertex knows its own identifier. The vertices do not know the topology or the edge weights of the entire network, but they know the weights of the edges incident to them. More precisely, a weight function $\omega : E \to \mathbb{N}$ associated with the graph assigns a nonnegative integer weight $\omega(e)$ to each edge $e = (u, v) \in E$. The weight $\omega(e)$ is known to the adjacent vertices, u and v. We assume that the edge weights are bounded by a polynomial in n (the number of vertices). The vertices can communicate only by sending and receiving messages over the communication

[2] The MST verification problem is considered in [24] from a different angle, closer to the notions of *local checking* [1, 3] or *computation with advice* [8, 11, 12]. Specifically, the focus therein is on the minimum size of a *label* (i.e., amount of information stored at a vertex) needed to allow verification of an MST in a single round, by exchanging the labels between neighboring vertices. The complexity of computing these labels are not in the scope of that paper.

links. Each vertex can distinguish between its incident edges. Moreover, if vertex v sends a message to vertex u along the edge $e = (v, u)$, then upon receiving the message, vertex u knows that the message was delivered over the edge e.

Similarly to [16, 27], we assume that the communication is carried out in a synchronous manner, i.e., all the vertices are driven by a global clock. Messages are sent at the beginning of each round, and are received at the end of the round. (Clearly, our lower bounds hold for asynchronous networks as well.) At most one B-bit message can be sent on each link in each direction on every round. Similarly to previous work, we assume that $B = O(\log n)$. The model also allows vertices to detect the absence of a message on a link at a given round, which can be used to convey information. Hence at each communication round, a link can be at one of $2^B + 1$ possible states, i.e., it can either transmit any of 2^B possible messages, or remain silent.

1.3 The distributed MST Verification problem

Formally, the *minimum-weight spanning tree (MST) verification* problem can be stated as follows. Given a graph $G(V, E)$, a weight function ω on the edges, and a subset of edges $T \subseteq E$, referred as the *MST candidate*, it is required to decide whether T forms a minimum spanning tree on G, i.e., a spanning tree whose total weight $w(T) = \sum_{e \in T} \omega(e)$ is minimal. In the distributed model, the input and output of the *MST verification problem* are represented as follows. Each vertex knows the weights of the edges connected to its immediate neighbors. A degree-d vertex $v \in V$ with neighbors $u_1, \ldots, u_d$ has d *weight variables* $W_1^v, \ldots, W_d^v$, with W_i^v containing the weight of the edge connecting v to u_i, i.e., $W_i^v = \omega(v, u_i)$, and d *boolean indicator variables* $Y_1^v, \ldots, Y_d^v$ indicating which of the edges adjacent to v participate in the MST candidate that we wish to verify. The indicator variables must be consistent, namely, for every edge (u, v), the indicator variables stored at u and v for this edge must agree (this is easy to verify locally). Similarly to previous work on MST construction, we assume that the verification algorithm is initiated by a designated source node. Let T_Y be the set of edges marked by the indicator variables (i.e., all edges for which the indicator variable is set to 1). The output of the algorithm at each vertex v is an assignment to a (boolean) output variable A^v that must satisfy $A^v = 1$ if T_Y is an MST of $G(V, E, \omega)$, and $A^v = 0$ otherwise.

1.4 Our Results

We establish asymptotically tight bounds for the time and message complexities of the MST verification problem. Specifically, in the positive direction we show the following:

▶ **Theorem 1.1.** *There exists a distributed MST verification algorithm that uses $\tilde{O}(\sqrt{n}+D)$ time and $\tilde{O}(|E|)$ messages.*

This upper bound is complemented by two lower bounds.

▶ **Theorem 1.2.** *Any distributed algorithm for MST verification requires $\Omega(|E|)$ messages.*

▶ **Theorem 1.3.** *Any distributed algorithm for MST verification requires $\tilde{\Omega}(\sqrt{n} + D)$ time.*

Theorem 1.2 is proved (in a rather straightforward manner) assuming that a vertex knows only its own identifier. We note, however, that it can be generalized to the model where each vertex knows also the identifiers of its neighbors, yielding $\Omega(|E|)$ lower bounds similar to those of [4]. Due to lack of space, the proof of Theorem 1.3 is deferred to the full paper.

2 An MST Verification Algorithm

2.1 Definitions and Notations

Following are some definitions and notations used in the description of the algorithm. For a graph $G = (V, E, w)$, an edge e is said to be *cycle-heavy* if there exists a cycle C in G that contains e, and e has the heaviest weight in C. For a graph $G = (V, E, w)$, a set of edges $F \subseteq E$ is said to be an *MST fragment* of G if there exists a minimum spanning tree T of G such that F is a subtree of T (i.e., $F \subseteq T$ and F is a tree). Similarly, a collection $\mathcal{F}$ of edge sets is referred to as an *MST fragment collection* of G if there exists an MST T of G such that (1) F_i is a subtree of T for every $F_i \in \mathcal{F}$, (2) $\bigcup_{F_i \in \mathcal{F}} V(F_i) = V$, and (3) $V(F_i) \cap V(F_j) = \emptyset$ for every $F_i, F_j \in \mathcal{F}$.

Consider a graph $G = (V, E, w)$, an MST fragment collection $\mathcal{F}$, a subgraph T of G and a vertex v in G. The *fragment graph* of G, denoted $G_{\mathcal{F}}$, is defined as a graph whose vertices are the MST fragments $F_i \in \mathcal{F}$, and whose edge set contains an edge (F_i, F_j) iff there exist vertices $u \in V(F_i)$ and $v \in V(F_j)$ such that $(u, v) \in E$. The *fragment graph induced by* T, denoted $T_{\mathcal{F}}$, is defined as a graph whose vertices are the MST fragments $F_i \in \mathcal{F}$, and whose edge set contains an edge (F_i, F_j) iff there exist vertices $u \in V(F_i)$ and $v \in V(F_j)$ such that $(u, v) \in T$. The edges of $T_{\mathcal{F}}$ are also referred to as the *inter-cluster* edges induced by T. For each fragment $F_i \in \mathcal{F}$, the set of *fragment internal* edges induced by T, denoted T_I^i, consists of all edges of T with both endpoints in $V(F_i)$, i.e., $T_I^i = \{e \mid e = (u, v) \in T \text{ and } u, v \in V(F_i)\}$. The fragment of v, denoted by $\mathcal{F}(v)$, is the fragment $F_i \in \mathcal{F}$ such that $v \in V(F_i)$. Denote by $E_{\mathcal{F}}(v)$ the set of fragment internal edges that are incident to v (i.e., $E_{\mathcal{F}}(v) = \{e| \ e = (u, v) \in E \text{ and } u \in V(\mathcal{F}(v))\}$). Similarly, denote by $E_{T_I}(v)$ the set of fragment internal edges induced by T and incident to v.

Throughout the description of the verification algorithm we assume that the edge weights are distinct. Having distinct edge weights simplifies our arguments since it guarantees the uniqueness of the MST. Yet, this assumption is not essential. Alternatively, in case the graph is not guaranteed to have distinct edge weights, we may use a modified weight function $\omega'(e)$, which orders edge weights lexicographically: first, by their original weight $\omega(e)$, then, by the indicator variable of the edge, and finally, by the identifiers of the edge endpoints. Under the weight function $\omega'(e)$, edges with indicator variable set to "true" will have lighter weight than edges with the same weight under $\omega(e)$ but with indicator variable set to "false" (i.e., for edges $e_1 \in T$ and $e_2 \notin T$ such that $w(e_1) = w(e_2)$, we have $w'(e_1) < w'(e_2)$). It follows that the given subgraph T is an MST of $G(V, E, \omega)$ iff T is an MST of $G(V, E, \omega')$. Moreover, since $\omega'(\cdot)$ takes into account the unique vertex identifiers, it assigns distinct edge weights.

The MST verification algorithm makes use of Procedures `DOM_Part` and `MAX_Label`, presented in [27] and [20] respectively. The distributed Procedure `DOM_Part`, used in [27], partitions a given graph into an *MST fragment collection (MFC)* $\mathcal{F}$, where each fragment is of size at least $k + 1$ and depth $O(k)$, for a specified parameter k. A *fragment leader* vertex is associated with each constructed fragment (the identifier of the fragment is the identifier of the fragment's leader). After Procedure `DOM_Part` is completed, each vertex v knows the identifier of the fragment to which it belongs and v's incident edges that belong to the fragment. (To abide by the assumption of [27] that each vertex knows the identifiers of its neighbors, before applying Procedure `DOM_Part`, the algorithm performs a single communication round that exchanges vertex identifiers between neighboring vertices.)

The labeling scheme `MAX_Label` of [20] which is designed for the family of weighted trees constructs an encoder algorithm $\mathcal{E}$ and a decoder algorithm $\mathcal{D}$ that satisfy the following: **1.** Given a weighted tree T, the encoder algorithm $\mathcal{E}$ assigns a label $L(v)$ to each vertex v of T.

2. Given two labels $L(u), L(v)$ assigned by $\mathcal{E}$ to two vertices u and v in some weighted tree T, the decoder algorithm $\mathcal{D}$ outputs $MAX(u, v)$, which is the maximum weight of an edge on the path connecting u and v in T. (The decoder algorithm $\mathcal{D}$ bases its answer on the pair of labels $L(u), L(v)$ only, without knowing any additional information regarding the tree T.) The labeling scheme `MAX_Label` produces $O(\log n \log W)$-bit labels, where W is the largest weight of an edge. Since W is assumed to be polynomial in n, the label size is $O(\log^2 n)$ bit.

2.2 The algorithm

The algorithm consists of three phases. The first phase starts by running the distributed Procedure `DOM_Part` of [27], which constructs an MST fragment collection (MFC) $\mathcal{F}$ for a fixed parameter k that will be specified later. The algorithm verifies that the set of edges contained in the constructed MFC is equal to the set of fragment internal edges induced by the MST candidate T, namely, $\bigcup_{F_i \in \mathcal{F}} F_i = \bigcup_{F_i \in \mathcal{F}} T_I^i$ (note that this is a necessary condition for correctness since the graph is assumed to have a unique MST).

In the second and third phases, the algorithm verifies that all remaining edges of T form an MST on the fragment graph $G_{\mathcal{F}}$. Let $T_{\mathcal{F}}$ be the fragment graph induced by T with respect to the MFC $\mathcal{F}$ found in the previous phase. In order to verify that $T_{\mathcal{F}}$ forms an MST on the fragment graph $G_{\mathcal{F}}$, it suffices to verify that $T_{\mathcal{F}}$ is a tree and that none of the edges of $T_{\mathcal{F}}$ is a *cycle heavy* edge in $G_{\mathcal{F}}$. The above is done by using the labeling scheme ($\mathcal{E}$,$\mathcal{D}$) of [20] (or [24]) on $T_{\mathcal{F}}$. Informally, the algorithm assigns a label $L(F_i)$ to each vertex F_i of $T_{\mathcal{F}}$ using the encoder algorithm $\mathcal{E}$ applied on $T_{\mathcal{F}}$. The label $L(F_i)$ is then efficiently delivered to each vertex in F_i. Recall, that given the labels of two fragments $L(F_i), L(F_j)$ it is now possible to compute the weight of the heaviest edge on the path connecting the fragments in $T_{\mathcal{F}}$ by applying the decoder algorithm $\mathcal{D}$. Once all vertices obtain the labels of their corresponding fragments, each vertex of G can verify (using the decoder $\mathcal{D}$) that each inter fragment edge incident to it and not participating in $T_{\mathcal{F}}$ forms a cycle when added to $T_{\mathcal{F}}$ for which it is a cycle heavy edge. Following is a more detailed description of the algorithm.

1. **a.** The source vertex s (that initiates the algorithm) constructs a BFS tree rooted at s, computes the values n and D and broadcasts a signal on the BFS tree instructing each vertex to send its identifier to all its neighbors.
   b. Perform Procedure `DOM_Part`(k), where k is specified later. (The result is an MFC $\mathcal{F}$, where each fragment $F \in \mathcal{F}$ is of size $|V(F)| > k$ and depth $O(k)$, having a fragment leader and a distinct fragment identifier known to all vertices in the fragment).
   c. Each vertex sends its fragment identifier to all its neighbors.
   d. Each vertex v identifies the sets of edges $E_{\mathcal{F}}(v)$ and $E_{T_I}(v)$.
   e. Verify that $\bigcup_{F_i \in \mathcal{F}} F_i = \bigcup_{F_i \in \mathcal{F}} T_I^i$ by verifying at each vertex v that $E_{\mathcal{F}}(v) \subseteq T$ and $E_{T_I}(v) \subseteq \mathcal{F}(v)$. (Else return "$T$ is not an MST".)
2. **a.** Vertex s counts the number of fragments by performing a convergecast on the BFS tree (only fragment leader vertices increase the counter of the convergecast). Let f be the number of fragments.
   b. Vertex s counts the number of inter fragment edges induced by T (i.e., the number of edges in $T_{\mathcal{F}}$) by performing a convergecast on the BFS tree. Then, Vertex s verifies that the number of edges is equal to $f - 1$. (Else return "T is not an MST".)
   c. All vertices send the description of all edges in $T_{\mathcal{F}}$ to s, by performing an upcast on the BFS tree. (The edges of $T_{\mathcal{F}}$ are all edges of T that connect vertices from different fragments.)

d. Vertex s verifies that $T_{\mathcal{F}}$ is a tree. (Else return "T is not an MST".)

3. **a.** Each fragment leader sends a message to vertex s over the BFS tree. Following these messages, all vertices save routing information regarding the fragment leaders. I.e., if v is a fragment leader and v is a descendant of some other vertex u in the BFS tree, then, after this step is applied, vertex u knows which of its children is on the path connecting it to the fragment leader v.

   b. Vertex s computes the labels $L(F_i)$ for all vertices F_i in $T_{\mathcal{F}}$ (i.e., assigns a label to each of the fragments) using the encoder algorithm $\mathcal{E}$. Subsequently, vertex s sends to each fragment leader its fragment label (the label of each fragment is sent to the fragment leader over the BFS tree using the routing information established in Step 3a above). Recall, each label is encoded using $O(\log^2 n)$ bits.

   c. Each fragment leader broadcasts its fragment label to all vertices in the fragment. The broadcast is done over the fragment edges.

   d. Each vertex sends its fragment label to all its neighbors.

   e. Each vertex v verifies for every neighbor u not belonging to v's fragment, and s.t. $(u,v) \notin T$, that $w(v,u) \geq MAX(\mathcal{F}(v), \mathcal{F}(u))$ (the value $MAX(F(v), F(u))$ is computed by applying the decoder algorithm $\mathcal{D}$ on labels $L(\mathcal{F}(v)), L(\mathcal{F}(u))$). (Else return "$T$ is not an MST".)

2.3 Complexity

Steps 1a and 1c clearly take $O(D)$ time and $O(|E|)$ messages. Step 1b, i.e., the execution of Procedure `DOM_Part`, requires $O(k \cdot \log^* n)$ time and $O(E \cdot \log k + n \cdot \log^* n \cdot \log k)$ messages. (A full analysis appears in [31].) The remaining steps of the first phase are local computations performed by all vertices. Thus, the first phase of the MST verification algorithm requires $O(D + k \cdot \log^* n)$ time and $O(E \cdot \log k + n \cdot \log^* n \cdot \log k)$ messages.

Since the fragments are disjoint and each fragment contains at least k vertices, the number of MST fragments constructed during the first phase of the algorithm is $f \leq n/k$. The table below summarizes the time and message complexities of the second and third phases of the algorithm.

Step	Description	Time	Messages
2a,2b	BFS convergecast	$O(D)$	$O(\|E\|)$
3d	Communication between neighbors	$O(\log n)$	$O(\log n \cdot \|E\|)$
2d,3e	Local computation	0	none
2c,3a	BFS upcast of f messages	$O(D + f)$	$O(f \cdot D)$
3b	BFS downcast of f messages (each of size $\log^2 n$)	$O(D + f \cdot \log n)$	$O(f \cdot \log n \cdot D)$
3c	Broadcast in each of the MST fragments	$O(k + \log n)$	$O(\log n \cdot n)$

Combining the above arguments, we get that the algorithm requires time $O(\frac{n}{k} \cdot \log n + k \cdot \log^* n + D)$ and $O(E \cdot \log n + n \cdot \log^* n \cdot \log k + \frac{n}{k} \cdot \log n \cdot D + n \cdot \log n)$ messages. Recall that after Step 1a is applied, the source vertex s knows the number of nodes n and the diameter D. By choosing $k = \sqrt{n}$ in case $D < \sqrt{n}$ and $k = D$ otherwise, we get the following, implying Theorem 1.1.

▶ **Lemma 2.1.** *The MST verification algorithm requires $O(\sqrt{n} \cdot \log n + D)$ time and $(|E| \cdot \log n)$ messages.*

2.4 Correctness

We now show that our MST verification algorithm correctly identifies whether the given tree T is an MST. We begin with the following claim.

▶ Claim 2.2. Let T be a spanning tree of G such that T contains all edges of the MFC $\mathcal{F}$ and $T_{\mathcal{F}}$ forms an MST on the fragment graph of G (with respect to the MFC $\mathcal{F}$). Then T is an MST on G.

Proof. Since $\mathcal{F}$ is an MST Fragment collection, there exists an MST T' of G such that T' contains all edges of $\mathcal{F}$. Due to the minimality of T', the fragment graph $T'_{\mathcal{F}}$ induced by T' necessarily forms an MST on the fragment graph of G (with respect to the MFC $\mathcal{F}$). Hence we get that $w(T) = w(T')$, thus T is an MST of G. ◀

Due to the assumption that edge weights are distinct, we get:

▶ Observation 2.3. The MST of G is unique.

By combining Claim 2.2 and Observation 2.3 we get the following.

▶ Claim 2.4. A spanning tree T of G is an MST if and only if T contains all edges of the MFC $\mathcal{F}$ and $T_{\mathcal{F}}$ forms an MST on the fragment graph of G (with respect to the MFC $\mathcal{F}$)

▶ **Lemma 2.5.** *The MST verification algorithm correctly identifies whether the given tree T is an MST of the graph G.*

Proof. By Claim 2.4, to prove the correctness of the algorithm it suffices to show that given an MST candidate T, the algorithm verifies that:
(1) T is a spanning tree of G,
(2) T contains all edges of $\mathcal{F}$, and
(3) $T_{\mathcal{F}}$ forms an MST on the fragment graph of G with respect to the MFC $\mathcal{F}$.

Since $\mathcal{F}$ as constructed by Procedure `DOM_Part` in the first phase is an MFC, it spans all vertices of G. Step 1e verifies that $\bigcup_{F_i \in \mathcal{F}} F_i = \bigcup_{F_i \in \mathcal{F}} T_I^i$, thus after this step, it is verified that T does not contain a cycle that is fully contained in some fragment $F_i \in \mathcal{F}$ (since every $F_i \in \mathcal{F}$ is a tree). On the other hand, step 2d verifies that T does not contain a cycle that contains vertices from different fragments. Hence, the algorithm indeed verifies that T is a spanning tree of G, and Property (1) follows. Property (2) is clearly verified by step 1e of the algorithm. Finally, to verify that $T_{\mathcal{F}}$ forms an MST on the fragment graph of G it suffices to verify that inter-fragment edges not in $T_{\mathcal{F}}$ are cycle heavy, which is done in step 3e. Property (3) follows. ◀

3 Message Complexity Lower Bound

We prove a message complexity lower bound of $\Omega(|E|)$ on the *Spanning Tree (ST) verification* problem, which is a relaxed version of the MST verification problem defined as follows. Given a graph $G = (V, E, \omega)$ and a subgraph T (referred to as the *ST candidate*), we wish to decide whether T is a spanning tree of G (not necessarily of minimal weight). Clearly, a lower bound on *ST verification* problem also applies to the *MST verification* problem. Since spanning tree verification is independent of the edge weights, for convenience we consider unweighted networks throughout the lower bound proof.

We begin with a few definitions. A *protocol* is a local program executed by all vertices in the network. In every step, each processor performs local computations, sends and receives messages, and changes its state according to the instructions of the protocol. A protocol

achieving a given task should work on every network G and every ST candidate T. Following [4], we denote the *execution* of protocol Π on network $G(V,E)$ with ST candidate T by $EX(\Pi,G,T)$. The *message history* of an execution $EX = EX(\Pi,G,T)$ is a sequence describing the messages sent during the execution EX. Consider a protocol Π, two graphs $G_0(V,E_0)$ and $G_1(V,E_1)$ over the same set of vertices V, and two ST candidates T_0 and T_1 for G_0 and G_1 respectively, and the corresponding executions $EX_0 = EX(\Pi,G_0,T_0)$ and $EX_1 = EX(\Pi,G_0,T_1)$. We say that the executions are *similar* if their message history is identical.

Let $G = (V,E)$ be a graph (together with an assignment of vertex identifiers), T be a subgraph and $e = (u,v)$ be one of its edges. Let $G' = (V',E')$ be some copy of $G = (V,E)$, where the identifiers of the vertices in V' are not only pairwise distinct but also distinct from the given identifiers on V. Consider the following graphs G^2 and G^e and the subgraph T^2.
• Graph G^2 is simply $G^2 = (V^2,E^2) = G \cup G' = (V \cup V', E \cup E')$. The subgraph T^2 of G^2 is defined as the union of the two copies of T, one in G and the other in G'.
• The graph G^e is a "cross-wired" version of G^2. Formally, $G^e = (V^2,E^e)$, where $E^e = E^2 \setminus \{(u,v),(u',v')\} \cup \{(u,v'),(u',v)\}$. (Observe, for $e \notin T$, T^2 is also a subgraph of G^e.)

Let Π be a protocol that correctly solves ST verification problem. Fix G to be some arbitrary graph, fix a copy G' of G, fix a spanning tree T of G, and fix a source vertex $s \in V$ initiating the execution of Π on either of the graphs G, G^2 and G^e with the ST candidates T, T^2 and T^2, respectively. We stress that G^2 (with candidate T^2) is not a valid input for the ST (or the MST) verification problem since it is not connected. Still, we can consider the execution $EX(\Pi,G^2,T^2)$, without requiring anything from its output.

▶ **Lemma 3.1.** *Let $e \in E \setminus E(T)$, such that no message is sent over the edges e and e' in execution $EX(\Pi,G^2,T^2)$. Then executions $EX(\Pi,G^2,T^2)$ and $EX(\Pi,G^e,T^2)$ are similar.*

Proof. Proof Sketch of lemma 3.1. We show that in both executions each vertex sends and receives identical sequences of messages in each communication round of the protocol. Note that at each round the messages sent by some vertex w is dependent on w's topological view (neighbors of w), w's initial input (the indicator variables of the edges incident to w), and the set of messages sent and received by w in previous communication rounds. Denote by EX^2 and EX^e executions $EX(\Pi,G^2,T^2)$ and $EX(\Pi,G^e,T^2)$ respectively. Note that any vertex $w \in V^2 \setminus \{u,v,u',v'\}$ has identical topological view and identical initial input in both executions. Vertex u has identical initial input and identical number of neighbors in both executions. Though the communication link connecting u to v in G^2 connects u to v' in G^e, vertex u is initially unaware of this difference between the executions since it does not know the identifiers of its neighbors. (Same holds for vertices v, u' and v'.) The proof is by induction on r, the number of communication rounds of protocol Π.
Induction base: For $r = 0$. In the first communication round, the messages sent by each vertex depend solely on its topological view and initial input. Let us analyze the sequence of messages sent by vertices in V (the vertices of graphs G^2 and G^e that belong to the first copy of G). Following are the possible cases.
Vertex $w \notin \{u,v\}$: Vertex w has identical topological view and identical initial input in both execution, thus it sends identical sequence of messages in the first round of both executions.
Vertex u: As mentioned above, although in execution EX^e vertex u is connected to v' instead of v, it has no knowledge of this difference. Thus u sends identical sequence of messages over each of its communication links. The fact that no messages are sent over edge e in execution EX^2, implies that in execution EX^e no message is sent by u to its neighbor v'. Thus, u sends identical sequence of messages in the first communication round of both

executions.
Vertex v: can be analyzed in the same manner as vertex u.
The above shows that vertices in V send the same sequence of messages in the first communication round of both executions. The induction base claim follows by applying the same argument on vertices of V'.

Inductive step: Can be shown using a similar case analysis as in the induction base. ◀

Theorem 1.2 follows as a consequence of the following Lemma.

▶ **Lemma 3.2.** *Execution* $EX(\Pi, G^2, T^2)$ *requires* $\Omega(|E^2 \setminus T^2|)$ *messages.*

Proof. Assume, towards contradiction, that there exists a protocol Π that correctly solves the ST verification problem for every graph G and ST candidate T, such that execution $EX(\Pi, G, T)$ sends fewer than $|E \setminus T|/2$ messages over edges from $E \setminus T$.

For the rest of the proof we fix $G = (V, E)$ to be an arbitrary connected graph and denote the ST candidate by T. We take T to be a spanning tree and not just any subgraph. (See Figure 1).

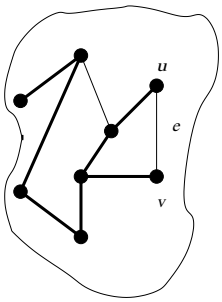

Figure 1 Graph G with ST candidate T (the bold edges belong to T)

Consider the graph G^2 as previously defined with ST candidate $T^2 = \{e \in T\} \cup \{(u', v') : e = (u, v) \in T\}$ (See Figure 2).

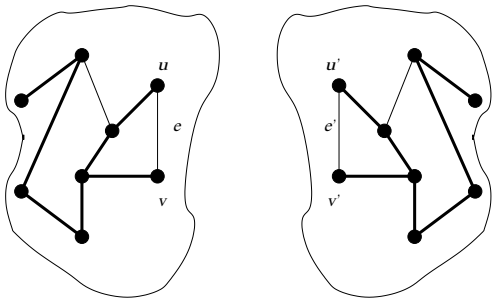

Figure 2 Graph G^2 with ST candidate T^2 (the bold edges belong to T^2)

Then by the assumption on Π, execution $EX^2 = EX(\Pi, G^2, T^2)$ sends fewer than $|E^2 \setminus T^2|/2$ messages over edges from $E^2 \setminus T^2$. Hence there exist $e = (u, v)$ and $e' = (u', v')$ such that $e, e' \in E^2 \setminus T^2$ and no message is sent over e and e' in execution EX^2. Consider the graph G^e with ST candidate T^2 as previously defined (See Figure 3).

By Lemma 3.1, executions EX^2 and $EX^e = EX(\Pi, G^e, T^2)$ are similar. Note that $e, e' \notin T^2$, thus T^2 is not a spanning tree of G^e (since the two copies of G contained in G^e

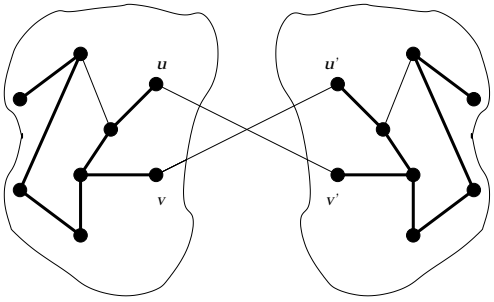

Figure 3 Graph G^e with ST candidate T^2 (the bold edges belong to T^2)

are connected solely by edges e and e'). Since Π correctly solves the ST verification problem, the output of all vertices in EX^e is "0" (i.e., the given ST candidate T^2 is not a spanning tree of the graph G^e). On the other hand, consider the execution $EX = (\Pi, G, T)$ with ST candidate T. Note that EX is exactly the restriction of EX^2 on the first copy of G contained in G^2. Since G^2 contains two disconnected copies of G the output of all vertices in execution EX^2 will be identical to the output of the same vertices in EX (since in both executions the vertices have identical topological view and the input variables contain identical values). Since executions EX^2 and EX^e are similar, the output of EX is "0", in contradiction to the correctness of Π. ◀

References

1. Y. Afek, S. Kutten, and M. Yung. The local detection paradigm and its applications to self stabilization. *Theoretical Computer Science*, 186(1-2):199–230, 1997.
2. B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proc. 19th ACM Symp. on Theory of computing (STOC)*, pages 230–240, New York, NY, USA, 1987. ACM.
3. B. Awerbuch and G. Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In *Proc. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 258–267, 1991.
4. B. Awerbuch, O. Goldreich, R. Vainish, and D. Peleg. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37(2):238–256, 1990.
5. A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, Robert E. Tarjan, and Jeffery R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008.
6. J. E. Burns. A formal model for message passing systems. Technical Report TR-91, Computer Science Dept., Indiana University, Bloomington, 1980.
7. I. Cidon, I. Gopal, M. Kaplan, and S. Kutten. A distributed control architecture of high-speed networks. *IEEE Trans. on Communications*, 43(5):1950–1960, 1995.
8. R. Cohen, P. Fraigniaud, D Ilcinkas, A Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. *ACM Trans. on Algorithms*, 4(4), 2008.
9. B. Dixon, M. Rauch, and R.E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992.
10. B. Dixon and R.E. Tarjan. Optimal parallel verification of minimum spanning trees in logarithmic time. *Algorithmica*, 17(1):11–18, 1997.

11 P. Fraigniaud, D Ilcinkas, and A. Pelc. Communication algorithms with advice. *J. Comput. Syst. Sci.*, 76(3-4):222–232, 2008.

12 P. Fraigniaud, A Korman, and E. Lebhar. Local mst computation with short advice. In *Proc. SPAA*, pages 154–160, 2007.

13 G. N. Frederickson and N. A. Lynch. The impact of synchronous communication on the problem of electing a leader in a ring. In *Proc. 16th ACM Symp. on Theory of computing (STOC)*, 493–503, (1984).

14 M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Proc. 31st IEEE FOCS*, pages 719–725, 1990.

15 R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

16 J. Garay, S. A. Kutten, and D. Peleg. A sub-linear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing*, 27(1):302–316, 1998.

17 R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Ann. Hist. Comput.*, 7(1):43–47, 1985.

18 D. Harel. A linear time algorithm for finding dominators in flow graphs and related problems. In *Proc. 17th ACM Symp. on Theory of computing (STOC)*, pages 185–194, 1985.

19 D.R Karger, P.N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, 1995.

20 M. Katz, N.A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2005.

21 V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18:263–270, 1997.

22 V. King, C.K Poon, V Ramachandran, and S. Sinha. An optimal erew pram algorithm for minimum spanning tree verification. *Information Processing Letters*, 62(3):153–159, 1997.

23 J. Komlòs. Linear verification for spanning trees. *Combinatorica*, 5:57–65, 1985.

24 A. Korman and S. Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007.

25 A. Korman, S. Kutten, and D Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010.

26 E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, New York, NY, USA, 1997.

27 S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

28 Z. Lotker, B. Patt-Shamir, and D. Peleg. Distributed mst for constant diameter graphs. In *Proc. 20th ACM Symp. on Principles of distributed computing (PODC)*, 63–71, (2001).

29 D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.

30 S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002.

31 V. Rubinovich. Distributed minimum spanning tree construction. Master's thesis, Department of CS. and Applied Mathematics, The Weizmann Institute of Science, 1999.

32 R.E. Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26:690–715, 1979.

33 R.E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.