# A Randomized Algorithm to Find Minimum Spanning Tree

Yvette (I-Ting) Tsai

Dec 5, 2013

In this report, we discuss a linear time algorithm to find a minimum spanning tree in a connected graph with weights. Compared to the log-linear deterministic algorithm to find a minimum spanning tree in a given graph, we show that by introducing randomness we can solve the problem in linear time.

## 1 Motivation

Imagine yourself as a CEO of a cable company. You want to enable all households in a city to have the ability to watch television via your cables. As a smart businessman, you want to maximize your profit by spending the minimal amount of money to set up cables yet covering all households in the city. Similarly, consider yourself as a governor of a bunch of islands where you plan to build bridges to connect all islands such that one can reach any other island from all islands. However, your budget is limit. What strategy would you use to solve these problems?

The previous two scenarios are some common problems that can be solved by transforming those problems into finding a minimum spanning tree. In the first scenario, consider each household as a vertex and the cost of setting up a cable between two household as the weight of the edge. To find the best way of setting up the cable is simply to solve the problem of finding a minimum spanning tree among the graph we just constructed. Additionally, the ability to efficiently solve the minimum spanning tree problem also benefits other areas such as clustering data, circuit design, approximation for traveling salesman problem, and taxonomy.

## 2 Problem Definition

We formally define the minimum spanning tree problem as follow. Given an undirected graph $G = (V, E)$, where $V$ is the set of vertices, $E$ is the set of edges, and for each edge $(u, v) \in E$, there is a weight $w(u, v)$ specifying the cost to connect $u$ and $v$[1]. Our goal is to find an acyclic subset $T \subset E$ such that $T$ connects all of the vertices and whose total weight $w(T)$ is minimized.

$$T = \text{argmin}_{\ T \subseteq E}\ \ w(T) = \sum_{(u,v) \in T} w(u, v)$$

[1]For simplicity, we assume all weights $w$ are distinct. When encounter non-distinct weight graph, we can always slightly alter the weights to make them all distinct

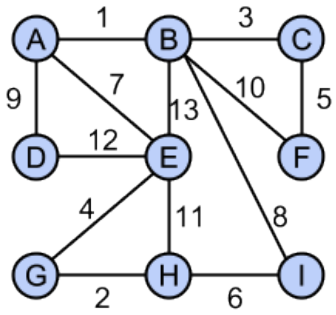


Figure 1: Graph $G$

Since $T$ is acyclic, it connects all of the vertices and minimizes the total weights, it must be a tree, which we call a minimum spanning tree (MST). The problem of finding such tree $T$ for a graph $G$ is known as **minimum spanning tree problem**.

# 3 Non-Randomized Algorithms

We show three deterministic algorithms to find a MST. The first two algorithms-Primes and Kruskals algorithms-are greedy algorithms that have been widely used to solve the problem. The third algorithm, Borůvka's algorithm finds MST by contracting edges. With each algorithm, we first describe how the algorithm works, then run an example with the algorithm, and finally analyze the runtime of such algorithm.

## 3.1 Kruskal's Algorithm

The main idea behinds Kruskal's algorithm is gradually grows the tree by adding the least weighted edge which connects any two unconnected trees. *Algorithm 1* following this paragraph shows how Kruskal's algorithm works. It takes an input graph $G = (V, E)$ where $|V| = n, |E| = m$ and a set of weights $w$ that corresponds to each edge $e \in E$. The MST $T$ starts out as an empty tree, and each vertex is itself a tree. The **for-loop** in line 6-10 examines each edge $(u, v)$ in order of weight, from lowest to highest. At each time, we check whether endpoints $u$ and $v$ belongs to the same tree. If they belong to the same tree, we discard such edge; otherwise we add it to $T$ and merge $u$ and $v$. The correctness of Kruskal's algorithm can be proved by induction and cut-property of minimum spanning tree [2].

**Example:** Consider *Figure 1* as the graph $G$ to run Kruskal's algorithm. We create MST by adding edges into $T$ in the following order. $T = \{e_{AB}, e_{GH}, e_{BC}, e_{EG}, e_{CF}, e_{HI}, e_{AE}, e_{AD}\}$ with total weight $w(T) = 37$.

[2]Since the proof for correctness of existing algorithm is not the focus of this report, we will skip the proof here. Please refer to [2, 8] for the complete proof.

**Algorithm 1 Kruskal_MST(G, w)**

```
1: T = ∅
2: for each v ∈ V do
3:     create a set {v}
4: end for
5: sort all edges e ∈ E in nondecreasing order by its weight w
6: for each edge (u, v) ∈ E taken in nondecreasing order do
7:     if {u} ≠ {v} then
8:         T = T ⋃ {(u, v)}
9:         union {u} and {v}
10:    end if
11: end for
12: return T
```

**Runtime:** Sorting all edges takes $O(m\ logm)$ time. In the worst case, we have to check through all edges in the graph to complete building MST $T$ which takes $O(m)$ time. Overall, Kruskal's algorithm takes $O(m\ logm) = O(m\ logn)$[3].

## 3.2 Prim's Algorithm

Prim's algorithm has a similar flavor to Dijkstra's algorithm for finding shortest paths in a graph. The algorithm gradually connects each vertex through the least weighted edge until all vertices are included in the tree. *Algorithm 2* following the paragraph shows how Prim's algorithm works. It takes an input graph $G = (V, E)$ where $|V| = n, |E| = m$ and a set of weights $w$ that corresponds to each edge $e \in E$. The MST $T$ starts out as an empty tree and $Visited$ starts out with a random vertex $v$. In line 3-7, while there is a vertex that is not visited, we keep selecting the lightest weight edge which connect the visited vertices to unvisited vertices. After all vertices have been visited, we stop and return $T$. Similarly, the correctness of Prim's algorithm can be proved by cut-property of minimum spanning trees [4].

**Algorithm 2 Prim_MST(G, w)**

```
1: T = ∅
2: Visited = {v} (a random vertex v ∈ V)
3: while Visited ≠ V do
4:     find the lightest edge (u, v) such that u ∈ Visited and v ∈ V \ Visited
5:     add {v} to Visited
6:     add edge (u, v) to T
7: end while
8: return T
```

**Example:** Again, consider *Figure 1* to run Prim's algorithm. Assuming we start with vertex $A$. Then we create MST by adding edges into $T$ in the following order. $T = \{e_{AB}, e_{BC}, e_{CF}, e_{AE}, e_{EG}, e_{GH}, e_{HI}, e_{AD}\}$

[3] In any given undirected graph, $|E|$ is at most $|V|^2$

[4] The complete proof of correctness of Prim's algorithm can be found [2, 11].

with total weight $w(T) = 37$.

**Runtime:** Prim's algorithm checks through all edges and search for the least weighted edge of a vertex. Assuming we use binary heap and adjacency list to store our graph $G$, Prim's algorithm runs in $O((m+n)logn) = O(m\ logn)$ time.

## 3.3 Borůvka's Algorithm

Unlike Kruskal and Prim, Borůvka's algorithm contracts the minimum weighted edge on each vertex simultaneously and then form a new contracted graph. Contracting step is repeated until only one vertex is left in the graph. It takes an input graph $G = (V, E)$ where $|V| = n, |E| = m$ and a set of weights $w$ that corresponds to each edge $e \in E$. MST $T$ starts out as an empty set and each vertex is an component, hence the initial size of $Comp$ is the number of vertices in graph $G$. In line 3-9, we repeatedly contract vertices via all its minimum weighted edges. We stop and return the tree $T$ when the size of $Comp$ is one. [5]

**Algorithm 3 Borůvka_MST(G, w)**

1: $T = \emptyset$
2: $Comp = \{\{v_1\}, \{v_2\}, ..., \{v_n\}\}$ (where $n$ is the size of $V$ and $v_i \in V$)
3: **while** size of $Comp \neq 1$ **do**
4: **for** each set $c \in Comp$ **do**
5: find the lowest weight edge $e$ from $c$ to $Comp \setminus c$
6: add edge $e$ to $T$
7: **end for**
8: contract any sets that connect via edges in $T$
9: **end while**
10: return $T$

**Example:** We consider *Figure 1* again to run Borůvka's algorithm. We show the component at each stage and edges being add into MST $T$ at each stage.

- Initial: $Comp = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}, \{G\}, \{H\}, \{I\}\}$, and $T = \emptyset$
- $1^{st}$ Iteration of while-loop: $Comp = \{\{A, B, C, D, F\}, \{E, G, H, I\}\}$, and $T = \{e_{AB}, e_{BC}, e_{AD}, e_{CF}, e_{EG}, e_{GH}, e_{HI}\}$
- $2^{nd}$ Iteration of while-loop: $Comp = \{\{A, B, C, D, F, E, G, H, I\}\}$, and $T = \{e_{AB}, e_{BC}, e_{AD}, e_{CF}, e_{EG}, e_{GH}, e_{HI}, e_{AE}\}$, with weight $w(T) = 37$.

**Runtime:** At each iteration of the while-loop, i.e. Borůvka step, the size of $Comp$ is reduced at least with a factor of two since for each set in $Comp$, there has to be at least one chosen edge. In the worst case, two distinct sets choose the same edge. Therefore it takes $O(logn)$ step to reach $Comp$

[5] The complete proof of correctness of Borůvka's algorithm can be found [6].

with size one and then at each step, we consider at most $m$ edges. Overall, Borůvka's algorithm runs in $O(m\ logn)$ time.

# 4 Randomized Algorithm

Here we present a randomized algorithm to find MST that is introduced by Karger, Klein, and Tarjan. The high level picture of this algorithm is to reduce the number of vertices and edges on each step. It reduces the number of vertices by Borůvka's step (i.e. step in line 4-8 from *Algorithm 3*), and reduces the number of edges via random sampling.

## 4.1 Overview Algorithm

The overview of the algorithm `Random_MST` [6] is as follow,

`Random_MST(G):`

1. Run two steps of Borůvka's step to reduce vertices by the factor of 4. Let the resulting contracted graph be $G_0$ and the set of contracted edges be $F_0$
2. For each edge $e$ in $G$, add $e$ in the subgraph $G_1$ with probability $p$. Run `Random_MST(`$G_1$`)` and obtain its minimum spanning tree $F_1$.
3. For each edge $e$ in $G$, if adding $e$ to $F_1$ makes $e$ the heaviest edge in the cycle, remove $e$ from $G$. Let the remaining graph be $G_2$.
4. Run `Random_MST(`$G_2$`)` and obtain its minimum spanning tree $F_2$.
5. Return $F_2 \cup F_0$

The central idea behinds the algorithm and the correctness of `Random_MST` relies on the following important points stated by Karger, Klein, and Tarjan,

**Lemma:** *If an edge $(u, v)$ is F-heavy in graph G, it cannot be part of the MST of G. The converse is not true. An edge $(u, v)$ is F-heavy if adding it to F creates a cycle and edge $(u, v)$ is the heaviest edge in that cycle. Otherwise edge $(u, v)$ is F-light*

This lemma can be proved easily by the cycle property of a minimum spanning tree [7]. With this lemma, we see that none of the edges we discard in step 2 are edges in the MST of $G$, therefore the MST must be some subset of the remaining edges (assuming that we have a linear time verification algorithm to help verifying *F-heavy* in a graph $G$) [8].

**Theorem:** *Let $G$ be a graph with $n$ vertices, and let $G'$ be a subgraph obtained by including each edge independently with probability $p$, and let $F$ be the minimum spanning forest of $G$. The*

[6] This algorithm actually solve the more general case of finding minimum spanning forest.

[7] Please refer to [4] for the complete proof

[8] In [2] they assume the existence of such linear time verification algorithm. For more detailed on the verification algorithm, please refer to [5, 7].

*expected number of F-light edges in G is at most $n/p$.*

The theorem can be easily proved by consider the following scenario, the expected number of coin flips in order to obtain $n$ heads where the coin turn up head with probability $p$. To obtain one head, the expected number of coin flips is $1/p$. From linearity of expectation, the expected number of coin flips to obtain $n$ heads is simply $\sum_{i=1}^{n} 1/p = \ n/p$. Given this, because a MST of $G$ can only have $n-1$ edges, therefore the number of *F-light* edges in $G$ is at most $n/p$ [9].

### 4.2 Analysis

For a given graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, we represent the running time as $T(m, n)$. The analysis can be further broke down to the following,

1. Two Borůvka's steps run in $O(m)$
2. Recursive call on $G_1$ runs in $T(m_1, n_1)$
3. *F-heavy* edges identification runs in $O(m_1)$
4. Recursive call on $G_2$ runs in $T(m_2, n_2)$
5. Concatenation of edges from in previous steps runs in $O(m)$

The number of edges in sub-graph, $m_1$ and $m_2$ is bounded by $m$, therefore the overall running time is $O(m)$ which only depends on the number edges in graph $G$. In the worst case, where the randomize step does not help speed up the process, this randomized algorithm simply becomes Borůvka's algorithm, which runs in $O(m\ logn)$ time.

### 4.3 Discussion

The randomized algorithm to find minimum spanning tree has the advantage of shorter running time, i.e. it runs in linear running (comparing to log-linear running time for non-randomized algorithm such as Kruskal, Prim, and Borůvka). Moreover, the worst case scenario for the randomized algorithm simply become the same as if running with Borůvka's algorithm. This advantage of decent worst case running time and expected linear running time allow the randomized algorithm to find minimum spanning tree be widely used.

## 5 Reference


1. Akhtar, N. and Wu, H. "Minimum Spanning Tree November 2013.
2. Cormen, T., Leiserson, C., Rivest, R., and Stein, C. "Introduction to Algorithms The MIT Press.
3. Ganesan, V. "Minimum Spanning Trees Algorithms and Applications MIT Course 18.304 Lecture Notes Spring 2013. http://math.mit.edu/ rothvoss/18.304.1PM/.../1-Varun-MSTPresentation.pdf


[9] Please refer to [1, 4] for the complete proof.

4. Karger, D., Klein, P. and Tarjan, R. “A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees pp 321-328 Journal of the Association for Computing Machinery. Vol 42, No 2. March 1995.

5. King, V. “A Simpler Minimum Spanning Tree Verification Algorithm.” Algorithmica (1997) 18: pp263-270

6. Kleinberg, J. and Tardos, E. ”Algorithm Design” Addison-Wesley. March 26, 2005

7. Komlós, J. “Linear Verification for Spanning Trees” Combinatorica 5: pp57-65

8. Kruskal, J. “On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem.” American Mathematical Society, Vol 7, No. 1 (Feb, 1956), pp. 4850

9. Mitzenmacher, M. and Upfal, E. “Probability and Computing - Randomized Algorithms and Probabilistic Analysis. Cambridge University Press.

10. Murali, T. “Applications of Minimum Spanning Trees. VT Course CS5114 Lecture Notes. February 14, 2013. http://courses.cs.vt.edu/cs5114/spring2013/lectures/lecture08-mst-applications.pdf

11. Prim, R. C. “Shortest connection networks and some generalizations” In: Bell System Technical Journal, 36 (1957), pp. 13891401

12. Wu, B. and Chao, K. “Spanning Trees and Optimization Problems Chapman and Hall/CRC Press, USA. 2004