Tight Bounds for Distributed Minimum-Weight Spanning Tree Verification

Liah Kor, Amos Korman, David Peleg

Tight Bounds For Distributed Minimum-weight Spanning Tree Verification

Liah Kor * Amos Korman † David Peleg *

Abstract

This paper introduces the notion of distributed verification without preprocessing. It focuses on the Minimum-weight Spanning Tree (MST) verification problem and establishes tight upper and lower bounds for the time and message complexities of this problem. Specifically, we provide an MST verification algorithm that achieves *simultaneously* $\tilde{O}(m)$ messages and $\tilde{O}(\sqrt{n}+D)$ time, where m is the number of edges in the given graph G, n is the number of nodes, and D is G's diameter. On the other hand, we show that any MST verification algorithm must send $\tilde{\Omega}(m)$ 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 $\tilde{\Omega}(m)$ messages and $\tilde{\Omega}(\sqrt{n}+D)$ time hold, but at the moment there is no known distributed algorithm that constructs an MST and achieves *simultaneously* $\tilde{O}(m)$ messages and $\tilde{O}(\sqrt{n}+D)$ time. Specifically, the best known time-optimal algorithm (using $\tilde{O}(\sqrt{n}+D)$ time) requires $O(m+n^{3/2})$ messages, and the best known message-optimal algorithm (using $\tilde{O}(m)$ 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.

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

1 Introduction

1.1 Background and Motivation

The problem of computing a Minimum-weight Spanning Tree (MST) received considerable attention in both the distributed setting as well as in the centralized setting. In the distributed setting, 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. This is particularly interesting in the $\mathcal{CONGEST}$ model of computation, where due to congestion constrains, each message can encode only a limited number of bits, specifically, this number is typically assumed to be $O(\log n)$, where n denotes the number of nodes in the network. The main measures considered for evaluating a distributed MST protocol are the *message* complexity, namely,

*Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. E-mail: {liah.kor,david.peleg}@weizmann.ac.il. Supported by a grant from the United States-Israel Binational Science Foundation (BSF).

†CNRS and LIAFA, Univ. Paris 7, Paris, France. E-mail: amos.korman@liafa.univ-paris-diderot.fr. Supported by the ANR project DISPLEXITY, and by the INRIA project GANG.

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 [17] and culminated in the $O(n)$ time and $O(m + n \log n)$ messages algorithm by Awerbuch [2], where m denotes the number of edges in the network. Both of these algorithms apply also to asynchronous systems. As pointed out in [2], the results of [4, 6, 15] establish an $\Omega(m + n \log n)$ 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 [18] 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 [29]. The tightness of this latter bound was shown by Peleg and Rubinovich [32] 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 [31] 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 [18, 29] are not message-optimal, i.e., they take asymptotically much more than $O(m + n \log n)$ messages. For example, the time-optimal protocol of [29] requires sending $O(m + n^{3/2})$ messages. The question of whether there exists an optimal distributed algorithm for MST construction that achieves *simultaneously* $\tilde{O}(m)$ messages and $\tilde{O}(\sqrt{n} + D)$ time still remains open.

This paper addresses the MST verification problem in the $\mathcal{CONGEST}$ model. Informally, the setting we consider is as follows. 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 a *marked subgraph*; see, e.g., [7, 17, 26, 27]. The verification task requires checking distributively whether the marked subgraph is indeed an MST of the given graph. Similarly to the papers dealing with sub-linear time MST construction [18, 29], we consider a synchronous environment.

1.2 Our Results

We consider the MST verification problem and establish asymptotically tight upper and lower bounds for the time and message complexities of this 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 $\tilde{O}(m)$ messages.*

This result appears to indicate that MST verification may be easier than MST construction, since, at the moment, it is not known whether there exists an algorithm that constructs an MST simultaneously in $\tilde{O}(\sqrt{n} + D)$ time and $\tilde{O}(m)$ messages. Conversely, we show that the verification problem is not much easier than the construction, by proving that the known lower bounds for MST construction also hold for the verification problem. Specifically, we prove the following two matching lower bounds.

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

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

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

To the best of our knowledge, this paper is the first to investigate distributed verification without assuming any kind of preprocessing (see Section 1.3.2). Following this paper, several other works on distributed verification have already been published. More specifically, a systematic study of distributed verification is established for various verification tasks [8]. Distributed verification in the $\mathcal{LOCAL}$ model has been studied in [13, 14], mostly focusing on computational complexity issues.

Our $\tilde{\Omega}(\sqrt{n}+D)$ time lower bound for verifying an MST is achieved by a (somewhat involved) modification of the corresponding lower bound for the computational task [32]. The idea behind our time lower bound was already found useful in several continuation studies. Specifically, a modification of our proof technique was used in [8] to yield time lower bounds for several other verification tasks, and for establishing a lower bound on the hardness of approximating an MST. Somewhat surprisingly, in [9], the proof technique was extended even further to yield a result in the (seemingly unrelated) area of distributed random walks.

1.3 Other related work

1.3.1 MST computation and verification in the centralized setting

In the centralized setting, there is a large body of literature concerning the problem of efficiently computing an MST of a given weighted graph. Reviews can be found, e.g., in the survey paper by Graham and Hell [19] or in the book by Tarjan [36] (Chapter 6). The fastest known algorithm for finding an MST is that of Pettie and Ramachandran [33], which runs in $O(m \cdot \alpha(m,n))$ time, where α is the inverse Ackermann function, n is the number of vertices and m 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 [16, 21].

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 [35]) 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 [35] was indeed superior to the best known bound on the computational problem. Improved verification algorithms in different centralized models were then given by Harel [20], Komlòs [25], and Dixon, Rauch, and Tarjan [10], and parallel algorithms were presented by Dixon and Tarjan [11] and by King, Poon, Ramachandran, and Sinha [24]. Even though it is not known whether there exists a deterministic algorithm that computes an MST in $O(m)$ time, the verification algorithm of [10] is in fact linear, i.e., runs in time $O(m)$ (the same result with a simpler algorithm was later presented by King [23] and by Buchsbaum [5]). For the centralized setting, this may indicate that the verification of an MST is indeed easier than its computation.

1.3.2 Distributed verification with preprocessing

Some previous papers investigated distributed verification tasks assuming that the algorithm designer can perform a preprocessing stage to help the verification, cf. [1, 3, 12, 26, 27]. Typically, in this preprocessing stage, data structures (i.e., labels, proofs) are provided to the nodes of the graph, and using these data structures, the verification proceeds in a constant number of rounds. The focus in those studies was on the minimum size of a data structure (i.e., amount of information

stored locally), while the complexities of the preprocessing stage providing the data structures were not analyzed.

2 Preliminaries

2.1 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. We denote by n the number of vertices in G, i.e., $n = |V|$, and let m denote the number of edges, i.e., $m = |E|$. 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 v knows its own identifier $\texttt{ID}(v)$. 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 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, that is, the weight $\omega((u, v))$ is known to u and v. Similarly to corresponding works on MST computation, we assume that the edge weights are bounded by a polynomial in n (this assumption implies that a weight of an edge can be encoded using $O(\log n)$ bits, and hence can be encoded in one message).

Similarly to previous work, we consider the $\mathcal{CONGEST}$ model. Specifically, the vertices can communicate only by sending and receiving messages over the communication 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. At most one $O(\log n)$ bits can be sent on each link in each message. Similarly to [18, 29], 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.) Relevant studies typically assume that computations start either at a single *source* node or simultaneously at all nodes. Our results hold for both of these settings.

2.2 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 $\omega(T) = \sum_{e \in T} \omega(e)$ 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). 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.

3 An MST Verification Algorithm

In this section we describe our MST verification algorithm, prove its correctness and analyze its time and message complexities.

First, we note that in this section we assume that the verification algorithm is initiated by a designated source node. The case in which all nodes wake up simultaneously can be reduced to the single source setting using the leader election algorithm of [30], which employs $\tilde{O}(m)$ messages and runs in $\tilde{O}(D)$ time.

3.1 Definitions and Notations

Following are some definitions and notations used in the description of the algorithm. For a graph $G = (V, E, \omega)$, 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 $G = (V, E, \omega)$, 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 $F_i, F_j \in \mathcal{F}$, where $i \neq j$.

Consider a graph $G = (V, E, \omega)$, 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) if and only if there exist vertices $u \in V(F_i)$ and $v \in V(F_j)$ such that $(u, v) \in E$. Similarly, 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) for $i \neq j$ if and only if 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 vertex v, let $E_T(v)$ denote the set of edges of T incident to v. For each fragment $F \in \mathcal{F}$, the set of *fragment internal* edges of F induced by T, denoted $E_{F,T}$, consists of all edges of T with both endpoints in $V(F)$, i.e., $E_{F,T} = \{e \mid e = (u, v) \in T \text{ and } u, v \in V(F)\}$. The fragment of v, denoted by $F(v)$, is the fragment $F \in \mathcal{F}$ such that $v \in V(F)$. Denote by $E_F(v)$ the set of edges in $F(v)$ that are incident to v (i.e., $E_F(v) = \{e| \ e = (u, v) \in E \text{ and } u \in V(F(v))\}$). Similarly, denote by $E_{F,T}(v)$ the set of fragment internal edges of F 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. At this point we would like to note that the standard technique (e.g., [17]) for obtaining unique weights is not sufficient for our purposes. Indeed, that technique orders edge weights lexicographically: first, by their original weight $\omega(e)$, and then, by the identifiers of the edge endpoints (say, first comparing the smaller of the two identifiers of the endpoints, and then the larger one). This yields a modified graph with unique edge weights, and an MST of the modified graph is necessarily an MST of the original graph. For construction purposes it is therefore sufficient to consider only the modified graph. However, this is not the case for verification purposes, as the given subgraph can be an MST of the original graph but not necessarily an MST

of the modified graph.

While we cannot guarantee than any MST of the original graph is an MST of the modified graph (having unique edge weights), we make sure that the particular given subgraph T is an MST of the original graph if and only if it is an MST of modified one. This condition is sufficient for our purposes, and allows us to consider only the modified graph. Specifically, to obtain the modified graph, we employ a slightly different technique than the classical one. That is, edge weights are lexicographically ordered as follows. For an edge $e = (v, u_i)$ connecting v to its ith neighbor u_i, consider first its original weight $\omega(e)$, next, the value $1 - Y_i^v$ where Y_i^v is the indicator variable of the edge e (indicating whether e belongs to the candidate MST to be verified), and finally, the identifiers of the edge endpoints, $\texttt{ID}(v)$ and $\texttt{ID}(u_i)$ (say, first comparing the smaller of the two identifiers of the endpoints, and then the larger one). Formally, let

$$\omega'(e) \;=\; \langle \omega(e), 1 - Y_i^v, \texttt{ID}_{min}(e), \texttt{ID}_{max}(e) \rangle \;\;,$$

where $\texttt{ID}_{min}(e) = \min\{\texttt{ID}(v), \texttt{ID}(u_i)\}$ and $\texttt{ID}_{max}(e) = \max\{\texttt{ID}(v), \texttt{ID}(u_i)\}$. Under this weight function $\omega'(e)$, edges with indicator variable set to 1 will have lighter weight than edges with the same weight under $\omega(e)$ but with indicator variable set to 0 (i.e., for edges $e_1 \in T$ and $e_2 \notin T$ such that $\omega(e_1) = \omega(e_2)$, we have $\omega'(e_1) < \omega'(e_2)$). It follows that the given subgraph T is an MST of $G(V, E, \omega)$ if and only if 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 [29] and [22] respectively. Before we continue, let us first recall what these procedures guarantee.

Procedure `DOM_Part`: The distributed Procedure `DOM_Part`, used in [29], 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 parameter k to be specified later (aiming to optimize the total costs). A *fragment leader* vertex is associated with each constructed fragment (the identifier of the fragment is defined as 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 [29] 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 execution of Procedure `DOM_Part`, requires $O(k \cdot \log^* n)$ time and $O(m \cdot \log k + n \cdot \log^* n \cdot \log k)$ messages. The parameter k to be used by our protocol, chosen as a suitable breakpoint so as to optimize the total costs, satisfies $k = \Theta(\sqrt{n} + D)$, hence, our invocation of Procedure `DOM_Part` will require $\tilde{O}(\sqrt{n} + D)$ time and $\tilde{O}(m)$ messages.

The `MAX_Label` labeling scheme: The labeling scheme `MAX_Label` of [22] is a centralized procedure designed for the family of weighted trees. The procedure involves two components: an encoder algorithm $\mathcal{E}$ and a decoder algorithm $\mathcal{D}$. These two components 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 $L(u)$ and $L(v)$ 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 $L(u)$ and $L(v)$ 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.

3.2 The algorithm

Overview: The algorithm consists of three phases. The first phase starts by letting the source node s construct a BFS tree rooted at s, and calculating the values of the number of nodes n and a 2-approximation d to the diameter D of the graph. Then, the distributed Procedure `DOM_Part` of [29] in invoked with parameter $k = \max\{\sqrt{n}, d\} = \Theta(\sqrt{n}+D)$, where D is the diameter of the graph. This procedure constructs an MST fragment collection (MFC) $\mathcal{F}$, where every fragment in $\mathcal{F}$ is of size at least $k+1$ and depth $O(k)$. As mentioned, our invocation of Procedure `DOM_Part` requires $\tilde{O}(\sqrt{n}+D)$ time and $\tilde{O}(m)$ messages. 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, i.e., $\bigcup_{F\in\mathcal{F}} E(F) = \bigcup_{F\in\mathcal{F}} E_{F,T}$ (this verifies that each $F \in \mathcal{F}$ is contained in T and that T does not contain additional fragment internal edges.) Note that this is a necessary condition for correctness since the graph is assumed to have a unique MST.

In the following 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 as follows.

In the second phase, the structure of $T_{\mathcal{F}}$ is aggregated over the BFS tree to the source node s, which in turn verifies that $T_{\mathcal{F}}$ is indeed a tree. Note that this aggregation is not very wasteful, and requires $O(D+f)$ time and $O(D \cdot f)$ messages, where f is the number of nodes in $T_{\mathcal{F}}$. As shown later, $f \leq n/k$, and hence, this aggregation can be done using $\tilde{O}(\sqrt{n}+D)$ time and $\tilde{O}(m)$ messages.

In the third phase, the source node s employs the (centralized) labeling scheme `MAX_Label` of [22] (or [26]) on $T_{\mathcal{F}}$. Informally, the labeling scheme 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. More specifically, the f labels are broadcasted over the BFS tree, so that eventually, each node in a fragment knows its corresponding label. This broadcasts costs $\tilde{O}(D+f) = \tilde{O}(\sqrt{n}+D)$ time and $\tilde{O}(D \cdot f) = \tilde{O}(m)$ messages. Recall, that given the labels of two fragments $L(F_i)$ and $L(F_j)$ 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 $\mathcal{D}$) by communicating with its neighbors only, 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. **Phase 1**

   (a) The source node s (which initiates the algorithm) constructs a BFS tree rooted at s, computes the values n and d, where n is the number of nodes and d is the depth of the BFS tree. (Note that d is a 2-approximation to the diameter D of the graph.) Subsequently, the source node s broadcasts a signal over the BFS tree containing the values n and d and instructing each vertex to send its identifier to all its neighbors.

This guarantees that all nodes know the values of n and d as well as the identifiers of their neighbors. In addition, to start the next step (i.e., Step 2.b) at the same time, the broadcast is augmented with a counter that is initialized by the source s to $d+1$ is decreased by one when delivered to a child in the BFS tree. Let $c(u)$ denote the counter received at node u. Before starting the next step each node u waits for $c(u)$ rounds after receiving the counter.

(b) Perform Procedure `DOM_Part`(k), with parameter $k = \max\{\sqrt{n}, d\} = \Theta(\sqrt{n} + D)$. 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) By comparing the fragment identifiers of its neighbors with its own fragment identifier, each vertex v identifies the sets of edges $E_F(v)$ and $E_{F,T}(v)$.

(e) Verify that $\bigcup_{F\in\mathcal{F}} E(F) = \bigcup_{F\in\mathcal{F}} E_{F,T}$ by verifying at each vertex v that $E_F(v) = E_{F,T}(v)$. (Else return "T is not an MST".)

2. **Phase 2**

(a) Vertex s counts the number of fragments, denoted by f. This is implemented by letting s broadcast a signal over the BFS tree instructing the nodes to perform a convergecast over the BFS tree. During this convergecast the number of fragments is aggregated to the root s by letting only fragment leader vertices increase the fragment counter. This guarantees that the fragment counter at the root s is f.

(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 another convergecast over the BFS tree. Then, s verifies that the number of edges is equal to $f-1$. (Else return "T is not an MST".)

(c) Vertex s broadcasts a signal over the BFS tree instructing all vertices to send the description of all their incident edges in $T_{\mathcal{F}}$ to s, by performing a convergecast over 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. **Phase 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, u knows which of its children is on the path connecting it to the fragment leader v.

(b) Vertex s computes the labels $L(F)$ for all vertices F in $T_{\mathcal{F}}$ (i.e., assigns a label to each of the fragments) using the encoder algorithm $\mathcal{E}$. Subsequently, 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 that 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 v sends its fragment label to all its neighbors in other fragments. (Recall that v already knows $E_F(v)$ by step 1d.)

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

3.3 Correctness

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

Claim 3.1 *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 $G_{\mathcal{F}}$, the fragment graph of G. Hence we get that $\omega(T) = \omega(T')$, thus T is an MST of G. ■

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

Observation 3.2 *The MST of G is unique.*

By combining Claim 3.1 and Observation 3.2 we get the following.

Claim 3.3 *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 $G_{\mathcal{F}}$.*

Lemma 3.4 *The MST verification algorithm correctly identifies whether the given tree T is an MST of the graph G.*

Proof By Claim 3.3, 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 $G_{\mathcal{F}}$.

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 $\bigcup_{F\in\mathcal{F}} E(F) = \bigcup_{F\in\mathcal{F}} E_{F,T}$, thus after this step, it is verified that T does not contain a cycle that is fully contained in some fragment $F \in \mathcal{F}$ (since every $F \in \mathcal{F}$ 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. ■

3.4 Complexity

Consider first the Phase 1. Steps 1a and 1c clearly take $O(D)$ time and $O(m)$ 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 [34].) The remaining steps of the first phase are local computations performed by all vertices. Thus, since k is set to $k = \Theta(\sqrt{n} + D)$, the first phase of the MST verification algorithm requires requires $\tilde{O}(\sqrt{n} + D)$ time and $\tilde{O}(m)$ messages.

Observe now, that since the fragments are disjoint and each fragment contains at least k vertices, we have the following.

Observation 3.5 *The number of MST fragments constructed during the first phase of the algorithm is* $f \leq n/k = O(\min\{\sqrt{n}, \frac{n}{D}\})$.

The first two steps of phase 2, namely, steps 2a and 2b consist of simple broadcasts over the BFS tree, hence they require $O(D)$ time and $O(m)$ messages. Step 2c consists of upcasting all the edges in $T_{\mathcal{F}}$ to s. Note that the number of edges in $T_{\mathcal{F}}$ can potentially be as high as m. However, given that the verification did not fail in Step 2b, we are guaranteed that the number of edges in $T_{\mathcal{F}}$ is $f - 1$. Thus, Step 2c amounts to upcasting $f - 1$ messages and can therefore be performed in $O(D + f)$ time and $O(fD)$ messages. Step 3a consists of an upcast of f messages over the BFS tree and thus requires the same time and message complexities as that of step 2c. Step 3b consists of a BFS downcast of f messages (each of size $\log^2 n$); it therefore requires $O(D + f \log n)$ time and $O(fD \log n)$ messages. Step 3c consists of a broadcast of a label (of size $O(\log^2 n)$) in each of the MST fragments; this can be performed in $O(k + \log n)$ time and $O(n \log n)$ messages. Finally, step 3d can be performed in $O(\log n)$ time and $O(m \log n)$ messages, and step 3e amounts to local computations. Table 1 below summarizes the time and message complexities of the second and third phases of the algorithm.

Step	Description	Time	Messages
2a,2b	BFS convergecast	$O(D)$	$O(m)$
2c, 3a	BFS upcast of f messages	$O(D + f)$	$O(f \cdot D)$
2d,3e	Local computation	0	none
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	$O(k + \log n)$	$O(n \cdot \log n)$
3d	Communication between neighbors	$O(\log n)$	$O(\log n \cdot m)$

Table 1: Time and message complexities of phases 2 and 3 of the algorithm. D is the diameter of the graph, and f is the number of MST fragments constructed in phase 1.

Combining the above arguments, and using the fact that $f = O(\min\{\sqrt{n}, \frac{n}{D}\})$ (see Observation 3.5), we obtain the following.

Lemma 3.6 *The algorithm describes above requires* $\tilde{O}(\sqrt{n} + D)$ *time and* $\tilde{O}(m)$ *messages.*

Theorem 1.1 follows by combining the lemma above with Lemma 3.4.

4 Time Complexity Lower Bound

In this section, we prove an $\tilde{\Omega}(\sqrt{n} + D)$ lower bound on the time required to solve the MST verification. Clearly, $\Omega(D)$ time is inevitable in the case of a single source node. The case in which

all nodes start at the same time is also very simple. Indeed, in this case, the $\Omega(D)$ time bound follows by considering a cycle C of size n with all edges having weight 1 except for two edges e and e' located at opposite sides. The given candidate spanning tree is $T = C \setminus \{e\}$. The two edges e and e' can have arbitrary weights, hence, to decide whether T is an MST one needs to transfer information along at least half of the cycle. This shows a lower bound of $\Omega(D)$ for the case where $D = \Theta(n)$. A similar argument can be applied for arbitrary values of D.

The difficult part is to prove a time lower bound of $\tilde{\Omega}(\sqrt{n})$. We prove this lower bound for the case where all nodes start at the execution at the same time. The lower bound for the case of a single source node follows directly from this lower bound, relying on the leader election algorithm of [30], which runs in $\tilde{O}(D)$ time.

In the remaining of this section we consider the case where all nodes start the execution simultaneously at the same time, and prove a lower bound of $\tilde{\Omega}(\sqrt{n})$. To show the lower bound, we first define a new problem named *vector equality*, and then show a lower bound for the time required to solve it. This is established in Section 4.1. More specifically, for the purposes of this lower bound proof, we consider the collection of graphs denoted F_m^2, for $m \geq 2$, as defined in [32]. Each graph graph F_m^2 consists of $n = \Theta(m^4)$ vertices, and its diameter is $\Theta(m)$. In Section 4.1.2, we establish a time lower bound of $\tilde{\Omega}(m^2) = \tilde{\Omega}(\sqrt{n})$ for solving vector equality in each graph F_m^2.

In Section 4.2, for $m \geq 2$, we consider a family $\mathcal{J}_m^2$ of weighted graphs (these are weighted versions of the graph F_m^2), and show that any algorithm solving the MST verification problem on the graphs in $\mathcal{J}_m^2$ can also be used to solve the vector equality problem on F_m^2 with the same time complexity. This establishes the desired $\tilde{\Omega}(\sqrt{n})$ time lower bound for the distributed MST verification problem.

4.1 A lower bound for the vector equality problem

The vector equality problem $\texttt{EQ}(G, s, r, b)$. Consider a graph G and two specified distinguished vertices s and r, each storing b boolean variables, $\bar{X}^s = \langle X_1^s, \ldots, X_b^s \rangle$ and $\bar{X}^r = \langle X_1^r, \ldots, X_b^r \rangle$ respectively, for some integer $b \geq 1$. An instance of the problem consists of initial assignments $\bar{X}^s = \{X_i^s \mid 1 \leq i \leq b\}$ and $\bar{X}^r = \{X_i^r \mid 1 \leq i \leq b\}$, where $X_i^s, X_i^r \in \{0, 1\}$, to the variables of s and r respectively. Given such an instance, the vector equality problem requires r to decide whether $\bar{X}^s = \bar{X}^r$, i.e., $X_i^s = X_i^r$ for every $1 \leq i \leq b$.

4.1.1 The graphs F_m^2

Let us recall the collection of graphs denoted F_m^2, for $m \geq 2$, as defined in [32]. (See Figure 1). The components used are the *ordinary path* $\mathcal{P}$ on $m^2 + 1$ vertices, where $V(\mathcal{P}) = \{v_0, \ldots, v_{m^2}\}$, $E(\mathcal{P}) = \{(v_i, v_{i+1}) \mid 0 \leq i \leq m^2 - 1\}$, and the *highway* $\mathcal{H}$ on $m + 1$ vertices, where $V(\mathcal{H}) = \{h_{im} \mid 0 \leq i \leq m\}$ and $E(\mathcal{H}) = \{(h_{im}, h_{(i+1)m}) \mid 0 \leq i \leq m - 1\}$. Each highway vertex h_{im} is connected to the corresponding path vertex v_{im} by a *spoke* edge (h_{im}, v_{im}).

The graph F_m^2 is constructed of m^2 copies of the path, $\mathcal{P}^1, \ldots, \mathcal{P}^{m^2}$, and connecting them to the highway $\mathcal{H}$. The two distinguished vertices are $s = h_0$ and $r = h_{m^2}$. The spoke edges are grouped into $m + 1$ *stars* S_i, $0 \leq i \leq m$, with each S_i consisting of the vertex h_{im} and the m^2 vertices $v_{im}^1, \ldots, v_{im}^{m^2}$ connected to it by spoke edges. Hence $V(S_i) = \{h_{im}\} \cup \{v_{im}^1, \ldots, v_{im}^{m^2}\}$ and $E(S_i) = \{(v_{im}^j, h_{im}) \mid 1 \leq j \leq m^2\}$. The graph F_m^2 consists of $V(F_m^2) = V(\mathcal{H}) \cup \bigcup_{j=1}^{m^2} V(\mathcal{P}^j)$ and $E(F_m^2) = \bigcup_{i=0}^{m} E(S_i) \cup \bigcup_{j=1}^{m^2} E(\mathcal{P}^j) \cup E(\mathcal{H})$. Thus F_m^2 has $n = \Theta(m^4)$ vertices and diameter $\Theta(m)$.

4.1.2 The lower bound for EQ

Our goal now is to prove that solving the vector equality problem on F_m^2 with a $b = m^2$-bit input strings $\bar{X}^s, \bar{X}^r$ requires $\Omega(m^2/B)$ time, assuming each message is encoded using B bits. (Later, we shall take $B = O(\log n)$.)

Consider some arbitrary algorithm $\mathcal{A}_{EQ}$, and let $\varphi(\bar{X}^s, \bar{X}^r)$ denote the execution of $\mathcal{A}_{EQ}$ on m^2-bit inputs $\bar{X}^s, \bar{X}^r$ on the graph F_m^2. For simplicity, we assume that $m^2/2$ is an integer. For $1 \leq i \leq m^2/2$, define the *i-middle set* of the graph F_m^2, denoted M_i, as follows. For every $1 \leq j \leq m^2$, define the *i-middle* of the path $\mathcal{P}^j$ as $M_i(\mathcal{P}^j) = \{v_l^j \mid i \leq l \leq m^2 - i\}$. Let $\beta(i)$ denote the least integer δ such that $\delta m \geq i$ and $\gamma(i)$ denote the largest integer κ such that $\kappa m \leq m^2 - i$. Define the *i-middle* of $\mathcal{H}$ as $M_i(\mathcal{H}) = \{h_{jm} \mid \beta(i) \leq j \leq \gamma(i)\}$. Now, the *i-middle set* of F_m^2 is the union of those middle sets, $M_i = M_i(\mathcal{H}) \cup \bigcup_j M_i(\mathcal{P}^j)$. (See Fig. 1.) For $i = 0$, the definition is slightly different, letting $M_0 = V \setminus \{h_0, h_{m^2}\}$.

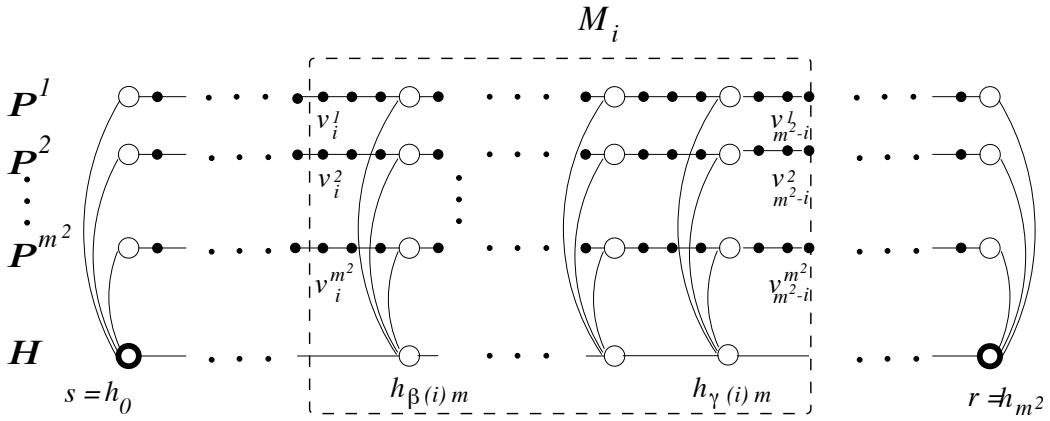

Figure 1: The middle set M_i in the graph F_m^2

Denote the *state* of the vertex v at the beginning of round t during the execution $\varphi(\bar{X}^s, \bar{X}^r)$ on the input $\bar{X}^s, \bar{X}^r$ by $\sigma(v, t, \bar{X}^s, \bar{X}^r)$. Without loss of generality, we may assume that in two different executions $\varphi(\bar{X}^s, \bar{X}^r)$ and $\varphi(\bar{X}'^s, \bar{X}'^r)$, a vertex reaches the same state at time t, i.e., $\sigma(v, t, \bar{X}^s, \bar{X}^r) = \sigma(v, t, \bar{X}'^s, \bar{X}'^r)$, if and only if it receives the same sequence of messages on each of its incoming links; for different sequences, the states are distinguishable.

For a given set of vertices $U = \{v_1, \ldots, v_l\} \subseteq V$, a *configuration* $C(U, t, \bar{X}^s, \bar{X}^r)$ is a vector $\langle \sigma(v_1, t, \bar{X}^s, \bar{X}^r), \ldots, \sigma(v_l, t, \bar{X}^s, \bar{X}^r) \rangle$ of the states of the vertices of U at the beginning of round t of the execution $\varphi(\bar{X}^s, \bar{X}^r)$. Denote by $\mathcal{C}[U, t]$ the collection of all possible configurations of the subset $U \subseteq V$ at time t over all executions $\varphi(\bar{X}^s, \bar{X}^r)$ of algorithm $\mathcal{A}_{EQ}$ (i.e., on all legal inputs $\bar{X}^s, \bar{X}^r$), and let $\rho[U, t] = |\mathcal{C}[U, t]|$.

Prior to the beginning of the execution (i.e., at the beginning of round $t = 0$), the input strings $\bar{X}^s, \bar{X}^r$ are known only to vertices s and r respectively. The rest of the vertices are found in some initial state, described by the configuration $C_{init} = C(M_0, 0, \bar{X}^s, \bar{X}^r)$, which is independent of $\bar{X}^s, \bar{X}^r$. Thus in particular $\rho[M_0, 0] = 1$. (Note, however, that $\rho[V, 0] = 2^{2m^2}$.)

Lemma 4.1 *For every* $0 \leq t < m^2/2$, $\rho[M_{t+1}, t+1] \leq (2^B + 1)^2 \cdot \rho[M_t, t]$.

Proof The lemma is proved by showing that in round $t+1$ of the algorithm, each configuration in $\mathcal{C}[M_t, t]$ branches off into at most $(2^B + 1)^2$ different configurations of $\mathcal{C}[M_{t+1}, t+1]$.

Fix a configuration $\hat{\mathcal{C}} \in \mathcal{C}[M_t, t]$, and let $\delta = \beta(t+1)$ and $\kappa = \gamma(t+1)$. The $(t+1)$-middle set M_{t+1} is connected to the rest of the graph by the highway edges $f_{\delta-1} = (h_{(\delta-1)m}, h_{\delta m})$ and $f_\kappa = (h_{\kappa m}, h_{(\kappa+1)m})$ and by the $2m^2$ path edges $e_t^j = (v_t^j, v_{t+1}^j)$,$e_{m^2-t}^j = (v_{m^2-t}^j, v_{m^2-t+1}^j)$ $1 \le j \le m^2$. (See Fig. 2.)

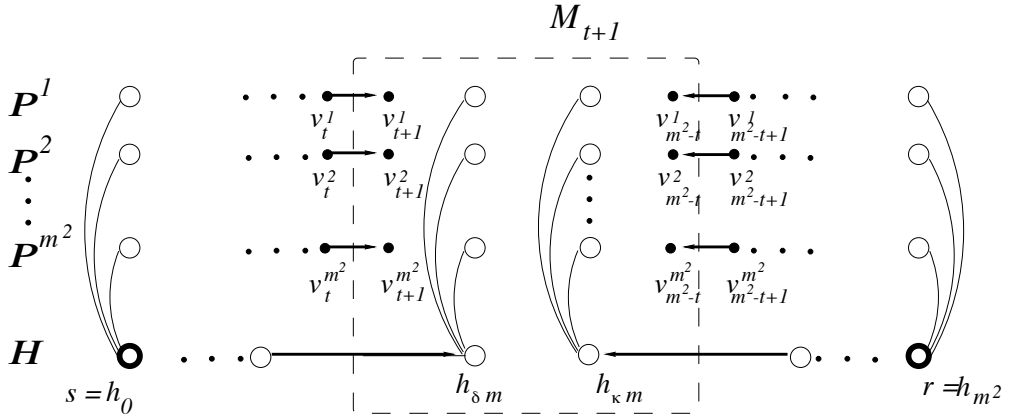

Figure 2: The edges entering the middle set M_{t+1}.

Let us count the number of different configurations in $\mathcal{C}[M_{t+1}, t+1]$ that may result of the configuration $\hat{\mathcal{C}}$. Starting from the configuration $\hat{\mathcal{C}}$, each vertex v_t^j is restricted to a single state, and hence it sends a single (well determined) message over the edge e_t^j to v_{t+1}^j thus not introducing any divergence in the execution. Similarly, each vertex $v_{m^2-t+1}^j$ is restricted to a single state, and hence it sends a single message over the edge $e_{m^2-t}^j$ to $v_{m^2-t}^j$. The same applies to all the edges internal to M_{t+1}. As for the highway edges $f_{\delta-1}, f_\kappa$, the vertices $h_{(\delta-1)m}$ and $h_{(\kappa+1)m}$ are not in the set M_t, hence they may be in any one of many possible states, and the values passed over these edges into the set M_{t+1} are not determined by the configuration $\hat{\mathcal{C}}$. However, due to the restriction of the B-bounded-message model, at most (2^B+1) different behaviors of $f_{\delta-1}$ can be observed by $h_{\delta m}$ and at most (2^B+1) different behaviors of f_κ can be observed by $h_{\kappa m}$. Thus altogether, the configuration $\hat{\mathcal{C}}$ branches off into at most $(2^B+1)^2$ possible configurations $\hat{\mathcal{C}}_1, \ldots, \hat{\mathcal{C}}_{2^B+1}^2 \in \mathcal{C}[M_{t+1}, t+1]$, differing only by the states $\sigma(h_{\delta m}, t+1, \bar{X}^s, \bar{X}^r)$, $\sigma(h_{\kappa m}, t+1, \bar{X}^s, \bar{X}^r)$. The lemma follows. ■

Applying Lemma 4.1 and the fact that $\rho[M_0, 0] = 1$, we get the following result.

Corollary 4.2 *For every* $0 \le t \le m^2/2$, $\rho[M_t, t] \le (2^B+1)^{2t}$.

To prove the time lower bound for the vector equality problem we introduce the following restricted model of computation as defined in [28]. A *two party communication model* consists of parties P_A and P_B connected via a bidirectional communication link. Let S_X, S_Y, S_Z be arbitrary finite sets. We say that Protocol Π computes function $f : S_X \times S_Y \to S_Z$ if, when given input $a \in S_X$ known to party P_A and input $b \in S_Y$ known to party B, the parties are able to determine $f(a,b)$ by a sequence of bit exchanges. The following observation follows from [28, 1.14], and is used to prove Lemma 4.4.

Observation 4.3 *Consider a protocol* Π. *Suppose that there exist inputs* $a, a' \in S_x$ *for party* P_A *and* $b, b' \in S_Y$ *for party* P_B *for which in the executions of* Π *on input* (a,b) *and on input*

(a', b'), *identical sequences of bits are exchanged by both parties. Then the same sequence of bits is exchanged in the execution of* Π *on input* (a', b).

Lemma 4.4 *For every* $m \geq 1$, *solving the vector equality problem* $\texttt{EQ}(F_m^2, h_0, h_{m^2}, m^2)$ *requires* $\Omega(m^2/B)$ *time.*

Proof Assume, towards contradiction, that there exists a protocol Π that correctly solves $\texttt{EQ}(F_m^2, h_0, h_{m^2}, m^2)$ and has worst case time complexity $T_\Pi < \frac{m^2}{2B}$. Let M_{T_Π} be the middle set of F_m^2, as previously defined, corresponding to $i = T_\Pi$. Let L_{T_Π} be the set of vertices that reside on the left side of M_{T_Π} in F_m^2 and R_{T_Π} be the set of vertices that reside on the right side of M_{T_Π}. Note that $M_{T_\Pi} \cup L_{T_\Pi} \cup R_{T_\Pi} = V$ and $M_{T_\Pi} \cap L_{T_\Pi} = M_{T_\Pi} \cap R_{T_\Pi} = L_{T_\Pi} \cap R_{T_\Pi} = \phi$. Consider a simulation of protocol Π by two parties P_A and P_B. The simulation works so that party P_A simulates the execution of Π in M_{T_Π} and in L_{T_Π}, and party P_B simulates the execution of Π in M_{T_Π} and in R_{T_Π}. At the end of the simulation, party P_B outputs the output of vertex r in the execution of Π. Every step of the distributed protocol Π is simulated by multiple bit exchanges between the parties P_A and P_B. Party P_A sends to party P_B all messages sent in Π by vertices in L_{T_Π} to vertices in M_{T_Π} and P_B party sends to party P_A all messages sent in Π by vertices in R_{T_Π} to vertices in M_{T_Π}. Hence, the parties P_A and P_B have full knowledge of the configuration of vertices in M_{T_Π} and are able to compute the configurations of vertices in L_{T_Π} and vertices in R_{T_Π} respectively. Thus, parties P_A and P_B correctly simulate Π.

Combining the assumption that $T_\Pi < \frac{m^2}{2B}$ with Corollary 4.2, we get that the number of possible configuration of M_{T_Π} in all possible executions of protocol Π is smaller than 2^{m^2}. Hence there exist inputs $x, x' \in \{0,1\}^{m^2}$ such that $x \neq x'$ and protocol Π reaches the same configuration of M_{T_Π} when executed with input $\bar{X}^s = \bar{X}^r = x$ or $\bar{X}^s = \bar{X}^r = x'$. Denote by Π_x the execution of Π on F_m^2 with input $\bar{X}^s = \bar{X}^r = x$, and by $\Pi_{x'}$ the execution of Π on F_m^2 with input $\bar{X}^s = \bar{X}^r = x'$. During both simulations of Π_x and $\Pi_{x'}$, parties P_A and P_B exchange the same sequence of messages (induced by the configuration of M_{T_Π}). By Obs. 4.3 we get that in the simulation of Π on input $\bar{X}^s = x', \bar{X}^r = x$, P_A and P_B exchange the same sequence of messages as in the simulation of Π_x. Thus in both executions of protocol Π (Π_x and Π with input $\bar{X}^s = x', \bar{X}^r = x$), vertex r has identical initial configuration and receives the same sequence of messages, hence it reaches the same decision, in contradiction to the correctness of Π. ■

4.2 A lower bound for the MST problem on $\mathcal{J}_m^2$

In this section we use the results achieved in the previous subsection, and show that the distributed MST verification problem cannot be solved faster than $\Omega(m^2/B)$ rounds on weighted versions of the graphs F_m^2. In order to prove this lower bound, we recall the definition given in [32] of a family of weighted graphs $\mathcal{J}_m^2$, based on F_m^2 but differing in their weight assignments. Then, we show that any algorithm solving the MST verification problem on the graphs of $\mathcal{J}_m^2$ can also be used to solve the vector equality problem on F_m^2 with the same time complexity. Subsequently, the lower bound for the distributed MST verification problem follows from the lower bound given in the previous subsection for the vector equality problem on F_m^2.

4.2.1 The graph family $\mathcal{J}_m^2$

The graphs F_m^2 defined earlier were unweighted. In this subsection, we define for every graph F_m^2 a family of weighted graphs $\mathcal{J}_m^2 = \{J_{m,\gamma}^2 = (F_m^2, \omega_\gamma) \mid 1 \leq \gamma \leq 2^{m^2}\}$, where ω_γ is an edge weight

function. In all the weight functions ω_γ, all the edges of the highway $\mathcal{H}$ and the paths $\mathcal{P}^j$ are assigned the weight 0. The spokes of all stars except S_0 and S_m are assigned the weight 4. The spokes of the star S_m are assigned the weight 2. The only differences between different weight functions ω_γ occur on the m^2 spokes of the star S_0. Specifically, each of these m^2 spokes is assigned a weight of either 1 or 3; there are thus 2^{m^2} possible weight assignments.

Since discarding all spoke edges of weight 4 from the graph $J^2_{m,\gamma}$ leaves it connected, and since every spoke edge of weight 4 is the heaviest edge on some cycle in the graph, the following is clear.

Lemma 4.5 *No spoke edge of weight 4 belongs to the MST of $J^2_{m,\gamma}$, for every $1 \leq \gamma \leq 2^{m^2}$.*

Let us pair the spoke edges of S_0 and S_m, denoting the jth pair (for $1 \leq j \leq m^2$) by $PE^j = \{(s, v_0^j), (r, v_{m^2}^j)\}$. The following is also clear.

Lemma 4.6 *For every $1 \leq \gamma \leq 2^{m^2}$ and $1 \leq j \leq m^2$, exactly one of the two edges of PE^j belongs to the MST of $J^2_{m,\gamma}$, namely, the lighter one. Moreover, for every $m \geq 2$ and $1 \leq \gamma \leq 2^{m^2}$, all the edges of the highway $\mathcal{H}$ and the paths $\mathcal{P}^j$, for $1 \leq j \leq m^2$, belong to the MST of $J^2_{m,\gamma}$.*

Note that the horizontal edges belong to the MST under any edge weight function. Of the remaining edges, exactly one of each pair will join the MST, depending on the particular weight assignment to the spoke edges of the star S_0.

4.2.2 The lower bound

Lemma 4.7 *Any distributed algorithm for MST verification on the graphs of the class $\mathcal{J}^2_m$, can be used to solve the $\texttt{EQ}(F^2_m, h_0, h_{m^2}, m^2)$ problem on F^2_m with the same time complexity.*

Proof Consider an algorithm $\mathcal{A}_{mst}$ for the MST verification problem, and suppose that we are given an instance of the $\texttt{EQ}(F^2_m, h_0, h_{m^2}, m^2)$ problem with input strings $\bar{X}^s, \bar{X}^r$ (stored in variables $\bar{X}^s$ and $\bar{X}^r$ respectively). We use the algorithm $\mathcal{A}_{mst}$ to solve this instance of vector equality problem as follows. Vertices $s = h_0$ and $r = h_m$ initiate the construction of an instance of the MST verification by turning F^2_m into a weighted graph from $\mathcal{J}^2_m$, setting the edge weights and marking the edges participating in the MST candidate as follows: for each $X^s_i \in \bar{X}^s$, $1 \leq i \leq m^2$, vertex s sets the weight variable W^s_i corresponding to the spoke edge $e_i \in E(S_0)$, to be

$$W^s_i = \begin{cases} 3, & \text{if } X^s_i = 1; \\ 1, & \text{if } X^s_i = 0. \end{cases}$$

In addition vertices s and r mark the edges participating in the MST candidate (that we wish to verify) in the following manner: for each $X^s_i \in \bar{X}^s$, $1 \leq i \leq m^2$, vertex s sets the indicator variable Y^s_i corresponding to the spoke edge $e_i \in E(S_0)$, to be

$$Y^s_i = \begin{cases} 0, & \text{if } X^s_i = 1; \\ 1, & \text{if } X^s_i = 0. \end{cases}$$

for each $X^r_i \in \bar{X}^r$, $1 \leq i \leq m^2$, vertex r sets the indicator variable Y^r_i corresponding to the spoke edge $e_i \in E(S_m)$, to be

$$Y^r_i = \begin{cases} 1, & \text{if } X^r_i = 1; \\ 0, & \text{if } X^r_i = 0. \end{cases}$$

The rest of the graph edges are assigned fixed weights as specified above. All path edges and highway edges are marked as participating in the MST candidate. All spoke edges not belonging to stars

S_0, S_m are marked as not participating in the MST candidate. Note that the weights of all the edges except those connecting s to its immediate neighbors in S_0 do not depend on the particular input instance at hand. Similarly, for all edges except the spoke edges belonging to S_0, S_m the indicator variable for participating in the MST candidate does not depend on the instance at hand. Hence a constant number of rounds of communication between s, r and their S_0, S_m neighbors suffices for performing this assignment; s assigns its edges weights and indicator variables according to its input string $\bar{X}^s$, and needs to send at most a constant number of messages to each of its neighbors in S_0, to notify it about the weight and the indicator variable of the spoke edge connecting them. (Same argument holds for vertex r). Every vertex v in the network, upon receiving the first message of algorithm $\mathcal{A}_{mst}$, assigns the values defined by the edge weight function ω_γ to its variables W_i^v and the corresponding indicator variable Y_i^v as described above. (As discussed earlier, this does not require v to know γ, as its assignment is identical under all weight functions ω_γ, $1 \leq \gamma \leq 2^{m^2}$). From this point on, v may proceed with executing algorithm $\mathcal{A}_{mst}$ for the MST verification problem.

Once algorithm $\mathcal{A}_{mst}$ terminates, vertex r determines its output for the vector equality problem by stating that both input vectors are equal if and only if the MST verification algorithm verified that the given MST candidate is indeed a minimum spanning tree. By Lemma 4.6, the lighter edge of each pair PE^j, for $1 \leq j \leq m^2$, belongs to the MST; thus, by the construction of the MST candidate and the weight assignment to the edges of F_m^2 the MST candidate forms a minimum spanning tree if and only if the input vectors $\bar{X}^s, \bar{X}^r$ for the vector equality problem are equal. Hence the resulting algorithm correctly solves the given instance of the vector equality problem. ■

Combined with Lemma 4.4, we now have:

Theorem 4.8 *For every $m \geq 1$, any distributed algorithm for solving MST verification problem on the graphs of the family $\mathcal{J}_m^2$ requires $\Omega(m^2/B)$ time.*

Corollary 4.9 *Any distributed algorithm for the MST verification problem requires $\Omega(\sqrt{n}/B)$ time on some n-vertex graphs of diameter $O(n^{1/4})$.*

Theorem 1.3 follows.

5 Message Complexity Lower Bound

We prove a message complexity lower bound of $\Omega(m)$ on the *Spanning Tree (ST) verification* problem, which is a relaxed version of the MST verification problem defined as follows. Given a connected 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[2] of G (not necessarily of minimal weight). Clearly, a lower bound on the *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 this lower bound proof.

The case of single source node is easy. Here, given a G with spanning tree T, if the execution does not send a message on some edge $e \in G \setminus \{T\}$, then we simply break e to two edges by adding another vertex in the middle of e. This brings us to the case where T is no longer spanning, but the execution (and hence the output of nodes) remains the same.

In what follows, we consider the somewhat more difficult case where all nodes start the execution at the same time. We begin with a few definitions. Recall that a *protocol* is a local program executed

[2]Equivalently, we may consider also disconnected graphs, and require T to be a spanning forest of G.

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.

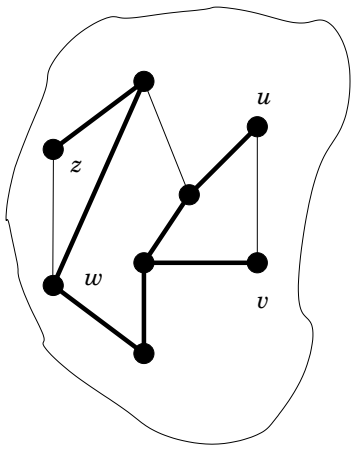

Figure 3: Graph G with ST candidate T (the bold edges belong to T)

Let $G = (V, E)$ be a graph (together with an assignment `ID` of vertex identifiers), T be a subgraph and let $e_1 = (u, v)$ and $e_2 = (z, w)$ be two of its edges. (See example in Figure 3.) 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^X 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'. See example in Figure 4. Note that G^2 is not connected, hence it is not a legal input to our problem.
- The graph G^X is a "cross-wired" version of G^2. Formally, $G^X = (V^2, E^X)$, where $E^X = E^2 \setminus \{(u, v), (z', w')\} \cup \{(u, w'), (v, z')\}$. (Observe that for $e_1, e_2 \notin T$, T^2 is also a subgraph of G^X.) See example in Figure 5.

Let Π be a protocol that correctly solves the ST verification problem. Fix G to be some arbitrary graph, fix a copy G' of G, fix a spanning tree T of G, and consider the execution of Π on either of the graphs G, G', G^2 and G^X with the ST candidates T, 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 5.1 *Let $e_1 \in E \setminus E(T)$ and $e'_2 \in E' \setminus E'(T)$, such that no message is sent over the edges e_1 and e'_2 in execution $EX(\Pi, G^2, T^2)$. Then executions $EX(\Pi, G^2, T^2)$ and $EX(\Pi, G^X, T^2)$ are similar.*

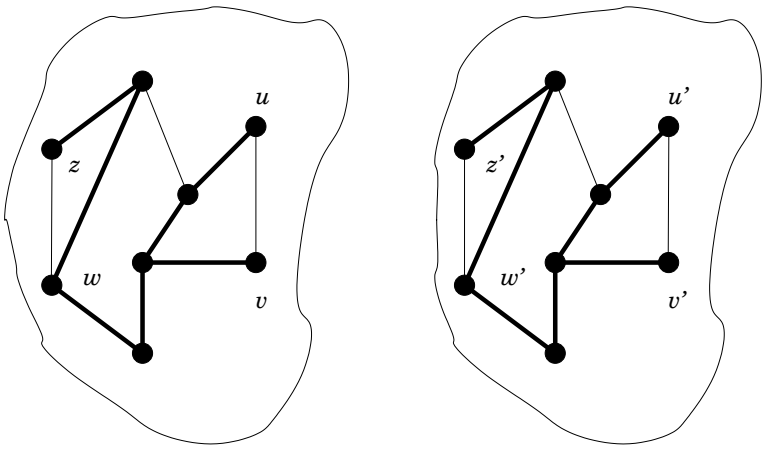

Figure 4: Graph G^2 with ST candidate T^2 (the bold edges belong to T^2)

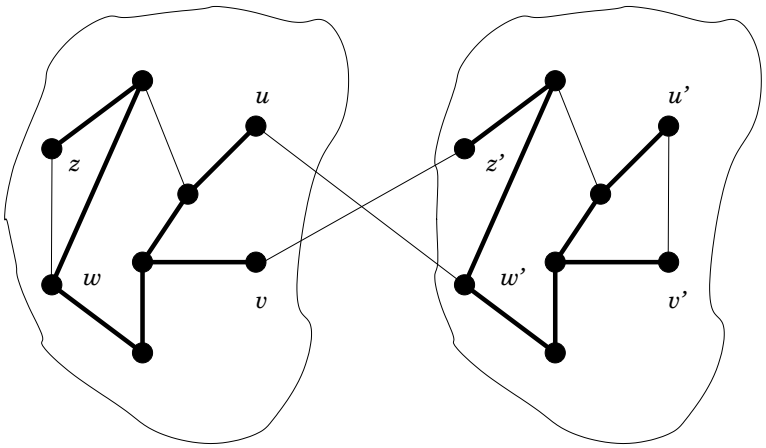

Figure 5: Graph G^X with ST candidate T^2 (the bold edges belong to T^2)

Proof 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 x are dependent on x's topological view (neighbors of x), x's initial input (its identity and the indicator variables of the edges incident to x), and the set of messages sent and received by x in previous communication rounds. Denote by EX^2 and EX^X executions $EX(\Pi, G^2, T^2)$ and $EX(\Pi, G^X, T^2)$ respectively. Note that any vertex $x \in V^2 \setminus \{u, v, z', w'\}$ 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. Although the communication link connecting u to v in G^2 connects u to w' in G^X, vertex u is initially unaware of this difference between the executions since it does not know the identifiers of its neighbors. (The same holds for vertices v, z' 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^X that belong to the first copy of G). Following are the possible cases.

- Vertex $x \notin \{u, v\}$: Vertex x has identical topological view and identical initial input in both execution, thus it sends identical sequences of messages in the first round of both executions.
- Vertex u: As mentioned above, although in execution EX^X vertex u is connected to w' instead of v, it has no knowledge of this difference. Thus u sends identical sequences 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^X no message is sent by u to its neighbor w'. Thus, u sends identical sequences 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 5.2 *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 3).

Consider the graph G^2 as previously defined with ST candidate $T^2 = \{e = (x, y) \in T\} \cup \{e' = (x', y') \mid e = (x, y) \in T\}$ (See Figure 4).

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 $e_1 = (u, v)$ and $e'_2 = (w', z')$ such that $e_1, e'_2 \in E^2 \setminus T^2$ and no message is sent over e_1 and e'_2 in execution EX^2. Consider the graph G^X with ST candidate T^2 as previously defined (See Figure 5).

By Lemma 5.1, executions EX^2 and $EX^X = EX(\Pi, G^X, T^2)$ are similar. Note that $e_1, e'_2 \notin T^2$, thus T^2 is not a spanning tree of G^X (since the two copies of G contained in G^X are connected solely by edges e_1 and e'_2). Since Π correctly solves the ST verification problem, the output of all vertices in EX^X is "0" (i.e., the given ST candidate T^2 is not a spanning tree of the graph G^X).

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^X 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)*, 230–240, NY, 1987.

[3] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In *Proc. IEEE Symp. on the Foundations of Computer Science*, 268–277, 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, R.E. Tarjan, and J.R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM J. 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] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proc. 43th ACM Symp. on Theory of computing (STOC)*, 2011.

[9] A. Das Sarma, D. Nanongkai, and G. Pandurangan. A tight unconditional lower bound on distributed random walk computation. In *Proc. 30th ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing (PODC)*, 2011.

[10] B. Dixon, M. Rauch, and R.E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM J. on Computing*, 21(6):1184–1192, 1992.

[11] B. Dixon and R.E. Tarjan. Optimal parallel verification of minimum spanning trees in logarithmic time. *Algorithmica*, 17(1):11–18, 1997.

[12] S. Dolev, M. Gouda, and M. Schneider. Requirements for silent stabilization. *Acta Informatica*, 36(6):447–462, 1999.

[13] P. Fraigniaud, A. Korman, and D. Peleg. Local Distributed Decision. FOCS 2011: 708-717

[14] P. Fraigniaud, A. Korman, M. Parter, and D. Peleg. Randomized Distributed Decision. DISC 2012: 371-385

[15] 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, NY, 1984.

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

[17] 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.

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

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

[20] 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)*, 185–194, 1985.

[21] 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.

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

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

[24] 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.

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

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

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

[28] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge Univ. Press, NY, 1997.

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

[30] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. Universal Bounds for Leader Election. Unpublished manuscript, 2012.

[31] 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, NY, 2001.

[32] 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.

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

[34] V. Rubinovich. Distributed minimum spanning tree construction. Master's thesis, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, 1999.

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

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