COMBINATORICA
Bolyai Society – Springer-Verlag

Combinatorica **26** (2) (2006) 207–230

AN INVERSE-ACKERMANN TYPE LOWER BOUND FOR ONLINE MINIMUM SPANNING TREE VERIFICATION*

SETH PETTIE[†]

Received April 15, 2003

Given a spanning tree T of some graph G, the problem of *minimum spanning tree verification* is to decide whether $T = MST(G)$. A celebrated result of Komlós shows that this problem can be solved with a linear number of comparisons. Somewhat unexpectedly, MST verification turns out to be useful in actually computing minimum spanning trees from scratch. It is this application that has led some to wonder whether a more flexible version of MST verification could be used to derive a faster deterministic minimum spanning tree algorithm. In this paper we consider the *online* MST verification problem in which we are given a sequence of *queries* of the form "Is e in the MST of $T \cup \{e\}$?", where the tree T is fixed. We prove that there are no linear-time solutions to the online MST verification problem, and in particular, that answering m queries requires $\Omega(m\alpha(m,n))$ time, where $\alpha(m,n)$ is the inverse-Ackermann function and n is the size of the tree. On the other hand, we show that if the weights of T are permuted randomly there is a simple data structure that preprocesses the tree in expected linear time and answers queries in constant time.

1. Introduction

The minimum spanning tree (MST) problem has seen a flurry of activity lately, driven largely by the success of a new approach to the problem. The recent MST algorithms [22,9,32,31], despite their superficial differences, are

Mathematics Subject Classification (2000): 05C38, 68R10, 68W01

* A preliminary version of this paper appeared in the proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS 2002), pages 155–163.

[†] This work was supported by Texas Advanced Research Program Grant 003658-0029-1999, NSF Grant CCR-9988160, and an MCD Graduate Fellowship.

all based on the idea of progressively improving an *approximately* minimum solution until the actual minimum spanning tree is found. It is still possible that this progressive improvement approach will continue to bear fruit. However the current techniques seem to have been pushed to the limit. For instance, the technique of random sampling combined with Komlós's MST verification routine, pioneered by Karger, Klein, and Tarjan [22], yields an expected linear-time randomized MST algorithm, settling the randomized complexity of the problem. Despite some success in reducing the dependence on random bits (see the algorithms of Pettie and Ramachandran [31]) it does not seem possible to fully derandomize the Karger et al. [22] algorithm. The MST algorithms of Chazelle [9] and Pettie and Ramachandran [32] are based on Chazelle's Soft Heap [10], a priority queue that can be construed as sampling in a *deterministic* fashion. Chazelle's algorithm runs in super-linear $O(m\alpha(m,n))$ time, where m and n are the number of edges and vertices, respectively, and α is the slowly growing inverse-Ackermann function. The running time of the Pettie-Ramachandran algorithm [32] is asymptotically equal to the decision-tree complexity of MST, and is therefore optimal. In light of Chazelle's algorithm [9] the decision-tree complexity of MST is somewhere between $\Omega(m)$ and $O(m\alpha(m,n))$.

It is not possible to improve the asymptotic performance of Komlós's MST verification algorithm [26], used in [22,31], nor Chazelle's Soft Heap [10], used in [9,32,31]. It is then natural to ask, assuming that Chazelle's algorithm [9] is not optimal, what data structure will be the basis of the next MST algorithm? In [9, p. 1029] Chazelle speculates on the subject:

> Given a spanning tree T, to verify that it is minimum can be done in linear time [Dixon et al. 1992; King 1997; Komlós 1985]. The problem is to check that any edge outside T is the most expensive along the cycle it forms with T. With real costs this can be viewed as a problem of computing over the semigroup $(\mathbb{R}, \max)$ along paths of a tree. Interestingly, this problem requires $\Omega(m\alpha(m,n))$ time over an arbitrary semigroup [Chazelle and Rosenberg 1991; Tarjan 1978]. This lower bound suggests that in order to improve upon our algorithm specific properties of $(\mathbb{R}, \max)$ will have to be exploited. This is done statically in [Dixon et al. 1992; King 1997; Komlós 1985]. We speculate that an answer might come from a dynamic equivalent.

In this paper we study the complexity of one such "dynamic equivalent" to MST verification, namely the *online* MST verification problem. Whereas the offline MST verification problem is given a tree T (the alleged MST) and a set of non-tree edges, an online MST verifier is presented each non-tree edge e one at a time. It must decide, before the next edge is presented, whether $e \in MST(T \cup \{e\})$. We prove that there is no linear-time online MST verifier, ruling out this sort of data structure as the basis of a faster explicit MST algorithm.

Theorem 1.1. *Any online minimum spanning tree verification algorithm performs $\Omega(m\alpha(m,n)+n)$ edge-weight comparisons, where m is the number of queries, n is the size of the fixed tree, and α is the inverse-Ackermann function.*

Perhaps restricting the MST verifier to answer one query at a time is too unrealistic. A simple corollary of Theorem 1.1 is that giving the MST verifier large *batches* of queries does not significantly affect the complexity of the problem.

Corollary 1.2. *Consider the problem of answering m MST verification queries, presented in k batches, where each batch of queries must be answered before receiving the next. Any algorithm for this problem performs $\Omega(m\alpha(\frac{m}{k}, \frac{n}{k})+n)$ edge-weight comparisons.*

For instance, if queries are grouped into $\log^* n$ batches of size $m/\log^* n$, Corollary 1.2 implies that the complexity of the problem remains $\Omega(m\alpha(m,n)+n)$.

1.1. Previous Work

The minimum spanning tree verification problem is ostensibly about minimum spanning trees; however, as Chazelle noted, it can also be viewed as a partial sums (or range searching) problem over the semigroup $(\mathbb{R}, \max)$. The range searching problem has been the subject of intense study for quite some time (see, e.g., the survey of Agarwal & Erikson [2]). In a typical formulation of the problem we are to preprocess a set system (or *range space*) $(X, \mathcal{Q})$, where X is a set of weighted elements, whose weights are drawn from some semigroup, and $\mathcal{Q}$ is a collection of subsets of X, so that given any *query set* $Q \in \mathcal{Q}$ we can quickly determine the sum of the weights of the elements in Q. In a commonly studied scenario the elements X are identified with points in $\mathbb{R}^d$ and $\mathcal{Q}$ corresponds to the set of all regions of $\mathbb{R}^d$ with some nice structure, such as the simplices or d-rectangles. Under the assumption of an *arbitrary* semigroup $(S, \circ)$, the only operation allowed on a semigroup variable x is of the form $x := y \circ z$, where $y, z \in S$. Thus, for a fixed set system $(X, \mathcal{Q})$ any program in the arbitrary semigroup model must be a straight-line program. However for the semigroups $(\mathbb{R}, \max)$ and $(\mathbb{R}, \min)$ it is most natural to assume the *decision-tree* model, where the program decides which comparison to make based on the outcomes of previous comparisons.

If we view the MST verification problem as a range searching problem, the set of weighted elements X coressponds to the weighted tree edges, $\mathcal{Q}$ corresponds to the set of tree paths, and edge weights are drawn from $(\mathbb{R}, \max)$. This is not a geometric problem in general, although it subsumes one as a special case. If the input tree is a single path, MST verification becomes isomorphic to a 1-dimensional range searching problem where the query sets correspond to intervals. We will call this problem *interval maximum* if the semigroup is $(\mathbb{R}, \max)$ and *interval sum* under arbitrary semigroups. Similarly, *tree sum* refers to the MST verification problem when generalized to arbitrary semigroups.

Tarjan [35] gave an algorithm for the *offline* tree sum problem and other algorithms for certain online variants of tree sum. All the algorithms in [35] are based on the same path-compression technique used in the standard union-find algorithm [33] and therefore run in time $\Theta(m\alpha(m,n))$ where n is the size of the tree and $m \geq n$ is the number of queries. Yao [39] proved that the query time of any online interval sum algorithm is $\Theta(\alpha(m,n))$ if it uses $m \geq n$ units of storage. See also [34,4] for related results. Chazelle [8] showed that algorithms on intervals (such as one solving interval sum) can be systematically translated into algorithms on trees. This yielded an online tree sum algorithm which answers queries in $\Theta(\alpha(m,n))$ time using $m \geq n$ units of storage. Chazelle and Rosenberg [11] strengthened the results of Yao [39] and Alon and Schieber [4] by showing that *offline* interval sum is just as hard as online interval sum. In other words, there exist m queries that can only be answered using $\Theta(m\alpha(m,n))$ operations. Tarjan [37] showed that the complexity of any offline subset sum problem over an arbitrary semigroup is precisely the same as its dual.[1] Tarjan's result was used to prove an $O(m\alpha(m,n))$ upper bound on the MST sensitivity analysis problem, which is the dual to MST verification. Together with [11] it also proves an $\Omega(m\alpha(m,n))$ lower bound on any *oblivious*[2] MST sensitivity analysis algorithm.

The complexity of all these problems seems to change when we substitute the specific semigroup $(\mathbb{R}, \max)$ in place of an arbitrary semigroup. Komlós [26] (see also [17,6]) gave a simple reduction from the (online) interval-

[1] A subset sum problem can be represented as a zero-one matrix M where $M[i,j]=1$ indicates element i is in set j. The dual of this subset sum problem is M^T, i.e. elements become sets and sets become weighted elements.

[2] In this model an oblivious algorithm is a fixed sequence of operations of the form $a := \max\{b, c\}$ where a, b, and c are variables or inputs. The output of the algorithm is a certain fixed set of variables.

maximum problem to the (online) least common ancestors problem, which, together with Harel and Tarjan's online LCA algorithm [20], implies that interval-maximum queries can be answered in $O(1)$ time after an $O(n)$-time preprocessing phase. Compare this with the inverse-Ackermann type lower bounds of [39, 4, 11]. Komlós [26], making use of his interval-maximum algorithm, showed that *offline* MST verification can be solved with $O(m+n)$ comparisons, where n and m are the number of tree-edges and non-tree edges, respectively. Much work has been devoted to finding a simple implementation of Komlós's algorithm in a simple model of computation [14, 23, 7, 5]. The MST sensitivity analysis problem was investigated in [14]; they gave two algorithms: a randomized, expected linear-time algorithm and a provably optimal algorithm with unknown running time. A recent improvement [30, 28] to the split-findmin data structure [16] implies that MST sensitivity analysis can be solved deterministically in $O(m \log \alpha(m,n))$ time.

Given this history one might be tempted to conjecture that *all* partial sums/range searching problems are significantly easier under $(\mathbb{R}, \max)$, as opposed to an arbitrary semigroup.[3] Our main result disproves this conjecture, and it is quite surprising given the known linear-time algorithms for similar problems. Our result represents the first inverse-Ackermann type lower bound on a *purely* comparison-based problem.

Because the elementary operation in our model is the *comparison*, as opposed to, e.g., the cell probe [15], the matrix query [24], or the pointer manipulation [36], our proof technique is information-theoretic in nature. This is in contrast to traditional inverse-Ackermann style lower bounds [36, 39, 4, 11, 24, 21, 15], which are based on mostly structural properties of the problem or problem instance. We suspect that our techniques will be useful in lower-bounding other problems in a comparison-based model of computation. We give two such candidates in Section 5.

1.2. Organization

Section 2 defines our notation and a class of hard problem instances. The lower bound proper appears in Section 3. In Section 4 we give almost matching upper bounds for online MST verification, and show that the problem

[3] This conjecture basically holds for *random* offline partial sums problems: nearly all such problems on n elements and m sets require $\Theta(mn/\log m)$ semigroup operations to solve (see [34]), whereas they require an expected $O(n \log \min\{n, \frac{m+n}{n}\})$ comparisons to solve [18] when the semigroup is $(\mathbb{R}, \max)$.

becomes significantly easier when the input edge-weights are permuted randomly. We discuss some open problems in Section 5.

2. Preliminaries

The problem is to answer, in an online fashion, a sequence of m minimum spanning tree verification queries. Each is of the form: given an edge e, decide whether $e \in MST(T \cup \{e\})$, where T is a *fixed*, edge-weighted tree. By the *cycle property* of minimum spanning trees this is equivalent to asking whether e is not the heaviest edge on the unique cycle in $T \cup \{e\}$. We restrict the types of input trees and queries as follows:

- The fixed tree T is a full, rooted binary tree.
- Every query edge connects a leaf to one of its ancestors.

For the sake of simpler notation we will actually consider trees that are vertex-weighted. (Hence we decide if e is heavier than all *vertices* in the unique cycle of $T \cup \{e\}$.) It is easy to transform an edge-weighted tree T' into an equivalent vertex-weighted tree T satisfying $|T| < 2|T'|$.[4] Since we are only considering leaf-to-ancestor queries we can use a simpler transformation that does not increase the size of the tree. We let the weight of a vertex v in T be the weight of the edge $(v, \text{parent}(v))$ in T'. The weight of the root is irrelevant. A query (u,v) in T' is equivalent to the query (u,v') in T, where v' is the child of v that is an ancestor of u.

It is clear that the query edge must participate in at least one comparison. The parameter $t \geq 1$ used throughout the paper represents the desired number of comparisons per query. We will prove that for each t there is an input distribution $\mathit{Distr}(t)$ such that for some query, either (a) Answering the query requires at least $t+1$ comparisons (worst case or amortized) or (b) The verification algorithm already performed $cn \log \lambda_t(n)$ comparisons preceeding the query, where c is an absolute constant and λ_t is the t^{th}-row inverse of a function similar to Ackermann's function. By judiciously setting $\tau(m,n) = \max\{t : cn \log \lambda_t(n) \geq tm\}$, this implies that answering m verification queries requires $m \cdot \tau(m,n)$ comparisons. It is then straightforward to show that $\tau(m,n)$ is always within an absolute constant of the traditional inverse-Ackermann function defined by Tarjan [33].

[4] The transformation: For each edge $(u,v) \in T'$, with v the parent of u, we add two edges (u,w) and (w,v), where w is a new vertex, u has weight $-\infty$ and w has the same weight as (u,v).

2.1. A Variation on Ackermann's Function

In the field of algorithms & complexity Ackermann's function [1] is rarely defined the same way twice [1,36,15,11,12,9,13]. We would not presume to buck such a well established precedent. Here is a slight variant:

$$\begin{array}{rcll} A(1,j) &=& 2^j & j \geq 0; \\ A(i+1,0) &=& A(i,1) & i \geq 1; \\ A(i+1,j+1) &=& A(i, 2^{2^{A(i+1,j)}}) & i \geq 1,\ j \geq 0. \end{array}$$

Let $\lambda_i(n)$ be the inverse of the i^{th} row of A, defined as:

$$\lambda_i(n) = \min\{j \ : \ A(i,j) \geq n\}.$$

2.2. The Input Distribution *Distr*(t)

We denote nodes of the fixed tree T with lower case letters. If q is a tree node we let $w(q)$ be the weight of q and size(q) be the number of leaf-descendants of q. Since T is a full binary tree, size(q) is always a power of two. Note also that $A(i,j)$ is always a power of two. The input distribution *Distr*(t) is fully characterized by Definitions 2.1 and 2.2 and Property 2.3.

Definition 2.1. In *Distr*(t), a non-leaf node p is an i-node if size$(p) = A(i,j)$ for $1 \leq i \leq t$ and some j, and an $\overline{i}$-node if it is an i-node but not an $(i+1)$-node. In *Distr*(t), leaf-nodes are $(t+1)$- and $\overline{(t+1)}$-nodes by definition.

Definition 2.2. Let p be an arbitrary $\overline{(i+1)}$-node and let q be the nearest $(i+1)$-node ancestor of p, or, if there is no such node, let q be the root. We define the set C_p to be those $\overline{i}$-nodes that lie on the path between p and q.

Property 2.3. Let p be a tree node. If p is a $\overline{1}$-node then $w(p)$, the weight of p, equals the height of p in T, i.e., $\log \text{size}(p)$. If p is an $\overline{i}$-node, for $i > 1$, then $w(p) = w(X_p)$, where X_p is a tree node selected uniformly at random from C_p, and independent of $\{X_q\}_{q \neq p}$ – see Figure 1.

In our analysis we will assume that the online MST verification algorithm knows that the permutation of tree weights was drawn from *Distr*(t). Therefore, a statement of the form "it is known that $w(p) < w(q)$" means the inequality $w(p) < w(q)$ follows from the inequalities discovered by the algorithm and the inequalities implicit in Property 2.3. For instance, the following Lemma follows from Property 2.3.

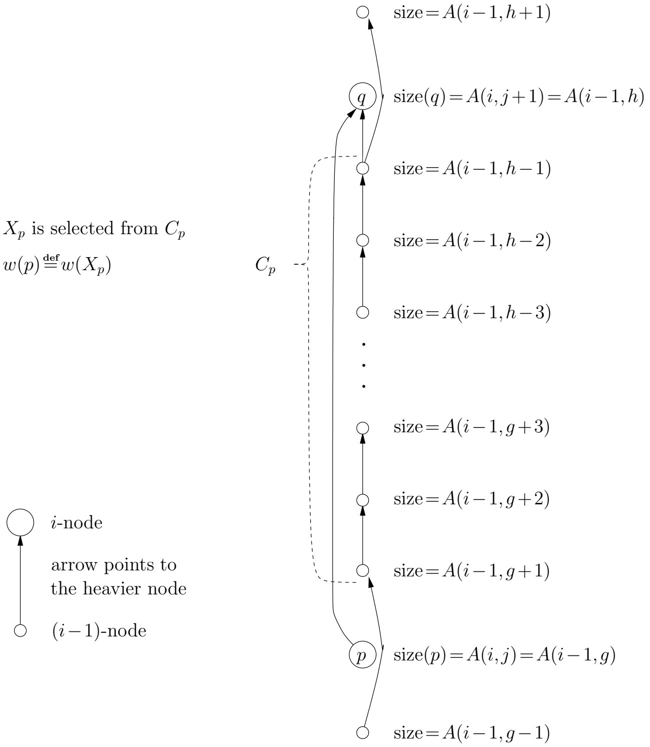

Figure 1. An $\overline{i}$-node p, where $i > 1$, and its associated set C_p. C_p consists of the $\overline{(i-1)}$-nodes between p and q, its nearest i-node ancestor. The weight of p is equal to that of some node in C_p selected uniformly at random.

Lemma 2.4. *Suppose p_1 is an $\overline{i_1}$-node, p_2 is an $\overline{i_2}$-node, $i_1 \leq i_2$, and p_1 appears at a lower level of T than p_2. Then $w(p_1) < w(p_2)$.*

Proof. Let size$(p_1) = A(i_1, j_1)$ and size$(p_2) = A(i_2, j_2)$. It follows from Property 2.3 that $w(p_1) \in [\log A(i_1, j_1), \log A(i_1, j_1 + 1))$ and $w(p_2) \in [\log A(i_2, j_2), \log A(i_2, j_2 + 1))$. Whether $i_1 = i_2$ and $j_1 < j_2$ or $i_1 < i_2$, it follows from the definition of A that $A(i_1, j_1 + 1) \leq A(i_2, j_2)$. ∎

A consequence of Lemma 2.4 is that, on any leaf-to-root path, the weights of the $\overline{i}$-nodes are monotonically increasing, for any i. This implies that any query can be answered in at most $t+1$ comparisons. If a query edge e connects a leaf to one of its ancestors there are at most $t + 2$ candidate maxima on the cycle in $T \cup \{e\}$: the most ancestral $\overline{i}$-node, for $1 \leq i \leq t+1$, plus the edge e itself.

2.3. A Measure of Information

Before the verification algorithm performs any comparisons it knows, by Property 2.3, that $w(q) = w(X_q)$, where q is any $\overline{i}$-node, $i > 1$, and X_q is uniformly distributed over C_q. We define, with respect to the current moment, the set $D_q \subseteq C_q$ as:

$$D_q \;=\; \{p \in C_q \;:\; w(q) = w(p) \text{ is consistent}\}.$$

That is, $C_q \backslash D_q$ consists of those $p \in C_q$ for which the equality $w(q) = w(p)$ is *impossible*, given Property 2.3 and all the inequalities discovered by the verification algorithm up until the current moment. It follows from Property 2.3 that D_q is non-empty. We will use the sizes of the C_q and D_q sets to roughly measure how much information is known about X_q. Define ϕ as:

$$\phi(q) = \begin{cases} 0 & \text{if } q \text{ is a } \overline{1}\text{-node}, \\ \log|C_q| + 2(t+1)^2 & \text{if } |D_q| = 1 \text{ and } \log|C_q| \geq 2(t+1)^2, \\ \log\frac{|C_q|}{|D_q|} & \text{otherwise.} \end{cases}$$

Define Φ as:

$$\Phi = \sum_{q \in T} \phi(q).$$

It would be preferable to define $\phi(q)$ as simply $\log(|C_q|/|D_q|)$. However we need to differentiate between the case when X_q is completely known and when there is a little uncertainty left in X_q (corresponding to $|D_q| = 1$ and $|D_q| > 1$, respectively.) Therefore, we add to $\phi(q)$ a little "bonus" of $2(t+1)^2$ whenever $|D_q|$ reaches 1. Lemma 2.5 relates the Φ measure to the number of bits of information learned about the verification algorithm.

Lemma 2.5. *$\Phi/2$ is a lower bound on the information learned about the tree-weights, as drawn from Distr(t).*

Proof. The number $\log(|C_q|/|D_q|)$ measures the bits of information learned about the variable X_q, in the special case when X_q is uniformly distributed over D_q and independent of $\{X_p\}_{p \neq q}$. This is certainly a lower bound on the *actual* number of bits of information learned about X_q. By definition, $\phi(q) \leq 2\log(|C_q|/|D_q|)$. Therefore $\Phi = \sum_q \phi(q)$ is at most twice the number of bits of information learned about the tree-weights. ∎

Our lower bound proof uses the Φ and ϕ values to argue about the existence of "hard" queries and to quantify their exact hardness. We show that if Φ is sufficiently small, where small is a function of n and t, then there exists a query that requires $t+1$ comparisons to be answered in the *worst case*. However there is a danger in continually forcing worst-case behavior. In so doing we may inadvertantly expose significantly more than one bit of information per comparison. That is, we would lose the ability to claim any relationship between comparisons and entropy/information. We prove amortized bounds by showing that either (a) the verification algorithm answers the query in at least $t+1$ comparisons or (b) after the query is answered, $\Phi/2$ increases by some amount much larger than $t+1$. Theorem 2.6 shows that if such a dichotomy exists then with high probability the amortized cost of the query is lower bounded by $t+1$ minus some small quantity.

Theorem 2.6. *Let π be an unknown permutation drawn from some known probability distribution. An algorithm performs a sequence of m operations by comparing elements of π. Let $\Psi_i \geq \Psi_{i-1}$ be a* lower bound *on the amount of information learned about π just before the ith operation. Suppose that it is known that the ith operation either performs at least ν comparisons or $\Psi_{i+1} - \Psi_i \geq \mu$. Let* $\mathrm{amort}(\nu,\mu,\epsilon) = \nu(1+(1+\epsilon)\nu/\mu)^{-1}$. *Let C be the total number of comparisons made in m operations and $M = m \cdot \mathrm{amort}(\nu,\mu,\epsilon)$. Then:*

$$\Pr[C \geq M] \geq 1 - 2^{-\epsilon M}.$$

Proof. Let I denote the number of bits of information learned about π after C comparisons. We have $\Psi_{m+1} \leq I$. It is simple to prove by induction[5] that $\Pr[I > (1+\delta)C] < 2^{-\delta C}$. Therefore, if $C < M$ then $\Pr[I > (1+\epsilon)M] < 2^{-\epsilon M}$. Let m_ν be the number of operations that perform at least ν comparisons and let m_μ be the number of operations that increase Ψ by at least μ. We have the following inequalities:

$$\begin{aligned} m &\leq m_\nu + m_\mu \\ &\leq C/\nu + I/\mu \\ &< M(1/\nu + (1+\epsilon)/\mu) \quad \text{(with prob. greater than } 1 - 2^{-\epsilon M}) \\ &= M(\mathrm{amort}(\nu,\mu,\epsilon))^{-1} = m\,. \end{aligned}$$

Therefore, $C < M$ only with probability less than $2^{-\epsilon M}$. ∎

In our lower bound proof $\Phi/2$ will take the role of Ψ in Theorem 2.6 and if we select the input from *Distr*(t) then ν will be $t+1$. We are basically free to set the other parameters. For $\epsilon = 1/t$ and $\mu = (t+1)^2$, Theorem 2.6 says

[5] A sketch of the proof: Let $\rho(c,f)$ be the probability that f bits of information are revealed in c comparisons. Then $\rho(1, f > 1) \leq 2^{-f}$ and $\rho(c, f \leq c) \leq 1$. Furthermore, ρ satisfies the relation: $\rho(c{+}1, f > c{+}1) \leq \max_{0<s\leq 1/2}\{s\rho(c, f{+}\log s) + (1{-}s)\rho(c, f{+}\log(1{-}s))\}$. It then follows by induction that $\rho(c, c+\delta) < 2^{-\delta}$.

that the actual amortized cost (C/m) is at least $\text{amort}(t+1,(t+1)^2,1/t)=t$ with probability $1-2^{-m}$. By increasing μ or decreasing ϵ the lower bound on the amortized cost can be driven arbitrarily close to $t+1$, with probability at least $1-2^{-\Omega(m)}$. For the sake of specificity, let the term *amortized cost* be w.r.t. $\epsilon=1/t$.

3. The Lower Bound

Theorem 1.1 will follow very easily from Lemma 3.1, given below. The remainder of this section will constitute a proof of Lemma 3.1.

Lemma 3.1. *Suppose that $\Phi < \frac{1}{8}n\log\lambda_t(n)$, where it is assumed that $\lambda_t(n) > 2^{3(t+1)^2}$. Then there exists a verification query whose amortized cost is at least t.*

The proof of Lemma 3.1 is structured as follows. We define a measure $\text{cost}(q)$ over the leaves q of the input tree and show that if any leaf's cost is sufficiently small, then there exists a "hard" query edge that connects that leaf to one of its ancestors. In particular, we generate a sequence of nodes $q_{t+1}, q_t, \ldots, q_1$ where q_{t+1} is a low-cost leaf (recall, leaves are $(t+1)$-nodes), and q_i is an $\bar{i}$-node ancestor of q_{i+1}. The query edge is then $e=(q_{t+1},q_1)$. We show that $q_{t+1}, q_t, \ldots, q_1$ are all candidate maxima on the unique cycle in $T\cup\{e\}$, which immediately implies that in the *worst case*, the verification algorithm must make $t+1$ comparisons to certify that e is heavier than $q_{t+1}, q_t, \ldots, q_1$. We then show that if, somehow, the verification algorithm got by with fewer than $t+1$ comparisons, then it must have caused Φ to increase by at least $2(t+1)^2$. By appealing to Theorem 2.6 we can then show that the amortized cost of the query is at least t.

Define $\text{cost}(q)$, where q is a leaf, as

$$\text{cost}(q) = \sum_{\substack{p \text{ ancestral to } q\\ (\text{including } q)}} \frac{\phi(p)}{\text{size}(p)} .$$

That is, we can think of p contributing $\phi(p)/\text{size}(p)$ to the cost of each of its $\text{size}(p)$ leaf-descendants. Clearly $\sum_q \text{cost}(q) = \sum_q \phi(q) = \Phi$. We choose a hard query as follows:

1. Let q_{t+1} be a leaf such that $\text{cost}(q_{t+1}) < \frac{1}{8}\log\lambda_t(n)$.
2. For i from t down to 1: let q_i be the second most ancestral node in $D_{q_{i+1}}$.
3. The query edge is $e=(q_{t+1},q_1)$. We fix $w(e)$ to be more than $w(q_1)$, but less than any other weight more than $w(q_1)$.

In (1), such a q_{t+1} can always be found because the average leaf cost is bounded by $\log\lambda_t(n)/8$. For (2) we clearly require $|D_{q_{i+1}}| \ge 2$. For our analysis to go through we will actually require $D_{q_{i+1}}$ to have at least $2^{2(t+1)^2}$ elements. We first prove bounds on $|C_{q_i}|, |D_{q_i}|$ and $\text{size}(q_i)$. We then bound the cost of answering the query (q_{t+1}, q_1).

Lemma 3.2. *If q is a leaf then $|C_q| = \lambda_t(n)$. If q is a non-leaf $\bar{i}$-node, where $i > 1$, and there exists some i-node above q, then $|C_q| \ge 2^{2^{\text{size}(q)}-1}$.*

Proof. When q is a leaf the set C_q consists of all t-node ancestors of q, of which there are $\lambda_t(n)$, by the definition of λ_t. Now suppose that q is an $\bar{i}$-node, for $i > 1$, and let p be the next i-node ancestor of q. Let j, j_1, j_2 be such that $\text{size}(q) = A(i,j) = A(i-1, j_1)$ and $\text{size}(p) = A(i, j+1) = A(i-1, j_2)$. The set $|C_q|$ consists of the $j_2 - j_1 - 1$ $(i-1)$-nodes between q and p. If $j = 0$ then by the definition of A, $|C_q| = 2^{2^{A(i,0)}} - 2$. If $j > 0$ then $|C_q| = 2^{2^{A(i,j)}} - 2^{2^{A(i,j-1)}} - 1$. In either case $|C_q| \ge 2^{2^{\text{size}(q)}-1}$ since $A(i, j-1) < \frac{1}{2}A(i,j)$ when $i > 1$. $\blacksquare$

Lemma 3.3. *Let p be an arbitrary leaf descendant of an $\bar{i}$-node q, where $i > 1$. Then*

$$|D_q| \ge \frac{|C_q|}{2^{\text{cost}(p)\ \text{size}(q)}}\,.$$

Proof. By definition of ϕ, $|D_q| \ge |C_q|/2^{\phi(q)}$. By the definition of $\text{cost}(p)$ we have $\text{cost}(p) \ge \phi(q)/\text{size}(q)$. The lemma follows. $\blacksquare$

Lemma 3.4. *For $1 \le i \le t$, $\text{size}(q_i) \ge \lambda_t(n) \ge 2^{3(t+1)^2}$.*

Proof. Since q_i is an ancestor of q_{i+1}, implying $\text{size}(q_i) > \text{size}(q_{i+1})$, we need only prove the lemma for $i = t$. We selected q_{t+1} for satisfying $\phi(q_{t+1}) < \log\lambda_t(n)/8$. Therefore, it follows from Lemmas 3.2 and 3.3 that $|D_{q_{t+1}}| \ge (\lambda_t(n))^{7/8}$. We chose q_t to be the second most ancestral node in $D_{q_{t+1}}$. Therefore, $\text{size}(q_t) \ge A(t, (\lambda_t(n))^{7/8} - 2)$, which is at least $\lambda_t(n)$ since $A(t,j) \ge 2^j$ and $(\lambda_t(n))^{7/8} - 2 \ge \log\lambda_t(n)$ for all but small constant values of $\lambda_t(n)$. (Recall that we assumed $\lambda_t(n) \ge 2^{3(t+1)^2} \ge 2^{12}$.) $\blacksquare$

Lemma 3.5. *For $1 < i \le t+1$, $|D_{q_i}| \ge 2^{2(t+1)^2}$.*

Proof. It was already shown in the proof of Lemma 3.4 that $D_{q_{t+1}} \ge (\lambda_t(n))^{7/8} \ge 2^{2(t+1)^2}$. Consider q_i, for $i \le t$. We have the inqualities:

$$|D_{q_i}| \ge \frac{|C_{q_i}|}{2^{\text{size}(q_i)\ \text{cost}(q_{t+1})}} \ge 2^{\left(2^{\text{size}(q_i)}-1-\text{size}(q_i)\log\lambda_t(n)\right)} \ge \text{size}(q_i) \ge 2^{2(t+1)^2}.$$

The first inequality follows from Lemma 3.3. The second follows the inequality $\text{cost}(q_{t+1}) < \log \lambda_t(n)/8$ and Lemma 3.2. The third and fourth follow from Lemma 3.4. ∎

Lemma 3.6. *The query $e = (q_{t+1}, q_1)$ requires $t+1$ comparisons to answer in the worst case.*

Proof. Consider the unique cycle in $T \cup \{e\}$. We will show that among the weighted elements on this cycle (weighted vertices and the edge e), the set of possible maxima is precisely $\{e, q_{t+1}, q_t, \dots, q_1\}$. Therefore, confirming that e is indeed the heaviest element will require $t+1$ comparisons in the worst case since for any comparison made by the verification algorithm, at least one outcome eliminates at most one possible maximum. Suppose that before the query algorithm began it was already known that $w(q_i) \le w(q_j)$, ruling out q_i as a potential maximum. We consider two cases, depending on the ordering of i and j. Assume first that $j < i$, that is, q_j is an ancestor of q_i. Let p be the most ancestral element in D_{q_i}. By our selection of q_{i-1} from D_{q_i}, q_{i-1} lies strictly below p, which implies $q_{i-1}, q_{i-2}, \dots, q_j, \dots, q_1$ lie strictly below p – see Figure 2 for a diagram. This leads to a contradiction since $p \in D_{q_i}$ implies $w(p) = w(q_i)$ is consistent with any known inequalities. However we know that $w(q_i) \le w(q_j) < w(p)$, where the first inequality is by assumption and the second by Lemma 2.4. The other case, when $i < j$, is similar but not exactly symmetrical. Let p be the least ancestral element in D_{q_j}. By our choice of q_{j-1} and Lemma 3.5, we know that p lies strictly below q_{j-1}. If $w(q_j) = w(p)$,

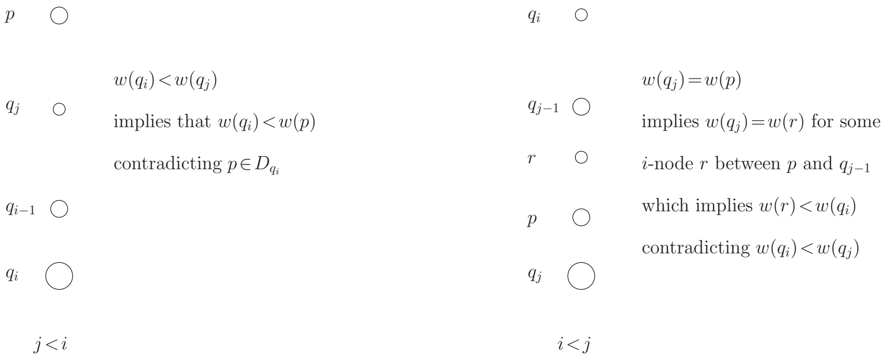
p
q_j
q_{i-1}
q_i
$w(q_i) < w(q_j)$
implies that $w(q_i) < w(p)$
contradicting $p \in D_{q_i}$
$j < i$
q_i
q_{j-1}
r
p
q_j
$w(q_j) = w(p)$
implies $w(q_j) = w(r)$ for some
i-node r between p and q_{j-1}
which implies $w(r) < w(q_i)$
contradicting $w(q_i) < w(q_j)$
$i < j$

Figure 2. Before answering the query the verification algorithm cannot know that $w(q_i) \le w(q_j)$. The two situations, $i < j$ and $j < i$, are diagrammed. Some of the nodes depicted may actually represent the same node, as in the case when $j = i-1$ (left) or $i = j-1$ (right).

which is possible since $p \in D_{q_j}$, then by Property 2.3 $w(q_j) = w(r)$ where r is some $\overline{i}$-node between p and q_{j-1}. By Lemma 2.4 $w(r) < w(q_i)$, giving us the inequalities $w(r) < w(q_i) \leq w(q_j) = w(r)$, a contradiction. ∎

In Lemmas 3.7 and 3.8 we show that if the query (q_{t+1}, q_1) is answered in fewer than $t+1$ comparisons, then Φ must increase by at least $2(t+1)^2$. Let Δ_Φ denote the change in Φ, measured before and after the query.

Lemma 3.7. *Either $\Delta_\Phi \geq 2(t+1)^2$ or $w(q_{t+1}) \leq w(q_t) \leq \cdots \leq w(q_1) < w(e)$.*

Proof. Let high(i) denote the event that $w(q_i) = w(p)$ where p is the most ancestral node in D_{q_i}. Recall that q_{i-1} was chosen to be the second most ancestral node in D_{q_i} and that $w(e) > w(q_1)$. Therefore, it must be the case that $w(q_{t+1}) \leq \cdots \leq w(q_1) < w(e)$ or high$(t+1) \vee$ high$(t) \vee \cdots \vee$ high(2). The event high(j) occurs iff $w(q_j) > w(q_1)$, which implies $w(q_j) > w(e)$. Since, by Lemma 3.6 the only possible maxima on the cycle in $T \cup \{e\}$ are $q_{t+1}, q_t, \ldots, q_1, e$, if e is not maximal then the verification algorithm must have discovered that $w(q_j) > w(e)$ for some j, i.e. it must have discovered that high(j) holds. According to Lemma 3.5 $|D(q_j)| \geq 2^{2(t+1)^2}$ before the query e is issued. However, after the query high(j) holds, which implies that $|D_{q_j}| = 1$. According to the definition of ϕ, $\phi(q_j)$ must increase by at least $2(t+1)^2$, implying Δ_Φ is at least that much. ∎

Lemma 3.8. *Either $\Delta_\Phi \geq 2(t+1)^2$ or each comparison made by the verification algorithm eliminates at most one candidate maximum from the set $\{q_{t+1}, q_t, \ldots, q_1, e\}$.*

Proof. Suppose that there exists a comparison that eliminates two or more candidate maxima from the unique cycle in $T \cup \{e\}$. By Lemma 3.6 the set of candidate maxima initially includes $\{q_{t+1}, q_t, \ldots, q_1, e\}$. Suppose that the comparison reveals the inequality $w(q_i) \geq w(x)$ and that it is already known that $w(x) \geq w(q_j), w(q_k)$, where q_j and q_k are candidate maxima at the time of the comparison. Let x be an $\overline{\ell}$-node. There are several cases, depending on the ordering of i, j, k, and ℓ. Lemma 3.7 lets us restrict our attention to the case when $w(q_{t+1}) \leq w(q_t) \leq \cdots \leq w(q_1) < w(e)$, which immediately implies $i < j < k$. The proof is structured as follows. Let p be the next $\overline{(k-1)}$-node above q_{k-1}, i.e., at the time the query e was issued, p was the most ancestral node in D_{q_k}. We will show that either $p \notin D_{q_k}$ just before the comparison, or the comparison causes Φ to increase by at least $2(t+1)^2$, which implies Δ_Φ is at least that much. The case $p \notin D_{q_k}$ leads to a contradiction because it implies, by the definition of D_{q_k}, that $w(q_k) \leq w(q_{k-1})$, meaning q_k was not a candidate maximum just before the comparison.

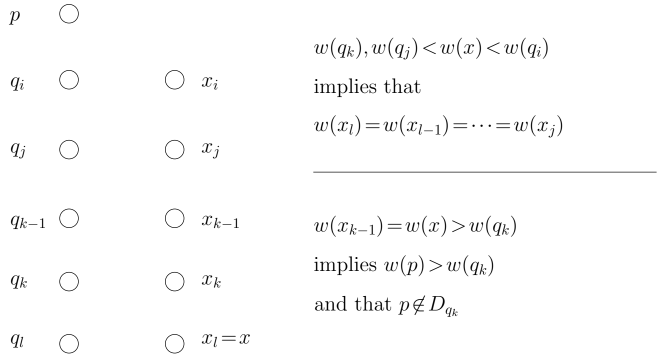
p
q_i
x_i
q_j
x_j
q_{k-1}
x_{k-1}
q_k
x_k
q_l
$x_l = x$
$w(q_k), w(q_j) < w(x) < w(q_i)$
implies that
$w(x_l) = w(x_{l-1}) = \cdots = w(x_j)$
$w(x_{k-1}) = w(x) > w(q_k)$
implies $w(p) > w(q_k)$
and that $p \notin D_{q_k}$

Figure 3. It was discovered that $w(q_j), w(q_k) \leq w(x) \leq w(q_i)$, ruling out q_j and q_k as potential maxima. The node $x = x_\ell$ is proved to be at the same level as q_ℓ, for some $\ell \geq k$. The nodes $x_{\ell-1}, \ldots, x_1$, are by definition the ancestors or x at the same levels as $q_{\ell-1}, \ldots, q_1$.

We will first show that $\ell \geq k$ and x is at the same level in T as q_ℓ. Suppose that $\ell < k$. If the inequalities $w(q_k) \leq w(x) \leq w(q_i)$ were a possibility before the comparison was made then x must lie in the band of T between the levels of q_k and p: any ℓ-nodes outside that band are, by Lemma 2.4, definitely lighter than q_k or heavier than q_i. However the inequality $w(x) \geq w(q_k)$ then implies, by Lemma 2.4, that $w(p) > w(q_k)$, that is, $p \notin D_{q_k}$, a contradiction. This shows that $\ell \geq k$. We now consider the height of x in T. We cannot have x lower in T than q_ℓ, for this would imply by Lemma 2.4 that $w(x) < w(q_\ell) \leq w(q_k)$. It also cannot lie above q_ℓ, as this implies $w(q_i) < w(x)$, again by Lemma 2.4. We define $x_\ell, x_{\ell-1}, \ldots, x_1$ to be the ancestors of x at the same levels as, respectively, $q_\ell, q_{\ell-1}, \ldots, q_1$, where $x = x_\ell$ – see Figure 3. Notice that if $w(q_j), w(q_k) \leq w(x) = w(x_\ell) \leq w(q_i)$, it must be the case that $w(x_\ell) = w(x_{\ell-1}) = \cdots = w(x_j)$, implying that $|D_{x_\ell}| = |D_{x_{\ell-1}}| = \cdots = |D_{x_{j+1}}| = 1$. If any one of these D-sets, say D_{x_r}, had strictly more than 1 element before the query e was issued then we argue that $\varDelta_\varPhi \geq 2(t+1)^2$. Since x_r is at the same level as q_r, it follows from Lemma 3.5 that $|C_{x_r}| \geq 2^{2(t+1)^2}$. Given this lower bound on $|C_{x_r}|$, the transition from $|D_{x_r}| > 1$ to $|D_{x_r}| = 1$ causes $\phi(x_r)$ to increase by at least $2(t+1)^2$, which causes $\varPhi$ to increase by at least that much. Therefore, if $|D_{x_r}| > 1$ before the query was issued then $\varDelta_\varPhi \geq 2(t+1)^2$. The last situation to consider is when $|D_{x_\ell}| = |D_{x_{\ell-1}}| = \cdots = |D_{x_{j+1}}| = 1$ *before* the query e was issued. In this situation the known inequality $w(x) = w(x_\ell) \geq w(q_k)$ implies $w(x_{k-1}) \geq w(q_k)$ since $j \leq k-1$. However Lemma 2.4 implies that $w(p) > w(x_{k-1})$, and consequently, that $p \notin D_{q_k}$ just before the comparison was made. This contradicts the claim that q_k was a candidate maximum. ∎

Lemma 3.8 implies that the query e is answered in $t+1$ comparisons or else increases Φ by at least $2(t+1)^2$. By invoking Theorem 2.6 with $\Psi = \Phi/2$, $\epsilon = 1/t$, $\nu = t+1$, $\mu = (t+1)^2$, the amortized cost of the query e is at least t. In the next section we prove that Lemma 3.1 implies Theorem 1.1.

3.1. Proof of Main Theorem

In this section we define a function $\tau(m,n)$ and prove a lower bound on the online MST verification problem in terms of m, n, and $\tau(m,n)$. We then show that $\tau(m,n)$ is within an absolute constant of the inverse-Ackemann function α as defined by Tarjan [33]. Define $\tau(m,n)$ as:

$$\tau(m,n) \;=\; \max\left\{t \;:\; \lambda_t(n) \;\geq\; 2^{16t^2\lceil m/n\rceil}\right\}.$$

It is true that $\tau(m,n)$ is undefined when $\lambda_1(n) < 2^{16\lceil m/n\rceil}$. For the sake of completeness, assume $\tau(m,n) = 1$ in this situation.

Lemma 3.9. *Any deterministic algorithm answering m MST verification queries on an n-node tree can make at least* $\max\{m\cdot\tau(m,n), n\}$ *comparisons. If the algorithm is randomized this bound holds with probability* $1-2^{-\Omega(m)}$.

Proof. If $\tau(m,n)=1$ or $m\cdot\tau(m,n)<n$ then the lemma is trivial. Any MST verification algorithm must perform 1 comparison per query and in the case where $T\cup\{e\}$ consists of a single cycle, confirming that $e\notin MST(T\cup\{e\})$ requires n comparisons. We assume that $\tau(m,n)>1$. By the definition of $\tau(m,n)$ it follows that $\frac{1}{16}n\log\lambda_t(n)\geq 2\cdot m\cdot\tau(m,n)$. If the MST verification algorithm made fewer than $m\cdot\tau(m,n)$ comparisons it follows from Lemma 2.5 that $\Phi<\frac{1}{8}n\log\lambda_t(n)$ with probability at least $1-2^{-m\cdot\tau(m,n)}$. It also follows from the definition of $\tau(m,n)$ that $\lambda_t(n)\geq 2^{3(t+1)^2}$. Therefore, the preconditions of Lemma 3.1 are met with high probability. According to Lemma 3.1 the amortized cost of each query is $\mathrm{amort}(t+1,(t+1)^2,t^{-1})=t$, and by Theorem 2.6 the probability that the amortized costs exceed the actual costs is no more than $2^{-\epsilon tm}=2^{-m}$. If the algorithm is deterministic then all high probability bounds become probability 1 on some worst case input. ∎

In [33] Tarjan defined a variant of Ackermann's function and a certain inverse which we denote by B and α, respectively. They are defined as:

$$\begin{array}{ll} B(0,j) = 2j & \text{for } j \geq 0, \\ B(i,0) = 0 & \text{for } i \geq 1, \\ B(i,1) = 2 & \text{for } i \geq 1, \\ B(i,j) = B(i-1, B(i,j-1)) & \text{for } i \geq 1, j \geq 2, \end{array}$$

$$\alpha(m,n) = \min\left\{ i : B\left(i, 4\left\lceil \frac{m}{n} \right\rceil\right) > \log n \right\}.$$

Lemma 3.10. $\alpha(m,n) - 3 \leq \tau(m,n) \leq \alpha(m,n) + 1$.

Proof. We will use several properties of A and B, given in Lines (1) through (4). They are all simple to prove by induction. The proofs, which are somewhat tedious, are left to the reader.

$$A(i,j) \geq B(i,j) \quad \text{for } i \geq 1 \text{ and } j \geq 0 \tag{1}$$

$$B(i,3j) \geq A(i,j) \quad \text{for } i \geq 1, j \geq 1 \tag{2}$$

$$B(i+1,j) \geq 2^{B(i,j)} \quad \text{for } i \geq 2, j \geq 3 \tag{3}$$

$$B(i+1,j) \geq 2^{B(i,j)/2} \quad \text{for } i \geq 1, j \geq 3 \tag{4}$$

We had defined $\tau(m,n) = \max\{t : \lambda_t(n) \geq 2^{16t^2\lceil m/n \rceil}\}$. One can see that this is equivilent to the definition:

$$\tau(m,n) = \min\left\{ t \geq 1 \ : \ A\left(t+1,\ 2^{16(t+1)^2\lceil m/n \rceil}\right) > n \right\},$$

which is more in line with the definition of α. We will first show that $\alpha(m,n) \leq \tau(m,n) + 3$. Consider the inequalities below, where $t = \tau(m,n)$.

$$A\left(t+1, 2^{16(t+1)^2\lceil m/n \rceil}\right) > n \tag{5}$$

$$B\left(t+1, 2^{17(t+1)^2\lceil m/n \rceil}\right) > \log n \tag{6}$$

$$B\left(t+3, 4\left\lceil \frac{m}{n} \right\rceil\right) > \log n \tag{7}$$

Line (5) follows from the definition of $\tau(m,n)$. Line (6) follows from Line (2). Line (7) follows from Line (3), though not directly. Let $x \uparrow y$ denote an exponential stack of x 2's with a y on top; e.g. $2 \uparrow 3 = 2^{2^3} = 256$. One can easily show that $B(1,j) = 2^j = 1 \uparrow j$. Lines (3) and (4) together imply that $B(i,j) \geq i \uparrow (j-1)$ for all $i \geq 1, j \geq 3$. Therefore, $B(t+3, 4\lceil \frac{m}{n} \rceil) = B(t+2, B(t+3, 4\lceil \frac{m}{n} \rceil - 1)) \geq B(t+2, (t+3) \uparrow (4\lceil \frac{m}{n} \rceil - 2))$, which is greater than $B(t+2, 2^{17(t+1)^2\lceil m/n \rceil})$ since $(t+2) \uparrow (4\lceil \frac{m}{n} \rceil - 2)$ always dominates $17(t+1)^2\lceil \frac{m}{n} \rceil$.

Therefore Line (7) follows from Line (6), and implies, by the definition of $\alpha(m,n)$, that $\alpha(m,n)\le t+3=\tau(m,n)+3$.

The bound $\tau(m,n)\le\alpha(m,n)+1$ is proved in a similar fashion. Consider the following inequalities, where α is short for $\alpha(m,n)$.

$$B\left(\alpha, 4\left\lceil \frac{m}{n}\right\rceil\right) > \log n \tag{8}$$

$$B\left(\alpha, 2^{8\lceil m/n\rceil}\right) > n \tag{9}$$

$$A\left(\alpha+1, 2^{16\alpha^2\lceil m/n\rceil}\right) > n \tag{10}$$

Line (8) is the definition of $\alpha(m,n)$. Line (9) follows from the inequality $B(i,j)\ge 2j$. Line (10) follows from Line (1) and the monotonicity of A and B. Line (10) is not given as $A(\alpha,\ldots)>n$ because α may be zero while $A(i,j)$ is only defined for $i\ge 1$. Therefore, $\tau(m,n)\le\max\{\alpha,1\}\le\alpha+1$. ∎

4. Upper Bounds

In this section we show that the actual complexity of online minimum spanning tree verification is not too far from the lower bound presented in Section 3. We also provide nearly tight bounds on the average case complexity of the problem, where the average is over all permutations of the tree weights. We will think of the MST verification algorithm as divided into two parts: a *preprocessing algorithm* that, given T, produces some fixed data structure, and a *query algorithm* that answers MST verification queries by referring only to the fixed structure. The complexities of interest are the number of preprocessing comparisons and the worst case number of comparisons per query.

Theorem 4.1. *Suppose the tree-weights are permuted randomly. With no more than $2n$ preprocessing comparisons (expected), MST verification queries can be answered with no more than 2 comparisons. If queries must be answered with 1 comparison then the expected preprocessing time is $\Theta(n\log n)$.*

Proof. First consider the 1-comparison case with randomly permuted edge weights. Let T be a star with center vertex c. For every two vertices $u,v\neq c$ the preprocessing algorithm must have determined the relative order of $w(u,c)$ and $w(v,c)$; otherwise it could not answer the query (u,v) in one comparison. The preprocessing algorithm sorts $(n-1)$ numbers and must therefore perform $\Omega(n\log n)$ comparisons. $O(n\log n)$ preprocessing time is also clearly sufficient for any tree T. (Remark: If all tree nodes have degree bounded by a constant, it seems likely that $o(n\log n)$ preprocessing would be required on average.)

For the 2-comparison case the preprocessing algorithm must reduce the number of candidate maxima on any query to at most 2 (not counting the query edge). We root the tree arbitrarily and divide any query (u,v) into two queries $(u, LCA(u,v))$ and $(v, LCA(u,v))$, where $LCA(u,v)$ is the least common ancestor of u and v. Therefore it will be sufficient to reduce the number of candidate maxima on a query (z,a) to one, where a is an ancestor of z. For any node $z \in T$ let $z = z_0, z_1, z_2, z_3, \ldots$ be the sequence of nodes from z up to the root. We must find the prefix-maxima of the sequences $\{(w(z_i))\}_{z\in T, i\geq 0}$, where $w(z_i)$ is the weight of the edge $(z_i, \text{parent}(z_i))$. This is tantamount to finding the subsequence $L(z) = (z_{i_1}, z_{i_2}, z_{i_3}, \ldots)$ where z_{i_p} has maximum weight among $(z_0, \ldots, z_{i_{p+1}-1})$. One can see that $L(z)$ is derived from $L(z_1) = (z_{j_1}, z_{j_2}, z_{j_3}, \ldots)$ by substituting a (possibly empty) prefix of $L(z_1)$ with $z = z_0$. After determining $L(z_1)$ we compute such a prefix in the obvious manner, by comparing $w(z_0)$ with $w(z_{j_1}), w(z_{j_2}), \ldots$ until j_q is found such that $w(z_0) < w(z_{j_q})$. (If there is no such z_{j_q} then for the sake of consistent notation we let it be the non-existent parent of the root.) The cost of this procedure, which is performed for every node in the input tree, is no more than q.[6] We analyze the behavior of q and j_q under the assumption that the tree edge-weights are randomly permuted. We have

$$\Pr[j_q = r] \leq 1/(r(r+1)), \tag{11}$$

$$\begin{aligned} \mathbb{E}[q \mid j_q = r] &\leq 1 + \sum_{i=1}^{r-1} \Pr\left[w(z_i) = \max_{1\leq k\leq i}\{w(z_k)\}\right] \\ &= 1 + H_{r-1}, \end{aligned} \tag{12}$$

$$\begin{aligned} \mathbb{E}[q] &= \sum_{r=1}^{\infty} \Pr[j_q = r] \cdot \mathbb{E}[q \mid j_q = r] \\ &\leq 1 + \sum_{r=2}^{\infty} \frac{H_{r-1}}{r(r+1)} \\ &= 1 + \sum_{i=1}^{\infty} \left(\frac{1}{i} \cdot \sum_{r=i+1}^{\infty} \frac{1}{r(r+1)} \right) \\ &= 1 + \sum_{i=1}^{\infty} \frac{1}{i(i+1)} = 2. \end{aligned} \tag{13}$$

[6] It is usually equal to q, unless z_{j_q} happens to be the parent of the root, in which case the comparison $w(e_0) < w(e_{j_q})$ never actually takes place.

Lines (11) and (12) are inequalities, rather than equalities, due to the finiteness of the $(z_i)_i$ sequence. Line (13) follows from Lines (11) and (12) and the identity $\sum_{i=1}^{k} 1/i(i+1) = 1 - 1/(k+1)$, which is easily proved by induction on k. ∎

Any (online) tree-sum algorithm for arbitrary semigroups can be used as an (online) MST verification algorithm. The constructions from [4,8] show that a tree T can be preprocessed in $O(n\lambda_t(n))$ time so that the sum of the weights on any path is computable with $2t-1$ semigroup operations. We sketch below how the preprocessing time can be reduced to $O(n \log \lambda_t(n))$ for the online MST verification problem, without affecting the query time. The [4,8] preprocessing algorithms implicitly generate a set of forests of rooted trees $F_1, F_2, \ldots, F_t$ with the property that the sum of the weights of any path in T is equal to the sum of the weights of $2t$ paths: 2 paths in each of $T_1 \in F_1, T_2 \in F_2, \ldots, T_t \in F_t$, where $T_1, \ldots, T_t$ are trees in their respective forests. The paths in $T_1, \ldots, T_t$ run from a leaf to one of its ancestors. Preprocessing the trees in $F_1, \ldots, F_t$ to answer leaf-to-ancestor queries is done in the obvious fashion: for a tree with size s and height h the preprocessing time is $O(sh)$. $F_1, \ldots, F_{t-1}$ are constructed so that their preprocessing time is $O(n)$, and F_t is guaranteed to be a single tree with size at most n and height at most $\lambda_t(n)$; therefore the total preprocessing time is $O(n\lambda_t(n))$. This algorithm can be improved, slightly, when the semigroup is $(\mathbb{R}, \max)$ and $t > 1$. Komlós's MST verification algorithm [26] can be thought of as an $O(s \log h)$-time preprocessing scheme for answering leaf-to-ancestor queries. Using it to preprocess F_t instead of the [4,8] algorithms yields an $O(n \log \lambda_t(n))$ preprocessing algorithm with query complexity $2t$ (not $2t-1$, as in [4,8], because the query edge must be compared against the tree-weights too!). We can reduce the query complexity to $2t-1$ by preprocessing the the trees in F_1 more effectively. Any tree $T_1 \in F_1$ with size s and height h has the property that $s \log s = O(sh)$. Therefore, we shall preprocess the F_1 trees by sorting their weights rather than use the [4,8] algorithm. The preprocessing time is unaffected but the query algorithm may reduce the number of comparisons by 1.

The above construction assumed $t > 1$, specifically that $F_1 \neq F_t$. What is the best preprocessing time we can achieve for query complexity 1, 2 and $2t$? For 1-comparison queries the optimal preprocessing time is clearly $\Theta(n \log n)$. For 2-comparison queries there is a simple algorithm with $O(n \log\log n)$ preprocessing time. King [23] showed that any tree T could be easily transformed into a new tree T' with several nice properties. First, any MST verification query on T can be mapped to an equivalent query in T'. Second, T' has size linear in $|T|$ and height logarithmic in $|T|$. Applying

Komlós's $O(n \log\log n)$-time preprocessing algorithm to T' lets us answer queries in 2 comparisons. It can be shown that this bound is optimal for 2-comparison queries, using the input distribution $\mathit{Distr}(1)$ and an argument similar to that of Lemma 3.1. If the query complexity is fixed at $2t$, where $t \geq 2$, we do not know of any faster preprocessing algorithm than the one for query complexity $2t-1$. Our results on the worst case complexity of online MST verification are summarized in Theorem 4.2.

Theorem 4.2. *For query complexity* 1 *the optimal online MST verification algorithm preprocesses the tree in* $\Theta(n \log n)$ *time. For query complexity* 2 *the optimal preprocessing time is* $\Theta(n \log\log n)$*. For query complexity* $2t-1 \geq 3$ *the optimal preprocessing time is* $O(n \log \lambda_t(n))$ *and* $\Omega(n \log \lambda_{2t-1}(n))$*.*

For query complexity $2t-1$ we suspect that the upper bound in Theorem 4.2 is tight. However for the special case of leaf-to-ancestor queries the lower bound from Section 3 is tight. Therefore, to improve our lower bounds one must necessarily consider more general types of queries.

5. Open Problems

Among natural comparison-based problems there are relatively few whose asymptotic complexities remain unresolved. The minimum spanning tree problem [22,9,32,31] is certainly the oldest such problem, though there are many others that seem equally difficult. In the *local sorting* problem [18] one is given a vertex-weighted undirected graph and asked to orient all the edges to point to the heavier endpoint. When generalized to hypergraphs the problem is known as *set maxima*. There are elegant, optimal algorithms for both local sorting and set maxima [18,31]; however they are all randomized. Surprisingly there are no known non-trivial deterministic solutions.

Besides online MST verification we are aware of no comparison-based problems for which inverse-Ackermann type lower bounds are very likely. However there are a couple such problems for which super-linear lower bounds seem at least *plausible*:

- The **split-findmin** problem is a data structuring problem. We are to maintain a set of disjoint sequences of n weighted elements under two types of operations. A *decrease-key* operation reduces the weight of some element, and a *split* operation divides a sequence into two disjoint sequences: one consisting of elements to the left of some designated element, and another consisting of the rest. The problem is to maintain, at all times, the minimum-weight element in each sequence. This peculiar data structure turns out to be useful in certain weighted matching algorithms [16] and several recent shortest path algorithms [38,19,30,27, 29]. It can also be used to solve the minimum spanning tree sensitivity analysis problem (see [37] for the definition of MST sensitivity analysis.) The fastest data structure to date [30,28] runs in $O(m\log\alpha(m,n))$ time, where m is the number of operations. It is a slight improvement over Gabow's data structure [16], which runs in $O(m\alpha(m,n))$ time.

- The problem of finding row-maxima in totally monotone matrices has many applications [3]. For complete $m\times n$ matrices the problem can be solved in $O(m+n)$ time [3]. However for *partial* matrices (where some elements may be blank) the complexity of the problem depends on the shape of the non-blank elements. Klawe [24] proved super-linear lower bounds of $\Omega(n\alpha(n))$ on the time to find row-maxima in an $n\times n$ matrix where the non-blank elements are either contiguous in each column (v-matrices) or in each row (h-matrices). For the more restrictive classes of $m{\times}n$ "staircase" or "skyline" matrices, there are known upper bounds of $O(m\alpha(m,n)+n)$ [25,24]. It is conceivable that these super-linear upper bounds are tight.

Acknowledgment. I thank Vijaya Ramachandran for many helpful comments, and Stephen Alstrup, Theis Rauhe, and Uri Zwick for reintroducing me to this problem.

References

[1] W. Ackermann: Zum Hilbertshen Aufbau der reelen Zahlen, *Math. Ann.* **99** (1928), 118–133.

[2] P. K. Agarwal and J. Erikson: Geometric range searching and its relatives, *Advances in Discrete and Computational Geometry* (B. Chazelle, J. E. Goodman, R. Pollack, eds.); *Contemporary Mathematics* **223** (1998), 1–56.

[3] A. Aggarwal, M. Klawe, S. Moran, P. Shor and R. Wilber: Geometric applications of a matrix-searching algorithm, *Algorithmica* **2** (1987), 195–208.

[4] A. Alon and B. Schieber: Optimal preprocessing for answering on-line product queries, Technical Report TR-71/87, Institute of Computer Science, Tel Aviv University, 1987.

[5] C. F. Bazlamaçci and K. S. Hindi: Verifying minimum spanning trees in linear time, in *Proc. Symp. on Operations Research (SOR)*, pages 139–144, 1997.

[6] M. A. Bender and M. Farach-Colton: The LCA problem revisited, in *Proceedings 4th Latin American Symp. on Theoretical Informatics (LATIN), LNCS Vol. 1776*, pages 88–94, 2000.

[7] A. L. Buchsbaum, H. Kaplan, A. Rogers and J. R. Westbrook: Linear-time pointer-machine algorithms for LCAs, MST verification, and dominators; in *Proc. 30th ACM Symposium on Theory of Computing (STOC'98)*, pages 279–288, May 23–26 1998.

[8] B. Chazelle: Computing on a free tree via complexity-preserving mappings, *Algorithmica* **2(3)** (1987), 337–361.

[9] B. Chazelle: A minimum spanning tree algorithm with inverse-Ackermann type complexity, *J. ACM* **47(6)** (2000), 1028–1047.

[10] B. Chazelle: The soft heap: an approximate priority queue with optimal error rate, *J. ACM* **47(6)** (2000), 1012–1027.

[11] B. Chazelle and B. Rosenberg: The complexity of computing partial sums off-line, *Internat. J. Comput. Geom. Appl.* **1(1)** (1991), 33–45.

[12] T. H. Cormen, C. E. Leiserson and R. L. Rivest: *Introduction to Algorithms*, MIT Press, Cambridge, Mass., 1990.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein: *Introduction to Algorithms*, MIT Press, 2001.

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

[15] M. L. Fredman and M. Saks: The cell probe complexity of dynamic data structures, in *Proc. 21st annual ACM Symposium on Theory of Computing*, pages 345–354, 1989.

[16] H. N. Gabow: A scaling algorithm for weighted matching on general graphs, in *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 90–100, 1985.

[17] H. N. Gabow, J. L. Bentley and R. E. Tarjan: Scaling and related techniques for geometry problems, in *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 135–143, 1984.

[18] W. Goddard, C. Kenyon, V. King and L. Schulman: Optimal randomized algorithms for local sorting and set-maxima, *SIAM J. Comput.* **22(2)** (1993), 272–283.

[19] T. Hagerup: Improved shortest paths on the word RAM, in *Proc. 27th Int'l Colloq. on Automata, Languages, and Programming (ICALP), LNCS vol. 1853*, pages 61–72, 2000.

[20] D. Harel and R. E. Tarjan: Fast algorithms for finding nearest common ancestors, *SIAM J. Comput.* **13(2)** (1984), 338–355.

[21] S. Hart and M. Sharir: Nonlinearity of Davenport–Schinzel sequences and of generalized path compression schemes, *Combinatorica* **6(2)** (1986), 151–177.

[22] D. R. Karger, P. N. Klein and R. E. Tarjan: A randomized linear-time algorithm for finding minimum spanning trees, *J. ACM* **42** (1995), 321–329.

[23] V. King: A simpler minimum spanning tree verification algorithm, *Algorithmica* **18(2)** (1997), 263–270.

[24] M. M. Klawe: Superlinear bounds on matrix searching, in *Proc. 1st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'90)*, pages 485–493, 1990.

[25] M. M. Klawe and D. J. Kleitman: An almost linear time algorithm for generalized matrix searching, *SIAM J. Discr. Math.* **3(1)** (1990), 81–97.

[26] J. Komlós: Linear verification for spanning trees, *Combinatorica* **5(1)** (1985), 57–65.

[27] S. Pettie: On the comparison-addition complexity of all-pairs shortest paths, in *Proc 13th Int'l Symp. on Algorithms and Computation (ISAAC'02)*, pages 32–43, 2002.

[28] S. Pettie: On the shortest path and minimum spanning tree problems (Ph.D. Thesis), Technical Report TR-03-35, The University of Texas at Austin, Department of Computer Sciences, August 2003.

[29] S. Pettie: A new approach to all-pairs shortest paths on real-weighted graphs, *Theor. Comp. Sci.* **312(1)** (2004), 47–74. Special issue of selected papers from 29th Int'l Colloq. on Automata Languages and Programming (ICALP 2002).

[30] S. Pettie and V. Ramachandran: A shortest path algorithm for real-weighted undirected graphs, *SIAM J. Comput.*, to appear.

[31] S. Pettie and V. Ramachandran: Minimizing randomness in minimum spanning tree, parallel connectivity and set maxima algorithms, in *Proc. 13th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 713–722, 2002.

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

[33] R. E. Tarjan: Efficiency of a good but not linear set merging algorithm, *J. ACM* **22** (1975), 215–225.

[34] R. E. Tarjan: Complexity of monotone networks for computing conjunctions, *Ann. Discrete Math.* **2** (1978), 121–133.

[35] R. E. Tarjan: Applications of path compression on balanced trees, *J. ACM* **26(4)** (1979), 690–715.

[36] R. E. Tarjan: A class of algorithms which require nonlinear time to maintain disjoint sets, *J. Comput. Syst. Sci.* **18(2)** (1979), 110–127.

[37] R. E. Tarjan: Sensitivity analysis of minimum spanning trees and shortest path problems (see also corrigendum *IPL* **23**(4), p. 219); *Info. Proc. Lett.* **14(1)** (1982), 30–33.

[38] M. Thorup: Undirected single-source shortest paths with positive integer weights in linear time, *J. ACM* **46(3)** (1999), 362–394.

[39] A. C. Yao: Space-time tradeoff for answering range queries, in *Proc. 14th ACM Symposium on Theory of Computing (STOC'82)*, pages 128–136, 1982.

Seth Pettie

Max Planck Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany
pettie@mpi-sb.mpg.de