An Inverse-Ackermann Style Lower Bound for MST Verification Queries[*]

Seth Pettie
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
seth@cs.utexas.edu

TR-02-22

April 23, 2002

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 $T \cup \{e\}$? It is well-known that *offline* minimum spanning tree verification is solvable in linear time. We demonstrate that this online variant of the problem is intrinsically harder. In particular, a preprocessing cost of $\Omega(n \log \lambda_t(n))$ is necessary to answer queries with at most t comparisons, where λ_t is the t^{th}-row inverse of Ackermann's function. For the case of linear preprocessing this implies a query lower bound of $\Omega(\alpha(n))$ comparisons. Our lower bound also holds for randomized preprocessing algorithms.

1 Introduction

The theoretically best minimum spanning tree algorithms [KKT95, Cha00a, PR02c] were made possible by even more foundational algorithms and data structures, namely the minimum spanning tree verification algorithm of Komlós [Kom85, DRT92, Kin97, BKRW98] and Chazelle's Soft Heap [Cha00b]. It has been speculated by some (see, e.g., [Cha00a, 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 [Tar79b, HS86, CR91] 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 [Kla90] $\Omega(n\alpha(n))$ lower bound on the time to find row-maxima in a totally monotone $2n \times n$ matrix, where the non-blank

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

elements are contiguous in each column. However, in [Kla90] the relevant operation is not the *comparison* but the *matrix query*.

The MST verification problem is actually part of a well-studied family of problems concerned with computing partial sums. In these problems there is an underlying set of n 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 under various types of queries — see [Fre81, Yao85, Cha89, Cha90, BCP93, CR96, Cha97] for lower bounds and references. Chazelle & Rosenberg [CR89, CR91] studied the case where the elements are packed into a d-dimensional array and queries take the form of d-rectangles (see also [Yao82, AS87] for $d = 1$.) In [CR91] 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 [Tar79a, Tar82] studied certain partial-sums algorithms based on path-compression.

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*,[1] where the algorithm chooses which comparisons to make based on the outcomes of previous comparisons. Naturally, many bounds which hold for arbitrary semigroups do not hold for $(\mathbb{R}, \max)$. For instance, the problem of answering interval-maximum queries in a 1-D array can be done in constant time with linear preprocessing [Kom85] (contrast this with the superlinear lower bound in [CR91] for arbitrary semigroups). Solving MST verification *offline* on arbitrary trees can be done in linear time [Kom85, DRT92, Kin97, BKRW98], and the dual to this problem, MST sensitivity analysis, can be solved in randomized linear time [GKKS93, DRT92] or deterministic $O(m \log \alpha(m, n))$ time.[2] All these problems have $\Omega(m\alpha(m, n))$ lower bounds when generalized to arbitrary semigroups. 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.

Nearly all inverse-Ackermann type lower bounds are 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: One defines a space Σ of problem solutions and argues that at least $\log |\Sigma|$ comparisons are required in the worst case. The challenge in lower bounding the online MST verification problem is in combining these two very different approaches. We suspect that our techniques could be used to lower bound other comparison-based problems; two candidates 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.

[1]The decision-tree model is equivalent to allowing the semigroup operations (max or min) and tests for equality.

[2]The *split-findmin* data structure [Gab85, PR02a] was known to be useful in certain weighted matching and shortest path algorithms. One application of *split-findmin* not mentioned in [Gab85, PR02a] is that it can solve MST sensitivity analysis in $O(m \log \alpha(m, n))$ time, an $\Omega(\alpha / \log \alpha)$ factor faster than Tarjan's path-compression-based algorithm [Tar82].

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 which 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 to first characterize the behavior of the optimal query algorithm. Regardless of what the preprocessing algorithm did, 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. $\square$

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 [Ack28] is rarely defined the same way twice (see e.g., [Ack28, Tar79b, CR91, CLR90, CLRS01]). We would not presume to buck such a well-established precedent. Here is a slight variant:

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