A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees

David R. Karger* Stanford University

Philip N. Klein† Brown University

Robert E. Tarjan‡ Princeton University and NEC Research Institute

July 7, 1994

Abstract

We present a randomized linear-time algorithm to find a minimum spanning tree in a connected graph with edge weights. The algorithm 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 [2]. An informative survey paper by Graham and Hell [12] describes the

*Department of Computer Science, Stanford University, Stanford, CA 94305.

Supported by a Hertz Foundation Graduate Fellowship, by NSF Grant CCR-9010517, and NSF Young Investigator Award CCR-9357849, with matching grants from IBM, Mitsubishi, the Schlumberger Foundation, the Shell Foundation, and the Xerox Corporation.

†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-89-20505; 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.

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 [10] (see also [11]), with a running time of $O(m \log \beta(m,n))$ on a graph of n vertices and m edges. Here $\beta(m,n) = \min\{i \mid \log^{(i)} n \leq m/n\}$.

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 [9] 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 [22] gave a verification algorithm running in $O(m\, \alpha(m,n))$ time, where α is a functional inverse of Ackerman's function. Later, Komlós [19] 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 [7] combined these algorithms with a table lookup technique to obtain an $O(m)$-time verification algorithm. King [17] recently obtained a simpler $O(m)$-time verification algorithm that combines ideas of Borůvka, Komlós, and Dixon, Rauch, and Tarjan.

In this paper we describe a randomized algorithm for finding a minimum spanning tree. It runs in $O(m)$ time with high probability in the restricted random-access model. The algorithm is a modification of one proposed by Karger [13, 15], who obtained a time bound of $O(n \log n + m)$. The $O(m)$ time bound is due to Klein and Tarjan [18]. The present paper is a revision of [18] that includes a tightened high-probability complexity analysis. Section 2 presents the random-sampling result that is the key to the $O(m)$ bound. Section 3 presents our algorithm, and Section 4 contains its analysis. Section 5 includes some final remarks. This section ends with some preliminaries.

1.1 Preliminaries

Our algorithm actually solves the slightly more general problem of finding a minimum spanning forest in a possibly disconnected graph. We assume that the input graph has no isolated vertices (vertices without incident edges).

If edge weights are not distinct, we can make them distinct by numbering the edges and breaking weight-ties according to the numbers. We therefore assume for simplicity that all edge weights are distinct. This assumption ensures that the minimum spanning tree is unique. The following properties are also well-known and correspond respectively to the red rule and the blue rule in [23].

Cycle property: For any cycle C in a graph, the heaviest edge in C does not appear in the minimum spanning forest.

Cut property: For any proper nonempty subset X of the vertices, the lightest edge with exactly one endpoint in X belongs to the minimum spanning forest.

Unlike most algorithms for finding a minimum spanning forest, our algorithm makes use of each property in a fundamental way.

2 A Sampling Lemma

Our algorithm relies on a random-sampling step to discard edges that cannot be in the minimum spanning tree. The effectiveness of this step is shown by a lemma that we present below. 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. Note that the edges of F are all F-light. For any forest F, no F-heavy edge can be in the minimum spanning forest of G. This is a consequence of the cycle property. 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 [7] describes the changes needed in the algorithm) or of that of King.

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. The expected number of F-light edges in G is at most n/p where n is the number of vertices of G.*

Proof. We describe a way to construct the sample graph H and its minimum spanning tree F simultaneously. The computation is a variant of Kruskal's minimum spanning tree algorithm [20]. Begin with H and F empty. Process the edges in increasing order by weight. To process an edge e, first test whether both endpoints of e are in the same connected component of F. If so, e is F-heavy, because every edge currently in F is lighter than e. Next, flip a coin that has probability

p of coming up heads. Include the edge e in H if and only if the coin comes up heads. Finally, if e is in H and is F-light, add e to the forest F.

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. An edge e that is F-heavy when it is processed remains F-heavy until the end of the computation, since F never loses edges. Similarly, an edge e that is F-light when processed remains F-light, since only edges heavier than e are added to F after e is processed. Our goal is to show that the number of F-light edges is probably small.

When processing an edge e, we know whether e is F-heavy before flipping a coin for e. Suppose for purposes of exposition we flip a penny for e if e is F-heavy and a nickel if it is not. The penny-flips are irrelevant to our analysis; the corresponding edges are F-heavy regardless of whether or not they are included in H. We therefore consider only the nickel-flips and the corresponding edges. For each such edge, if the nickel comes up heads, the edge is placed in F. The size of F is at most $n-1$. Thus at most $n-1$ nickel-tosses have come up heads by the end of the computation.

Now imagine that we continue flipping nickels until $n-1$ heads have occured, and let Y be the total number of nickels flipped. Then Y is an upper bound on the number of F-light edges. The distribution of Y is exactly the negative binomial distribution with parameters $n-1$ and p [8]. The expectation of a random variable that has a negative binomial distribution is $(n-1)/p$ [8]. It follows that the expected number of F-light edges is at most $(n-1)/p$. $\square$

Remark. The above proof actually shows that the number of F-light edges is stochastically dominated by a variable with a negative binomial distribution.

Remark. Lemma 1 directly generalizes to matroids. See [15].

3 The Algorithm

The minimum spanning forest algorithm intermeshes steps of Borůvka's algorithm, called *Borůvka steps*, with random-sampling steps. Each Borůvka step reduces the number of vertices by at least a factor of two; each random-sampling step discards enough edges to reduce the density (ratio of edges to vertices) to a fixed constant with high probability.

The algorithm is recursive. It generates two subproblems, but with high probability the total size of these subproblems is at most a constant fraction less than one of the size of the original problem. This fact is the basis for the probabilistic linear bound on the running time of the algorithm.

We begin by describing a Borůvka step.

Borůvka Step. 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 deleting all 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.

A Borůvka step reduces the number of vertices by at least a factor of two.

Now we describe the minimum spanning forest algorithm. If the graph is empty, return an empty forest. Otherwise, proceed as follows.

Step 1. Apply two successive Borůvka steps to the graph, thereby reducing the number of vertices by at least a factor of four.

Step 2. In the contracted graph, choose a subgraph H by selecting each edge independently with probability 1/2. Apply the algorithm recursively to H, producing a minimum spanning forest F of H. Find all the F-heavy edges (both those in H and those not in H) and delete them.

Step 3. Apply the algorithm recursively to the remaining graph to compute a spanning forest F'. Return those edges contracted in Step 1 together with the edges of F'.

We prove the correctness of the algorithm by induction. By the cut property, every edge contracted during Step 1 is in the minimum spanning forest. Hence the remaining edges of the minimum spanning forest of the original graph form a minimum spanning forest of the contracted graph. It remains to show that the recursive call in Step 3 finds the minimum spanning forest of the contracted graph.

By the cycle property, the edges deleted in Step 2 do not belong to the minimum spanning forest. By the inductive hypothesis, the minimum spanning forest of the remaining graph is correctly determined in the recursive call of Step 3.

Remark. Our algorithm can be viewed as an instance of the generalized greedy algorithm presented in [23], from which its correctness follows immediately.

4 Analysis of the Algorithm

We begin our analysis by making some observations about the worst-case behavior of the algorithm. Then we show that the expected running time of the algorithm is linear, by applying Lemma 1 and the linearity of expectations. Finally, we show that the algorithm runs in linear time with all but exponentially small probability, by developing a global version of the analysis in the proof of Lemma 1 and using a Chernoff bound [1, 4, 21].

Consider a single invocation of the algorithm. The total time spent in Steps 1–3, excluding the time spent on recursive subproblems, is linear in the number of edges: Step 1 is just two steps of Borůvka's algorithm, which takes linear time using straightforward graph-algorithmic techniques, and Step 2 takes linear time using the modified Dixon-Rauch-Tarjan verification algorithm, as noted in Section 2. The total running time is thus bounded by a constant factor times the total number of edges in the original problem and in all recursive subproblems. Thus our objective is to estimate this total number of edges.

Suppose the algorithm is initially applied to a graph with n vertices and m edges. Since the graph contains no isolated vertices, $m \geq n/2$. Each invocation of the algorithm generates at most two recursive subproblems. Consider the entire binary tree of recursive subproblems. The root is the initial problem. For a particular problem, we call the first recursive subproblem, occuring in Step 2, the *left child* of the parent problem, and the second recursive subproblem, occuring in Step 3, the *right child*. At depth d, the tree of subproblems has at most 2^d nodes, each a problem on a graph of at most $n/4^d$ vertices. Thus the depth of the tree is at most $\log_4 n$, and there are at most $\sum_{d=0}^{\infty} 2^d n/4^d = \sum_{d=0}^{\infty} n/2^d = 2n$ vertices total in the original problem and all subproblems.

Theorem 1 *The worst-case running time of the minimum-spanning-forest algorithm is $O(\min\{n^2, m \log n\})$, the same as the bound for Borůvka's algorithm.*

Proof. We estimate the worst-case total number of edges in two different ways. First, since there are no multiple edges in any subproblem, a subproblem at depth d contains at most $(n/4^d)^2/2$ edges. Summing over all subproblems gives an $O(n^2)$ bound on the total number of edges. Second, consider the left and right children of some parent problem. Suppose the parent problem is on a graph of v vertices. Every edge in the parent problem ends up in exactly one of the children (the left if it is selected in Step 2, the right if it is not), with the exception of the edges in the minimum spanning forest F of the sample graph H, which end up in both subproblems, and the edges that are removed in Step 1, which end up in no subproblem. If v' is the number of vertices in the graph after Step 1, then F contains $v' - 1 \leq v/4$ edges. Since at least $v/2$ edges are removed in Step 1, the total number of edges in the left and right subproblems is at most the number of edges in the parent problem.

It follows that the total number of edges in all subproblems at any single recursive depth d is at most m. Since the number of different depths is $O(\log n)$, the total number of edges in all recursive subproblems is $O(m \log n)$. $\square$

Theorem 2 *The expected running time of the minimum spanning forest algorithm is* $O(m)$.

Proof. Our analysis relies on a partition of the recursion tree into left paths. Each such path consists of either the root or a right child and all nodes reachable from this node through a path of left children. Consider a parent problem on a graph of X edges, and let Y be the number of edges in its left child. Since each edge in the parent problem is either removed in Step 1 or has a chance of $\frac{1}{2}$ of being selected in Step 2, $E[Y|X = k] \leq k/2$. It follows by linearity of expectation that $E[Y] \leq E[X]/2$. That is, the expected number of edges in a left subproblem is at most half the expected number of edges in its parent. It follows that, if the expected number of edges in a problem is k, then the sum of the expected numbers of edges in every subproblem along the left path descending from the problem is at most $\sum_{i=0}^{\infty} k/2^i = 2k$.

Thus the expected total number of edges is bounded by twice the sum of m and the expected total number of edges in all right subproblems. By Lemma 1, the expected number of edges in a right subproblem is at most twice the number of vertices in the subproblem. Since the total number of vertices in all right subproblems is at most $\sum_{d=1}^{\infty} 2^{d-1}\, n/4^d = n/2$, the expected number of edges in the original problem and all subproblems is at most $2m + n$. $\square$

Theorem 3 *The minimum spanning forest algorithm runs in* $O(m)$ *time with probability* $1 - e^{-\Omega(m)}$.

Proof. We obtain the high-probability result by applying a global version of the analysis in the proof of Lemma 1. We first bound the total number of edges in all right subproblems. These are exactly the edges that are found to be F-light in Step 2 of the parent problems. Referring back to the proof of Lemma 1, let us consider the nickel-tosses corresponding to these edges. Each nickel that comes up heads corresponds to an edge in a spanning forest in a right subproblem. The total number of edges in all such spanning forests in all right subproblems is at most the number of vertices in all such subproblems, which in turn is at most $n/2$ as shown in the proof of Theorem 2. Thus $n/2$ is an upper bound on the total number of heads in nickel-flips. The probability that there are more than $3m$ F-light edges is at most the probability that fewer than $n/2$ heads occur in a sequence of $3m$ nickel-tosses. By a Chernoff bound [1, 4, 21], this probability is $e^{-\Omega(m)}$ since $m \geq n/2$.

We now consider the edges in left subproblems. The edges in a left subproblem are obtained from the parent problem by sampling; i.e., a coin is tossed for each edge in the parent problem not deleted in Step 1, and the edge is copied to the subproblem if the coin comes up heads and is not copied if the coin comes up tails. To put it another way, an edge in the root or in a right subproblem gives rise to a sequence of copies in left subproblems, each copy resulting from a coin-flip coming up heads. The sequence ends if a coin-flip comes up tails. The number of occurrences of tails is at most the number of sequences, which in turn is at most the number m' of edges in the root problem and in all right subproblems. The total number of edges in all these sequences is equal to the total number of heads, which in turn is at most the total number of coin-tosses. Hence the probability that this number of edges exceeds $3m'$ is the probability that at most m' tails occur in a sequence of more than $3m'$ coin-tosses. Since $m' \geq m$, this probability is $e^{-\Omega(m)}$ by a Chernoff bound.

Combining this with the previous high-probability bound of $O(m)$ on m', we find that the total number of edges in the original problem and in all subproblems is $O(m)$ with probability $1-e^{-\Omega(m)}$.

$\square$

5 Remarks

In work with Richard Cole [5], Klein and Tarjan have adapted the randomized algorithm to run in parallel. The parallel algorithm does linear expected work and runs in $O(\log n \; 2^{\log^* n})$ expected time on a CRCW PRAM [16]. This is the first parallel algorithm for minimum spanning trees that does linear work. In contrast, Karger [13] gives an algorithm running on an EREW PRAM that requires $O(\log n)$ time and $m/\log n + n^{1+\epsilon}$ processors for any constant $\epsilon > 0$. Also, Cole and Vishkin [6] give an algorithm running on a CRCW PRAM that requires $O(\log n)$ time on $O((n+m)\log\log n/\log n)$ processors.

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 or some other technique 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 [3] and minimum-cut [14] problems.

Acknowledgments

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

[3] 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.

[4] H. Chernoff, "A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Annals of Mathematical Statistics*, 23, 1952, pp. 493–509.

[5] R. Cole, P. N. Klein, and R. E. Tarjan, "A linear-work parallel algorithm for finding minimum spanning trees," to appear in *Proc., 6th Symposium on Parallel Algorithms and Architectures*, 1994.

[6] R. Cole and U. Vishkin, "Approximate and exact parallel scheduling with applications to list, tree, and graph problems," *Proc. 27th Annual IEEE Symp. on Foundations of Computer Science*, Computer Society Press, Los Alamitos, CA, 1986, pp. 478-491.

[7] 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.

[8] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. John Wiley and Sons, 3rd edition, 1968.

[9] 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.

[10] 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. 347-357.

[11] 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.

[12] 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.

[13] D. R. Karger, "Approximating, verifying, and constructing minimum spanning forests," manuscript, 1992.

[14] 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.

[15] 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, pp. 84-93.

[16] 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.

[17] V. King, "A simpler minimum spanning tree verification algorithm," manuscript, 1993.

[18] Philip N. Klein and Robert E. Tarjan, "A linear-time algorithm for finding minimum spanning trees," *Proceedings of the* 26^{th} *ACM Symposium on Theory of Computing*, 1994, pp. 9–15 .

[19] J. Komlós, "Linear verification for spanning trees," *Combinatorica* 5, 1985, pp. 57-65.

[20] 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.

[21] 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, p. 54.

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

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