Minimum Spanning Tree

MA252 Project

Name: Rathod Vijay Mahendra

Roll No: 180123037

Name: Ayaz Anis

Roll No: 180123007

1 Aim

The objective of this project is to learn and understand the state of art in Minimum Spanning Tree and look forward if it is possible to extend ideas further.

2 Background

A spanning tree of a connected undirected graph G with n vertices is a subset of n-1 edges of G that, together with the vertices of G, form a tree. The Minimum Spanning Tree problem is to, given a connected undirected graph with edge weights, find a spanning tree of G the sum of whose edge weights is the smallest possible. Minimum Spanning Tree (hereafter called MST) has a host of applications, from travelling-salesman approximations to circuit design to utility networks. With this in mind, it's useful to develop MST algorithms that scale well to the amount of concurrency available on modern-day machines.

The history of the minimum spanning tree (MST) problem is long and rich, going as far back as Borůvka's work [2], long before computers even existed. In fact, MST is perhaps the oldest open problem in computer science. According to Nešetřil (1997), "this is a cornerstone problem of combinatorial optimization and in a sense its cradle." Borůvka's algorithm was published as a method of constructing an efficient electricity network. A second algorithm is Prim's algorithm, which was invented by Vojtěch Jarník in 1930 and rediscovered by Prim [13] and Dijkstra in 1959. A third algorithm commonly in use is Kruskal's algorithm, given by Joseph Kruskal [11]. A fourth algorithm, not as commonly used, is the reverse-delete algorithm, which is the reverse of Kruskal's algorithm.

Textbook algorithms run in $O(m \log n)$ time, where n and m denote, respectively, the number of vertices and edges in the graph. Time Complexity of Boruvka's algorithm is $O(m \log n)$ which is the same as Kruskal's and Prim's algorithms. Runtime of the reverse-delete algorithm is $O(m \log n (\log \log n)^3)$. Improvements to $O(m \log \log n)$ were given independently by Yao [15] and by Cheriton and Tarjan [4]. In the mid-eighties, Fredman and Tarjan [7] lowered the complexity to $O(m\beta(m, n))$, where $\beta(m, n)$ is the number of log-iterations necessary to map n to a number less than m/n.The complexity was further reduced to $O(m \log \beta(m, n))$ by Gabow et al. [9].

Several researchers have tried to find more computationally-efficient algorithms. In a comparison model, in which the only allowed operations on edge weights are pairwise comparisons, Karger, Klein & Tarjan [10] found a linear time randomized algorithm based on a combination of Borůvka's algorithm and the reverse-delete algorithm. If the edge costs are integers and the model allows bucketing and bit manipulation, it is possible to solve the problem in linear time deterministically, as was shown by Fredman and Willard [8]. To achieve a similar result in a comparison-based model has long been a high-priority objective in the field of algorithms. The reason why is, first, the illustrious history of the MST problem; second, the fact that it goes to the heart of matroid optimization.

Recently Seth Pettie and Vijaya Ramachandran [12] gave a deterministic, comparison-based MST algorithm that runs in $O(T*(m, n))$, where $T*(m, n)$ is the number of edge weight comparisons needed to determine the MST. Because of the nature of their algorithm, its exact running time is unknown. The source of their algorithm's mysterious running time, and its optimality, is the use of precomputed "MST decision trees" whose exact depth is unknown but nonetheless provably optimal.

3 Work Focused

The fastest non-randomized comparison-based algorithm with known complexity was given by Bernard Chazelle [3] which is based on the soft heap, an approximate priority queue. Its running time is $O(m\ \alpha(m,n))$, where α is the classical functional inverse of the Ackermann function. The function α grows extremely slowly, so that for all practical purposes it may be considered a constant no greater than 4; thus Chazelle's algorithm takes very close to linear time. However, since it uses a soft heap and an approximate priority queue, it is difficult to implement.

In this report, we will look at the algorithm given by Karger et al. [10] for solving the minimum spanning tree problem.

3.1 Algorithm by Karger et al.

This algorithm is a randomized algorithm for computing the minimum spanning forest of a weighted graph with no isolated vertices. It was developed by David Karger, Philip Klein, and Robert Tarjan in 1995. The algorithm relies on techniques from Borůvka's algorithm along with an algorithm for verifying a minimum spanning tree in linear time. It combines the design paradigms of divide and conquer algorithms, greedy algorithms, and randomized algorithms to achieve expected linear performance.

We first state two important properties. We will assume for simplicity that all edge weights are distinct. If edge weights are not distinct, we can make them distinct by numbering the edges and breaking weight-ties according to the numbers.This assumption ensures that the minimum spanning tree is unique. We will also assume that the input graph has no isolated vertices (vertices without incident edges).

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.

3.1.1 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 (without proof). We need a little terminology. Let G be a graph with weighted edges. We denote by $w(x, y)$ the weight of the 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 et al. [6] (page 1188 in that paper describes the changes needed in the algorithm).

Lemma 3.1. *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.*

3.1.2 The Algorithm

The minimum spanning forest algorithm intermeshes steps of Borůvka's algorithm, called Borůvka steps, with random-sampling steps. Each Boriivka 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 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 from the contracted graph.

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'.

3.1.3 Correctness

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)**.

3.1.4 Analysis of the Algorithm

We will show that the expected running time of the algorithm is linear, by applying **Lemma 3.1** and the linearity of expectations.

If we consider a single invocation of the algorithm, the total time spent in **Steps (l)-(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 straight-forward graph-algorithm techniques, and **Step (2)** takes linear time using the modified Dixon-Rauch-Tarjan verification algorithm [6] as noted in section **3.1.1**.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.

We suppose the algorithm is initially applied to a graph with n vertices and m edges. Since the graph contains no isolated vertices, $m > n/2$. Each invocation of the algorithm generates at most two recursive subproblems. We now consider the entire binary tree of recursive subproblems. The root is the initial problem. For a particular problem, we call the first recursive subproblem, occurring in **Step (2)**, the *left child* of the parent problem, and the second recursive subproblem, occurring 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 $\Sigma^{\infty}_{d=0} 2^d n/4^d = 2n$ vertices total in the original problem and all subproblems.

Theorem 3.1. *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. We consider a parent problem on a graph of X edges, and we let Y to 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 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 $\Sigma^{\infty}_{t=0} k/2^t = 2k$.

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 3.1**, 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 $\Sigma^{\infty}_{d=1} 2^{d-1} n/4^d = n/2$, the expected number of edges in the original problem and all subproblems is at most $2m + n$. Since n is at most $2m$ for a graph with no isolated vertices, the algorithm runs in expected time $O(m)$.

The worst case runtime is equivalent to that of Borůvka's algorithm which is $O(\min\{n^2, m\log n\})$. This occurs if all edges are added to either the left or right subproblem on each invocation.

We can further 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 3.1** and using a Chernoff bound [1, 5, 14].

4 Conclusion

In this report, we have seen that the algorithm given be Karger et al. runs in expected time $O(m)$ for a graph with m edges and n vertices, and we have also noted that in the worst case its runtime is $O(\min\{n^2, m\log n\})$ which occurs with an exponentially small probability. Hence in practical cases we can expect the algorithm to run in linear time except for a small fraction of cases.

Minimum spanning trees (MST) finds its application in a range of fields. There have been quite a lot of algorithms for solving the MST problem, however, as practical problems are often quite large (road networks sometimes have billions of edges), performance is a key factor. Hence several researchers have been trying to find more computationally-efficient algorithms. One of them is the algorithm given by Bernard Chazelle in 2000 which is the fastest non-randomized comparison-based algorithm (with known complexity) till date. In addition to providing a new complexity bound, Chazelle's work also contributes largely to the introduction of a non greedy approach to matroid optimization, which proves useful beyond minimum spanning trees.

It remains an open problem whether a linear-time algorithm exists for finding a minimum spanning tree. Thus a trivial lower bound for the MST problem is $\Omega(m)$; and the best upper bound is $O(m\alpha(m, n))$.

5 References

[1] ALON, N., AND SPENCER, J. H. 1992. The Probabilistic Method. Wiley, New York, p. 223.

[2] BORŮVKA, OTAKAR (1926). "O jistém problému minimálním" [About a certain minimal problem]. *Práce Mor. Přírodověd. Spol. V Brně III* (in Czech and German). 3: 37–58.

[3] CHAZELLE, BERNARD 2000. A minimum spanning tree algorithm with inverse-Ackermann type complexity. Journal of the ACM, 47 (6): 1028–1047.

[4] CHERITON, D. AND TARJAN, R. E. 1976. Finding minimum spanning trees. SIAM J. Comput., 5:724–742.

[5] CHERNOFF, H. 1952. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Stat. 23, 493–509.

[6] DIXON, B., RAUCH, M., AND TARJAN, R. E. 1992. Verification and sensitivity analysis of minimum spanning trees in linear time. SIAM J. Comput. 21, 1184– 1192.

[7] FREDMAN, M. L., AND TARJAN, R. E. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34, 596–615.

[8] FREDMAN, M. L., AND WILLARD, D. E. 1993. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comput. Syst. Sci. 48, 424–436.

[9] GABOW, H. N., GALIL, Z., SPENCER, T., AND TARJAN, R. E. 1986. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica 6, 109–122.

[10] KARGER, D. R., KLEIN, P. N., AND TARJAN, R. E. 1995. A randomized linear-time algorithm to find minimum spanning trees. J. ACM 42, 321–328.

[11] KRUSKAL, J. B. 1956. On the shortest spanning subtree of a graph and the travelling salesman problem. Proc. Amer. Math. Soc., 7:48–50.

[12] PETTIE, S., RAMACHANDRAN V. 2002. An optimal minimum spanning tree algorithm. J. ACM, 49:16–34

[13] PRIM., R. C. 1957. Shortest connection networks and some generalizations. Bell. Syst. Tech. J., 36:1389–1401.

[14] RAGHAVAN, P. 1990. Lecture notes on randomized algorithms. Res. Rep. RC 15340 (#68237). Computer Science/Mathematics. IBM Research Division, T. J. Watson Research Center, Yorktown Heights, N. Y., p. 54

[15] YAO, A. 1975. An O(|E| log log |V |) algorithm for finding minimum spanning trees. Inf. Process. Lett., 4:21–23.