An Inverse-Ackermann Style Lower Bound for the Online Minimum Spanning Tree Verification Problem

Seth Pettie*
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712

Abstract

We consider the problem of preprocessing an edge-weighted tree T in order to quickly answer queries of the following type: does a given edge e belong in the minimum spanning tree of $T \cup \{e\}$? Whereas the offline minimum spanning tree verification problem admits a lovely linear time solution, we demonstrate an inherent inverse-Ackermann type tradeoff in the online MST verification problem. In particular, any scheme that answers queries in t comparisons must invest $\Omega(n \log \lambda_t(n))$ time preprocessing the tree, where λ_t is the inverse of the t^{th} row of Ackermann's function. This implies a query lower bound of $\Omega(\alpha(n))$ for the case of linear preprocessing time. We also show that our lower bound is tight to within a factor of 2 in the t parameter.

1 Introduction

The theoretically best minimum spanning tree algorithms [23, 10, 33] were made possible by even more fundamental algorithms and data structures, namely Komlós's minimum spanning tree verification algorithm [27, 17, 24, 5] and Chazelle's Soft Heap [11]. It has been speculated by some (see, e.g., Chazelle [10, p. 1029]) that the key to a faster MST algorithm is some interesting new data structure. In this paper we show that there are no linear solutions to the *online* minimum spanning tree verification problem, ruling out this type of data structure in a faster MST algorithm. In particular, we show that a preprocessing time of $\Omega(n \log \lambda_t(n))$ is necessary in order to answer queries with t comparisons, where n is the size of the tree and λ_t is the t^{th}-row inverse of Ackermann's function.

Inverse-Ackermann type lower bounds are not too common (see [35, 22, 13] for some fundamental ones) and in the domain of purely comparison-based problems they were, to our knowledge, previously non-existent. The closest related result is Klawe's [25] $\Omega(n\alpha(n))$ lower bound on the time to find row-maxima in a totally monotone $2n \times n$ matrix, where the non-blank elements are contiguous in each column. However, in [25] the relevant operation is not the *comparison* but the *matrix query*.

Although the MST verification problem is nominally about minimum spanning trees, its closest cousins in the literature fall within a well-studied family of problems concerned with computing partial sums. In these problems there is an underlying set of weighted elements, where the weights are drawn from some (commutative) semigroup $(S, \circ)$. The problem is to answer a set of queries, where a query asks for the cumulative weight of some subset of the underlying elements. The case where elements are points in $\mathbb{R}^d$ has been studied extensively under various types of queries. (There are too many papers to cite; see [18, 38, 39, 7, 8, 4, 14, 9] for lower bounds and more references.) Chazelle & Rosenberg [12, 13] studied the case where the elements are packed into a d-dimensional array and queries take the form of d-rectangles (see also [38, 2] for $d = 1$.) In [13] a tight lower bound of $\Omega(n + m\alpha(m, n))$ semigroup operations is proved for the 1-dimensional *offline* version of the problem, where n is the size of the array and m the number of queries. This lower bound obviously extends to the online problem, and it relates to the MST verification problem because a 1-dimensional array is just a kind of tree. For general trees, Tarjan [34, 36] studied certain offline partial-sums algorithms based on path-compression. Online variants were studied in [6, 2].

The lower bounds cited above assume that semigroup elements are only accessible via the semigroup operator $\circ$. A consequence of this — which is key to previous lower bounds — is that any algorithm solving such a problem can be written as 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 algorithm chooses

[*] Email: seth@cs.utexas.edu. This work was supported by Texas Advanced Research Program Grant 003658-0029-1999, NSF Grant CCR-9988160, and an MCD Graduate Fellowship.

which comparisons to make based on the outcomes of previous comparisons. Naturally, many bounds that hold for arbitrary semigroups do not hold for $(\mathbb{R}, \max)$. For instance, the problem of answering interval-maximum queries in a 1-dimensional array can be done in constant time with linear preprocessing [27] (contrast this with the superlinear lower bound in [13] for arbitrary semigroups). Solving MST verification *offline* on arbitrary trees can be done in linear time [27, 17, 24, 5], and the dual to this problem, MST sensitivity analysis, can be solved in randomized linear time [20, 17] or deterministic $O(m \log \alpha(m, n))$ time.[1] All these problems have $\Omega(m\alpha(m, n))$ lower bounds when generalized to arbitrary semigroups [13]. Given this history it is somewhat startling that the problem we consider, online MST verification, does not admit a linear solution in the decision tree model.

Inverse-Ackermann type lower bounds are generally proved by appealing purely to the structure of certain fixed combinatorial objects. Contrast this with most lower bounds on decision tree complexity, which are information-theoretic in nature. The challenge in lower bounding the online MST verification problem is in combining these two very different approaches. We suspect that our techniques might yield inverse-Ackermann type lower bounds in other comparison-based problems; two candidate problems are given in Section 5.

1.1 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 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 preprocess an edge-weighted tree T so that given any *query edge* e, we can determine if $e \in MST(T \cup \{e\})$. This is tantamount to deciding whether e is not the heaviest edge on the only cycle in $T \cup \{e\}$. For the sake of simpler notation we consider input trees that are *vertex*-weighted rather than edge weighted. (A query then decides if e is heavier than all *vertices* in the unique cycle of $T \cup \{e\}$.) To further simplify matters we restrict the types of inputs and queries, as described below. Assertions 2.2 and 2.3, given in Section 2.3, provide further restrictions on the input.

1. The input tree T is a full, rooted binary tree.
2. The query edge will connect a leaf to one of its ancestors.
3. The answer to the query e will be *no*, $e \notin MST(T \cup \{e\})$. Therefore, the query algorithm need only verify this fact.

In a query it is clear that the query edge must participate in at least one comparison; the parameter $t \geq 0$ used throughout the paper represents the desired number of *additional* comparisons per query. The terms "query complexity" and "preprocessing complexity" refer to the number of comparisons performed by the query and preprocessing algorithms, respectively.

2.1 A Basic Lemma

We characterize the limits of the preprocessing algorithm later. It is important first to characterize the behavior of the optimal query algorithm. Regardless of what the preprocessing algorithm does, for any query some subset S of the vertices on the query path are candidate maxima. The *natural query algorithm* determines the actual maximum with $|S| - 1$ comparisons in the obvious manner, then compares this maximum with the weight of the query edge.

Lemma 2.1 *The natural query algorithm is optimal.*

Proof: Comparing two candidates, or a candidate with the query edge, can eliminate only one candidate from consideration. Now consider a comparison $w_a : w_b$ involving two weights, one of which, say w_a, is not a candidate. If w_a is known to be larger (smaller) than a candidate maximum, then the case $w_a > w_b$ ($w_a < w_b$) eliminates no candidates. In all other cases the comparison can go either way without eliminating candidates. □

It is conceivable that the natural query algorithm could be improved under some measure besides worst-case performance.

2.2 Ackermann's Function

In the field of algorithms & complexity, Ackermann's function [1] is rarely defined the same way twice (see e.g., [1, 35, 13, 15, 16]). We would not presume to buck such a well-established precedent. Here is a slight variant:

[1] The *split-findmin* data structure [19, 31] was known to be useful in certain weighted matching [19] and shortest path algorithms [37, 21, 31, 29, 30]. One application of *split-findmin* not mentioned in [19, 31] is MST sensitivity analysis, which it solves in $O(m \log \alpha(m, n))$ time, an $\Omega(\alpha/\log \alpha)$ factor faster than Tarjan's path-compression-based algorithm [36].

$$\begin{aligned} A(0,j) &= 2^j \\ A(i+1,0) &= A(i,1) \\ A(i+1,j+1) &= A(i,2^{2^{A(i+1,j)}}) \end{aligned}$$

Let $\alpha(m,n)$ be the inverse-Ackermann function and $\lambda_i(n)$ be the i^{th}-row inverse, defined as follows:

$$\begin{aligned} \alpha(m,n) &= \min\{i \,:\, A(i,\lceil \tfrac{m}{n} \rceil) \geq n\} \\ \lambda_i(n) &= \min\{j \,:\, A(i,j) \geq n\} \end{aligned}$$

and let $\alpha(n)$ be short for $\alpha(n,n)$. An equivalent definition of α, which is frequently more intuitive, can be had without direct appeal to Ackermann's function; see [26].

2.3 The Input Distribution

If a is a tree node we let $w(a)$ be the weight of a, and *size*(a) be the number of leaf-descendants of a. We call a an i-node if *size*$(a) = A(i,j)$ for $i \leq t$ and some j and an $\overline{i}$-node if it is an i-node but not an $(i+1)$-node.

If a is an $\overline{i}$-node, $i > 0$, we let C_a be the sequence of $\overline{(i-1)}$-nodes between a and its nearest i-node ancestor. If a is a leaf then C_a is the sequence of $\overline{t}$-nodes between a and the root; see Figure 1.

Assertion 2.2 *Let a be a non-leaf $\overline{i}$-node and b be a $\overline{k}$-node ancestor of a, where $k \geq i$. Then $w(a) < w(b)$ is known before the preprocessing algorithm begins.*

Assertion 2.2 establishes some a priori knowledge about the input. We are purposefully giving the preprocessing algorithm information so that we may succinctly characterize what it "knows" later on.

Assertion 2.2 implies that the sequence of $\overline{i}$-node ancestors of any node is monotonically increasing; see Figure 1. It also follows from Assertion 2.2 that for any query there are at most $t+2$ candidate maxima: the most ancestral $\overline{i}$-node in the query path, for $0 \leq i \leq t$, and the leaf involved in the query. Note that leaves were specifically excluded from Assertion 2.2.

Assertion 2.3 *For all leaves a, or $\overline{i}$-nodes a, $i > 0$, $w(a) = w(X_a)$ where X_a is a node selected uniformly at random from C_a, independent of all X_b, where $b \in T$, $b \neq a$.*

One can see by Assertion 2.3 that the weight of every $\overline{i}$-node, $i > 0$, is set (randomly) to equal that of some ancestral $\overline{0}$-node. The exact weights of the $\overline{0}$-nodes are not particularly important, so long as they accord with Assertion 2.2.

First, we show that Assertions 2.2 & 2.3 are mutually consistent and realizable. The danger in Assertion 2.3 is that it is, perhaps, impossible to satisfy the independence condition.

Lemma 2.4 *There is a weight-distribution consistent with Assertions 2.2 & 2.3.*

Proof: Assertion 2.3 defines the independent variables $\{X_a\}_a$. We only need to show no inconsistencies arise when combined with Assertion 2.2.

Consider an auxiliary graph with the same vertex set as the input tree, whose edges represent known (in)equalities. Assertion 2.2 places a number of uni-directional edges in this graph, and Assertion 2.3 places (randomly) some *bi*-directional edges in the graph, representing equalities. An inconsistency in Assertions 2.2 and 2.3 manifests itself in the auxiliary graph as a cycle, at least one edge of which is uni-directional. We show that no such cycles can exist. Assuming the contrary, let $\mathcal{C}$ be such a cycle with minimum length and b be a vertex in $\mathcal{C}$ such that no proper descendant of b is in $\mathcal{C}$. All uni-directional edges implied by Assertion 2.2 go from nodes to their ancestors. Furthermore, by Assertion 2.3 there is exactly one bi-directional edge connecting b to an ancestor. Therefore, if (a,b) and (b,c) are the edges incident on b in $\mathcal{C}$, exactly one of them, say (b,c), must be bi-directional — see Figure 2. Suppose b is an $\overline{i}$-node and let d be the closest i-node ancestor of b. By Assertion 2.3, c is a $\overline{(i-1)}$-node descendant of d. By Assertion 2.2, a is a $\overline{k}$-node ancestor of d (not necessarily proper), for some $k \geq i$. Therefore, by Assertion 2.2 there is a uni-directional edge (c,a); by replacing (c,b), (b,a) with (c,a) we obtain a smaller cycle. $\square$

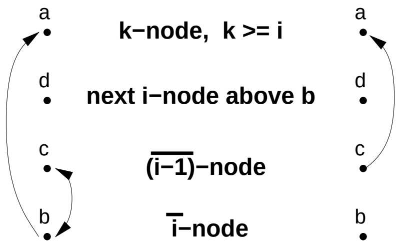

Figure 2. Left: the two edges incident on b in $\mathcal{C}$. Right: replaced by one edge.

2.4 A Measure of Information

We define D_a to be the elements of C_a that could have weight equal to a, given Assertions 2.2 & 2.3 and all the comparisons made by the preprocessing algorithm. It follows from Assertion 2.3 that D_a is non-empty. Define ϕ, Φ as

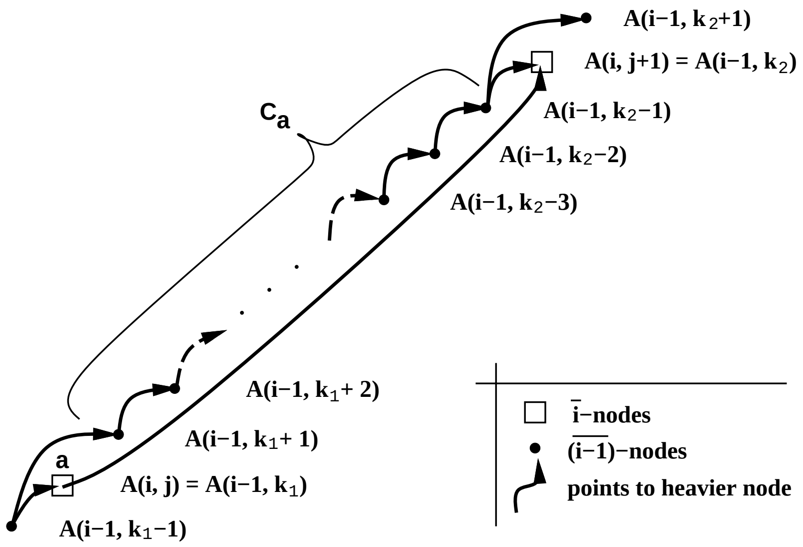

Figure 1. An $\bar{\imath}$-node a and its associated set C_a. Inequalities implied by Assertion 2.2 are marked with an arrow. To the right of each node is its *size* in Ackermann notation.

$$\phi(a) = \log \frac{|C_a|}{|D_a|} \quad \text{and} \quad \Phi = \sum_{a \in T} \phi(a).$$

This following lemma may be obvious. We prove it for completeness' sake.

Lemma 2.5 *If the preprocessing algorithm has a budget of Ψ comparisons, then $\mathbb{E}[\Phi] \leq \Psi$, where the expectation is over the input distribution and any random choices made by the algorithm.*

Proof: Consider, from Assertion 2.3, the set of random variables $X = \{X_a\}_a$. The expected number of bits of information one could derive about X in Ψ comparisons is clearly no more than Ψ. Now, $\phi(a) = \log \frac{|C_a|}{|D_a|}$ measures the number of bits known about X_a, for the special case when X_a is uniformly distributed over D_a. Therefore $\phi(a)$ is never *more* than the actual number of bits known about X_a, and, by the independence of the $\{X_a\}_a$ (Assertion 2.3 and Lemma 2.4), $\Phi = \sum_a \phi(a)$ is never more than the number of bits known about X. The Lemma follows. □

Φ measures a certain kind of information. A consequence of Lemma 2.6, proved below, is that for the right kind of query, any other information gathered by the preprocessor is not useful.

Lemma 2.6 *Let g be an arbitrary node, and let a, b, z be the first, second and last nodes of D_g (i.e., z ancestral to b, b ancestral to a). Then for any node e between g and z it is not known that $w(g) \leq w(e)$; for e between b and z it is not known that $w(g) \geq w(e)$.*

Proof: We only consider the case when $|D_g| \geq 3$, so there *are* distinct nodes a,b and z. Let g be an $\overline{(i+1)}$ node, a, b, z be $\bar{\imath}$-nodes and suppose e is a $\bar{k}$-node, $k \leq i$. If e is between g and z and it is known that $w(g) \leq w(e)$, then $w(g) < w(z)$ also follows, since from Assertion 2.2 $w(e) < w(z)$. This contradicts the fact that $z \in D_g$. For the second case, e lies between b and z — see Figure 3. The definition of $C_g \supseteq D_g$ implies $k \leq i$. Suppose that $w(g) \geq w(e)$ is known. Then by Assertion 2.2 g is heavier than all $\bar{k}$-nodes between a and b. However, by Assertion 2.3, *if* $w(g) = w(a)$ then $w(g) = w(f)$ for *some* $\bar{k}$ node between a and b. Therefore $w(g) \geq w(e)$ implies $a \notin D_g$, a contradiction. □

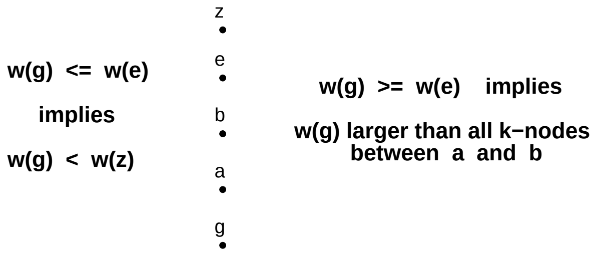

Figure 3. Ancestor relationship indicated by height on the page.

Lemma 2.6 greatly simplifies our lower bound proof. Consider any query path that includes g. Lemma 2.6 says that if the upper endpoint is not too close to the upper end of D_g, then no ancestor of g in the query is known to be heavier than g, and no ancestor of g at or above b is known to be lighter than g. These two facts will prove useful in Section 3.

3 The Lower Bound

Our main Theorem is stated below. The remainder of this section constitutes its proof.

Theorem 3.1 *Any (randomized) preprocessing algorithm for the online MST verification problem making at most $c\,n\log\lambda_t(n)$ comparisons has query complexity at least $t+1$ (with probability $1-\epsilon$ for any constant $\epsilon > 0$), for some constant c.*

Corollary 3.2 *Any linear-time preprocessing algorithm for the online MST verification problem has query complexity $\Omega(\alpha(n))$.*

The outline of the proof is as follows. We generate a query edge (f, a_0) by first finding an appropriate leaf f, then finding a sequence of nodes $a_t, a_{t-1}, \ldots, a_0$, where a_t is a $\overline{t}$-node ancestor of f and a_i is an $\overline{i}$-node ancestor of a_{i+1}. If we can then show that $\{f, a_t, \ldots, a_0\}$ are all candidate maxima for the query edge (f, a_0), by Lemma 2.1 the query algorithm must use at least $t+1$ comparisons.

For deterministic preprocessing algorithms Theorem 3.1 could be proved with $c = \frac{1}{8}$. We set $c = \frac{\epsilon}{8}$ and consider randomized preprocessing algorithms as well. We assume w.l.o.g. that $\lambda_t(n) \geq 8$.

Define *cost*(l), where l is a leaf, as

$$\textit{cost}(l) = \sum_{\substack{a \text{ ancestral to } l \\ \text{(including } l)}} \frac{\phi(a)}{\textit{size}(a)}$$

Clearly $\Phi = \sum_{a\in T}\phi(a) = \sum_l \textit{cost}(l)$ and from Lemma 2.5 we know that $\mathbb{E}[\Phi] \leq cn\log\lambda_t(n)$. By Markov's inequality $\Pr[\Phi \leq \epsilon^{-1}cn\log\lambda_t(n)] \geq 1-\epsilon$. Since Theorem 3.1 is only guaranteed with probability $1-\epsilon$, we assume below that $\Phi \leq \epsilon^{-1}cn\log\lambda_t(n) = \frac{1}{8}n\log\lambda_t(n)$.

Our query edge (f, a_0) is chosen as follows

1. Let $f = a_{t+1}$ be a leaf such that *cost*$(f) \leq \frac{\log\lambda_t(n)}{4}$
2. For $0 \leq i \leq t$, let a_i be the second most ancestral node in $D_{a_{i+1}}$.

In (1), such an f can always be found because the average leaf cost is bounded by $\frac{\epsilon^{-1}cn\log\lambda_t(n)}{(n+1)/2} \leq \frac{\log\lambda_t(n)}{4}$. For (2) we clearly require $|D_{a_{i+1}}| \geq 2$. We will actually require it to have at least 3 elements, in order to apply Lemma 2.6 later. Lemma 3.3 shows the feasibility of (2).

Lemma 3.3 *For $0 < i \leq t+1$, $|D_{a_i}| \geq 3$.*

Proof: We first show $|D_{a_{t+1}}| = |D_f| \geq 3$. By definition of $\phi(f) = \log\frac{|C_f|}{|D_f|}$, and the fact that $|C_f| = \lambda_t(n)$, we have $|D_f| = \lambda_t(n)/2^{\phi(f)}$, and since $\phi(f) \leq \textit{cost}(f)$, $|D_f| \geq (\lambda_t(n))^{\frac{3}{4}} \geq 3$. Recall that $\lambda_t(n)$ was assumed to be at least 8; the last inequality follows since $r^{\frac{3}{4}} \geq 3$ for $r \geq 8$.

Now to prove that $|D_{a_i}| \geq 3$ for $i \leq t$. Let j be such that *size*$(a_i) = A(i,j)$. Recall that C_{a_i} includes all $\overline{(i-1)}$-nodes between a_i and its next i-node ancestor. Therefore, $|C_{a_i}| = j_2 - j_1 - 1$ where

$$A(i,j) = A(i-1, j_1) \quad\text{and}\quad A(i,j+1) = A(i-1,j_2)$$

For $j > 0$, $|C_{a_i}| = 2^{2^{A(i,j)}} - 2^{2^{A(i,j-1)}} - 1$, and for $j = 0$, $|C_{a_i}| = 2^{2^{A(i,j)}} - 2$. In either case, if $|D_{a_i}| < 3$ then $\phi(a_i) = \log\frac{|C_{a_i}|}{|D_{a_i}|} \geq 2^{A(i,j)} - 2$ and hence

$$\textit{cost}(f) \;\geq\; \frac{\phi(a_i)}{\textit{size}(a_i)} \;\geq\; \frac{2^{A(i,j)}-2}{A(i,j)} \;\geq\; \lambda_t(n)$$

The last inequality follows from Lemma 3.4, below, stating that $A(i,j) \geq \lambda_t(n) \geq 8$. We now have a contradiction because we specifically chose f such that *cost*$(f) \leq \log\lambda_t(n)/4$.

□

The following lemma is only used in the proof of Lemma 3.3.

Lemma 3.4 *For $0 < i \leq t$, size$(a_i) \geq \lambda_t(n)$*

Proof: (sketch) Since a_i is an ancestor of a_{i+1} and hence *size*$(a_i) >$ *size*(a_{i+1}), we need only prove the lemma for $i = t$. By our selection of a_t, the lemma will follow from the inequality $A(t, \lceil r^{\frac{3}{4}}\rceil - 2) \geq r$, which holds for all $t \geq 0$, $r \geq 8$. One can prove by induction that for all i, j, $A(i,j) \geq 2^j$, and that $\lceil r^{\frac{3}{4}}\rceil - 2 \geq \log r$ for all $r \geq 8$.

□

Lemma 3.5, given below, establishes that $\{f, a_t, \ldots, a_0\}$ are all candidate maxima, and by Lemma 2.1, at least $t+1$ comparisons will be needed to answer the query (f, a_0). This will conclude our proof of Theorem 3.1.

Lemma 3.5 *On the query (f, a_0), the set of candidate maxima includes $\{f, a_t, \ldots, a_0\}$.*

Proof: Assuming the contrary, let $a_k \in \{f = a_{t+1}, a_t, \ldots, a_0\}$ and y be a node in the query path such that $w(y) > w(a_k)$ is known. By Lemma 2.6 y cannot be an ancestor of a_k. Suppose y is an $\overline{i}$-node descendant of a_k. (If $a_k = f$ this is clearly impossible.) By Assertion 2.2, clearly $i > k$. Since, also by Assertion 2.2, $w(y) > w(a_k)$ implies all $\overline{i}$-node ancestors of y are heavier than a_k, we may as well assume $y = a_i$ as a_i is the most ancestral $\bar{i}$-node on the query path. By Lemma 3.3 $|D(a_i)| \geq 3$ and by our choice of a_{i-1}, the upper endpoint of the query path has at least 2 elements from $D(a_i)$ below it. This allows us to apply the second part of Lemma 2.6, implying that it is not known whether $w(a_i) > w(a_k)$, a contradiction. Therefore a_k is a candidate maximum on the query path (f, a_0). □

3.1 One Last Detail

We proved our lower bound with a slightly non-standard version of Ackermann's function. Below we define B, following a more typical definition of Ackermann's function, and compare its row-inverse ρ_i with λ_i. Lemma 3.6, given below, can be repeated with little modification for any of the other definitions of Ackermann's function found in the literature [1, 35, 13, 15, 16].

$$\begin{aligned} B(0,j) &= 2^j \\ B(i+1,0) &= B(i,1) \\ B(i+1,j+1) &= B(i, B(i+1,j)) \end{aligned}$$

$$\rho_i(n) = \min\{j \ : \ B(i,j) \geq n\}$$

We omit the proofs that A and B are ascending in both arguments.

Lemma 3.6 *For any t, n, $\rho_t(n) \leq 3\lambda_t(n) + 1$.*

Proof: We will prove by induction that $B(i, 3j+1) \geq A(i,j)$ for all i, j, giving the lemma. Notice first that since $B(0,j) = A(0,j) = 2^j$, the Lemma holds for $i = 0$. Assume, henceforth, that $i > 0$. The following inequalities are easily proved by induction.

$$\begin{aligned} B(i,0) &\geq 4 \qquad \{\text{for } i \geq 2\} && (1) \\ B(i,j) &\geq 2^{2^j} \qquad \{\text{for } i \geq 1\} && (2) \end{aligned}$$

The case $j = 0$: $B(i, 3j+1) = B(i,1) = B(i-1, B(i,0))$. If $i = 1$ then $B(0, B(1,0)) > A(1,0)$ and we are done, so assume $i > 1$. By Inequality 1, $B(i-1, B(i,0)) \geq B(i-1, 4) \geq A(i-1, 1) = A(i,0)$, where the second inequality is by our inductive assumption.

Now for the general case: $i, j \geq 1$.

$$\begin{aligned} B(i, 3j+1) &= B(i-1, B(i-1, B(i-1, B(i, \\ &\qquad 3(j-1)+1)))) \quad \{\text{Def. of } B\} \\ \{i = 1\} \quad &\geq B(i-1, 2^{2^{A(i,j-1)}}) \\ &\qquad \{\text{Inductive assumption}\} \\ &= A(i,j) \qquad \{B(0,\cdot) = A(0,\cdot)\} \\ \{i > 1\} \quad &\geq B(i-1, 2^{2^{2^{2^{A(i,j-1)}}}}) \\ &\qquad \{\text{Induct. assumpt., Inequality 2}\} \\ &\geq A\left(i-1, \left(2^{2^{2^{2^{A(i,j-1)}}}} - 1\right)/3\right) \\ &\qquad \{\text{Inductive assumption}\} \\ &\geq A(i-1, 2^{2^{A(i,j-1)}}) \\ &= A(i,j) \qquad \{\text{Def. of } A\} \end{aligned}$$

□

4 Upper Bounds

In this section we show that the lower bound established in Section 3 is within a factor of 2 of optimal. We also show that under the assumption of randomly permuted edge-weights, an expected linear number of preprocessing comparisons is sufficient to answer MST verification queries with $t = 1$ additional comparisons. Bear in mind that we are mainly concerned with *decision tree* complexity in this section. The tools required to actually implement these algorithms are non-trivial.

Recall from Section 2 that we give the query algorithm one comparison for free. Therefore a preprocessing algorithm with parameter $t = 0$ means zero *additional* comparisons are necessary.

Theorem 4.1 *Suppose the edge-weights of a tree T are permuted randomly. With no more than $2n$ preprocessing comparisons (expected), MST verification queries on T can be answered with $t = 1$ additional comparisons. For $t = 0$, $\Theta(n \log n)$ preprocessing is necessary.*

Proof: Our notation here will reflect *edge*-weighted trees, rather than the vertex-weighted ones from Section 3.

First consider the $t = 0$ case with random edge-weights. If the input tree T is a star then we need to know the relative ordering of all edge-weights, which obviously requires $\Omega(n \log n)$ preprocessing comparisons; $O(n \log n)$ is also sufficient. (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. Could linear preprocessing suffice?)

For $t = 1$ the preprocessing algorithm must reduce the number of candidate maxima (on any query) to at most two. We root the tree arbitrarily and divide the query (u, v) into two queries $(u, LCA(u,v))$ and $(v, LCA(u,v))$. Hence, 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. Fix the node $z = z_0$; let $z_1, z_2, z_3, \ldots$ be the sequence of ancestors of z up to the root, and let $e_i = (z_i, z_{i+1})$. We must find the prefix-maxima of the sequence $(w(e_i))_i$, which is tantamount to finding the subsequence $L_0 = (e_{i_1}, e_{i_2}, e_{i_3}, \ldots)$ where e_{i_p} has maximum weight among $(e_0, \ldots, e_{i_{p+1}-1})$. We compute L_0 from $L_1 = (e_{j_1}, e_{j_2}, \ldots)$, where L_1 is the sequence for z_1 (z's parent). One can see that L_0 is derived from L_1 by substituting a (possibly empty) prefix of L_1 with e_0. We find such a prefix in the obvious manner, by comparing $w(e_0)$ with $w(e_{j_1}), w(e_{j_2}), \ldots$ until j_q is found such that $w(e_0) < w(e_{j_q})$. (If there is no such e_{j_q} then for the sake of consistent notation we let it be a dummy edge connecting the root to its nonexistent parent.) The comparison-cost of this procedure, which is performed for every edge in the input tree, is no more than q.[2] We analyze the behavior of q and j_q under the assumption that the tree edge-weights are randomly permuted. We have

$$\begin{align}
\Pr[j_q = r] &\le 1/r(r+1) \tag{3}\\
\mathbb{E}[q \mid j_q = r] &\le 1 + \sum_{i=1}^{r-1} \Pr[w(e_i) = \max_{1\le k\le i}\{w(e_k)\}] \notag\\
&= 1 + H_{r-1} \tag{4}\\
\mathbb{E}[q] &= \sum_{r=1}^{\infty} \Pr[j_q = r]\cdot \mathbb{E}[q \mid j_q = r] \notag\\
&\le 1 + \sum_{r=2}^{\infty} \frac{H_{r-1}}{r(r+1)} \notag\\
&= 1 + \sum_{i=1}^{\infty} \left(\frac{1}{i} \cdot \sum_{r=i+1}^{\infty} \frac{1}{r(r+1)} \right) \notag\\
&= 1 + \sum_{i=1}^{\infty} \frac{1}{i(i+1)} \quad = \quad 2 \tag{5}
\end{align}$$

Lines 3 and 4 are inequalities, rather than equalities, due to the finiteness of the $(e_i)_i$ sequence. Line 5 follows from Lines 3 and 4 and the identity $\sum_{i=1}^{k} 1/i(i+1) = 1 - 1/(k+1)$, which is easily proved by induction on k. □

In the general case, Theorem 4.2, given below, has a preprocessing cost of the form $O(n \log \lambda_k(n))$; this is an improvement over the previous constructions for arbitrary semigroups [2, 6], which have a preprocessing cost of $O(n\lambda_k(n))$. Both [2, 6] are generalizations of Yao's construction [38] for linear arrays. The proof of Theorem 4.2 is abbreviated; the techniques used are fully fleshed-out in [27, 2, 6, 24].

Theorem 4.2 *We wish to answer MST verification queries with t additional comparisons. For $t = 0$ a preprocessing cost of $\Theta(n \log n)$ is necessary and sufficient. For $t = 1$ a preprocessing cost of $O(n \log\log n)$ is sufficient, and for $t = 2k$, $k > 0$, a preprocessing cost of $O(n \log \lambda_k(n))$ is sufficient.*

Proof: (sketch) For $t = 0$, the argument is identical to Theorem 4.1.

For $t = 1$ we use King's reduction [24], which produces an equivalent tree (for the purpose of MST verification) with height no more than $\log n$. We then run a version[3] of Komlós's MST verification algorithm [27] directly on this tree, which takes $O(n \log h) = O(n \log\log n)$ comparisons on height h trees. Komlós's algorithm lets us answer node-to-ancestor queries in 0 comparisons, and hence arbitrary queries with 1 comparison.

For $t = 2k$ we use the same tree-decomposition technique used in [2] and [6]. The idea is to generate $k+1$ forests $F_0, F_1, F_2, \ldots, F_k$. A query on the original tree is then translated into $k+1$ queries: for each i, one query on some tree $T_i \in F_i$. We preprocess the F_0 trees for $t' = 0$ just by sorting their edge weights. For the remaining trees in $F_1, \ldots, F_k$ we preprocess them for $t' = 1$ using Komlós's algorithm [27]. For any query the number of candidate maxima is reduced to $2k+1$, as required. The time required to generate $F_0, \ldots, F_k$ is $O(n)$; the rest of the preprocessing is $O(n \log \lambda_k(n))$. □

5 Open Problems

There are several natural comparison-based problems which remain unresolved. Chief among them are the set maxima problem [20, 3, 32] and the minimum spanning tree problem; see [23, 10, 33, 32] for recent progress. We identify here two comparison-based problems for which slightly super-linear lower bounds seem plausible.

- The **split-findmin** problem is to maintain a set of intervals, made up of n weighted *elements*, under *split* operations, which split an interval in two, and $m \ge n$ *decrease-key* operations, which lower the weight of some element. The interval-minima must be known at all times. This peculiar data structure turns out to be very useful in certain weighted matching algorithms [19] and several recent shortest path algorithms [37, 21, 31, 29, 30]. It can also be used to solve the minimum spanning tree and shortest path tree sensitivity analysis problems. This last application is an unpublished result; see [36] for the definitions of the sensitivity analysis problems.

  Gabow's [19] implementation of split-findmin takes $O(m\alpha(m, n))$ time on a pointer machine, which is optimal (for pointer machines) by a result of LaPoutré

[2] It is usually equal to q, unless e_{j_q} happens to be the "dummy" edge, in which case the comparison $w(e_0) < w(e_{j_q})$ never takes place.

[3] The first stage of Komlós's algorithm is simply a preprocessing algorithm for answering MST verification queries selected from a fixed graph G. Setting G to be the complete graph makes it an all-purpose preprocessing algorithm for online MST verification.

[28]. Pettie and Ramachandran [31] recently showed that the number of *comparisons* required to implement the split-findmin structure is only $O(m \log \alpha(m,n))$. It is conceivable that $\Omega(m \log \alpha(m,n))$ is a lower bound as well.

- In [26] an $O(m\alpha(m,n))$ time algorithm is given to find the row-maxima in an $n \times m$ totally monotone "staircase" matrix. Is there a matching lower bound?

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.*, 99:118–133, 1928.

[2] 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.

[3] A. Bar-Noy, R. Motwani, and J. Naor. A linear time approach to the set maxima problem. *SIAM J. Discr. Math.*, 5(1):1–9, 1992.

[4] H. Brönnimann, B. Chazelle, and J. Pach. How hard is half-space range searching? *Discrete Comput. Geom.*, 10(2):143–155, 1993.

[5] 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.

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

[7] B. Chazelle. Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc.*, 2(4):637–666, 1989.

[8] B. Chazelle. Lower bounds for orthogonal range searching: II. the arithmetic model. *J. ACM*, 37(3):439–463, July 1990.

[9] B. Chazelle. Lower bounds for off-line range searching. *Discrete Comput. Geom.*, 17(1):53–65, 1997.

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

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

[12] B. Chazelle and B. Rosenberg. Computing partial sums in multidimensional arrays. In *Proc. 5th Annual Symposium on Computational Geometry (SCG'89)*, pages 131–139, 1989.

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

[14] B. Chazelle and B. Rosenberg. Simplex range reporting on a pointer machine. *Comput. Geom.*, 5(5):237–247, 1996.

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

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

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

[18] M. L. Fredman. A lower bound on the complexity of orthogonal range queries. *J. ACM*, 28(4):696–705, 1981.

[19] 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.

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

[21] T. Hagerup. Improved shortest paths on the word RAM. In *Proc. 27th Int'l Colloq. on Automata, Languages, and Programming (ICALP'00), LNCS vol. 1853*, pages 61–72, 2000.

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

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

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

[25] M. M. Klawe. Superlinear bounds on matrix searching. In D. Johnson, editor, *Proc. 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'90)*, pages 485–493, 1990.

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

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

[28] H. LaPoutré. Lower bounds for the union-find and the split-find problem on pointer machines. *J. Comput. Syst. Sci.*, 52:87–99, 1996.

[29] S. Pettie. A faster all-pairs shortest path algorithm for real-weighted sparse graphs. In *Proc. 29th Int'l Colloq. on Automata, Languages, and Programming (ICALP'02), LNCS vol. 2380*, pages 85–97, 2002.

[30] 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 ??–??, 2002.

[31] S. Pettie and V. Ramachandran. Computing shortest paths with comparisons and additions. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 267–276, 2002.

[32] 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.

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

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

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

[36] 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):30–33, 1982.

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

[38] 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.

[39] A. C. Yao. On the complexity of maintaining partial sums. *SIAM J. Comput.*, 14(2):277–288, 1985.