Minimum Spanning Trees

Yan Liu
LDCSEE
West Virginia University,
Morgantown, WV
{yanliu@csee.wvu.edu}

1 Statement of Problem

Let $\mathbf{G} =< \mathbf{V}, \mathbf{E} >$ *be a connected graph with real-valued edge weights:* $w : E \to R$*, having* n *vertices and* m *edges. A spanning tree in* $\mathbf{G}$ *is an acyclic subgraph of* $\mathbf{G}$ *that includes every vertex of* $\mathbf{G}$ *and is connected; every spanning tree has exactly* $n-1$ *edges. A minimum spanning tree (***MST***) is a spanning tree of minimum weight which is defined to be the sum of the weights of all its edges. Our problem is to find the* **MST** *of* $\mathbf{G}$*.*

2 Minimum Spanning Forests

Some algorithms we present here will deal with subgraphs that are not necessarily connected. Thus, we introduce the definition of minimum spanning forest.

1. A forest $\mathbf{F}$ is an acyclic subgraph of $\mathbf{G}$ that consists of a collection of disjoint trees in $\mathbf{G}$.
2. A spanning forest $\mathbf{F}$ is a forest whose trees are spanning trees for the connected components of the graph $\mathbf{G}$.
3. A minimum spanning forest $F = (V, E')$ is a subgraph of $G(V, E)$ such that the sum of all the weights $w(e)$ for all the edges e in E' is minimal.

3 Deterministic Algorithms

3.1 Kruskal's Algorithm

Algorithm MST:

Input: $\mathbf{G}(\mathbf{V}, \mathbf{E})$
Output: $\mathbf{T}$, the **MST**

1: $\mathbf{T} \leftarrow$ empty graph
2: **for** $(i = 1$**to** $|V|\ )$ **do**
3: Let e be the minimum weight edge in $\mathbf{G}$ that does not form a cycle with $\mathbf{T}$. $\mathbf{T} \leftarrow \mathbf{T} \bigcup \{\mathbf{e}\}$
4: **end for**

Algorithm 3.1: Kruskal's Algorithm

The only difference between these two algorithms is that Kruskal's Algorithm doesn't require that the edge e to be connected to the evolving tree T, which makes T be a forest other than a tree. The Kruskal's Algorithm

also runs in $O(m \log n)$ time.

3.2 Boruvka's Algorithm

There is another greedy strategy for **MST** called Boruvka's algorithm, which also runs in $O(m \log n)$ time. Later we will show that using randomization in conjunction with the algorithm leads to a linear-time algorithm. Boruvka's algorithm is based on the following lemma.

Lemma: 3.1 *Let $v \in V$ be any vertex in G. The* **MST** *for* **G** *must contain the edge (v, w) that is the minimum weight edge incident on v.*

Proof: *Suppose that (v, w), the minimum weight edge incident on v is not contained in the* **MST** *T of* **G**, *then we must have another edge (v, u) in T such that vertex v can be covered in T. By adding (v, w) into T, we will form a path of cycle in T which passes v. Since (v, w) is the minimum weight edge incident on v, we can remove (v, u) and thus keep (v, w) in T', resulting in that the weights of T' is less than that of T. This reaches the contradiction with that T is the* **MST** *of* **G**. *Therefore, the* **MST** *for* **G** *must contain the edge (v, w).* □

Definition: 3.1 *The basic idea in Boruvka's algorithm is to contract simultaneously the minimum weight edges incident on each of the vertices in* **G**. *In a contraction, only the minimum weight edge needs to be retained out of any set of multiple edges. The process of contracting the minimum weight edge for each vertex in the graph is called the Boruvka phase.*

A good implementation of a Boruvka phase is the following:

1. mark the edges to be contracted;
2. determine the connected components formed by the marked edges;
3. replace each connected component by a single vertex;
4. finally, eliminate the self-loops and multiple edges created by these contractions.

Lemma: 3.2 *The set of edges marked for contraction during a Boruvka phase induces a forest in* **G**.

Proof: *From the steps listed above, we can compute the running time of implementing such a Boruvka phase. Step 1 takes $O(m+n)$ time; in step 2 and step 3 , we may use a Depth-First Search (DFS) [2] to find the connected components and create a new vertex that correspond to each connected component, and associate each vertex with that new vertex, all these take $O(m + n)$ time; finally, step 4 takes $O(m)$ time. Thus, the whole process of a Boruvka phase needs $O(m + n)$ time.* □

Lemma: 3.3 *The set of edges marked for contraction during a Boruvka phase induces a forest in* **G**.

Proof: *By Lemma 3.1, each of the edges marked for contraction must be one of the edges in the* **MST** *of* **G**. *During each Boruka phase, the edges we marked is a subgraph of the* **MST** *of* **G**. *Therefore, these edges might not be connected, but must be acyclic. This makes them form a forest of* **G**. □

We claim that the graph G' obtained from the Boruvka Phase has at most $\frac{n}{2}$ vertices. This is because that each contracted edge can be the minimum incident edge on at most two vertices. The number of marked edges is thus at least $\frac{n}{2}$. Since each vertex chooses exactly one edge to mark, it is easy to verify that each marked edge must eliminate a distinct vertex. The number of edges in G' is no more than m since no new edges are created during this process. By Lemma 3.1, we know that each of the contracted edges must belong to the **MST** of **G**. In fact, the forest induced by the edges marked for contracting is a subgraph of the **MST**.

Lemma: 3.4 *Let G' be the graph obtained from G after a Boruvka phase. The* **MST** *of G is the union of the edges marked for contraction during this phase with the edges in the* **MST** *of G'.*

Proof: *By Lemma 3.3, we know that during each Boruvka phase, the edges we marked is a forest of G, and these edges belong to the* **MST** *of G. Each marked edge eliminate a distinct vertex of* **G**, *thus the uncovered vertices are still remained in G'. The union of the* **MST** *of G' with these marked edges will cover all the vertices with no cycle. Furthermore, since the* **MST** *of G' will also be induced by Boruvka phases, the* **MST** *of G' is also a subgraph of the* **MST** *of G. Therefore, the union of the edges marked for contraction during this phase with the edges in the* **MST** *of G' is the* **MST** *of G.* □

Algorithm MST:

Input: $G(V, E)$
Output: T, the **MST**

```
1: T ← empty graph
2: for each v in V do
3:    Let e be the minimum weight edge in G that is incident on v
4:    T ←T + {e}
5: end for
6: G' ← G with all edges in T contracted
7: T' ← recursively compute the minimum spanning tree of G'
8: return T + T'
```

Algorithm 3.2: Boruvka's Algorithm

Boruvka's algorithm reduces the **MST** problem in an n-vertex graph with m edges to the **MST** problem in an $(\frac{n}{2})$-vertex graph with at most m edges. The time required for the reduction is only $O(m + n)$. The worst case running time is $O(m \log n)$.

4 Heavy Edges and MST Verification

Before describing how randomization can be used to decrease the running time of Boruvka's algorithm, we develop a technical lemma on random sampling of edges from graph **G**.
Let **F** be any forest in graph **G** and consider any pair of vertices $(u, v) \in V$. Let $w_F(u, v)$ denote the maximum weight of any edge along the unique path in **F** from u to v. If there is no path exists, then $w_F(u, v) = \infty$. Note that if an edge (u, v) exists in **G**, the normal weight of this edge is denoted as $w(u, v)$.

Definition: 4.1 *An edge $(u, v) \in E$ is said to be* **F-heavy** if $w(u, v) > w_F(u, v)$. The edge (u, v) is said to be **F-light** if $w(u, v) \leq w_F(u, v)$.

In particular, all the edges in **F** must be **F**-light. An edge (u, v) is **F**-heavy if the forest **F** contains a path from u to v using only edges of weight smaller than that of (u, v) itself.

Lemma: 4.1 *Let F be any forest in graph G. If an edge (u,v) is F-heavy, then it does not lie in the* **MST** *of G. The converse is not true.*

Proof: *By the definition of F-heavy edges, we know that F must contain a path from u to v using the edges of weight smaller than (u,v). Suppose (u,v) belongs to the* **MST** *of G, then by replacing (u,v) with the existed path in F from u to v into the* **MST**, *we can obtain another* **MST** *of G with the smaller weights. Therefore, there is contradiction thus (u,v) cannot lie in the* **MST** *of G. However, F may contain isolated vertex, say u. There is no path existed in F from any other vertex to u. By the definition, $w_F(u, v_i) = \infty$, $v_i \in V_u$ and (u, v_i) is F-light. It is easy to see that for a certain edge not in the* **MST** *of G, it might be F-light. Thus, the converse is not true.* □

A verification algorithm for the **MST** can be viewed as taking as input a tree T in a graph **G**, and checking that the only T-light edges are the edges in T itself. This is equivalent to verifying T is the **MST**. There exists linear-time verification algorithms that can be adapted to identify all **F**-heavy and **F**-light edges with respect to any forest **F**. The performance of these algorithms are summarized in the following theorem.

Theorem: 4.1 *Given a graph G and a forest F, all F-heavy edges in G can be identified in time O(n+m).*

5 Random Sampling for MSTs

In order to reduce the number of edges in the graph, we will use the fact that a random subgraph of **G** has a "similar" minimum spanning tree. To be precise, consider a (random) graph $G(p)$ obtained by independently including each edge of **G** in $G(p)$ with probability p. The graph $G(p)$ has n vertices and expected number of edges mp. Note that there is no guarantee that $G(p)$ is connected.

Let **F** be the minimum spanning tree for $G(p)$. We expect very few edges in **G** to be **F**-light such that **F** would be a good approximation to the **MST** of **G**. This expectation is explained in details in the lemma presented below. First we recall some probability definition and theory.

Definition: 5.1 *Let $X_1, X_2, ..., X_n$ be independent random variables whose common distribution is the geometric distribution with parameter p,. The random variable $X = X_1 + X_2 + ... + X_n$ denotes the number of coin flips needed to obtain n HEADS. The random variable X has the* **negative binomial distribution** *with parameters n and p. The density function for this distribution is defined only for $x = n, n+1, n+2...$:*

$$\mathbf{Pr}[X = x] = \binom{x-1}{n-1} p^n q^{x-n}$$

The Characteristics are: $E[X] = \frac{n}{p}$, $var[X] = \frac{nq}{p^2}$, and $G(z) = (\frac{pz}{(1-qz)})^n$.

Definition: 5.2 *For independent variables X and Y, we say that random variable X* **stochastically dominates** *variable Y if, for all $z \in R$, $Pr[X > z] \geq Pr[Y > z]$.*

Proposition: 5.1 *Let X and Y be random variables with finite expectations. If X stochastically dominates Y, then $E[X] \geq E[Y]$; equality holds if and only if X,Y are identically distributed.*

Lemma: 5.1 *Let X have the negative binomial distribution with parameters n_1 and p, and Y have the negative binomial distribution with parameters n_2 and p. For $n_1 \geq n_2$, X stochastically dominates Y.*

Proof: *Without loss of generality, we assume that z is a positive integer.*

$$Pr[X > z] = 1 - Pr[X \leq z] = 1 - (Pr[X = z] + Pr[X = z-1] + .. + Pr[X = n_1])$$

$$Pr[Y > z] = 1 - Pr[Y \leq z] = 1 - (Pr[Y = z] + Pr[Y = z-1] + .. + Pr[Y = n_2])$$

For $n_1 \geq n_2$,

$$(Pr[X = z] + Pr[X = z-1] + .. + Pr[X = n_1]) \leq (Pr[Y = z] + Pr[Y = z-1] + .. + Pr[Y = n_2]).$$

Therefore,

$$Pr[X > z] \geq Pr[Y > z].$$

□

Lemma: 5.2 *Let F be the minimum spanning forest in the random graph G(p) obtained by independently including each edge of G with probability p. Then the number of F-light edges in G is stochastically dominated by a random variable X that has the negative binomial distribution with parameters n and p. In particular, the expected number of F-light edges in G is at most $\frac{n}{p}$.*

Proof: *Let e_1, e_2 be the edges of G arranged in order of increasing weight. Suppose that we construct G(p) by traversing the list of edges in this order, flipping a coin with probability of HEADS equal to p for each edge, and including an edge e_i in G(p) if the ith coin flip turns up HEAD.*

The minimum spanning forest F for G(p) can be constructed online during this process as following:

1. *Initially F is empty.*

2. *At step i, after the coin is flipped for $e_i = (u, v)$, if e_i is chosen for G(p), then e_i is considered for inclusion in F.*

3. *The edge is added to F if and only if u and v belong to different connected parts of F.*

Given the order of the examination of the edges, by the definition of F-light edge, an edge is F-light when examined if and only if its end-points lie in different connected components.

The crucial observations are:

1. *the F-lightness of e_i depends only on the outcome of the coin flips for the edges preceding it in the ordering.*

2. *edges are never removed from F during this process;*

3. *and the edge e_i is F-light at the end if and only if it is F-light at the start of step i.*

Define phase k as starting after the forest F has $k-1$ edges and continuing until it has k edges. Every edge that is F-light during this phase has probability p of being included in G(p), and hence of being added to F. The phase ends exactly when an F-light edge ia added to G(p) for the first time during the phase. It follows that the number of F-light edges considered during this phase has the geometric distribution with parameter p.

Suppose the forest F grows in size from 0 to s. Then the total number of F-light edges processed till the end of phase s is distributed as the sum of s independent geometrically distributed random variables, each with parameter p. To account for the F-light edges processed after that but not chosen for G(p), we continue flipping coins until a total of n HEADS have appeared. The total number of coin flips is a random variable which has the negative binomial distribution with parameters n and p. Since s is at most $n-1$, it follows that the total number of F-light edges is stochastically dominated by the random variable which represents the total number of coin flips. The expected number of F-light edges is bounded from above by the expectation of this random variable, which is $\frac{n}{p}$. □

6 The Linear-Time MST Algorithm

The randomized linea time **MST** algorithm interleaves Boruvka phases that reduce the number of vertices with random sampling phases that reduce the number of edges. After a random sampling phase, the minimum spanning forest F of the sampled edges is computed using recursion, and the verification algorithm is used to eliminate all but the F-light edges. Then, the MST with respect to the residual F-light edges is computed using another recusive invocation of the algorithm. This is summarized in the following Algorithm MST.

Linear time algorithm :

Input: Weighted, undirected graph $G(V, E)$ with n vertices and m edges.
Output: Minimum spanning forest **F** for **G**.

1: Using three applications of Boruvka phases interleaved with simplication of the contracted graph, compute a graph G_1 with at most $\frac{n}{8}$ vertices and let C be the set of edges contracted during the three phases. **If G** is empty **then** exit and **return** $F = C$.
2: Let $G_2 = G_1(p)$ be a randomly sampled subgraph of G_1, where $p = \frac{1}{2}$.
3: Recursively applying Algorithm **MST** , compute the minimum spanning forest F_2 of the graph G_2.
4: Using a linear-time verification algorithm, identify the F_2-heavy edges in G_1 and deleted them to obtain a graph G_3.
5: Recursively applying Algorithm **MST** , compute the minimum spanning forest F_3 of the graph G_3.
6: Return forest $F = C \bigcup F_3$.

Algorithm 6.1: Algorithm MST

Theorem: 6.1 *The expected running time of Algorithm* **MST** *is* $O(n+m)$.

Proof: *Let* $T(n,m)$ *denote the expected running time of Algorithm MST for a graph* $G(V,E)$ *with* n *vertices and* m *edges. Consider the cost of various steps in this algorithm for such input.*

1. *At step* 1*, the three invocations of Boruvka algorithm, which runs in* $O(n+m)$ *time, run in deterministic time* $O(n+m)$*. After this step, a graph* G_1 *with at most* $\frac{n}{8}$ *vertices and* m *edges is produced.*
2. *At step* 2*, the algorithm performs a random sampling to produce the graph* $G_2 = G_1(\frac{1}{2})$ *with* $\frac{n}{8}$ *vertices and an expected number of edges equal to* $\frac{m}{2}$*. This step also takes* $O(m+n)$ *time.*
3. *Finding the minimum spanning forest of* G_2 *in step* 3 *has the expected running time of* $T(\frac{n}{8}, \frac{m}{2})$*.*
4. *The linear-time algorithm verification in step* 4 *runs in time* $O(n+m)$ *and produces a graph* G_3 *with at most* $\frac{n}{8}$ *vertices and an expected number of edges at most* $\frac{n}{4}$*, by Lemma* 5.2*.*
5. *Finding the minimum spanning forest of* G_3 *in step* 3 *has the expected running time of* $T(\frac{n}{8}, \frac{n}{4})$*.*
6. *Finally,* $O(n)$ *time is needed for step* 6*.*

Adding up all these steps give us the recurrence:

$$T(n,m) \leq T(\frac{n}{8}, \frac{m}{2}) + T(\frac{n}{8}, \frac{n}{4}) + c(n+m)$$

for some constant c*.*

A solution satisfying this recurrence is $2c(n+m)$*. We have base case of* $n = 1$ *and* $m = 0$*:* $T(1,0) = O(1)$*. Suppose that* $T(\frac{n}{8}, \frac{m}{2}) \leq 2c(\frac{n}{8} + \frac{m}{2})$ *and* $T(\frac{n}{8}, \frac{n}{4}) \leq 2c(\frac{n}{8} + \frac{n}{4})$*, then by induction:*

$$T(n,m) \leq T(\frac{n}{8}, \frac{m}{2}) + T(\frac{n}{8}, \frac{n}{4}) + c(n+m)$$

$$T(n,m) \leq 2c(\frac{n}{8} + \frac{m}{2}) + 2c(\frac{n}{8} + \frac{n}{4}) + c(n+m)$$

$$T(n,m) \leq 2c(\frac{n}{4} + \frac{n}{4} + \frac{m}{2}) + c(n+m))$$

$$T(n,m) \leq 2c(n+m)$$

This implies that the expected running time of the **MST** *algorithm is* $O(n+m)$*.* □

References

[1] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, England, June 1995.

[2] T. Cormen, C. Leiserson and R. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1999.