VERIFICATION AND SENSITIVITY ANALYSIS OF MINIMUM SPANNING TREES IN LINEAR TIME

Brandon Dixon
Monika Rauch
Robert E. Tarjan

CS-TR-289-90

July 1990

Verification and Sensitivity Analysis of Minimum Spanning Trees in Linear Time

Brandon Dixon[1,2]

Monika Rauch[1,3]

Robert E. Tarjan[1,4,5]

July 27, 1990

Abstract

Komlós has devised a way to use a linear number of binary comparisons to test whether a given spanning tree of a graph with edge costs is a minimum spanning tree. The total computational work required by his method is much larger than linear, however. We describe a linear-time algorithm for verifying a minimum spanning tree. Our algorithm combines the result of Komlós with a preprocessing and table look-up method for small subproblems and with a previously known almost-linear-time algorithm. Additionally, we present an optimal deterministic algorithm and a linear-time randomized algorithm for sensitivity analysis of minimum spanning trees.

---

[1]Department of Computer Science, Princeton University, Princeton, New Jersey 08544.

[2]Research partially supported by a National Science Foundation Graduate Fellowship.

[3]Research supported by the German Fellowship Foundation, *Studienstiftung des deutschen Volkes.*

[4]NEC Research Institute, Princeton, New Jersey 08540

[5]Research at Princeton University partially supported by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science Foundation Science and Technology Center, grant NSF-STC88-09648, and the Office of Naval Research, contract N00014-87-K-0467.

1. Introduction

Suppose we wish to solve some problem for which we know in advance the size of the input data, using an algorithm from some well-defined class of algorithms. For example, consider sorting n numbers, when n is fixed in advance, using a binary comparison tree. Given a sufficient amount of preprocessing time and storage space, we can in a preprocessing step compute a minimum-depth comparison tree, store it explicitly, and then solve any instance of the sorting problem by using the precomputed comparison tree.

This technique is of course generally useless because it is prohibitively expensive in preprocessing time and storage space, both being at least exponential in n. There are situations in which this idea can be used to advantage, however. This is the case in problems susceptible to very efficient divide-and-conquer. The idea is to split the problem to be solved into subproblems, which are categorized into classes. If the subproblems are small enough, they can be solved efficiently as follows: An optimal algorithm for each class is precomputed and stored in a look-up table, and each instance of a subproblem is solved by looking up and running the algorithm for its class. For this technique to pay off, solving all the subproblems must reduce the original problem sufficiently that it can be solved quickly with respect to the size of the original problem by using a non-optimal algorithm.

This paper presents an application of this general technique to two problems concerning minimum spanning trees. This approach was first proposed explicitly by Larmore [12], who used it to solve a convex matrix searching problem. Related techniques were used by Gabow and Tarjan [6] to solve a disjoint set union problem and by Harel and Tarjan [9] to find nearest common ancestors in a tree.

We present an algorithm that verifies a minimum spanning tree in an n-vertex, m-edge graph in $O(m)$ time. We also give algorithms performing sensitivity analysis of minimum spanning trees in worst-case time minimum to within a constant factor and in linear expected time. Our computer model is a unit-cost random-access machine with word size $O(\log n)$ bits. The verification algorithm uses the comparison bound of Komlós [11] for the subproblems and Tarjan's $O(m\alpha(m,n))$ algorithm [14] for the reduced problem. For sensitivity analysis we solve the subproblems using a result of Goddard, King, and Schulman [7] in the randomized case and enumeration of all possible algorithms in the deterministic case. In both cases Tarjan's $O(m\alpha(m,n))$-time sensitivity analysis algorithm [15] processes the reduced problem. We describe the algorithms in Sections 2 and 3. Section 4 contains concluding remarks.

2. Verification of Minimum Spanning Trees

Let $G = (V, E)$ be a connected, undirected graph with vertex set V of size n and edge set E of size m. Suppose every edge $\{v, w\} \in E$ has a real-valued *cost* $c(v, w)$. A *minimum spanning tree* of G is a spanning tree whose total edge cost is minimum. The *minimum spanning tree verification problem* is that of determining whether a specific spanning tree T is a minimum spanning tree. Since G is connected, $m \geq n - 1$. To simplify time bounds, we assume that $m \geq n$; otherwise, G itself is a tree.

Several results concerning the minimum spanning tree verification problem are known. There are many efficient algorithms for *finding* a minimum spanning tree, given only the graph G and the edge costs; see the survey paper by Graham and Hell [8] or the monograph by Tarjan [16, Chapter 6]. The fastest known algorithm for finding a minimum spanning tree is that of Gabow, et al. [5], which runs in $O(m \log \beta(m, n))$ time, where $\beta(m, n) = \min\{i \mid \log^{(i)} n \leq m/n\}$, and $\log^{(i)} n$ is defined recursively by $\log^{(0)} n = n$, $\log^{(i+1)} n = \log \log^{(i)} n$. The verification problem was considered by Tarjan [14] and subsequently by Komlós [11]. Tarjan proposed a verification algorithm running in $O(m\alpha(m, n))$ time, where α is a functional inverse of Ackermann's function. Komlós showed that a minimum spanning tree can be verified in $O(m)$ binary comparisons between edge costs. Unfortunately, his method requires nonlinear time to determine which comparisons to make. Here we describe an algorithm that verifies a minimum spanning tree in $O(m)$ time.

Let T be a spanning tree whose minimality we wish to test. For any pair of vertices v, w, we denote by $T(v, w)$ the path in T from v to w. T is minimum if and only if, for every nontree edge $\{v, w\}$, $c(v, w) \geq \max\{c(x, y) \mid \{x, y\} \in T(v, w)\}$. In order to efficiently verify this condition, we replace each nontree edge $\{v, w\}$ by a set of up to six replacement edges, each of cost $c(v, w)$. This replacement leaves invariant the minimality of T. Edge replacement is a two-stage process. To begin the first stage, we choose an arbitrary vertex r and root T at r. We denote by $p(v)$ the parent of vertex v in the rooted version of T. For each nontree edge $\{v, w\}$, we compute the nearest common ancestor of v and w in T, say u. If v and w are unrelated in T (i.e., $u \notin \{v, w\}$), we replace $\{v, w\}$ by the pair of edges $\{u, v\}$, $\{u, w\}$, each with cost $c(v, w)$. Such replacement leaves invariant the minimality of T, at most doubles the number of nontree edges, and results in a graph such that every nontree edge joins two related vertices in T. The time to perform this replacement is $O(m)$ using either of the known linear-time algorithms for computing nearest common ancestors [9],[13].

We can now assume that each nontree edge $\{u, v\}$ is such that vertex u is an ancestor

of vertex v. In the second stage, we replace each such edge by a set of up to three edges. In order to determine the edge replacements, we partition T into a collection of edge-disjoint subtrees. Let $g \geq 1$ be an integer parameter, whose value we shall specify later. The subtrees have two properties:

(i) there are at most $(n-1)/g + 1$ subtrees; and

(ii) deletion from any subtree of its root and all edges incident to the root leaves a collection of smaller subtrees, called *microtrees*, each containing at most g vertices.

We compute the collection of subtrees in $O(n)$ time, as follows. We process all the vertices except r in postorder [16]. (This order guarantees that a parent is processed after all of its children.) When processing a vertex v, we compute an integer value $s(v)$ for it; and, in addition, we may mark it as a subtree root. The computed value of $s(v)$ is the number of descendants of v in T (including v itself) that are in the same microtree as v. Initially all vertices are unmarked. The vertex processing step is as follows:

process(v): Compute $h = 1 + \sum\{s(w) \mid w \text{ is a child of } v\}$. If $h \leq g$ then let $s(v) = h$; otherwise, mark v as a subtree root and let $s(v) = 1$.

Once the vertex processing is completed we mark r, the root of T, as a subtree root. Condition (ii) is immediate from the definition of the vertex processing. Condition (i) is also immediate: each subtree, except possibly the one rooted at r, contains more than g vertices and hence contains at least g edges, which means that there are at most $(n-1)/g + 1$ subtrees.

Let T' be the tree whose vertices are the marked vertices of T, with v the parent of w in T' if v is the deepest marked proper ancestor of w in T (i.e., the first marked vertex encountered on the path from w to r in T). We call T' the *macrotree*. By (i), T' has $O(n/g)$ vertices. Tree T' can be computed in $O(n)$ time by doing a depth-first traversal of T and maintaining the set of marked proper ancestors of the currently visited vertex on a stack; when the search visits a vertex v, the deepest marked proper ancestor of v, which we denote by $p'(v)$, is on top of the stack. (We adopt the convention that $p'(r)$ is undefined.)

We use the macrotree to define the replacement edges for each nontree edge. Let $\{u, v\}$ be such a nontree edge, with u an ancestor of v. Let $r_1 = p'(u)$ if u is unmarked or u if u is marked. Similarly, let $r_3 = p'(v)$ if v is unmarked or v if v is marked. If $r_1 = r_3$, we do not replace $\{u, v\}$. If $r_1 \neq r_3$, let r_2 be the child of r_1 in T' that is an

ancestor of r_3; replace $\{u,v\}$ by $\{u,r_2\}$, $\{r_2,r_3\}$, $\{r_3,v\}$, deleting any of these edges that is a loop (an edge of the form $\{x,x\}$ for some x). Each new edge has a cost of $c(u,v)$. This replacement leaves invariant the minimality of T and at most triples the number of nontree edges.

We can compute the replacement edges for every nontree edge in a total of $O(m)$ time, as follows. A depth-first traversal of T as described above allows us to compute $p'(v)$ for each vertex $v \neq r$. This gives r_1 and r_3 in the edge replacement construction. It remains to compute the r_2-vertices in the edge replacement construction. The computation of these vertices requires answering $O(m)$ queries of the following form on T': given a vertex z and another vertex y that is a proper ancestor of z, determine the child of y that is an ancestor of z. These queries can be answered in $O(m)$ time by performing a depth-first traversal of T', maintaining a stack of the ancestors of the currently visited vertex, and answering the query for a pair y,z when visiting z during the search, by reporting as the answer to the query the vertex just above y on the stack.

Having computed all the replacement edges, we must test, for each replacement edge $\{w,x\}$, whether $c(w,x) \geq \max\{c(y,z) \mid \{y,z\} \in T(w,x)\}$. In the rest of this section we describe how to perform this test for all replacement edges.

For each vertex $v \neq r$, we compute a value $high(v)$ equal to the maximum cost of an edge on the path $T(p'(v),v)$. These values can be computed for all vertices by doing a separate depth-first traversal of each of the subtrees of T that were determined by the partitioning process described previously. During the traversal of the subtree rooted at a vertex u, we maintain the path of edges from u to the currently visited vertex as a stack with heap order [16]; the values that are heap-ordered are the edge costs, and the *high*-values are computed using *find-max* operations. This data structure requires $O(1)$ amortized time per *push*, *pop*, or *find-max* operation [16]. Hence the total time to compute all *high*-values is $O(n)$. The *high*-values suffice to perform the required test for each of the $\{r_3,v\}$-replacement edges, in $O(1)$ time per edge: for such an edge, $high(v) = \max\{c(y,z) \mid \{y,z\} \in T(r_3,v)\}$.

We deal with the $\{r_2,r_3\}$-replacement edges by adding all of these edges to T' to form a graph G', giving each edge $\{p'(v),v\}$ in T' a cost $c(p'(v),v) = high(v)$, and verifying that T' is a minimum spanning tree in G'. To verify the minimality of T' we use the algorithm of Tarjan [14], which runs in $O(m\alpha(m,n'))$ time, where n' is the number of vertices in T'. If $g = \Omega(\log^{(i)} n)$ for any fixed positive integer i, then $n' = O(n/\log^{(i)} n)$, and $\alpha(m,n') = O(1)$ [16]. Thus verifying the minimality of T' takes $O(m)$ time.

The remaining edges that must be tested are the $\{u,r_2\}$-replacement edges. Each

such edge has u and r_2 in the same microtree. Let $T_1, T_2, \ldots, T_k$ be the microtrees. For $1 \leq i \leq k$, we form a graph G_i by adding each $\{u, r_2\}$-replacement edge to the tree T_i such that u and r_2 are in T_i. Together the graphs $G_1, G_2, \ldots, G_k$ contain n vertices and $O(m)$ edges. By (ii), each G_i contains at most g vertices. We complete the task of verifying the minimality of T by verifying that T_i is a minimum spanning tree of G_i, for each i in the range $1 \leq i \leq k$.

To verify the minimality of the microtrees, we use a preprocessing and table look-up technique. For each possible connected graph with no more than g vertices and specified spanning tree, we construct a short integer encoding by numbering the vertices consecutively from 1, encoding each edge by the pair of numbers of its end vertices, and concatenating the encodings of the edges, listing the spanning tree edges first. (It does not matter that this encoding is not unique.) The encoding for a graph-tree pair contains at most $\lceil \log g \rceil g^2/2$ bits, since there are fewer than $g^2/2$ edges. The total number of possible code strings (not all of which are legal encodings of graphs) is not more than $2^{\lceil \log g \rceil g^2/2}$. We will choose g such that each graph encoding fits into one computer word and such that there are at most $\sqrt{n}$ possible code strings. Choosing $g \leq c_2(\log n)^{1/3}$ for a suitably small value of c_2 more than suffices for this purpose.

Consider a connected graph with at most g vertices and $e < g^2/2$ edges and having a specified spanning tree T^*. The result of Komlós [11] implies that there is a decision tree D whose nodes represent binary comparisons of edge costs that will verify the minimality of T^* and has a depth of at most $c_1 e$, for some sufficiently large c_1. The number of nodes in D is at most $2^{c_1 e + 1}$. Furthermore, an inspection of the construction of Komlós shows that D can easily be constructed in $O(g^2)$ time per node, for a total of $O(g^2 2^{c_1 g^2/2})$ time.

Choosing $g \leq c_2(\log n)^{1/3}$ for a suitably small value of c_2 guarantees that the construction time for one decision tree is $O(\sqrt{n})$, and the total time required to construct decision trees for all possible graphs with at most g vertices is $O(n)$. Furthermore, the space needed to store all the decision trees is $O(n)$.

We construct one decision tree for each possible graph with at most g vertices and then build a table that maps code strings for graphs to the corresponding decision trees. Then we use the table to verify the minimality of the microtrees T_i in the respective graphs G_i, by computing a code string for each G_i, T_i pair, accessing the decision tree corresponding to the code string, and following the path through the decision tree determined by the edge costs of G_i. The total time to perform all the verifications is $O(m)$. This completes the verification of T.

The only constraints imposed on the choice of g in this construction are $g = \Omega(\log^{(i)} n)$

for some fixed positive integer i and $g \leq c_3(\log n)^{1/3}$ for $c_3 = \min\{c_0, c_2\}$. Thus it suffices to choose $g = c_3(\log n)^{1/3}$.

3. Sensitivity Analysis of Minimum Spanning Trees

An extension of the minimum spanning tree verification problem is the sensitivity analysis problem. Let G be an undirected graph with edge costs and let T be a minimum spanning tree of G. The *sensitivity analysis problem* is to compute, for each edge $\{v, w\}$ of G, by how much $c(v, w)$ can change without affecting the minimality of G. Tarjan [15] has extended his verification algorithm to an algorithm that solves the sensitivity analysis problem in $O(m\alpha(m, n))$ time. For the special case of planar graphs, Booth and Westbrook [2] have given an algorithm running in $O(m)$ time. We shall describe a randomized $O(m)$-time algorithm and a deterministic algorithm that runs in time minimum to within a constant factor, although all that we can say for sure about the running time of the latter algorithm is that it is $O(m\alpha(m, n))$ and $\Omega(m)$. Our technique is the same as that of Section 2; namely, we reduce the original problem in $O(m)$ time to a collection of subproblems, each of which is small enough to solve by using a decision tree selected from a precomputed set of such trees.

Let $\{v, w\}$ be a nontree edge. Let $a(v, w) = \max\{c(x, y) \mid \{x, y\} \in T(v, w)\}$. Then T remains minimum until the edge cost of $\{v, w\}$ decreases by more than $c(v, w) - a(v, w)$. Similarly, let $\{v, w\}$ be a tree edge. Let $b(v, w) = \min\big\{c(x, y) \mid \{x, y\}$ is a non-tree edge such that $\{v, w\} \in T(x, y)\big\}$. Then T remains minimum until the edge cost of $\{v, w\}$ increases by more than $b(v, w) - c(v, w)$. (See [15].)

The value of $a(v, w)$ for every nontree edge $\{v, w\}$ can be computed in $O(m)$ time by a simple extension of the verification algorithm in Section 2: instead of verifying that $c(v, w) \leq a(v, w)$, we compute $a(v, w)$ explicitly.

Computing $b(v, w)$ for every tree edge $\{v, w\}$ is harder. We first replace the nontree edges exactly as in Section 2: each nontree edge $\{x, y\}$ is replaced by a set of up to six nontree edges, each of cost equal to $\{x, y\}$, in a way that preserves $b(v, w)$ for every tree edge $\{v, w\}$. In the process of performing this replacement, we choose a root r of T and compute subtree roots, subtrees, and microtrees exactly as in Section 2. After the replacement, each nontree edge $\{x, y\}$ is such that x and y are related in T, say x is an ancestor of y. In addition, such an edge is of exactly one of three types:

Type 1: x is a subtree root, y is not a subtree root, and $x = p'(y)$, where p' is defined as in Section 2: $p'(y)$ is the deepest ancestor of y that is a subtree root.

Type 2: x and y are subtree roots.

Type 3: x and y are in the same microtree.

We compute each value $b(v,w)$ using the equation

$$b(v,w) = \min\{b_1(v,w),\, b_2(v,w),\, b_3(v,w)\}\,,$$

where b_i for $i = 1,2,3$ is defined exactly like $b(v,w)$ but with the minimum taken only over nontree edges of type i.

To compute the b_1-values, we begin by computing, for each vertex $v \neq r$, the value $min_1(v) = \min\big\{c(x,y) \,|\, \{x,y\}$ is a type-1 edge such that $x = p'(y)$ is a proper ancestor of v and v is an ancestor of $y\big\}$. The min_1-values can be computed in $O(m)$ time by visiting the vertices of T in postorder and applying the recurrence

$$min_1(v) = \begin{cases} \infty & \text{if } v \text{ is a subtree root;} \\ \min\big(\{c(x,v) \,|\, \{x,v\} \text{ is a type-1 edge}\} & \text{if } v \text{ is not a subtree root.} \\ \quad \cup\, \{min_1(w) \,|\, p(w) = v\}\big) & \end{cases}$$

Then, for every vertex $v \neq r$, $b_1(p(v),v) = min_1(v)$.

We compute the b_2-values in three steps. First, we form the graph G' as in Section 2 by adding to the macrotree T' each type-2 edge. Second, we compute, for each tree edge $\{v,w\}$ of the macrotree T', the value $b'(v,w) = \min\big\{c(x,y) \,|\, \{x,y\}$ is a type-2 edge such that $(v,w) \in T'(x,y)\big\}$. All the b'-values can be computed in $O(m\alpha(m,n'))$ time by applying the sensitivity analysis algorithm of Tarjan [15] to the graph G' and the tree T'. Choosing g (the size parameter for the macrotrees) to be $\Omega(\log^{(i)} n)$ for any fixed positive integer i results in an $O(m)$ time bound for this computation. Third, we compute, for each vertex $v \neq r$ in T, the value $min_2(v) = \min\big\{b'(p'(y),y) \,|\, \{p'(y),y\}$ is an edge of T' such that y is a descendant of v and $p'(y)$ is a proper ancestor of $v\big\}$. The min_2-values can be computed in $O(n)$ time by visiting the vertices of T in postorder and applying the recurrence

$$min_2(v) = \begin{cases} b'(p'(v),v) & \text{if } v \text{ is a subtree root and} \\ \min\{min_2(w) \,|\, p(w) = v\} & \text{if } v \text{ is not a subtree root.} \end{cases}$$

Then, for every vertex $v \neq r$, $b_2(p(v),v) = min_2(v)$.

All that remains is to compute the b_3-values. Since the definition of a type-3 edge $\{x,y\}$ implies that x and y are in the same microtree, we can compute the b_3-values by adding the type-3 edges to the appropriate microtrees to form graphs $G_1, G_2, \ldots, G_k$ as in Section 2 and then process each G_i, T_i pair separately. We again use preprocessing to

construct a fast decision-tree algorithm for each possible graph-tree pair and then use table lookup to select the correct algorithm for each actual pair G_i, T_i.

It is only in the construction of the decision trees that the randomized and deterministic algorithms differ. We first consider the deterministic case. A decision tree for the sensitivity analysis problem consists of a binary tree, each internal node of which specifies a comparison between the costs of two edges, and each leaf x of which provides a mapping f_x from the tree edges of the problem graph to the nontree edges, such that the b_3-value of any edge e is $c(f_x(e))$, assuming that the edge costs are consistent with the outcome of the comparisions leading to leaf x. The algorithm of Tarjan [15] implies the existence of an $O(m\alpha(m,n))$-depth decision tree for the sensitivity analysis of an n-vertex, m-edge graph and given spanning tree. Since $\alpha(g^2, g) = O(1)$ [16], these decision trees have depth $O(g^2)$.

For each possible connected graph with no more than g vertices and specified spanning tree, we construct a minimum-depth decision tree for sensitivity analysis by brute-force enumeration. We restrict our attention to complete binary trees, enumerating all possible decision trees of each possible depth in increasing order by depth until finding a correct one. A complete binary decision tree of depth d has $2^d - 1$ internal nodes and 2^d leaves. Each internal node corresponds to one of the less than g^4 possible binary comparisions of edge costs; each leaf can correspond to one of the less than g^{2g} possible mappings of the tree edges to the nontree edges. Thus there are less than $(g^4)^{2^d-1}(g^{2g})^{2^d} < g^{g2^{d+2}}$ possible decision trees of depth d, assuming $g \geq 2$. The total number of trees that must be considered before encountering a correct one is $O(g^{g2^{c_3 g^2}})$ for some suitably large constant c_3. This is $O(2^{2^{g^3}})$. The space needed to store a decision tree of depth d is $O(2^d g \log g) = O(2^{g^3})$, if $d = O(g^2)$. To determine whether a particular decision tree correctly solves the sensitivity analysis problem, it suffices to test that the correct answer is obtained for each of the at most $(g^2)! = O(2^{g^3})$ possible permutations of edge costs. Testing one permutation requires $O(g^2)$ time. The time to test a particular decision tree is thus $O(g^2 2^{g^3}) = O(2^{g^4})$, and the time to find a minimum-depth decision tree is $O(2^{g^4} 2^{2^{g^3}}) = O(2^{2^{g^4}})$. If we choose $g = c_4(\log\log n)^{1/4}$ for some sufficiently small constant c_4, then the time to find minimum-depth decision trees for all possible graph-tree pairs is $O(n)$, as is the space needed to store them in a table.

We compute the b_3-values for all tree edges in a microtree T_i by indexing the lookup table with the code string of the pair G_i, T_i to get a decision tree and evaluating the decision tree with the given edge costs.

The total time needed for sensitivity analysis is $O(m)$ plus time proportional to the

sum of the minimum numbers of comparisons needed to perform sensitivity analysis for all of the G_i, T_i pairs. Performing sensitivity analysis for all of the G_i, T_i pairs is at most a constant factor more time-consuming than performing sensitivity analysis for a worst-case n-vertex, m-edge graph. Thus the sensitivity analysis algorithm runs in time minimum to within a constant factor, assuming that only binary comparisons between edge costs are used as tests.

In the randomized case, we replace the deterministic decision trees used for sensitivity analysis of the microtrees by randomized decision trees. In a randomized decision tree, each internal node corresponds either to a comparison of two edge costs or to a test of a distinct random bit. As in the deterministic case, we require every path of the decision tree to give the correct answer, but as a measure of the complexity of the tree we use the weighted average depth of a leaf, rather than the worst-case depth, taking the weight of a leaf to be $1/2^i$, where i is the number of tests of random bits along the path from the root to the leaf.

Goddard, King and Shulman [7] have found a randomized algorithm to compute the maxima of n subsets of an ordered universe of size n in $O(n)$ comparisions on the average. Their result, together with the observation of King [10] that their algorithm needs only $O(n)$ random bits on the average, implies the existence of a randomized decision tree of average depth $O(m)$ for the sensitivity analysis problem. Such a decision tree can be converted into a decision tree of $O(m)$ average depth and $O(m \log m)$ worst-case depth by trimming the decision tree at depth $m \log m$ and replacing each subtree that was cut out by a decision tree that merely sorts by cost the edges of the problem graph. A brute-force enumeration can be used to find minimum-average-depth randomized decision trees for all possible microtree problems. The details mimic the deterministic case, so we omit them. The Goddard-King-Shulman result then implies that the resulting randomized sensitivity analysis algorithm runs in $O(m)$ expected time, for a suitable choice of the microtree size bound g.

4. Concluding Remarks

We have illustrated by means of two related examples a general technique of speeding up divide-and-conquer algorithms by a preprocessing and table lookup technique. A curious phenomenon is that the technique can give algorithms running in time minimum to within a constant factor, but for which we can not presently offer a tight asymptotic time analysis. This is the case for our deterministic minimum spanning tree sensitivity analysis algorithm and for Larmore's convex matrix searching algorithm[12]; both

have running times somewhere between linear and an inverse Ackerman function times linear. Providing tight analysis of these algorithms amounts to bounding the number of comparisons needed to solve the corresponding problems. Obtaining tight bounds remains open. A related question is whether the randomized maxima-finding algorithm of Goddard, King, and Shulman can be made deterministic.

The technique we have illustrated is not limited to comparison-based problems. We can allow arithmetic operations in the decision trees used to solve the subproblems. Testing the correctness of such a decision tree amounts to testing the validity of a first-order sentence about the real numbers. Such sentences can be tested in double-exponential time[1], which suffices for the use of the method: we merely reduce the size of the subproblems to double-logarithmic, triple-logarithmic, or further, as needed. As an example, the technique can be applied to the $O(n \log^* n)$-time algorithm of Chazelle[3] for triangulating a simple n-sided polygon, to produce an algorithm running in time minimum to within a constant factor. The bound for this algorithm is in fact $O(n)$, because of the even more recent result of Chazelle[4] giving an explicitly linear-time algorithm. Further applications remain to be discovered.

References

[1] M. Ben-Or, D. Kozen, and J. Reif, The Complexity of Elementary Algebra and Geometry, *J. Comput. System Sci.* **32**(2) (1986) pp. 251–264.

[2] H. Booth and J. Westbrook, Linear Algorithms for Analysis of Minimum Spanning and Shortest Path Trees in Planar Graphs, Yale University, Department of Computer Science, TR-768, Feb. 1990.

[3] B. Chazelle, Efficient Polygon Triangulation, Princeton University, CS-TR-249-90, Feb. 1990.

[4] B. Chazelle, Efficient Polygon Triangulation to *Proc. 31nd Annual IEEE Sympos. on Foundation of Comput. Sci.*, 1990, to appear.

[5] H.N. Gabow, Z. Galil, T. Spencer, and R.E. Tarjan, Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs, *Combinatorica* **6**(2) (1986) pp. 109-122.

[6] H.N. Gabow and R.E. Tarjan, A Linear-Time Algorithm for a Special Case of Disjoint Set Union, *J. Comput. System Sci.* **30**(2) (1985) pp. 209–221.

[7] W. Goddard, V. King, and L. Schulman, Optimal Randomized Algorithms for

Local Sorting and Set-Maxima, in *Proc. 22nd Annual ACM Sympos. on Theory of Computing*, 1990, pp. 45-53.

[8] R.L. Graham and P. Hell, On the History of the Minimum Spanning Tree Problem, *Ann. Hist. Comput.* **7**(1) (1985) pp. 43–47.

[9] D. Harel and R.E. Tarjan, Fast Algorithms for Finding Nearest Common Ancestors, *SIAM J. Comput.* **13**(2) (1984) pp. 338–355.

[10] V. King, personal communication.

[11] J. Komlós, Linear Verification for Spanning Trees, *Combinatorica* **5** (1985) pp. 57–65.

[12] L.L. Larmore, An Optimal Algorithm with Unknown Time Complexity for Convex Matrix Searching, *Information Processing Letters*, to appear.

[13] B. Schieber and U. Vishkin, On Finding Lowest Common Ancestors: Simplification and Parallelization, *SIAM J. Comput.* **17**(6) (1988) pp. 1253–1262.

[14] R.E. Tarjan, Applications of Path Compressions on Balanced Trees, *J. Assoc. Comput. Mach.* **26**(4) (1979) pp. 690–715.

[15] R.E. Tarjan, Sensitivity Analysis of Minimum Spanning Trees and Shortest Path Trees, *Information Processing Letters* **14**(1) (1982) pp. 30–33. Corrigendum, Ibid **23**(4) (1986), p.219.

[16] R.E. Tarjan, *Data Structures and Network Algorithms,* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.