A Randomized Linear-Time Algorithm for Finding Minimum Spanning Trees

Philip N. Klein*

Robert E. Tarjan†

October 12, 1993

Abstract

We present a randomized linear-time algorithm for finding a minimum spanning tree in a connected graph with edge weights. The algorithm is a modification of one proposed by Karger and uses random sampling in combination with a recently discovered linear-time algorithm for verifying a minimum spanning tree. Our computational model is a unit-cost random-access machine with the restriction that the only operations allowed on edge weights are binary comparisons.

1 Introduction

We consider the problem of finding a minimum spanning tree in a connected graph with real-valued edge weights. This problem has a long and rich history; the first fully realized algorithm was devised by Borůvka in the 1920's [3]. An informative survey paper by Graham and Hell [9] describes the history of the problem up to 1985. In the last two decades faster and faster algorithms were found, the fastest being an algorithm of Gabow, Galil, and Spencer [7] (see also [8]), with a running time of $O(m \log \beta(m,n))$ on a graph of n vertices and m edges. Here $\beta(m,n) = \min\{i \mid \log^{(i)} n \leq m/n\}$.

*Department of Computer Science, Brown University, Providence, RI 02912-1910. Research partially supported by an NSF PYI award, CCR-9157620, together with PYI matching funds from Thinking Machines Corporation, Xerox Corporation, and Honeywell Corporation. Additional support provided by DARPA Contract No. N00014-91-J-4052, ARPA Order No. 8225, and by the NEC Research Institute, Princeton, NJ.

†Department of Computer Science, Princeton University, Princeton, NJ and the NEC Research Institute, Princeton, NJ. Research at Princeton University partially supported by the National Science Foundation, Grant No. CCR-8920505; the Office of Naval Research, Contract No. N0014-91-J-1463; and DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science and Technology Center, Grant No. NSF-STC88-09648.

This and earlier algorithms used as a computational model the sequential unit-cost random-access machine with the restriction that the only operations allowed on the edge weights are binary comparisons. Fredman and Willard [6] considered a more powerful model that allows bit manipulation of the binary representations of the edge weights. In this model they were able to devise a linear-time algorithm. Still, the question of whether a linear-time algorithm exists for the restricted random-access model remained open.

A problem related to finding minimum spanning trees is that of verifying that a given spanning tree is minimum. Tarjan [16] gave a verification algorithm running in $O(m\ \alpha(m,n))$ time, where α is a functional inverse of Ackermann's function. Later, Komlós [13] showed that a minimum spanning tree can be verified in $O(m)$ binary comparisons of edge weights, but with nonlinear overhead to decide which comparisons to make. Dixon, Rauch and Tarjan [5] recently combined these algorithms with a table lookup technique to obtain an $O(m)$-time verification algorithm.

In this paper we describe a randomized algorithm for finding a minimum spanning tree. It has an expected running time of $O(m)$ in the restricted random-access model. The algorithm is a modification of one proposed by Karger [10], who obtained an expected time bound of $O(m + n \log n)$. Our algorithm uses the random-sampling technique of Karger in combination with the verification algorithm of Dixon, Rauch, and Tarjan to discard edges that cannot be in the minimum spanning tree.

Our algorithm uses no algorithmic ideas that are not part of Karger's algorithm. The key to our improvement of Karger's result is a random-sampling lemma that improves his corresponding lemma by a logarithmic factor. In the next section we present this lemma and its proof, which uses a technique different from Karger's. In Sections 3 and 4 we describe and analyze our algorithm. Section 5 contains some final remarks.

Throughout the paper, we assume for simplicity that all edge weights are distinct. This implies that the minimum spanning tree is unique. Ties in edge weights can be broken by numbering the edges and choosing the edge of smaller number whenever a tie occurs. We also assume that the input graph has no isolated vertices (vertices without incident edges). Our algorithm actually solves the slightly more general problem of finding a minimum spanning forest in a possibly disconnected graph. We assume familiarity with standard results on minimum spanning trees, as presented, for example, in [17].

2 A Sampling Lemma

In order to present the sampling result, we need a little terminology. Let G be a graph with weighted edges. We denote by $w(x, y)$ the weight of edge $\{x, y\}$. If F is a forest in G, we denote by $F(x, y)$ the path (if any) connecting x and y in F, and by $w_F(x, y)$ the maximum weight of an edge on $F(x, y)$, with the convention that $w_F(x, y) = \infty$ if x and y are not connected in F. We say an edge $\{x, y\}$ is *F-heavy* if $w(x, y) > w_F(x, y)$ and F-light otherwise. For any forest F, no F-heavy edge can be in the minimum spanning forest of G. This is a consequence of the "red rule" for minimum spanning tree construction ([17], p.71). Given a forest F in G, the F-heavy edges of G can be computed in time linear in the number of edges of G, using an adaptation of the verification algorithm of Dixon, Rauch, and Tarjan. (Page 1188 in [5] describes the changes needed in the algorithm.)

Lemma 1 *Let H be a subgraph obtained from G by including each edge independently with probability p, and let F be the minimum spanning forest of H. Then, for any k, the number of F-light edges of G exceeds k with probability at most*

$$\sum_{i=0}^{n} \binom{k}{i} p^i (1-p)^{k-i} \tag{1}$$

where n is the number of vertices of G.

Proof. Let $e_1, e_2, ..., e_m$ be the edges of G, arranged in increasing order by weight. Consider the following computation, a variant of Kruskal's minimum spanning tree algorithm [14]. Begin with an empty forest F. Process the edges e_j in increasing order on j. To process an edge e_j, first flip a coin that has probability p of coming up heads. If the coin does come up heads, include e_j in H. Then test whether both endpoints of e_j are in the same connected component of F. If so, e_j is F-heavy; discard it. If not, add e_j to F if e_j belongs to H. The forest F produced by this computation is the forest that would be produced by Kruskal's algorithm applied to the edges in H, and is therefore exactly the minimum spanning forest of H. Our goal is to show that the number of edges not discarded is probably small.

The coin-flips corresponding to discarded edges are irrelevant; such edges are F-heavy regardless of whether they are included in H. We therefore consider only coin-flips corresponding to non-discarded edges. For each non-discarded edge, if the coin comes up heads, the edge is placed in F. The size of F is at most $n - 1$. Hence the probability that the number of non-discarded edges exceeds k is at most the probability that k coin-flips take place before the n^{th} occurence of heads. Since the number of occurences of heads could be any i from 0 to $n - 1$, the probability of this happening is at most $\sum_{i=0}^{n-1} \binom{k}{i} p^i (1-p)^{k-i}$.

$\square$

Remark. Lemma 1 has an immediate generalization to matroids. See [11].

3 The Algorithm

The algorithm intermeshes steps of Borůvka's minimum-spanning-tree algorithm [3] with random-sampling steps that discard edges that cannot be in the minimum spanning tree. Each Borůvka step reduces the number of vertices by at least a factor of two. A random-sampling step is performed only if the graph density (the number of edges divided by the number of vertices) is sufficiently high, in which case the step reduces the number of edges by a constant factor with high probability.

The algorithm is recursive. In the case of a dense graph, it performs two recursive calls, but the total size of the two subproblems is with high probability no more than a fraction less than one of the size of the original problem. It follows that the expected running time is linear. The recurrence relation resembles the one arising in the analysis of a linear-time selection algorithm [2].

Here is a complete specification of the algorithm.

Step 1. For each vertex, select the minimum-weight edge incident to the vertex. Contract all the selected edges, replacing by a single vertex each connected component defined by the selected edges and eliminating resulting isolated vertices, loops (edges both of whose endpoints are the same) and all but the lowest-weight edge among each set of multiple edges.

Step 2. If the density of the remaining graph is less than 6, go directly to Step 3; otherwise, proceed as follows. Choose a subgraph H by including each edge independently with probability 1/2. Apply the algorithm recursively to H, producing a minimum spanning forest F of this subgraph. Find all the F-heavy edges in the entire graph and delete them.

Step 3. Apply the algorithm recursively to the remaining graph to compute a minimum spanning forest F'. The minimum spanning forest for the original graph consists of those edges contracted in Step 1 plus the edges of F'.

4 Analysis of the Algorithm

We shall show that the algorithm runs in linear time except with exponentially small probability, from which it follows immediately that the expected running time of the algorithm is linear. We begin with some observations about the worst-case behavior of the algorithm.

Suppose the algorithm is applied to a graph with n vertices and m edges. The total time spent in Steps 1–3, excluding the time spent on the recursive subproblems, is $O(m)$: Step 1 is just a single step of Borůvka's algorithm, which takes $O(m)$ time using straightforward graph-algorithmic techniques; Step 2 takes $O(m)$ time using the modified Dixon-Rauch-Tarjan verification algorithm, as noted in Section 2.

Let n_0 and m_0 be the number of vertices and the number of edges, respectively, in the graph remaining after Step 1. At least $n/2$ edges are contracted in Step 1, since one edge is selected for each vertex but an edge can be selected twice, once for each of its endpoints. This implies that $n_0 \leq n/2$ and $m_0 \leq m - n/2$. Combining these inequalities yields $m_0 \leq m - n_0$. If the density m_0/n_0 of the graph remaining after Step 1 is less than 6, then $m_0 < m - m_0/6$, so $m_0 < 6m/7$. In this case the recursive invocation in Step 3 is applied to a graph with less than $\frac{6}{7}m$ edges.

Suppose m_0/n_0 is at least 6. In this case the random-sampling step is performed. Let m_1 be the number of edges in the sample graph H, and let m_2 be the number of edges left after deletion of the F-heavy edges. The forest F constructed in Step 2 has at most $n_0 - 1$ edges; these edges are the only ones in H not deleted by Step 2. It follows that $m_1 + m_2 \leq m_0 + n_0 - 1 < m$. This inequality allows us to derive a bound of m on the number of recursive invocations of the algorithm. If $I(m)$ is the worst-case number of invocations as a function of m, we have the following recurrence:

$$
\begin{aligned}
I(1) &= 1 \\
I(m) &\leq 1 + \max\{I(m_1) + I(m_2) | m_1 + m_2 < m\} \text{ for } m > 1 \qquad (2)
\end{aligned}
$$

It follows by induction that $I(m) \leq m$.

Next we bound the worst-case running time of the algorithm. If $T(n, m)$ is the worst-case running time on a graph of at most n vertices and at most m edges, $T(n, m)$ satisfies the following recurrence:

$$
\begin{aligned}
T(1, m) &= O(m) \\
T(n, m) &\leq O(m) + \max\{T\lfloor n/2\rfloor, m_1) + T(\lfloor n/2\rfloor, m_2) | m_1 + m_2 < m\} \text{ for } n > 1 \qquad (3)
\end{aligned}
$$

A proof by induction yields $T(n, m) = O(m \log n) = O(m \log m)$.

Remark. The alternative recurrence $T(1, m) = O(1), T(n, m) \leq O(n^2) + 2\,T(\lfloor n/2 \rfloor, m)$ for $n > 1$, which follows from the inequality $m \leq \binom{n}{2}$, yields the bound $T(n, m) = O(n^2)$. Combining bounds gives a worst-case running time of $O(\min\{m \log m, n^2\})$, the same as the bound for Borůka's algorithm.

We are now ready to undertake a probabilistic analysis of the algorithm. The key is to show that $m_1 + m_2 \leq (1 - \epsilon)n$ with high probability, for a sufficiently small positive constant ϵ. An appropriate recurrence then yields a running time that is $O(m)$ with high probability.

The number of edges m_1 in the sample graph H is binomially distributed with mean $m_0/2 \leq m/2$. A standard bound on the tail of the binomial distribution [1, 15] implies that the probability that $m_1 > \frac{m}{2}(1 + \delta_1)$ is exponentially small, namely $\exp(-\Omega(m))$, for any constant $\delta_1 > 0$. Choosing $\delta_1 = 1/10$, we have $m_1 > 11\,m/20$ with probability $\exp(-\Omega(m))$.

Next we estimate the number of F-light edges, m_2. Lemma 1 applies with $p = 1/2$ to give an upper bound of $(\frac{1}{2})^k \sum_{i=0}^{n_0} \binom{k}{i}$ on the probability that m_2 exceeds k. This sum is the probability of at most n_0 heads occuring in a sequence of k flips of an unbiased coin. The expected number of heads is $k/2$. Let $k = \frac{m}{3}(1 + \delta_2)$, where $\delta_2 > 0$ is a constant to be specified later. Since $n_0 \leq m_0/6 \leq m/6$, we have $n_0 \leq \frac{k}{2}(1 + \delta_2)^{-1}$. The bound on the tail of the binomial distribution cited above implies that the probability of at most n_0 heads occuring in k unbiased coin flips is $\exp(-\Omega(m))$. Choosing $\delta_2 = 1/20$, we find that $m_2 > 7m/20$ with probability $\exp(-\Omega(m))$.

We call an invocation of the algorithm a *failure* if Step 2 is executed bu $m_1 > 11m/20$ or $m_2 > 7m/20$. The analysis above shows that the probability of failure is $\exp(-\Omega(m))$. Let r be a parameter to be determined. We call an invocation of the algorithm *large* if the number of edges in the corresponding problem graph exceeds r. We analyze the running time of the algorithm under the assumption that no large invocation fails. We show later that this assumption holds with high probability.

Let $A(m)$ denote the running time of the algorithm on a graph of at most m edges assuming that no large invocation fails. Using the worst-case time bound of $O(m \log m)$ and the definition of failure, we obtain the following recurrence for $A(m)$, where c_0 and c_1 are suitable positive constants:

$$\begin{aligned}
A(m) &\leq c_0 m \log m && \text{if} m \leq r \\
A(m) &\leq c_1 m + \max\{A(11m/20) + A(7m/20),\ A(6m/7)\} && \text{if } m > r
\end{aligned} \tag{4}$$

Lemma 2 *$A(m) \leq 10c_1\, m + c_0 m^{1-\epsilon} r^{\epsilon} \log r$ where $\epsilon < 1$ is a suitable positive constant.*

Proof. The proof is by induction on m. If $m \leq r$ then $c_0 m^{1-\epsilon} r^{\epsilon} \log r \geq c_0 m \log m$, so the inequality follows from the base case of the recurrence. Suppose $m > r$. By the induction hypothesis,

$$\begin{aligned} A(11m/20) &\leq 11c_1m/2 + c_0(11m/20)^{1-\epsilon}r^\epsilon \log r \\ &\leq 11c_1m/2 + c_0(3/5)m^{1-\epsilon}r^\epsilon \log r \end{aligned}$$

if ϵ is chosen so that $(11/20)^{1-\epsilon} \leq 3/5$.

Similarly,

$$A(7m/20) \leq 7c_1m/2 + c_0(2/5)m^{1-\epsilon}r^\epsilon \log r$$

if ϵ is chosen so that $(7/20)^{1-\epsilon} \leq 2/5$.

It follows that

$$c_1m + A(11m/20) + A(7m/20) \leq 10c_1m + c_0m^{1-\epsilon}r^\epsilon \log r.$$

The inequality

$$c_1m + A(6m/7) \leq 10c_1m + c_0m^{1-\epsilon}r^r \log r$$

also follows easily from the induction hypothesis. Thus the inequality of the lemma holds, and the proof is complete. $\square$

A few observations complete the analysis. For a graph of m edges we choose $r = m/\log^{1/\epsilon} m$. Lemma 2 implies that the running time of the algorithm is $O(m)$ unless some large invocation among those in the recursion fails. The probability of one such invocation failing is $\exp(-\Omega(r))$. There are at most m recursive invocations altogether, which means that the probability of even one failure occuring is $m\exp(-\Omega(r)) = \exp(-\Omega(r))$. Thus the algorithm runs in $O(m)$ time except with probability $\exp(-m/\log^{O(1)} m)$. In particular, the expected running time of the algorithm is $O(m)$.

5 Remarks

The algorithm can be changed in many ways without affecting the high-probability $O(m)$ time bound. For example, the random-sampling step can be performed unconditionally, rather than just when the graph is dense. The running time analysis becomes more complicated if this change is made, however.

In joint work with Richard Cole to be described in a future paper, we have adapted our algorithm to run in parallel. The parallel algorithm does linear work and runs in $O(\log n 2^{\log^* n})$ time on a PRIORITY PRAM [12]. In contrast, Karger [11] claims an algorithm running on an EREW PRAM that requires $O(\log n)$ time and $m/\log n + n^{1+\epsilon}$ processors for any constant $\epsilon > 0$.

Among remaining open problems, we note especially the following three:

1. Is there a deterministic linear-time minimum spanning tree algorithm in the restricted random-access model?

2. Can randomization be used to simplify the linear-time verification algorithm?

3. Can randomization be used fruitfully to solve other network optimization problems, such as the shortest-path problem? Randomization has already proved valuable in solving the maximum-flow [4] and minimum-cut [10] problems.

Acknowledgments

We thank David Karger, Satish Rao, and David Zuckerman for fruitful discussions.

References

[1] N. Alon and J. H. Spencer, *The Probabilistic Method*, John Wiley & Sons, Inc., New York, N. Y., 1992, p. 223.

[2] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan, "Time bounds for selection," *J. Comput. System Sci.* 7, 1973, pp. 448-461.

[3] O. Borůvka, "O jistém problému minimálním, *Práca Moravské Přírodovědecké Společnosti* 3, 1926, pp. 37-58. (In Czech.)

[4] J. Cheriyan, T. Hagerup, and K. Mehlhorn, "Can a maximum flow be computed in $O(nm)$ time?", *Proc. 17th International Colloquium on Automata, Languages, and Programming*, published as *Lecture Notes in Computer Science*, Vol. 443, Springer-Verlag, New York, 1990, pp. 235-248.

[5] B. Dixon, M. Rauch, and R. E. Tarjan, "Verification and sensitivity analysis of minimum spanning trees in linear time," *SIAM J. on Computing* 21, 1992, pp. 1184-1192.

[6] M. Fredman and D. E. Willard, "Trans-dichotomous algorithms for minimum spanning trees and shortest paths," *Proc. 31st Annual IEEE Symp. on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 719-725.

[7] H. N. Gabow, Z. Galil, and T. H. Spencer, "Efficient implementation of graph algorithms using contraction," *Proc. 25th Annual IEEE Symp. on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1984, pp. 338-346.

[8] 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, 1986, pp. 109-122.

[9] R. L. Graham and P. Hell, “On the history of the minimum spanning tree problem,” *Annals of the History of Computing* 7, 1985, pp. 43-57.

[10] D. R. Karger, “Global min-cuts in RNC and other ramifications of a simple mincut algorithm,” *Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, Association for Computing Machinery, New York, NY, and Society for Industrial and Applied Mathematics, Philadelphia, PA, 1993, pp. 21-30.

[11] D. R. Karger “Random sampling in matroids, with applications to graph connectivity and minimum spanning trees,” *Proc. 34st Annual IEEE Symp. on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, , 1993, to appear.

[12] R. M. Karp and V. Ramachandran, “A survey of parallel algorithms for shared-memory machines,” Chapter 17 in *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity* J. van Leeuwen, ed., MIT Press, Cambridge, Mass., 1990, pp. 869–941.

[13] J. Komlós, “Linear verification for spanning trees,” *Combinatorica* 5, 1985, pp. 57-65.

[14] J. B. Kruskal, “On the shortest spanning subtree of a graph and the traveling salesman problem,” *Proc. Amer. Math Soc.* 7, 1956, pp. 48-50.

[15] P. Raghavan, “Lecture Notes on Randomized Algorithms,” Research Report RC 15340 (#68237), Computer Science/Mathematics IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, 1990, 54.

[16] R. E. Tarjan, “Applications of path compression on balanced trees,” *J. Assoc. Comput. Mach.* 26, 1979, pp. 690-715.

[17] R. E. Tarjan, *Data Structures and Network Algorithms*, Chapter 6, Society for Industrial and Applied Mathematics, Philadelphia, 1983.