Randomized Algorithm for Minimum Spanning Trees

by Nabeel Akhtar

Nov. 2013 - CS537 Randomized Algorithms - Prof. Steve Homer

1 Problem Definition

Given a undirected and weighted graph $G = (E, V, w)$, E is the set of edges, V is the set of vertices, w is the weight function. Construct a minimum spanning tree $G' = (E', V, w)$ so that G' is connected, and $\sum_{e \in E'} w(e)$ is minimum (that is, total weight is minimum).

2 Minimum Spanning Forest

A forest F is an acyclic subgraph of G that consists of a collection of disjoint tree in G. If the graph G is not connected, a MST does not exist and we generalize the notion of minimum spaning tree to that of the minimum spanning forest.

3 Properties of MST and their proof

3.1 Number of Edges

The MST of G will have $n-1$ edges if $|V| = n$.

3.2 Possible multiplicity

There may be several minimum spanning trees of the same weight having a minimum number of edges e.g. Graph with all edges of weight 1. Any tree of such graph is a minimum spanning tree.

3.3 Uniqueness Property

If each edge has a distinct weight then there will be only one, unique minimum spanning tree. Adding an edge to a tree will form a loop. Removing an edge of a tree will result in two disconnected subtrees.

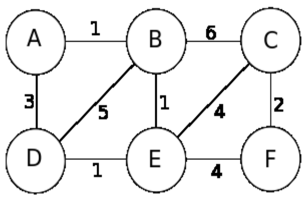
A proof of uniqueness by contradiction is as follows

1. Suppose there are two different MSTs A and B.
2. Let e_1 be the edge of least weight that is in one of the MSTs and not the other. Without loss of generality, assume e_1 is in A but not in B.
3. As B is a MST, $\{e1\} \bigcup B$ must contain a cycle C.
4. Then C has an edge e2 whose weight is greater than the weight of e1, since all edges in B with less weight are in A by the choice of e1.
5. Replacing e2 with e1 in B yields a spanning tree with a smaller weight.
6. This contradicts the assumption that B is a MST.

3.4 Cut Property

For any cut C in the graph, if the weight of an edge e of C is strictly smaller than the weights of all other edges of C, then this edge belongs to all MSTs of the graph.

Proof by assuming contrary



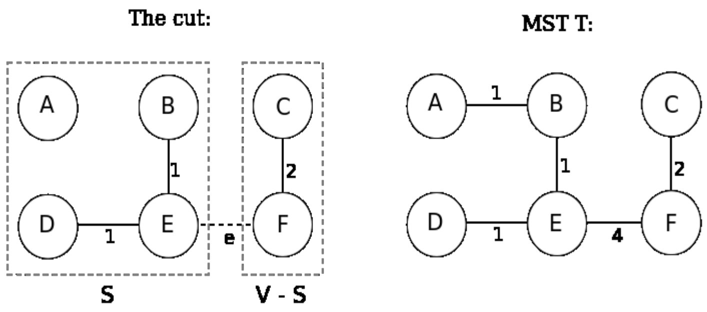


Figure 1.

1. In the figure above, make edge BC (weight 6) part of the MST T instead of edge e (weight 4).
2. Adding e to T will produce a cycle, while replacing BC with e would produce MST of smaller weight.
3. Thus, a tree containing BC is not a MST, a contradiction that violates our assumption.

3.5 Cycle Property

Given G, and a cycle C in G. The heaviest edge e on C is not in the MST of G.

Proof: Assuming the contrary

- If e belongs to an MST T1, then deleting e will break T1 into two subtrees with the two ends of e in different subtrees.
- The remainder of C reconnects the subtrees, hence there is an edge f of C with ends in different subtrees, i.e., it reconnects the subtrees into a tree T2 with weight less than that of T1, because the weight of f is less than the weight of e.

3.6 Minimum Cost Edge

If the edge of a graph with the minimum cost e is unique, then this edge is included in any MST.

If e was not included in the MST, removing any of the (larger cost) edges in the cycle formed after adding e to the MST, would yield a spanning tree of smaller weight.

4 Deterministic Algorithm - Boruvka's

Lemma 1. *Let $v \in V$ be any vertex in G, the MST of G must contain edge (v, w) that is the minimum weight edge incident on v.*

Proof. Consider the cut defined by that vertex v and all other vertices. Use the Cut Property. □

Boruvka's algorithm is to concontract simultaneously the minimum weight edges incident on each vertex, and form a new contracted graph with contracted vertices. Repeat this step until a single vertex. Or in some other text, starting from n blue trees (consisting of n vertices in V), select a minimum weight edge incident to each blue tree, and color all selected edges blue. Repeat until there is only one single blue tree.

Algorithm

1. start from n vertices as n individual blue trees
2. for every blue tree, mark the edge with minimum weight incident to it as blue
3. repeat until a single blue tree
4. output all blue edges, and that is the MST

Note. The coloring phase is called Boruvka's Step/Phase.

At each step, no of bluse trees reduces by al least factor of two, so we need $O(\log n)$ steps to get to a single tree. And in every step, we need to scan at most $O(m)$edges. Therefore, the overall complexity is $O(m \log n)$.

5 Heavy Edges and MST Verification

We introduce some symbols. Given any forest F of G, $w_F(u, v)$ denotes the edge with maximum weight among all edges on the path in F from u to v. If there is no path, $w_F(u, v) = \infty$. $w(u, v)$ is the weight of edge (u, v) of G. Let $p(u, v) = \{e | e \in F, e \text{ is on the path from } u \text{ to } v\}$

Definition. *An edge $(u, v) \in E$ is F-heavy if $w(u, v) > w_F(u, v)$. Otherwise, it is F-light.*

Note. F-heavy edges can't be in F. Because every edge in F satisfy $w(e \in p(u, v)) \leqslant w_F(u, v)$.

Lemma 2. *Let F be any forest in G, if an edge (u, v) is F-heavy, then it doesn't lie in the MST of G. The converse is not true.*

Proof. We can prove it using contradiction. Consider to replace $p(u, v)$ with (u, v). □

5.1 MST Verification

This is not our focus, but since the randomized algorithm needs a linear time verification, we briefly introduce some backgroud.

An important property of some of these linear-time verification algorithms is that when input tree F is not a MST, they produce a list of edges in G any of which can be used to improve F.

Basic Idea:

The basic idea of verification is for every edge $e \notin F$, test if it is F-heavy. The algorithm adopts Komlos's algorithm together with some specially designed data structures to achieve linear time. And we are not going to cover this.

What we should know is we can identify F-heavy edges in $O(m + n)$ time.

6 Preliminary

6.1 Negative Binomial Distribution

Suppose X_1, X_2, $\cdots$, X_n are independent random variables with geometric distribution with parameter p. Then $X = \sum_i X_i$ stands for the number of coin flips we need to get n heads. It is a random variable with negative binomial distribution with parameter n and p. And we have

$$E[X] = \frac{n}{p}$$

6.2 Stochastic Dominance

A random variable X stochastically dominates another random variable Y if, $\forall z \in \mathbb{R}, \Pr(X > z) \geqslant \Pr(Y > z)$. And if X stochastically dominates Y we have

$$E[X] \geqslant E[Y]$$

And for negative binomial r.v. $X \sim (n_1, p)$ and $Y \sim (n_2, p)$, if $n_1 \geqslant n_2$, X stochastically dominates Y.

Proof. (Hint)

$$\Pr(X > z) = 1 - \Pr(X \leqslant z) = 1 - \sum_{i=n_1}^{z} \Pr(X = i)$$

$$\Pr(Y > z) = 1 - \Pr(Y \leqslant z) = 1 - \sum_{i=n_2}^{z} \Pr(Y = i)$$

□

6.3 Random Graph

Given a graph G, $G(p)$ is a random graph generated by individually including each edge of G with probability p. Expected number of edges is mp.

Note. $G(p)$ may not be connected.

7 Linear Time MST

In Boruvka's MST, we reduce the number of vertices in each phase. Here, we use randomness to reduce the number of edges in each phase.

And, the basic idea comes from the fact that for reasonably large p, the minimum spanning forest of $G(p)$ is actually a very good approximation of the MST of G, which we will prove in the following sections.

7.1 Random Sampling Lemma

Here, we are dealing with $G = (V, E)$, $G(p) = (V, E(p))$, and F (the MSF of $G(p)$). And we are going to prove that for such F, F-light edges in G is bound by n/p. ($|V| = n$).

Note. The bound is used for complexity analysis, but the F is used for removing F-heavy edges in the algorithm.

Note. Once $G(p)$ is fixed, F is fixed. All edges in F is F-light, all remaining edges in $G(p)$ is F-heavy, but other edges in G could be F-heavy or F-light.

A F-light edge is included in F if it has been chosen by $G(p)$ during construction of $G(p)$. Therefore, among all F-light edges in G, each such edge is included in F with probability p. Suppose $|F| = s$. Then include one such edge in F is like flipping p-coin multiple times until we get a head. So including all s such edges in F is to flip p-coin until we get s head. Therefore the total number of F-light edges is represented by the total number of coin flips, which is a r.v. with negative binomial distribution $X \sim (s, p)$.

Furthermore, s is bounded by $n - 1$ by the definition of MSF. We can use $Y \sim (n, p)$ which stochastic dominates X to bound it.

Finally we get $E[Y] = n/p$, which is the expected number of F-light edges in G. So we have this.

Lemma 3. *Given $G, G(p)$ and the minimum spanning forest F of $G(p)$, $|V| = n$, then the number of F-light edges in G is bounded by n/p.*

7.2 The Algorithm

Algorithm

Input: $G = (V, E, w), |V| = n, |E| = m$
Output: MSF of G

1. If G is empty, return empty forest.
2. Perform three Boruvka's step on G to get G_1 with at most $n/8$ vertices. Let C be those edges being colored blue in these three steps. Note after coloring, we are contracting each components into a signle node.
3. Let $G_2 = G_1(p)$ with $p = 0.5$.
4. Recursively running this algorithm on G_2 to get the minimum spanning forest F_2 of G_2.
5. Running linear time verification algorithm on G_1 to remove F_2-heavy edges, and get G_3
6. Recursively running this algorithm on G_3 to get the minimum spanning forest F_3 of G_3
7. Return $C \cup F_3$.

7.3 Analysis

Let $T(n, m)$ denote the running time on a graph with n vertices and m edges.

1. Step 2 runs three Boruvka's steps. Coloring takes $O(m)$ time while contracting takes $O(n)$ time. Therefore this step takes $O(n + m)$ time.
2. Step 3 is a random sampling, which reads $O(n)$ vertices, and samples $O(m)$ edges. Takes $O(n + m)$ time.
3. Step 4 takes $T(n/8, m/2)$ expected running time.
4. Step 5 we use linear verification algorithm that takes $O(n + m)$ time. And G_3 has expected $n/8$ vertices and $\frac{n}{8} \times 2 = n/4$ edges by the sampling lemma.
5. Step 6 takes $T(n/8, n/4)$ time.
6. Step 7 takes $O(n)$ time.

Therefore,

$$T(n, m) \leqslant T(n/8, m/2) + T(n/8, n/2) + c \times O(m + n)$$

Solve it and we get *2c(n+m)* which implies that expected running time is $O(n + m)$.

Note 4. Expected running time is O(n+m)

Worst-case behaviour:

Worst case running time of the minimum-spanning forest algorithm is *O(min{n*^{2}*,m log n})*, same as the bound for the Boruvka's algorithm.

Proof. Total running time is bounded by a contant factor times the total number of edges in the original problem. Thus, estimate of the total number of edges can give us the running time.

There are no multiple edges in any subproblem. A subproblem at depth d contains at most $\mathbf{(n/4^d)^2/2}$ edges. Summing over all subproblems gives an O(n^2) bound on the total number of edges.

Total number of edges in all the subproblems at any single recursive depth d is at most m. Since number of different depth is O(log n), the total number of edges in all the recursive subproblems is O(m log n). □

Running time is O(n+m) with high probability:

Algorithm runs in O(n+m) with probabiliy 1 - exp(-$\Omega(m)$). This can be proven using Lemma 3 and Chernoff bound. We will cover Chernoff bound in the next class so we are not giving the proff here.

8 Related Problems:

1. **K-minimum spanning tree**: A tree of minimum cost that has exactly k vertices and forms a subgraph of a larger graph. It is also called the k-MST or edge-weighted k-cardinality tree.
2. **Distributed minimum spanning tree:** Problem involves the construction of a minimum spanning tree by a distributed algorithm, in a network where nodes communicate by message passing. It is radically different from the classical sequential problem, although the most basic approach resembles Borůvka's algorithm.
3. C**apacitated minimum spanning tree:** The capacity constraint ensures that all subtrees (maximal subgraphs connected to the root by a single edge) incident on the root node r have no more than c nodes.
4. **Degree constrained minimum spanning tree**: Is a minimum spanning tree in with each vertex is connected to no more than d other vertices, for some given number d.
5. **Directed Graph MST:** The minimum spanning tree problem is called the Arborescence problem and can be solved in quadratic time using the Chu–Liu/Edmonds algorithm.
6. **Dynamic MST:** Problem concerns the update of a previously computed MST after an edge weight change in the original graph or the insertion/deletion of a vertex.