Distrib. Comput. (2007) 20:253–266
DOI 10.1007/s00446-007-0025-1

SPECIAL ISSUE PODC 06

Distributed verification of minimum spanning trees

Amos Korman · Shay Kutten

Received: 14 August 2006 / Accepted: 19 March 2007 / Published online: 15 May 2007

Abstract The problem of verifying a Minimum Spanning Tree (MST) was introduced by Tarjan in a sequential setting. Given a graph and a tree that spans it, the algorithm is required to check whether this tree is an MST. This paper investigates the problem in the distributed setting, where the input is given in a distributed manner, i.e., every node "knows" which of its own emanating edges belong to the tree. Informally, the distributed MST verification problem is the following. Label the vertices of the graph in such a way that for every node, given (its own state and label and) the labels of its neighbors only, the node can detect whether these edges are indeed its MST edges. In this paper, we present such a verification scheme with a maximum label size of $O(\log n \log W)$, where n is the number of nodes and W is the largest weight of an edge. We also give a matching lower bound of $\Omega(\log n \log W)$ (as long as $W > (\log n)^{1+\epsilon}$ for some fixed $\epsilon > 0$). Both our bounds improve previously known bounds for the problem.

For the related problem of tree sensitivity also presented by Tarjan, our method yields rather efficient schemes for both the distributed and the sequential settings.

Keywords Network algorithms · Graph property verification · Labeling schemes · Minimum spanning tree · Proof labeling schemes · Self stabilization

A preliminary version of this work was presented in ACM PODC 2006.
A. Korman was supported in part at the Technion by an Aly Kaufman fellowship.
S. Kutten was supported in part by a grant from the Israeli Ministry for Science and Technology.

A. Korman (✉) · S. Kutten
Information Systems Group, Faculty of IE&M,
The Technion, Haifa, 32000 Israel
e-mail: pandit@tx.technion.ac.il

S. Kutten
e-mail: kutten@ie.technion.ac.il

1 Introduction

1.1 Background and motivation

The Minimum Spanning Tree (MST) verification problem was introduced by Tarjan [32,34] in the context of sequential algorithms. A weighted graph is given, together with a tree that spans it, and it is required to decide whether this tree is indeed an MST of the graph. Improved algorithms in different sequential models were given by Harel [17], Komlòs [23], and Dixon et al. [7] who improved the original results of Tarjan. (The same result with a simpler algorithm was later presented by King [26].) Parallel algorithms were presented by Dixon and Tarjan [8] and by King et al. [27]. One application of the algorithm of [7] is as a subroutine for an algorithm that computes an MST, see e.g. [22,31]. A related motivation for the problem is that the verification seems easier than the computation. A linear (in the number of edges) time algorithm for computing an MST is known only in certain cases, or by a randomized algorithm [12,22]. On the other hand, the verification algorithm of [7] is linear (in its sequential running time).

The motivation for verification is even stronger in a distributed setting. There, the tree is given in a distributed manner, such that every vertex marks locally some of its own emanating edges, and the collection of the marked edges (of all the nodes) is the tree, see e.g. [5,6,13]. Computing such a tree distributively requires a computation that involves all the network nodes, and involves messages sent to remote nodes and waiting for replies. Intuitively, it is much harder than computing an MST sequentially. On the other hand, in the verification task, it is desired that every node, looking only at itself and at its neighbors, is able to determine whether its edges that are marked as belonging to the MST, indeed belong to it. Initial upper and lower bounds for the distributed verification of an MST are given in [24] together with the model for distributed verification we follow here. It turns out that neither of the bounds presented therein is optimal.

A common application of local distributed verification is in the context of self stabilization. See, for example, the *local detection* [1], the *local checking* [3], or the *silent stabilization* [9]. Self stabilization deals with algorithms that must cope with faults that are rather sever, though of a type that does occur in reality [18,19]. The faults may cause the states of different nodes to be inconsistent with each other. For example, the collection of marked edges may not be a tree, or may not be an MST. Self stabilizing algorithms thus often use distributed verification repeatedly. If the verification fails, then the output (e.g. the MST) is recomputed. An efficient verification algorithm thus saves repeatedly in communication.

Another motivation is theoretical. A fundamental issue studied for distributed computing is the following question: how efficient is the translation from results for sequential models to results in distributed models? Quite a few papers dealing with the (distributed) time cost of the translations appear in the literature, e.g. [25,28–30]. In particular, interest has been shown in translations that take one (or a constant) time unit. The current paper deals with a unit time translation (for Tarjan's MST verification problem) and with its cost in terms of the size of the labels in nodes, as well as in terms of the communication needed to compare this information between neighboring nodes. A (semi) automatic paradigm for translating a sequential verification algorithm to a distributed one can be derived from [24]. Unfortunately, when translating the known sequential MST verification algorithms, the resulting distributed verification algorithms turn out to be inefficient.

Some techniques used for solving the MST verification problem (in the sequential setting) were previously shown to be useful for the related problem of sensitivity testing. Given a graph and an MST of the graph, the task of this problem is the following. Label every edge (u, v) with the minimum number $c(u, v)$ such that if the weight of the edge changes by $c(u, v)$ (and the weights of the rest of the edges remain the same) then the given tree is no longer minimum. Note, that the output of any algorithm solving the sensitivity testing problem must use $\Omega(|E| \cdot \log W)$ bits in some cases, where W is the maximum edge weight.

In [32], Tarjan has extended his MST verification algorithm to an algorithm solving the sensitivity testing problem in $O(|E| \cdot \alpha(|E|, n))$ time, where α is the functional inverse of Ackermann's function. For the special case of planar graphs, Booth and Westbrook [4] gave an algorithm solving the sensitivity problem in $O(|E|)$ time. In [7], they describe a randomized $O(|E|)$-time algorithm and a deterministic algorithm that runs in time within a constant factor of the unknown optimum (which is between $O(|E| \cdot \alpha(|E|, n))$ and $\Omega(|E|)$). These results assume a model for time which allows very simple operations on edge costs to be performed in unit time. Let us note that in a stronger model for time, in which bit manipulations of edge costs are possible in unit time, somewhat stronger results where achieved. In particular, Harel [17] showed that if the edge costs are polynomial in n then the sensitivity problem can be solved by a deterministic algorithm that runs in $O(|E|)$ time.

1.2 Our results

We first present an algorithm solving the distributed MST-verification problem using $O(\log n \log W)$-bit labels, where n is the number of nodes and W is the largest weight of an edge. We note that our scheme can be applied to any given MST, even when the MST is not unique. This result improves the corresponding $O(\log^2 n + \log n \log W)$ size scheme presented in [24]. We also show a matching lower bound of $\Omega(\log n \log W)$ (as long as $W > (\log n)^{1+\epsilon}$ for some fixed $\epsilon > 0$). This improves the corresponding lower bound of $\Omega(\log n + \log W)$ given in [24] and answers a question left open there. Proving this lower bound was the main technical difficulty overcome in this study.

Let us also mention that one of the tools constructed here for the upper bound proof also improves the previously known upper bound for implicit labeling schemes supporting the $FLOW$ function on trees [21].

Returning to the sequential setting, our distributed MST verification algorithm yields a result for a slightly weaker version of the sensitivity problem. We relax the required output as follows: instead of writing the sensitivity of each edge explicitly, we write some auxiliary information. Later, when queried about the sensitivity of an edge, we are allowed to perform a constant time computation to derive the sensitivity of that edge, using the above auxiliary information. We provide a rather efficient algorithm for this version of the sensitivity problem as well as for a distributed version of this problem.

2 Preliminaries

The model described in this paper is taken from [24]. We consider distributed systems that are represented by weighted undirected connected graphs $G = \langle V, E\rangle$. For an edge $e \in E$, let $\omega(e)$ denote the (integral) weight of e. Every node v has internal ports, each corresponding to one of the edges attached to v. The ports are numbered from 1 to $\deg(v)$ (the degree of v) by an internal numbering known only to node v. For every vertex v, let $N(v)$ denote the set of edges adjacent to v. Note, that $\deg(v) = |N(v)|$. In addition, we assume that for each graph G, each vertex $v \in G$ has a unique *identity* $id(v)$ (i.e., for every pair of vertices v and u in G, it is given that $id(u) \neq id(v)$), which is encoded using $O(\log n)$.

Let $\mathcal{F}(n, W)$ (respectively, $\mathcal{T}(n, W)$) denote the family of all graphs (resp., trees) with at most n vertices whose edge weights are bounded from above by W.

Given a vertex v, let s_v denote the state of v and let $v_s = (v, s_v)$. A *configuration graph* corresponding to a graph $G = \langle V, E\rangle$ is a graph $G_s = \langle V_s, E_s\rangle$, where $V_s = \{v_s \mid v \in V\}$ and $(v_s, u_s) \in E_s$ iff $(v, u) \in E$. bits.

A *family of configuration graphs* $\mathcal{F}_s$ corresponding to a graph family $\mathcal{F}$ consists of configuration graphs $G_s \in \mathcal{F}_s$ for each $G \in \mathcal{F}$. Given a set S, let $\mathcal{F}_S$ be the largest possible family of configuration graphs corresponding to F, such that the states of the vertices in each graph G_s in F_s belong to S. When S is not specified, we consider S as the set of natural numbers. When it is clear from the context, we use the term "graph" instead of "configuration graph". We may also use the notation v instead of v_s.

We consider distributed representations of subgraphs, encoded in the collection of the nodes' states. There can be many such representations. For simplicity, we focus on the case that an edge is included in the subgraph if it is explicitly pointed at by the state of one of its endpoints. This is formalized in the following definition.

Definition 1 Given a configuration graph $G_s = \langle V_s, E_s\rangle$, the *subgraph induced by the states* of G_s is defined as follows. The set of vertices of the subgraph is V. The state of each node contains (a) specific field (or fields) for storing names (that is, numbers) of ports. An edge $(u, v) \in G$ belongs to the subgraph iff the state of one of its endpoints (say s_u) includes (in one of the above specific fields) the name of u's port number that points at v.

For example, a spanning tree can be represented as follows. The state of every vertex u contains a field that is u's port number leading to its parent in the tree. This field at the root is empty.

Consider a graph G. A distributed problem $Prob$ is the task of selecting a state s_v for each vertex v, such that G_s satisfies a given predicate f_{Prob}. This induces the problem $Prob$ on a graph family $\mathcal{F}$ in the natural way. We say that f_{Prob} is the *characteristic function* of $Prob$ over $\mathcal{F}$. In this paper, $Prob$ is the following problem-MST: assign states to the vertices of a given graph G such that the subgraph induces by the states of G is a minimum spanning tree (MST) of G.

Proof labeling schemes deal with the task of adding labels to configuration graphs in order to maintain a (locally checkable) distributed proof that the given configuration graph satisfies a given predicate f_{Prob}. Informally, a proof labeling scheme includes a *marker* algorithm M that generates a label for every node, and a *verifier* algorithm that compares labels of neighboring nodes. If a configuration graph satisfies f_{Prob}, then the verifier at every two neighboring nodes declares their labels (produced by marker M) "consistent" with each other. However, if the configuration graph does *not* satisfy f_{Prob}, then for *any possible* marker algorithm L, the verifier must declare "inconsistencies" between *some* neighboring nodes in the labels produced by the marker L. It is not required that the marker be distributed. However, the verifier is distributed and *local*, i.e., every node can check only the labels of its neighbors (and its own label and state).

More formally, A *marker* algorithm L is an algorithm that given a graph $G_s \in \mathcal{F}_s$, assigns a label $L(v_s)$ to each vertex $v_s \in G_s$. For a marker algorithm L and a vertex $v_s \in G_s$, let $N'_L(v)$ be a set of $\deg(v)$ fields, one field per neighbor. Each field $e = (v, u)$ in $N'_L(v)$, corresponding to edge $e \in N(v)$, contains the following.

- The port number of e in v.
- The weight of e.
- The label $L(u)$.

Let $N_L(v) = \langle (s_v, L(v)), N'_L(v)\rangle$. Informally, $N'_L(v)$ contains the labels given to all of v's neighbors along with the port number and the weights of the edges connecting v to them. $N_L(v)$ contains v's state and label as well as $N'_L(v)$. A *verifier* algorithm $\mathcal{V}$ is an algorithm which is applied separately at each vertex $v \in G$. When $\mathcal{V}$ is applied at a vertex v, its input is $N_L(v)$ and its output, $\mathcal{V}(v, L)$, is boolean.

Let f be some characteristic function of a problem over some family $\mathcal{F}$. Let $\mathcal{F}_s$ be some family of configuration graphs corresponding to $\mathcal{F}$. A *proof labeling scheme* $\pi = \langle \mathcal{M}, \mathcal{V}\rangle$ for $\mathcal{F}_s$ and f is composed of a *marker* algorithm $\mathcal{M}$ and a *verifier* algorithm $\mathcal{V}$, such that the following two properties hold.

1. For every $G_s \in \mathcal{F}_s$, if $f(G_s) = 1$ then $\mathcal{V}(v, \mathcal{M}) = 1$ for every vertex $v \in G$.
2. For every $G_s \in \mathcal{F}_s$, if $f(G_s) = 0$ then for every marker algorithm L there exists a vertex $v \in G$ so that $\mathcal{V}(v, L) = 0$.

We note that the proof labeling schemes constructed in this paper use a polynomial (sequential) time verifier algorithm. The *size* of a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{V}\rangle$ is the maximum number of bits used in a label $\mathcal{M}(v_s)$ over all $v_s \in G_s$ and all $G_s \in \mathcal{F}_s$.

For some of our proofs we make use of a particular different type of a distributed representation referred to as an *implicit labeling scheme* [15,20]. These were introduced in the past for a different purpose. For example, they can be viewed as a generalization of a routing table: given the labels of nodes u and v, find the first node (other than u) on the route from u to v.

Formally, let g be a function on pairs of vertices of a graph. An *implicit labeling scheme* $\gamma = \langle \mathcal{E}, \mathcal{D} \rangle$ supporting function g on a family of graphs $\mathcal{F}$ is composed of the following components:

1. An *encoder* algorithm $\mathcal{E}$ that given a graph in $\mathcal{F}$, assigns labels to its vertices.
2. An *decoder* algorithm $\mathcal{D}$ that given the labels $\mathcal{E}(u)$ and $\mathcal{E}(v)$ (assigned by the encoder $\mathcal{E}$ to two vertices u and v, not necessarily neighbors) outputs $g(u, v)$.

The *size* of an implicit labeling scheme is the maximum number of bits in a label assigned by the encoder $\mathcal{E}$ to any vertex in any graph in $\mathcal{F}$.

Note, that unlike the case of the verifier (in proof labeling schemes), instead of receiving the labels of neighbors as input, the decoder (in implicit labeling schemes) receives two labels of any two nodes as input. An additional difference between proof labeling schemes and implicit labeling scheme is the following. In an implicit labeling scheme, the answer of the decoder is some function of the two specific nodes given as inputs. It does not seem easy to define such a function for a proof labeling scheme. For example, if the answer of a correct verifier for some graph and function should be zero (e.g., the states of the graph do not encode an *MST*) then one correct verifier V_1 may output zero for some node, while another correct verifier V_2 may output 1 for that node, but zero for some other node. Hence, for a proof labeling scheme it is hard to claim that what the verifier finds is a function of some specific nodes (unlike the case of a decoder in implicit labeling schemes). Still, we find it interesting that techniques from each of these kinds of schemes helped, or inspired our proofs and constructions for the other kind.

Let us define two functions for which we construct and use implicit labeling schemes. Given a tree $T \in \mathcal{T}(n, W)$ and two vertices u and v in T, $MAX(u, v)$ is the maximum weight of an edge on the path connecting u and v in T. Similarly, $FLOW(u, v)$ is the minimum weight of an edge on the path connecting u and v in T. (Our constructions are based on the methods used for the labeling schemes in [11,14].)

In our schemes, the labels given to the vertices may contain several fields of different sizes. We assume that it is possible to perform operations on individual fields, e.g., to compare an individual field between the labels of two neighbors. Clearly, this can be performed with the aid of a sequential data structure that does not increase the order of the size of a label.

Regarding the sensitivity problem, we use the same model for time as used in [7]. I.e., we allow edge costs to be compared, added, or subtracted at unit cost, and side computations to be performed on a unit-cost random-access machine with word size $\Omega(\log n)$ bits.

Let us illustrate the definitions using an example presented in [24], and is in the spirit of [35]. Note, that v's neighbors cannot 'see' the state of v but they can see v's label.

Agreement in anonymous families Problem [24] Assign all the nodes identical states. Let $S = \{1, 2, \ldots, 2^m\}$.

Note, that the corresponding computation task, that of assigning every node the same state, requires only states of size 1. The verification is harder, as was shown in the following lemma.

Lemma 1 [24] *The proof size of $\mathcal{F}_S^{all}$ (the family of all the graphs) and $f_{Agreement}$ is $\Theta(m)$.*

Proof We first describe a trivial proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ of the desired size m. Given G_s such that $f_{Agreement}(G_s) = 1$, for every vertex v, let $\mathcal{M}(v) = s_v$. I.e., we just copy the state of node v into its label. Then, $\mathcal{D}(v, L)$ simply verifies that $L(v) = s_v$ and that $L(v) = L(u)$ for every neighbor u of node v. It is clear that π is a correct proof labeling scheme for $\mathcal{F}_S^{all}$ and $f_{Agreement}$ of size m. We now show that the above bound is tight up to a multiplicative constant factor even assuming that $\mathcal{F}_S^{all}$ is id-based. Consider the connected graph G with two vertices v and u. Assume, by way of contradiction, that there is a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{F}_S^{all}$ and $f_{Agreement}$ of size less than $m/2$. For $i \in S$, let G_s^i be G modified so that both u and v have state $s(u) = s(v) = i$. Obviously, $f_{Agreement}(G_s^i) = 1$ for every i. For a vertex x, let $\mathcal{M}^i(x)$ be the label given to x by marker $\mathcal{M}$ applied on G_s^i. Let $L^i = (\mathcal{M}^i(v), \mathcal{M}^i(u))$. Since the number of bits in L^i is assumed to be less than m, there exist $i, j \in S$ such that $i < j$ and $L^i = L^j$. Let G_s be G modified so that $s_u = i$ and $s_v = j$. Let L be the marker algorithm for G_s in which $L(u) = \mathcal{M}^i(u)$ and $L(v) = \mathcal{M}^j(v)$. Then, for each vertex x, $\mathcal{D}(x, L) = 1$, contradicting the fact that $f(G_s) = 0$. □

3 Upper bound for the MST problem

In this section, we construct a proof labeling scheme $\pi_{mst} = \langle \mathcal{M}_{mst}, \mathcal{V}_{mst} \rangle$ for f_{MST} and $\mathcal{F}_S(n, W)$ with size $O(\log n \cdot \log W)$. Recall that the main technical difficulty overcome in this paper is the proof of the lower bound, which is presented later.

Like the verification scheme of [7], our proof labeling scheme π_{mst} is based on the well known fact that given a graph G, a spanning tree T (of G) is an MST iff for every edge $e = (u, v) \in G$, its weight $\omega(e)$ is at least as large as $MAX(u, v)$ (calculated on T) [33].

It was proved in [24], that the problem of verifying an MST can be split into two: (1) verifying that it is a spanning tree, and (2) verifying that the spanning tree is minimal. Moreover, in [24] they show how to solve problem (1) above using a relatively simple construction. Intuitively, in the construction of [24] (a direct translation of existing self stabilizing tree protocols such as [1,2,10]), the label includes the unique identity of some root node, as well as the distance to that root and a pointer to the parent towards that root. Clearly, the size of the label is $O(\log n)$. The decoder at each node v needs to verify that (1) the root of each of its neighbors is the same as the root of v, (2) that the distance of the parent is smaller by 1 than the distance of v, and (3) that if the node is at distance 0 then its own identity is the identity of the root (and it has no parent). Hence, we assume that the given tree is indeed a spanning tree and concentrate on proving whether T is minimal or not.

We first establish a family Γ of implicit labeling schemes supporting the function $MAX(u, v)$ on $\mathcal{T}(n, W)$. Second, we select an implicit labeling scheme $\gamma \in \Gamma$ and the proof labeling scheme labels every vertex in T according to γ. For the proof size to be small, we find a specific $\gamma_{small} \in \Gamma$ of small size, namely $O(\log n \cdot \log W)$. The reason we define both Γ and γ_{small} is rather subtle, and is explained below.

At first glance, applying any $\gamma \in \Gamma$ may seem enough to perform the distributed verification. If the vertices are labeled according to any such γ then the distributed verification can be performed as follows. Let $I(v)$ be the label given to v in the implicit labeling scheme. Supposedly $I(v)$ has been given according to γ. Each vertex v just compares the weight of each of its incident edges (v, u) in G to $MAX(v, u)$ (which is calculated on T using $I(u)$ and $I(v)$).

Unfortunately, for that to suffice, we also need to locally verify that a given set of labels I has indeed been given according to γ (or some other implicit labeling scheme supporting $MAX(\cdot, \cdot)$ on T). That is, we need to construct a proof labeling scheme to verify the implicit labeling scheme. We do not know how to construct a small size proof labeling scheme to verify that I has been given according to γ_{small}. Fortunately, it is enough to verify that there exists *some* implicit labeling scheme $\gamma \in \Gamma$ such that the labels at the vertices are indeed given according to γ. This already verifies that the the value computed by v above for edge (v, u) is $MAX(v, u)$.

Even though we cannot verify that the specific $\gamma \in \Gamma$ used is γ_{small}, our marker does use γ_{small} nevertheless, in order to obtain a proof labeling scheme of a small size.

Let us remark that γ_{small} can easily be transformed into an implicit labeling scheme supporting the $FLOW$ function on weighted trees with size $O(\log n \cdot \log W)$; this improves the previously known upper bound $O(\log^2 n + \log n \cdot \log W)$ shown (for general graphs) in [21]. We also note that similar techniques can be used to provide compact proof labeling schemes for various implicit labeling schemes on trees, such as routing, distance etc.

Let us start with some preliminaries. A *separator decomposition* of a tree T is defined recursively as follows. At the first stage we choose some vertex v in T to be the *level-1* separator of T. Vertex x may be any vertex (not chosen before). Different choices of v lead to different decompositions. By removing v, T breaks into disconnected subtrees $T^1(v), T^2(v), \ldots, T^p(v)$. These subtrees are referred to as the subtrees *formed* by v. Each such subtree is decomposed recursively by choosing some vertex to be a level-2 separator, etc. A separator decomposition is termed *perfect* if every such separator v mentioned above is chosen in such a way that $|T^j(v)| \leq |T|/2$ for every j. It is easy to see that every tree has a perfect separator decomposition.

Define the *separator tree* T^{sep} to be the tree rooted at the level-1 separator of T, with the level-2 separators as its children, and generally, with each level $j+1$ separator as the child of the level j separator above it in the decomposition. For a vertex v in T, define the *level-j separator of* v to be the ancestor of v in T^{sep} at depth j.

Given a separator decomposition *Sep_Decomp* of a tree T, we define the function $Sep-level$ on pairs of vertices as follows. For every two vertices u and v in T, $Sep{-}level(u, v)$ is the depth of the nearest common ancestor of u and v in T^{sep}, i.e., the maximum level of a separator common to both u and v.

3.1 Implicit labeling schemes supporting $MAX(\cdot, \cdot)$ on $\mathcal{T}(n, W)$

We first describe a family Γ of implicit labeling schemes, each supporting the function $MAX(\cdot, \cdot)$ on $\mathcal{T}(n, W)$. In fact, in the construction (and later in the marker algorithm), we shall use a specific scheme γ_{small} in Γ. As mentioned above, the reason we define both Γ and γ_{small} (rather than just γ_{small} is somewhat subtle. Intuitively, our verifier will not check for γ_{small}, because proving that γ_{small} was used is costly. Instead, the proof is that **some** scheme in Γ is used.

3.1.1 The family Γ of implicit labeling schemes

We define Γ as the collection of all the implicit labeling schemes $\gamma = \langle \mathcal{E}_\gamma, \mathcal{D}_\gamma \rangle$ described as follows. Given a tree $T \in \mathcal{T}(n, W)$, perform a separator decomposition *Sep_Decomp* on T. (The choice of the specific separator decomposition to use is one of the factors that differentiates the members of Γ from one another). Note, that given any separator decomposition, each vertex is a separator of some level. For every vertex v, the encoder algorithm $\mathcal{E}_\gamma$ assigns each vertex v a label $\mathcal{E}(v)$ composed of two sublabels, namely, $\mathcal{E}^{sep}(v)$ and $\mathcal{E}^{\omega}(v)$. Let us first describe the first sublabel, $\mathcal{E}^{sep}(\cdot)$, which is designed so that given the sublabels $\mathcal{E}^{sep}(u)$ and $\mathcal{E}^{sep}(w)$ of two vertices u and w in T, one can determine $Sep-level(u, w)$. Specifically, for every level-l separator v, $\mathcal{E}^{sep}(v)$ consists of l fields, namely, $\mathcal{E}^{sep}(v) = (\mathcal{E}_1^{sep}(v), \mathcal{E}_2^{sep}(v), \ldots, \mathcal{E}_l^{sep}(v))$. It will follow from the description of the encoder algorithm $\mathcal{E}_\gamma$, that the following property holds for every two vertices u and w.

The $Sep - level$ **property:** For every $1 \leq k \leq i$, $\mathcal{E}_k^{sep}(u) = \mathcal{E}_k^{sep}(w)$ iff u and w share the same level-i separator in the separator decomposition Sep_Decomp.

We start by initializing $\mathcal{E}^{sep}(v)$ to be the same arbitrary number for every vertex v. Next, we apply the following recursive protocol. Let $T^1(v), T^2(v), \ldots, T^p(v)$ for some p be the subtrees formed by (the removal of) v from a tree T (T itself may be a subtree created earlier during the algorithm by the removal of another node). For every $1 \leq j \leq p$, let $\rho(j)$ be a unique number (in the sense that if $j \neq g$ then $\rho(j) \neq \rho(g)$). The specific values of $\rho(j)$ is the other factor differentiating the members of Γ from one another.

For every vertex $u \in T^j(v)$ update $\mathcal{E}^{sep}(u)$ as follows- $\mathcal{E}^{sep}(u) = \mathcal{E}^{sep}(u), \rho(j)$ (where the comma stands for concatenation). The protocol is then applied recursively on each subtree $T^j(v)$ formed by v.

For each level-l separator v, its second sublabel $\mathcal{E}^\omega(v)$ also consists of l fields, namely, $\mathcal{E}^\omega(v) = (\mathcal{E}_1^\omega(v), \mathcal{E}_2^\omega(v), \ldots, \mathcal{E}_l^\omega(v))$. For every $1 \leq i \leq l$, $\mathcal{E}_i^\omega(v)$ is set to $MAX(v, v^i)$, where v^i is the level-i separator of v in the separator decomposition Sep_Decomp.

Given the labels $\mathcal{E}(u)$ and $\mathcal{E}(v)$ of two vertices u and v, the decoder $\mathcal{D}_\gamma$ first calculates the largest index i such that for every $1 \leq k \leq i$, $\mathcal{E}_k^{sep}(u) = \mathcal{E}_k^{sep}(v)$, and then outputs $\max\{\mathcal{E}_i^\omega(u), \mathcal{E}_i^\omega(v)\}$. Note, that the decoder $\mathcal{D}_\gamma$ operates in the same manner for every scheme $\gamma \in \Gamma$.

Claim Every scheme $\gamma \in \Gamma$ is a correct implicit labeling scheme supporting function $MAX(\cdot, \cdot)$ on $\mathcal{T}(n, W)$.

Proof Fix a scheme $\gamma = \langle \mathcal{E}_\gamma, \mathcal{D}_\gamma \rangle$ in Γ. Let $\mathcal{E}(u)$ and $\mathcal{E}(v)$ be the labels assigned by the encoder algorithm $\mathcal{E}_\gamma$ to two vertices u and v in some tree $T \in \mathcal{T}(n, W)$. By the description of the encoder algorithm $\mathcal{E}_\gamma$, the $Sep - level$ property is satisfied for u and v. Therefore, $Sep - level(u, v)$ is the largest index i such that for every $1 \leq k \leq i$, $\mathcal{E}_k^{sep}(u) = \mathcal{E}_k^{sep}(v)$. Consequently, x, the level-i separator common to both u and v resides on the path connecting u and v in T. Therefore, $MAX(u, v) = \max\{MAX(u, x), MAX(v, x)\}$ and the correctness of the implicit labeling scheme γ follows. □

Let us remark, however, that the size of γ may be large. Next, we describe a particular implicit labeling scheme $\gamma_{small} \in \Gamma$ whose size is small, namely $O(\log n \cdot \log W)$.

3.1.2 The size $O(\log n \log W)$ implicit labeling scheme $\gamma_{small} \in \Gamma$

Scheme γ_{small} is constructed using a similar method to the one described in [14] for the different purpose of constructing approximate distance labeling schemes in trees. Scheme γ_{small} is a refinement of Γ as follows. The separator decomposition chosen by scheme γ_{small} is a perfect separator decomposition $Perfect\_Decomp$. The specification of how to assign a unique number to each of the subtrees $T^1(v)$, $T^2(v), \ldots, T^p(v)$ formed by the removal of each separator v, is done according to the method described in [14]. As proved therein, for every vertex v, the number of bits used in $\mathcal{E}^{sep}(v)$ is $O(\log n)$ (there, a different terminology was used for $\mathcal{E}^{sep}(v)$). Note, that since the separator decomposition $Perfect\_Decomp$ is perfect, the level of each separator is bounded from above by $\log n + 1$. Therefore, for every vertex v, $\mathcal{E}^\omega(v)$ contains $O(\log n)$ fields. Since each such field can be encoded using $O(\log W)$ bits, we obtain that for every vertex v, the number of bits used in $\mathcal{E}^\omega(v)$ is $O(\log n \cdot \log W)$. Therefore, by following similar steps to the ones described in Lemma 2.5 in [14], we obtain the following lemma.

Lemma 2 *γ_{small} is a correct implicit labeling scheme supporting the function $MAX(\cdot, \cdot)$ on $\mathcal{T}(n, W)$ with size $O(\log n \cdot \log W)$. Moreover, given the labels assigned by γ_{small} to two vertices u and v in some $T \in \mathcal{T}(n, W)$, the value $MAX(u, v)$ can be computed in constant time.*

Next, we establish a proof labeling scheme π_Γ on $\mathcal{T}_S(n, W)$ whose goal is to verify, given a configuration tree T_s, whether there exists an implicit labeling scheme $\gamma \in \Gamma$ such that the states of the vertices in T_s are the same as the corresponding labels assigned by γ.

3.2 The proof labeling scheme π_Γ for $\mathcal{T}_S(n, W)$

Let $Prob(\Gamma)$ be the following problem: assign states to the vertices of a tree so that there exists an implicit labeling scheme $\gamma = \langle \mathcal{E}, \mathcal{D} \rangle \in \Gamma$ such that for every vertex $v \in T$, $s_v = \mathcal{E}_\gamma(v)$.

Lemma 3 *There exists a proof labeling scheme $\pi_\Gamma = \langle \mathcal{M}_\Gamma, \mathcal{V}_\Gamma \rangle$ for $f_{Prob(\Gamma)}$ and $\mathcal{T}_S(n, W)$ such that given any tree $T_s \in \mathcal{T}_S(n, W)$, the maximum number of bits used in a label given by $\mathcal{M}_\Gamma$ is asymptotically the same as the maximum number of bits used in a state of a vertex in T_s.*

Before proving, let us try to give an informal high level description of the proof. At the label of each vertex v, we specify for each separator u of v, whether the direction from v towards u is up or down the tree. By verifying consistency between the labels at neighboring nodes and by verifying at each separator v that the numbers assigned to the subtrees formed by v are disjoint, we obtain that the states are indeed given by some separator decomposition Sep_Decomp. Then, the fact that for each separator u of v, the maximum weight of an edge on the path connecting u and v is as indicated in the corresponding place in the state of v, is verified along the path from v to u.

We stress that this does not verify that the implicit labeling scheme γ_{small} is used. Fortunately, we do not have to prove that, for γ_{small} to be useful for us.

Proof We describe Scheme $\pi_\Gamma = \langle \mathcal{M}_\Gamma, \mathcal{V}_\Gamma \rangle$ as claimed. First, it was shown in [24] (Lemma 2.4) that any proof labeling scheme π on undirected trees can be partitioned into three steps: (1) transform the tree into a rooted tree where every node knows the orientation towards the root; (2) construct a proof labeling scheme for this orientation; and (3) construct π assuming the tree has such an orientation. Moreover, the first two steps are already given in [24] using $O(\log n)$-bit labels. Hence, we assume that an orientation is given on the tree, i.e., the root r knows it is the root and every non-root vertex v knows which of its port numbers leads to its parent $p(v)$ in the tree.

Constructing the marker algorithm: Given a tree T_s such that $f_{Prob(\Gamma)}(T_s) = 1$, let $\gamma \in \Gamma$ be the corresponding implicit labeling scheme. Recall that γ is based on some separator decomposition Sep_Decomp on T. For every level-l separator v (in Sep_Decomp), the label $\mathcal{M}_\Gamma(v)$ given by marker $\mathcal{M}_\Gamma$ is composed of two sublabels, namely, $\mathcal{M}^{orient}(v)$ and $\mathcal{M}^{state}(v)$.

Informally, the first sublabel $\mathcal{M}^{orient}(\cdot)$ is used to verify that the states are given by some implicit labeling scheme according to some separator decomposition. This is implemented by indicating, at the label of each vertex v, whether the direction from v towards each separator of v is up or down the tree. Formally, for every level-l separator v, $\mathcal{M}^{orient}(v)$ contains l fields, i.e., $\mathcal{M}^{orient}(v) = (\mathcal{M}_1^{orient}(v), \mathcal{M}_2^{orient}(v), \ldots, \mathcal{M}_l^{orient}(v))$. For every $1 < k \leq l$, let $sep^k(v)$ denote the level-k separator of v, and set

$$\mathcal{M}_k^{orient}(v) = \begin{cases} 0\text{ , } sep^k(v) \text{ is a descendant of } v \text{ in } T \\ *\text{ , } k = l \\ 1\text{ , otherwise} \end{cases}$$

The second sublabel $\mathcal{M}^{state}(v)$ is simply a copy of the state s_v. Since the state s_v is supposed to be a label assigned by some $\gamma \in \Gamma$, we may assume that it is composed of two components (which correspond to the sublabels $\mathcal{E}^{sep}(\cdot)$ and $\mathcal{E}^{\omega}(\cdot)$ of $\mathcal{E}_\gamma(\cdot)$). We therefore consider the sublabel $\mathcal{M}^{state}(\cdot)$ as composed of two components, namely, $\mathcal{M}^{sep}(\cdot)$ and $\mathcal{M}^{\omega}(\cdot)$, i.e., for every vertex v, $\mathcal{M}^{state}(v) = \mathcal{M}^{sep}(v), \mathcal{M}^{\omega}(v)$. (The comma stands for the concatenation operator.)

Constructing the verifier algorithm: Given some marker algorithm L, for every vertex v, let $l(v)$ denote the number of fields in sublabel $L^{orient}(v)$. (We assume that some delimiter method is used to define the fields; clearly, the delimiter method does not increase the order of the size of the label.)

Given $N_L(v)$, the verifier $\mathcal{V}$ outputs 1 iff for every $1 \leq k \leq l(v)$, the following conditions are satisfied. Informally, the conditions imply that the label indeed looks as if it has been given by Marker $\mathcal{M}_\Gamma$ as defined above. Recall, that $\mathcal{M}_\Gamma$ assigns $\mathcal{M}^{state}(v)$ to be the state of v- this is checked below by condition 1. Then, some conditions check that L^{orient} matches the semantics of $\mathcal{M}^{orient}$ of the case that the separator is an ancestor of v (condition 2), the case that the separator is a descendant of v (condition 3), and the case that v itself is the separator (conditions 4 and 6). The $Sep-level$ property is also checked (based on Claim 3.1.1) by conditions 4, 5, and 6. The remaining conditions check that the computation of the maximum weight edge on the route between v and each separator is consistent among the nodes.

1. $L^{state}(v) = s_v \neq \emptyset$
2. If $L_k^{orient}(v) = 1$ then v is not the root and $l(p(v)) \geq k$. Moreover, for every child u of v, $L_k^{orient}(u) = 1$.
3. If $L_k^{orient}(v) = 0$ then there exists precisely one child u of v such that $L_k^{orient}(u)$ is either 0 or $*$ and if v is not the root then $L_k^{orient}(p(v)) = 0$.
4. If $L_k^{orient}(v) = *$ then both $L^{orient}(v)$, $L^{sep}(v)$ and $L^{\omega}(v)$ contain precisely k fields.
5. If w is a neighbor of v then for every $1 \leq j \leq \min\{l(v), l(w)\}$, $L_j^{sep}(v) = L_j^{sep}(w)$.
6. If $k = l(v)$ then $L_k^{orient}(v) = *$ and for every neighbor w of v such that $l(w) \geq l(v)$, the following hold.
   (a) $l(w) > l(v)$.
   (b) If v is a non-root node then $L_k^{orient}(p(v)) = 0$ and if v is a non-leaf node then $L_k^{orient}(u) = 1$ for every child u of v.
   (c) For any other neighbor w' of v such that $l(w') \geq l(v)$, $L_{k+1}^{sep}(w) \neq L_{k+1}^{sep}(w')$.
7. If $L_k^{orient}(v) = 1$ then let ω be the weight of the edge leading from v to its parent $p(v)$.
   Then, if $L_k^{orient}(p(v)) = *$ then $L_k^{\omega}(v) = \omega$, otherwise, $L_k^{\omega}(v) = \max\{L_k^{\omega}(p(v)), \omega\}$.
8. If $L_k^{orient}(v) = 0$ then let ω be the weight of the edge leading from v to its unique child u satisfying $L_k^{orient}(v) \neq 1$. Then, if this child u satisfies $L_k^{orient}(v) = *$ then $L_k^{\omega}(v) = \omega$, otherwise, $L_k^{\omega}(v) = \max\{L_k^{\omega}(u), \omega\}$.

The fact that the maximum number of bits used in a label given by $\mathcal{M}_\Gamma$ is asymptotically the same as the maximum number of bits used in a state of a vertex in T_s is clear from the description of $\mathcal{M}_\Gamma$.

Let us now turn to prove that π_Γ is a correct proof labeling scheme for $f_{Prob(\Gamma)}$ and $\mathcal{T}_S(n, W)$. Clearly, given a tree $T_s \in \mathcal{T}_S(n, W)$ such that $f_{Prob(\Gamma)}(T_s) = 1$, for every vertex $v \in T$, the verifier $\mathcal{V}_\Gamma$ applied on $N_{\mathcal{M}_\Gamma}(v)$ outputs 1. Assume now, by way of contradiction, that T_s is such that $f_{Prob(\Gamma)}(T_s) = 0$ and there exists a marker algorithm L such that for every vertex $v \in T$, $\mathcal{V}_\Gamma(N_L(v)) = 1$. The contradiction is achieved once we show that there exists an implicit labeling scheme $\gamma \in \Gamma$ such that for every vertex $v \in T$, $s_v = \mathcal{M}_\gamma(v)$.

Let us first show that there exists a unique vertex v such that $l(v) = 1$. By Condition 2 in the description of $\mathcal{V}_\Gamma$, for the root r, $L_1^{orient}(r)$ is either 0 or $*$. If $L_1^{orient}(r) \neq *$ then by Condition 3, there must exists a vertex x such that $L_1^{orient}(x) = *$. It follows that there must exist a vertex v_1 such that $L_1^{orient}(v_1) = *$. By Conditions 4 and 1, $l(v_1) = 1$ and therefore, by Conditions 6.b and 1, for every child u of v_1, $L_1^{orient}(u) = 1$ and if v_1 is not the root then $L_1^{orient}(p(v_1)) = 0$. Therefore, by Condition 2, for every descendant w of v_1, $L_1^{orient}(w) = 1$. It follows by Condition 6 that if v_1 is the root then v_1 is the only vertex satisfying $l(v) = 1$. Otherwise, if v_1 is not the root, then $L_1^{orient}(p(v_1)) = 0$ and by Condition 3, $L_1^{orient}(w) = 0$ for every vertex w on the path from v_1 to the root. Therefore, by Conditions 2 and 3, $L_1^{orient}(x) = 1$ for every vertex $x \neq v_1$ which is not an ancestor of v_1. We therefore get that v_1 is the only vertex satisfying $l(v) = 1$. Consider v_1 as the level-1 separator in the separator decomposition Sep_Decomp performed by the desired scheme γ. Since v_1 is the only vertex with just one field in its state, it follows from Condition 5 that by removing v_1 from the tree, T breaks into subtrees $T^1(v_1), T^2(v_2), \ldots, T^p(v_1)$, such that for every $1 \leq j \leq p$, all labels $L^{sep}(v)$ of vertices $v \in T^j(v_1)$ have the same value $\rho(j)$ in their second field. Moreover, by Condition 6.c, for every $1 \leq j \neq g \leq p$, $\rho(j) \neq \rho(g)$. The fact that for each vertex v, $L_1^\omega(v)$ is $MAX(v, v_1)$ follows from Conditions 7 and 8.

Following the same arguments as before we obtain that for each subtree $T^j(v_1)$, there exists a unique vertex v_2^j whose state contains precisely two fields and that for each vertex $v \in T^j(v_1)$, $L_2^\omega(v)$ is $MAX(v, v_2^j)$. For each j, consider v_2^j as a level-2 separator in the separator decomposition Sep_Decomp performed by the desired scheme γ. Using the same arguments as before, the lemma follows by induction on the depth of the separator decomposition Sep_Decomp. □

3.3 The proof labeling scheme π_{mst} for f_{MST} and $\mathcal{F}(n, W)$

Theorem 1 *There exists a proof labeling scheme π_{mst} for f_{MST} and $\mathcal{F}(n, W)$ with size $O(\log n \cdot \log W)$.*

Proof As mentioned before, it was shown in [24] (Lemma 4.3) that a proof labeling scheme for f_{MST} and $\mathcal{F}(n, W)$ can be partitioned into two steps: (1) verify that the subgraph induced by the states is a spanning tree; and (2) verify that this spanning tree is minimal. Moreover, the first step is already given in [24] using $O(\log n)$-bit labels. Hence, we assume that the subgraph induced by the states is a spanning tree T and concentrate on verifying whether it is minimal.

We describe a proof labeling scheme $\pi_{mst} = \langle \mathcal{M}_{mst}, \mathcal{V}_{mst} \rangle$ as claimed. In the case where T is an MST, the marker algorithm $\mathcal{M}_{mst}$ assigns each vertex v a label $\mathcal{M}(v)$, composed of three sublabels. The first sublabel at every node $\mathcal{M}_{span}(\cdot)$ is assigned by the marker algorithm of the proof labeling scheme described in Lemma 2.3 of [24] (to verify that the given tree is indeed a tree). The second sublabel $\mathcal{M}_{\gamma_{small}}(\cdot)$ is the same as the label assigned to the corresponding vertex by the encoder algorithm $\mathcal{E}_{\gamma_{small}}$ of the implicit labeling scheme γ_{small} applied on T. By composing with the proof labeling scheme π_Γ, mentioned in the above lemma (encoded in the third sublabel $\mathcal{M}_\Gamma(\cdot)$), we can verify that the second sublabel is the label assigned by some implicit labeling scheme $\gamma \in \Gamma$ applied on T. Note, that in terms of correctness, it is not necessary to verify that the second sublabel is the label assigned by the specific scheme $\gamma_{small} \in \Gamma$. However, since an arbitrary scheme $\gamma \in \Gamma$ may have large size, we use the specific scheme γ_{small} in the marker algorithm, to achieve a proof labeling scheme of small size.

As mentioned before, using the first sublabels, verifier $\mathcal{V}_{mst}$ first verifies that T is a spanning tree of G. Then, using the third sublabels of the vertices, verifier $\mathcal{V}_{mst}$ verifies that the second sublabels can be used by some implicit labeling scheme γ supporting the $MAX(\cdot, \cdot)$ function on T. Then, verifier $\mathcal{V}_{mst}$ at vertex v computes $MAX(v, u)$ for every neighbor u of v, using the decoder of γ_{small} (which is the same for any scheme $\gamma \in \Gamma$) applied on the second sublabels of v and u. Finally, verifier $\mathcal{V}_{mst}$ at vertex v verifies for every neighbor u of v, that $\omega(v, u)$ is at least as large as $MAX(v, u)$.

The correctness of the proof labeling scheme π_{mst} follows from Claim 3.1.1 and Lemmas 2 and 3. The fact that the size of π_{mst} is $O(\log n \cdot \log W)$ follows from Lemmas 2 and 3. □

4 Lower bound for MST verification

In this section we establish a lower bound of $\Omega(\log n \cdot \log W)$ on the label size of any proof labeling scheme for f_{MST} and $\mathcal{F}_S(n, W)$ as long as $W > (\log n)^{1+\epsilon}$ for some fixed $\epsilon > 0$. Though the specific parts of our proof are rather different, our approach is inspired by the lower bound proofs of [16,21]. The lower bound proof of [16] was constructed for implicit distance labeling schemes in trees. This proof was later modified in [21] to construct a lower bound proof for implicit $FLOW$ labeling schemes. Our proof is somewhat closer to the modified proof of [21]. Specifically, the proof in [21] uses a class of weighted complete binary trees called (h, μ)-trees. The main lemma therein shows that one can derive labels for the *leaves* of any $(h-1, \mu^2)$-tree from the labels given to the leaves of some (h, μ)-tree. In fact, the class of (h, μ)-trees can be partitioned into μ disjoint subclasses, such that one can label the leaves in $(h-1, \mu^2)$-trees using the labels given to the leaves in trees even of one (any one) of the above mentioned subclass. (This shows that the set of labels used for leaves in (h, μ)-trees is large, and thus leads to a lower bound on the size of a label.)

Our proof uses a new combinatorial structure termed (h, μ)-*hypertrees* that is a combination of (h, μ)-trees and a hypercube. That is, an (h, μ)-hypertree is constructed by connecting (via a weighted path) every node in one $(h-1, \mu)$-hypertree to the corresponding node in another $(h-1, \mu)$-hypertree. Intuitively, the proof needed that the combinatorial structure has the following properties, which the original construct of [21] does not provide: (1) we needed to create many cycles; and (2) we needed to move nodes in different subtrees closer to each other. Our proof follows the general structure of [21] in the sense that labels for some $(h-1, \mu^2)$-hypertree H' are computed using the labels for some (h, μ)-hypertree H. However, it may be useful to understand some of the differences (in addition to the difference between the combinatorial structure). First, in contrast to [21], in which only the leaves of the trees are labeled, we needed to label all the vertices of our hypertrees. This is because proof labeling schemes rely on the cooperation of many vertices, checking a property transitively. Second, the additional complexity of our structure poses additional difficulties not only at the state of computing labels, but also at the stage of verifying them. A new trick we introduce here is that the verifier described in the construction below, at any node v, has to *guess* labels for some other nodes.

Let us start by defining the family of (h, μ)-hypertrees. For $i \geq 0$, let

$$Q_i(\mu) = \{\mu \cdot i + j \mid 0 \leq j \leq \mu - 1\}.$$

An (h, μ)-hypertree H is constructed inductively on h. We note that it will follow from our construction that given h, all (h, μ)-hypertrees H are identical if we consider them as unweighted. Furthermore, it will follow from our construction, that the subgraph induced by the states of a hypertree H (see definition 1) is a spanning tree of H (not necessarily minimal). We therefore refer to the subgraph induced by the states of a hypertree H as the *spanning tree induced* by the states of H. (The edges of the spanning tree are drawn as directed edges in Fig. 1) A $(1, \mu)$-hypertree is a single vertex with an empty state. Given two (rooted) $(h-1, \mu)$-hypertrees H_0 and H_1, we construct an (h, μ)-hypertree H as follows. (Fig. 1 illustrates the following construction of H).

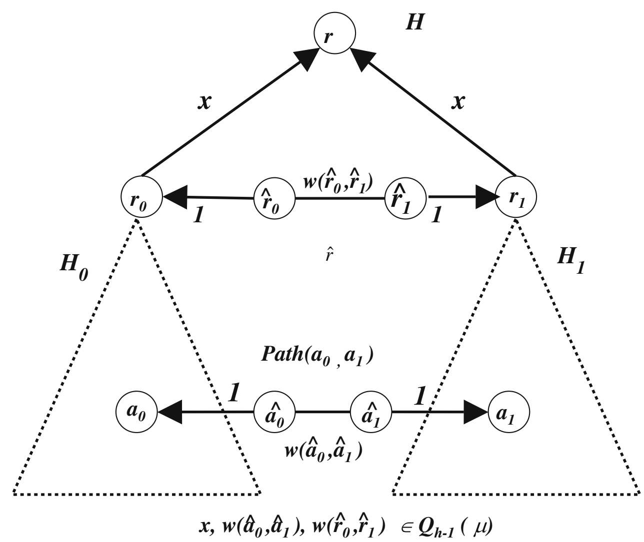

Fig. 1 Constructing a hypertree out of two trees

1. Create a new root vertex r and choose some fixed $x \in Q_{h-1}(\mu)$. Connect the roots of H_0 and H_1 to r, and assign them states such that they point at r. Let the weight of both edges incident to r be x.
2. For every vertex $a_0 \in H_0$, let a_1 be its homologous vertex in H_1, and add to H the path $Path(a_0, a_1)$ defined as follows. $Path(a_0, a_1) = (a_0, \hat{a}_0, \hat{a}_1, a_1)$ consists of four vertices (a_0 and a_1 together with two new vertices $\hat{a}_0$ and $\hat{a}_1$) and three edges, namely, $(a_0, \hat{a}_0)$, $(\hat{a}_0, \hat{a}_1)$ and $(\hat{a}_1, a_1)$. Let the state at $\hat{a}_0$ (respectively, $\hat{a}_1$) point at a_0 (resp., a_1).
3. For every new path $Path(a_0, a_1)$ defined in Step 2 above, let the weights $\omega(a_0, \hat{a}_0) = \omega(\hat{a}_1, a_1) = 1$ and let the value of $\omega(\hat{a}_0, \hat{a}_1)$ be taken from $Q_{h-1}(\mu)$. We refer to $\omega(\hat{a}_0, \hat{a}_1)$ as the *weight* of $Path(a_0, a_1)$. Note, that the weight of the path can be any fixed value from $Q_{h-1}(\mu)$. However, path $Path(a_0, a_1)$ is termed *legal* if its weight equals x, the value given in step 1 above (to the top most edges of H).
4. Consider the spanning tree induced by the states of H (the directed edges of Fig. 1). Perform a preorder on this spanning tree starting at the root. Let the new identity $id(v)$ of each vertex v be a number according to this preorder. In particular, $id(r) = 1$.

Given an (h, μ)-hypertree H constructed as shown above, let $\mathcal{P}$ denote the set of paths $Path(u, v)$ added to H throughout the inductive construction. (Note, that a path is added also between vertices that were created for paths added earlier.) An (h, μ)-hypertree H is termed *legal* if the collection of paths $\mathcal{P}$ consists only of legal paths.

The proof of the following claim is straightforward.

Claim 1. Given a hypertree H, the weight of any legal path $Path(u, v) \in H$ equals $MAX(u, v)$, where $MAX(u, v)$

is calculated on the spanning tree induced by the states of H.

2. The spanning tree induced by the states in a legal (h,μ)-hypertree H is an MST of H.

Let $\mathcal{C}(h,\mu)$ be the family of (h,μ)-hypertrees. Given $x \in Q_{h-1}(\mu)$, let $\mathcal{C}(h,\mu,x)$ be the subfamily of $\mathcal{C}(h,\mu)$ consisting of (h,μ)-hypertrees with x being the weight of the edges adjacent to the root. In other words, $\mathcal{C}(h,\mu) = \bigcup_{x=\mu(h-1)}^{\mu h-1} \mathcal{C}(h,\mu,x)$.

A proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{V} \rangle$ satisfies the *identity property* if the identity $id(v)$ of each vertex v is encoded in the leftmost field of the label $\mathcal{M}(v)$, assigned to v by $\mathcal{M}$. (We shall show that it is enough to speak only of schemes with this property.) Given a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{V} \rangle$ for f_{MST} and $\mathcal{C}(h,\mu)$ satisfying the identity property, let $X(\pi,h,\mu)$ denote the set of all labels assigned by $\mathcal{M}$ to nodes of hypertrees in $\mathcal{C}(h,\mu)$. Let $g(h,\mu)$ denote the minimum cardinality $|X(\pi,h,\mu)|$ over all proof labeling schemes for f_{MST} and $\mathcal{C}(h,\mu)$ which satisfy the identity property.

Hereafter, we fix $\hat{\pi} = \left\langle \hat{\mathcal{M}}, \hat{\mathcal{V}} \right\rangle$ to be some proof labeling scheme (satisfying the identity property) for f_{MST} and $\mathcal{C}(h,\mu)$ attaining $g(h,\mu)$, i.e., such that $|X(\pi,h,\mu)| = g(h,\mu)$.

Note, that a legal (h,μ)-hypertree H is completely defined by H_0, H_1, x, where H_0 and H_1 are the two (legal) $(h-1,\mu)$-hypertrees hanging from the root's children and x is the weight of the edges incident to the root of H. Let $X(x)$ denote the set of all possible pairs of labels assigned by $\hat{\mathcal{M}}$ to some nodes $a_0, a_1 \in H = (H_0, H_1, x)$, where $a_0 \in H_0$, $a_1 \in H_1$ and H is a legal hypertree in $\mathcal{C}(h,\mu,x)$. Let $\mathcal{X} = \bigcup_{x=\mu(h-1)}^{\mu h-1} X(x)$. As $\mathcal{X} \subseteq X(\hat{\pi},h,\mu) \times X(\hat{\pi},h,\mu)$, we have,

Claim $|\mathcal{X}| \le g(h,\mu)^2$.

Lemma 4 *For every $\mu(h-1) \le x \ne x' < h\mu$, the sets $X(x)$ and $X(x')$ are disjoint.*

Proof Consider two different weights $\mu(h-1) \le x \ne x' < h\mu$, and assume by way of contradiction that there exists a pair $(\lambda_1,\lambda_2) \in X(x) \cap X(x')$. Then there exists a legal (h,μ)-hypertree $H = (H_0, H_1, x)$ which uses the label λ_1 for some vertex $a_0 \in H_0$ and the label λ_2 for some vertex $a_1 \in H_1$, and there exists a legal (h,μ)-hypertree $H' = (H'_0, H'_1, x')$ which uses the label λ_1 for some vertex $a'_0 \in H'_0$ and the label λ_2 for some vertex $a'_1 \in H'_1$. By our assumption, that the identity of a vertex is encoded in the leftmost field of its label, and by the fact that given h, all $(h-1,\mu)$-hypertrees are identical if we consider them as unweighted, we obtain that $a_0 = a'_0$ and $a_1 = a'_1$. It follows that the path $Path(a_0,a_1)$ in H is the same as the path $Path(a'_0,a'_1)$ in H', except that the weight of $Path(a_0,a_1)$ is x whereas the weight of $Path(a'_0,a'_1)$ is x'. Assume W.L.O.G that $x > x'$ and modify H into a new hypertree H'' by replacing the weight of path $Path(a_0,a_1)$ in H by x'.

Since, by Claim 4, the weight x' of $Path(a_0,a_1)$ in H'' is smaller than $x = MAX(a_0,a_1)$ (calculated on the spanning tree induced by H''), we obtain that the spanning tree induced by the states of H'' is not minimal, i.e., $f_{MST}(H'') = 0$. We now show that it is possible to mark H'' by a combination of the labeling $\hat{\mathcal{M}}$ uses for H' and the labeling it uses for H such that the claimed verifier is actually fooled (to output 1) at each node.

We describe a labeling assignment $\mathcal{L}$ to the vertices of H''. For every vertex $v \notin Path(a_0,a_1)$ (that is, all the vertices except for four), let $L(v)$ be the label assigned to v by the marker algorithm $\hat{\mathcal{M}}$ applied on H and for every vertex $v \in Path(a_0,a_1)$, let $L(v)$ be the label assigned to v by the marker algorithm $\hat{\mathcal{M}}$ applied on H'.

Since $f_{MST}(H) = 1$ (by Claim 4), and since for every vertex $v \notin Path(a_0,a_1)$, $N_L(v)$ in H'' is the same as $N_{\hat{\mathcal{M}}}(v)$ in H, we obtain that at every vertex $v \notin Path(a_0,a_1)$, the verifier at v satisfies $\hat{\mathcal{V}}(N_L(v)) = 1$. Similarly, since $f_{MST}(H') = 1$, and since for every vertex $v \in Path(a_0,a_1)$, $N_L(v)$ in H'' is the same as $N_{\hat{\mathcal{M}}}(v)$ in H', we obtain that at every vertex $v \in Path(a_0,a_1)$, the verifier at v satisfies $\hat{\mathcal{V}}(N_L(v)) = 1$. This contradicts the correctness of $\hat{\pi}$ (assumed in the definition of $X(x)$), since $f_{MST}(H'') = 0$. □

Lemma 5 *For every $x \in Q_h(\mu)$, $|X(x)| \ge g(h-1,\mu^2)$.*

Before proving, let us first outline the proof. Given an $(h-1,\mu^2)$ hypertree H', we associate with it a hypertree $H \in C(h,\mu,x)$ (for some $x \in Q_{h-1}(\mu)$) such that H is legal if and only if H' is legal. Moreover, the states of the two hypertrees (H and of H') induce two subgraphs. We show that one of these subgraphs is a minimum spanning tree if and only if the other is. This is summarized in Claim 4.5.

Next, given a marking of an assumed optimal proof labeling scheme for H (for f_{MST}), we show how to translate these marking $\hat{\mathcal{M}}$ to a marking $\mathcal{M}'$ for the vertices of H' (for f_{MST}). That is, we translate the labels of a pair of vertices in H to a label of one vertex in H'. (Each vertex of H participates in only one pair). Hence, we obtain in Claim 4 that the number of labels used to label H' is at most $X(x)$. This implies the lemma (by the definition of g), provided that we prove that the above marking is a part of a correct proof labeling scheme. For that, we first construct the verifier part $\mathcal{V}'$ for the above marker $\mathcal{M}'$. We then conclude the proof by showing that scheme $(\mathcal{M}',\mathcal{V}')$ is a correct proof labeling scheme for f_{MST} for H'. That is, if the tree induced by the states of the vertices in H' is a minimum spanning tree than $\mathcal{V}'$ outputs 1 everywhere in H', and vice versa.

Proof Given a hypertree H, and a vertex $v \in H$, let $Neigh_H(v)$ denote the set of vertices in H which are at hop distance at most 1 from v, i.e., the set of all neighbors of v (including the vertex v itself).

In any $(h-1, \mu^2)$-hypertree, a weight $\omega_i \in Q_i(\mu^2)$, $\omega_i = i \cdot \mu^2 + j$, for $0 \le j \le \mu^2 - 1$, can be represented by the pair of weights

$$y_0 = j \bmod \mu \quad \text{and} \quad y_1 = \left\lfloor \frac{j}{\mu} \right\rfloor,$$

such that $y_0, y_1 \in [0, \mu - 1]$ and $\omega_i = y_0 + \mu \cdot y_1 + \mu^2 \cdot i$.

Consequently, one can associate with any hypertree $H' \in \mathcal{C}(h-1, \mu^2)$ a hypertree $H = (H_0, H_1, x) \in \mathcal{C}(h, \mu, x)$ as follows. Every vertex a' of H' is now associated with two homologous vertices of H, namely, the vertex a_0 (occurring in the left part of H, i.e., H_0), and the vertex a_1 (occurring in H_1). For any edge e' of H' with weight $\omega_{e'} = y_0 + \mu \cdot y_1 + \mu^2 \cdot i$ (for some i), let the weight of e_0 (respectively, e_1), the corresponding edge of e in H_0 (resp., H_1), be $\omega_{e_0} = y_0 + \mu \cdot i$ (resp., $\omega_{e_1} = y_1 + \mu \cdot i$) for the i defined above. Note, that ω_{e_0} and ω_{e_1} are uniquely determined by $\omega_{e'}$ and vice versa. In addition, for every two homologous vertices $a_0 \in H_0$ and $a_1 \in H_1$, let the weight of the path $Path(a_0, a_1)$ in H be x. See Fig. 2 for illustrating how to associate H' with H.

The following claim is straightforward.

Claim 1. The hypertree $H' \in \mathcal{C}(h-1, \mu^2)$ is legal iff the associated hypertree $H \in \mathcal{C}(h, \mu, x)$ is legal,
2. $f_{MST}(H) = 1$ iff $f_{MST}(H') = 1$.

Using the marking of H to derive a marking for H': We use the claim above to derive a proof labeling scheme $\pi' = \langle \mathcal{M}', \mathcal{V}' \rangle$ for f_{MST} and $\mathcal{C}(h-1, \mu^2)$ (satisfying the identity property) using at most $|X(x)|$ labels. This will show that $X(x)$ is large, as claimed in the lemma. Given an $(h-1, \mu^2)$-hypertree H', consider the hypertree $H \in \mathcal{C}(h, \mu, x)$ associated with H' and use the marker algorithm $\hat{\mathcal{M}}$ to label $H = (H_0, H_1, x)$. Denote by $\hat{L}(v, H)$ the label assigned to a vertex $v \in H$ by marker $\hat{\mathcal{M}}$ applied on H. We now use these labels to define the marker algorithm $\mathcal{M}'$ for the nodes of H' as follows.

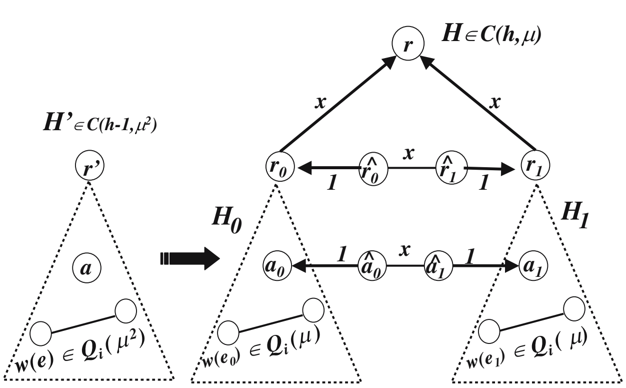

Fig. 2 Associating H' with H

For every vertex $a' \in H'$, let a_0 and a_1 be the two homologous vertices in H associated with a'. Vertex $a' \in H'$ receives $\mathcal{M}'(a') = \left\langle id(a'), \hat{L}(a_0, H), \hat{L}(a_1, H), x \right\rangle$ as its label. Note, that for every vertex $a' \in H'$, the label $\mathcal{M}'(a')$ contains four sublabels, namely, $\mathcal{M}'(a') = \langle \mathcal{M}'_1(a'), \mathcal{M}'_2(a'), \mathcal{M}'_3(a'), \mathcal{M}'_4(a') \rangle$. Clearly, given $id(a')$, one can reconstruct the identities $id(a_0)$ and $id(a_1)$ and vise versa. (Recall that the id of a vertex in a hypertree is the preorder number of the node in the spanning tree of the hypertree.) Therefore, since the identities of the vertices are encoded in the left fields of the corresponding labels, the set $\{\left\langle id(a'), \hat{L}(a_0, H), \hat{L}(a_1, H), x \right\rangle \mid a' \in H'$ and $H' \in \mathcal{C}(h-1, \mu^2)\}$ contains the same number of items as the set $\{\left\langle \hat{L}(a_0, H), \hat{L}(a_1, H) \right\rangle \mid a' \in H'$ and $H' \in \mathcal{C}(h-1, \mu^2)\} \subseteq X(x)$. The following claim follows:

Claim The number of labels used by marker $\mathcal{M}$ is at most $|X(x)|$.

It remains to show that $\mathcal{M}'$ is the marker algorithm of a correct proof labeling scheme π for MST.

Constructing verifier $\mathcal{V}'$: Given a labeling algorithm L' applied on H', let $L'(a') = \langle L'_1(a'), L'_2(a'), L'_3(a'), L'_4(a') \rangle$ denote the label assigned to a' by L'. The verifier $\mathcal{V}'(N_{L'}(a'))$ at a vertex $a' \in H'$ outputs 1 iff the following conditions hold. (Informally, the conditions check that L' could really result from performing M' as defined above.)

1. $L'(a') = id(a')$.
2. For every vertex $b' \in Neigh_{H'}(a')$, $L'_4(b') = L'_4(a')$. (In the case where L' is $\mathcal{M}'$, this value is x).
3. (The guessing stage:) There exists a labeling assignment L for the nodes in $Neigh_H(a_0) \cup Neigh_H(a_1)$ with the following properties. Consider any vertex $b' \in Neigh_{H'}(a')$, and let b_0, b_1 be the two homologous vertices in H associated with b' as defined above. Labeling assignment L is required to be such that $L(b_0) = L'_2(b')$ and $L(b_1) = L'_3(b')$. Moreover, assuming the weight of $Path(a_0, a_1)$ is $L'_4(a')$, verifier $\mathcal{V}'$ performs a simulated execution of verifier $\hat{\mathcal{V}}$ using L. Verifier $\mathcal{V}'$ then checks that for the simulated execution, the following hold:
   (a) If $a' \neq r'$ (i.e., $id(a') \neq 1$), then the condition here is satisfied if in the simulated execution of verifier $\hat{\mathcal{V}}$ at $a_0, \hat{a}_0, \hat{a}_1$ and a_1 the following holds: $\hat{\mathcal{V}}(N_L(a_0)) = \hat{\mathcal{V}}(N_L(\hat{a}_0)) = \hat{\mathcal{V}}(N_L(\hat{a}_1)) = \hat{\mathcal{V}}(N_L(a_1)) = 1$.
   (b) If $a' = r'$ (i.e., $id(a') = 1$), then the condition here holds if in the simulated execution at $r_0, \hat{r}_0, \hat{r}_1, r_1$ and r the following holds: $\hat{\mathcal{V}}(N_L(r_0)) = \hat{\mathcal{V}}(N_L(\hat{r}_0)) = \hat{\mathcal{V}}(N_L(\hat{r}_1)) = \hat{\mathcal{V}}(N_L(r_1)) = \hat{\mathcal{V}}(N_L(r)) = 1$.

Proving scheme $(\mathcal{M}', \mathcal{V}')$: Let us now show that scheme $\pi' = \langle \mathcal{M}', \mathcal{V}' \rangle$ is a correct proof labeling scheme for f_{MST} and $\mathcal{C}(h-1, \mu^2)$ (satisfying the identity property). Let H' be a hypertree in $\mathcal{C}(h-1, \mu^2)$ such that $f_{MST}(H') = 1$. Clearly, for every vertex $a' \in H'$, Conditions 1 and 2 in the description of verifier $\mathcal{V}'$ are satisfied if the labels are given by marker $\mathcal{M}'$. Let H be the corresponding hypertree in $\mathcal{C}(h, \mu, x)$ and for each vertex $v \in H$, let $\hat{\mathcal{M}}(v)$ be the label assigned to v by marker algorithm $\hat{\mathcal{M}}$. By Claim 4, $f_{MST}(H) = 1$ and therefore, by the correctness of proof labeling scheme $\hat{\pi}$, we obtain that for every vertex $v \in H$, $\hat{\mathcal{V}}(v) = 1$. It follows that for every vertex $a' \in H'$, Condition 3 in the description of verifier $\mathcal{V}'$ is also satisfied. Altogether, we obtain that for every vertex $a' \in H'$, $\mathcal{V}'_{\mathcal{M}'}(a') = 1$.

Assume now that for every vertex $a' \in H'$, $\mathcal{V}'_{L'}(a') = 1$ for some labeling algorithm L' applied on H'. Our goal is to show that $f_{MST}(H') = 1$. Let H be the corresponding hypertree in $\mathcal{C}(h, \mu, x)$. We now define a marker algorithm L assigning each vertex $v \in H$ a label $L(v)$ defined as follows. For every vertex $a' \in H'$, let $L(a_0) = L'_2(a')$ and let $L(a_1) = L'_3(a')$. For every non-root vertex $a' \in H'$, we know that $\mathcal{V}'$ guessed some labels that satisfied condition 3(a) (since we assume here that $\mathcal{V}'_{L'}(a') = 1$). Let these labels be $L(\hat{a_0})$ and $L(\hat{a_1})$ of $\hat{a_0}$ and $\hat{a_1}$, respectively. Similarly, let $L(\hat{r_0})$, $L(\hat{r_1})$ and $L(r)$ (where r is the root of H) be the labels of $\hat{r_0}$, $\hat{r_1}$ and r respectively, such that by guessing them $\mathcal{V}'$ found that Condition 3(b) is satisfied. If follows that for every vertex $v \in H$, $\hat{\mathcal{V}}_L(v) = 1$ and therefore, by the correctness of $\hat{\pi}$, $f_{MST}(H) = 1$. By Claim 4, $f_{MST}(H') = 1$ and the correctness of π' follows.

Since scheme $\pi' = \langle \mathcal{M}', \mathcal{V}' \rangle$ uses at most $|X(x)|$ labels, we obtain that $|X(x)| \geq g(h-1, \mu^2)$. $\square$

Combining Claim 4 and Lemmas 4 and 5, we obtain the following;

Corollary 1 $g(h, \mu) \geq \sqrt{\mu} \cdot \sqrt{g(h-1, \mu^2)}$.

Subsequently, we obtain the following lemma.

Lemma 6 $g(h, \mu) \geq \mu^{h/2}$.

This allows us to conclude with the lower bound.

Theorem 2 *In the case where* $W > (\log n)^{1+\epsilon}$ *for some fixed* $\epsilon > 0$*, the size of any proof labeling scheme for* f_{MST} *and* $\mathcal{F}(n, W)$ *is* $\Omega(\log n \cdot \log W)$.

Proof For $W > (\log n)^{1+\epsilon}$ for some fixed $\epsilon > 0$, let π be any proof labeling scheme for f_{MST} and $\mathcal{F}(n, W)$ satisfying the identity property. Let $h = \log(n+1)$ and let $\mu = (W+1)/h$. Scheme π is in particular a proof labeling scheme for f_{MST} and $\mathcal{C}(h, \mu)$. Therefore, the size of π is at least $\log(g(h, \mu))$. By Lemma 6, the size of π is at least $\frac{h}{2} \log \mu = \frac{\log(n+1)}{2} \cdot \log\left(\frac{W+1}{h}\right) = \frac{\log(n+1)}{2} \cdot \log(W+1) - \frac{\log(n+1)}{2} \cdot \log\log(n+1)$. Since $W > (\log n)^{1+\epsilon}$ for some fixed $\epsilon > 0$, it follows that $\log\log n = o(\log W)$ and we obtain that the size of π is $\Omega(\log n \cdot \log W)$. The theorem follows by the fact that with an extra additive cost of $O(\log n)$ to the label size, any proof labeling scheme for f_{MST} and $\mathcal{F}(n, W)$ can be transformed into a proof labeling scheme for f_{MST} and $\mathcal{F}(n, W)$ satisfying the identity property. $\square$

5 Sensitivity testing

As mentioned before, in the sequential setting, a related problem to MST verification is the following sensitivity testing problem: given a graph and an MST of the graph G, label every edge (u, v) with the minimum number $c(u, v)$ such that if the weight of the edge changes by $c(u, v)$ (and the weights of the rest of the edges remain the same) then the given tree is no longer minimum. The number of bits used to store the output of an algorithm π per edge is referred to as the space complexity of π. Observe that if each $c(u, v)$ must be given explicitly (in binary), then any algorithm solving this problem must outputs at least $\log c(u, v)$ bits, leading to a space complexity of $\Theta(\log W \cdot |E|)$, for example in graphs where the weights of all the edges are within a constant multiplicative factor of each other.

In this paper, we consider a slightly weaker variant of the sensitivity problem, in which the requirement from the output is relaxed as follows. Instead of writing the sensitivity $c(u, v)$ of each edge (u, v) explicitly, we write some auxiliary information. Later, when queried about the sensitivity of an edge, we are allowed to perform a constant time computation to derive the sensitivity of that edge, using the above auxiliary information and the given graph G. Our sensitivity testing problem is referred to as the *weak sensitivity problem*.

We describe a deterministic algorithm solving the weak sensitivity problem. For dense graphs (specifically, when $n \log n = o(|E|)$), this improves the best possible space complexity of any algorithm solving the original sensitivity problem. Moreover, let π be any algorithm solving the original sensitivity problem and let $Time(\pi)$ be the time complexity of π (using the same model for time as the one used in [7]). Given a graph and an MST, our computation of the auxiliary information incurs $O(Time(\pi) + n \log n)$ time. Hence, for graphs for which we improve the space complexity, the running time of our algorithm is not worse than the running time of π.

Lemma 7 *Let* π *be any algorithm solving the original sensitivity problem and let* $Time(\pi)$ *be the time complexity of* π*. There exists an algorithm solving the weak sensitivity problem with space complexity* $O(\log n \log W)$*-bits per node, and time complexity* $O(Time(\pi) + n \log n)$.

Proof Given a graph and an MST T of the graph, the auxiliary information consists of assigning a label $L(v)$ to each node v in the graph. For every node v, the label $L(v)$ contains two sublabels, namely $L_T(v)$ and $L_{\gamma_{small}}(v)$.

We first run π, but instead of writing $c(v, u)$ on each edge (v, u), we do the following. Let v be a non-root vertex in T and let $p(v)$ be v's parent in the tree. The sublabel $L_T(v)$ is the value $c(v, p(v))$ as given by π. If r is the root of T then $L_T(r)$ is empty. The sublabel $L_{\gamma_{small}}(v)$ of each vertex v is simply the label given to v by the implicit labeling scheme γ_{small} as described in Sect. 3.1.2.

When queried about the sensitivity $c(u, v)$ of a some edge (u, v), the following happens. If (u, v) is a tree edge and u is a child of v in T, then $c(u, v)$ is written in $L_T(u)$. If (u, v) is a non-tree edge, then $MAX(u, v)$ can be extracted using the decoder of γ_{small} (which runs in a constant time). The sensitivity $c(u, v)$ is then extracted using the fact that $c(u, v) = \omega(u, v) - Max(u, v)$, where $\omega(u, v)$ is the weight of (u, v) in G.

Clearly, a sensitivity query can be answered in constant time using our auxiliary information and the given graph G. Furthermore, the fact that the auxiliary information uses $O(\log n \cdot \log W)$ bits per node follows from Lemma 2. The time required to construct the sublabels $L_T(\cdot)$ is $Time(\pi)$. In addition, it can be easily shown that the time required to construct the sublabels $L_{\gamma_{small}}(\cdot)$ is $O(n \log n)$. (Clearly, in a perfect decomposition, a vertex participates in up to $\log n$ partitions; in each, the vertex has to be marked by the unique name of the subtree and by the heaviest edge on the route to the root of the subtree; those can be calculated for the whole subtree by passing once on every edge of the subtree, walking down from the subtree root.) □

Returning to the distributed setting, one can define the sensitivity problem in several ways. We define the *distributed sensitivity problem* as follows. Given a graph G and an MST of G, label each vertex v by a label $L(v)$, such that the following holds. For every vertex u and each neighbor v of u, the sensitivity $c(u, v)$ can be extracted in constant distributed time, using a distributed algorithm that is invoked at u. Note, that the preprocessing algorithm that labels the vertices is not required to be distributed. However, the sensitivity query is distributed in the sense that when queried about the sensitivity of an adjacent edge (u, v), vertex u is only allowed to look at its own label $L(v)$ and to invoke a distributed algorithm (that is subject to the constraints in the common message passing model, for example, the only way to move information between nodes is between neighboring nodes, and only by using messages).

We evaluate an algorithm solving the distributed sensitivity problem by its label size, i.e, the maximum number of bits used in a label. Using the same labels as given by the algorithm described in the previous lemma, the following lemma can easily be obtained.

Theorem 3 *There exists an algorithm solving the distributed sensitivity problem with label size* $O(\log n \log W)$.

6 Conclusion

Our bounds are tight only for W such that $\log\log n = o(\log W)$. The lower bound proof does not carry to the cases of a smaller W, and the question of tight bounds in these cases are still an open problem. Note, that for a large W the existing lower bound depends on W, while for a small W this dependence disappears. This seems counter intuitive. At first glance, it seems that **some** factor that depends on W should become easier to prove in the lower bound when W shrinks.

Another interesting direction of research would be to construct efficient *distributed* marker algorithms, as opposed to the algorithm described here that was not designed for an efficient distributed implementation.

We used some tools taken from the the theory of implicit labeling schemes. We used them both for proving the lower bound and for constructing the algorithm for the upper bound. See [20] for implicit labeling schemes for adjacency queries, and [15] for a comprehensive survey of implicit labeling schemes. This may raise the hope that the theory of implicit labeling scheme may be shown useful for studying proof labeling schemes further.

Finally, we should mention that not much is known about the complexity of proof labeling schemes in general. We hope that the current paper can serve as an example that problems in this area are an interesting subject of study.

References

1. Afek, Y., Kutten, S., Yung, M.: The local detection paradigm and its applications to self stabilization. Theor. Comput. Sci. **186**, 199–230 (1997)
2. Arora, A., Gouda, M.G.: Distributed reset. IEEE Trans. Comput. **43**(9), 1026–1038 (1994)
3. Awerbuch, B., Varghese, G.: Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In: Proceedings of the 32nd annual symposium on foundations of computer science, pp. 258–267 (1991)
4. Booth, H., Westbrook, J.: Linear algorithms for analysis of minimum spanning and shortest paths trees in planar graphs. Algorithmica **11**(4), 341–352 (1994)
5. Cidon, I., Gopal, I., Kaplan, M., Kutten, S.: A distributed control architecture of high-speed networks. IEEE Trans. Commun. **43**(5), 1950–1960 (1995)
6. Deering, S., Estrin, D., Farinacci, D., Jacobson, V., Liu, C., Wei, L.: Protocol independent multicast (PIM): motivation and architecture. Internet Draft draft-ietf-pim-arch-01.ps (1995)
7. Dixon, B., Rauch, M., Tarjan, R.E.: Verification and sensitivity analysis of minimum spanning trees in linear time. SIAM J. Comput. **21**(6), 1184–1192 (1992)
8. Dixon, B., Tarjan, R.E.: Optimal parallel verification of minimum spanning trees in logarithmic time. Algorithmica **17**(1), 11–18 (1997)

9. Dolev, S., Gouda, M.G., Schneider, M.: Memory requirements for silent stabilization. Acta Informatica **36**(6), 447–462 (1999)
10. Dolev, S., Israeli, A., Moran, S.: Uniform dynamic self-stabilizing leader election. IEEE Trans. Parallel Distrib. Syst. **8**(4), 424–440 (1997)
11. Fraigniaud, P., Gavoille, C.: Routing in trees. In: Proceedings 28th international colloq. on automata, languages & prog. LNCS 2076, 757–772 (2001)
12. Fredman, M.L., Willard, D.E.: Trans-Dichotomous algorithms for minimum spanning trees and shortest paths. In: Proceedings of the 31st IEEE annual symposium on foundations of computer science, Los Alamitos, CA, pp. 719–725 (1990)
13. Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. ACM Trans. Prog. Lang. Syst. **5**, 66–77 (1983)
14. Gavoille, C., Katz, M., Katz, N.A., Paul, C., Peleg, D.: Approximate distance labeling schemes. In 9th European symposium on algorithms, Aarhus, Denmark, SV-LNCS 2161, pp. 476–488 (2001)
15. Gavoille, C., Peleg, D.: Compact and localized distributed data structures. Distrib. Comput. **16**(2–3), 111–120 (2003)
16. Gavoille, C., Peleg, D., Pérennes, S., Raz, R.: Distance labeling in graphs. In: Proceedings of the 12th ACM-SIAM symposiun on discrete algorithms, pp. 210–219 (2001)
17. Harel, D.: A linear time algorithm for finding dominators in flow graphs and related problems. In: Proceedings of the 17th annual ACM symposium on theory of computing, Salem, MA. pp. 185–194 (1985)
18. Jayaram, M., Varghese, G.: The complexity of crash failures. In: Proceedings of the ACM symposium on principles of distributed computing, pp. 179–188 (1997)
19. Jayaram, M., Varghese, G.: Crash failures can drive protocols to arbitrary states. In: Proceedings of the ACM symposium on principles of distributed computing, pp. 247–256 (1996)
20. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. SIAM J. Discrete Math **5**, 596–603 (1992)
21. Katz, M., Katz, N.A., Korman, A., Peleg, D.: Labeling schemes for flow and connectivity. In: Proceedings of the 19th ACM-SIAM symposium on discrete algorithms, pp. 927–936 (2002)
22. Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm to find minimum spanning trees. J. ACM **42**(2), 321–328 (1955)
23. Komlòs, J.: Linear verification for spanning trees. Combinatorica **5**, 57–65 (1985)
24. Korman, A., Kutten, S., Peleg, D.: Proof Labeling Schemes. In: Proceedings of the 24th annual ACM symposium on principles of distributed computing, Las Vegas, NV, USA (2005)
25. Kutten, S., Peleg, D.: Fast distributed construction of small k-dominating sets, and applications. J. Algorithms **28**(1), 40–66 (1998)
26. King, V.: A simpler minimum spanning tree verification algorithm. Algorithmica **18**, 263–270 (1997)
27. King, V., Poon, C.K., Ramachandran, V., Sinha, S.: An optimal EREW PRAM algorithm for minimum spanning tree verification. Inform. Process. Lett. **62**(3), 153–159 (1997)
28. Kuhn, F., Watenhoffer, R.: Constant time distributed dominating set approximation. Distrib. Comput. **17**(4), 303–310 (2005)
29. Kuhn, F., Moscibroda, T., Watenhoffer, R.: What cannot be computed locally! In: Proceedings of the ACM symposium on principles of distributed computing, pp. 300–309 (2004)
30. Naor, M., Stockmeyer, L.J.: What can be computed locally?. SIAM J. Comput. **24**(6), 1259–1277 (1995)
31. Pettie, S., Ramachandran, V.: An optimal minimum spanning tree algorithm. J. ACM **49**(1), 16–34 (2002)
32. Tarjan, R.E.: Sensitivity analysis of minimum spanning trees and shortest paths trees. Inform. Process. Lett. **14**, 30–33 (1982)
33. Tarjan, R.E.: Data Structures and Network Algorithms. SIAM, Philadelphia (1983)
34. Tarjan, E.E.: Applications of path compression on balanced trees. J. ACM **26**, 690–715 (1979)
35. Yao, A.C.: Some complexity questions related to distributed computing. In: Proceedings of 11th annual ACM symposium on theory of computing, pp. 209–213 (1979)