Randomized Minimum Spanning Trees

Kent Quanrud

September 23, 2020

These notes are (clearly) a work in progress.

1 Introduction

Which is harder, computing the minimum spanning tree over m edges or sorting n numbers?

1.1 Computing a minimum spanning tree

Let $G = (V, E)$ be a weighted graph.

1. $O(m \log n)$ – "Borůvka's algorithm"
2. $O(m + n \log n)$ – Fibonacci heaps

Call an edge "light" if it is in the MST and "heavy" if it is not. Both of the above algorithms are based on the following sufficient condition for a light edge.

Lemma 1. *If e is the minimum weight edge in a cut $C = \partial(S)$, then e is light.*

We point out that there is also a simple way to identify heavy edges.

Lemma 2. *If e is the maximum weight edge in a cycle, then e is heavy.*

The second running time is remarkable because it is faster then sorting m numbers.

> *Which is harder, computing the minimum spanning tree over $O(n)$ edges and vertices or sorting n numbers?*

1.2 Verifying a spanning tree

Theorem 3 (Dixon, Rauch, and Tarjan [1])**.** *In $O(m + n)$ time, one can verify if a set of edges F is a minimum spanning forest of G.*

Question.

Can one compute an MST as fast as one can verify it?

Before proceeding, we mention that the verification algorithm above can be extended to not only decide if a set of edges is the MST, but also find the "violating edges". Given a forest F and an edge e, we say that e is **heavy w/r/t *F*** if e is heavy in $F + e$.

Theorem 4. *Let $G = (V, E)$ be a weighted, undirected graph and let $F \subseteq E$ be a forest. In $O(m+n)$ time, one can compute the subset of edges that are heavy w/r/t F.*

2 High-level overview

Let $G = (V, E)$ be an undirected graph with edge weights $w \in \mathbb{R}^E$. Without loss of generality we may assume that all edge weights are distinct.

Definition 5. *Let $E' \subset E$ be a subset of edges. We say that e is* **heavy w/r/t F** *if e is heavy in $F + e$.*

Karger, Klein, and Tarjan [2] proposed the following randomized algorithm for MST.
Consider the following randomized algorithm..

1. Let $E' \subset E$ randomly sample each edge with probability 1/2.
2. Remove all edges that are heavy w/r/t E'.

Since heavy edges are never in the MST, this algorithm will eventually converge to a solution. Is it fast?

Consider one iteration. Sampling E' takes linear time. To identify edges that are heavy w/r/t E', however, might seem difficult. Recall that the minimum spanning forest F' over E' allows us to quickly identify heavy edges: an edge e is heavy w/r/t E' iff e is heavy w/r/t F'. How do we quickly find F? Recursively!

To make the recursion work in our favor, it is helpful to decrease the number of vertices in the graph slightly before recursing. Consider now the following algorithm, given by Karger, Klein, and Tarjan [2].

1. Run Borůvka's algorithm for two iterations.
2. Let $E' \subset E$ randomly sample each edge with probability 1/2.
3. Let $F' \subseteq E'$ be the minimum spanning forest of E', obtained recursively.
4. Remove all edge that are heavy w/r/t F', and repeat.

Lemma 6. *The expected number of F-light edges is $2n$.*

Proof. The proof is very cute. Imagine building a minimum spanning tree greedily over E, except with the following alteration. Whenever we are about to add an edge to our spanning tree, flip a coin. If the coin comes up heads, then add the edge to our spanning tree. If not, then skip it. This random process is the exact same as flipping all the coin tosses first to assemble E', and then finding the minimum spanning tree in E'.

The proof is now just 3 simple observations.

1. *The number of F-light edges is exactly the total number of coin flips.*
2. *The total number of heads is at most $n - 1$.*
3. *The expected number of fair coin tosses until $n - 1$ heads is $2n - 1$.*

■

2.1 Running time analysis

Consider the recursion tree, with "left children" corresponding to recursing in step 3, and "right children" corresponding to repeating after step 4. The running time analysis is broken down as follows.

Lemma 7. *The running time is proportional to the number of edges over all the subproblems.*

Lemma 8. *If a problem has m edges, then the left subproblem has $m/2$ edges in expectation.*

Lemma 9. *If a subproblem has m edges, then the total expected number of edges over itself and all recursively left subproblems is $2m$.*

Lemma 10. *The running time is proportional to the original number of edges plus the number of edges over all right subproblems.*

Lemma 11. *The expected number of edges in a right subproblem is twice the number of vertices.*

Lemma 12. *The running time is proportional to the original number of edges plus the number of vertices in all right subproblems.*

Lemma 13. *The total number of vertices in all right subproblems is $n/2$.*

3 Exercises

Exercise 1. Design a "data structure free" randomized linear time MST algorithm that does not require a linear time verifier such as [1] as a black box.

References

[1] Brandon Dixon, Monika Rauch, and Robert Endre Tarjan. "Verification and Sensitivity Analysis of Minimum Spanning Trees in Linear Time". In: *SIAM J. Comput.* 21.6 (1992), pp. 1184–1192.

[2] David R. Karger, Philip N. Klein, and Robert Endre Tarjan. "A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees". In: *J. ACM* 42.2 (1995), pp. 321–328.