COMPSCI 590.1: Graph Algorithms March 17, 2015

Lecture 18

Lecturer: Debmalya Panigrahi *Scribe: Allen Xiao*

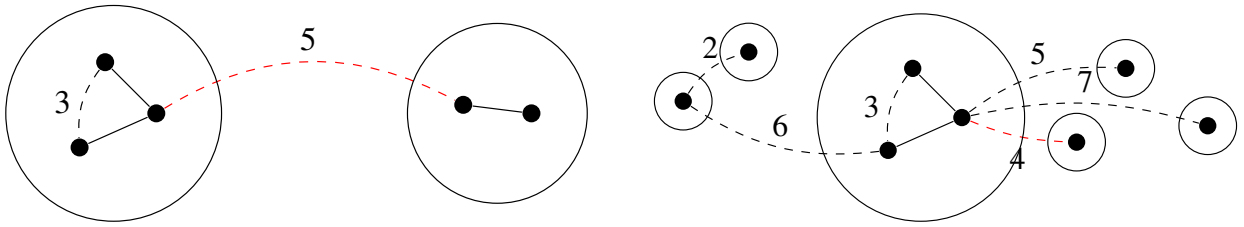
1 Overview

This lecture we begin our discussion of *network design problems*, where we are given a graph and try to build an optimal subgraph subject to some constraints. Two problems in this space are *shortest s-t path* and *minimum spanning tree*. These, in some sense, lie at different extremes for network design problems (one connected pair versus all vertices). In a following lecture we will examine a problem somewhere in between, called *Steiner forest*.

In this lecture, we review algorithms for minimum spanning tree: Kruskal's, Prim's, Boruvka's, and finally the randomized linear time algorithm of Karger, Klein, and Tarjan.

2 Minimum Spanning Tree

In the minimum spanning tree problem, we are given a graph $G = (V, E)$ with edge weights $w(e)$, and asked to compute a spanning tree of minimum weight. Let $n = |V|, m = |E|$. For our discussion, we will assume that edge weights are unique (or at least uniquely orderable) – it is possible to generalize.



(a) Kruskal's algorithm. (b) Prim's algorithm.

Figure 1: Kruskal's algorithm and Prim's algorithm for minimum spanning tree. The red edges are added this iteration.

2.1 Kruskal's Algorithm

Kruskal's algorithm maintains a spanning forest (starting with only singletons) and on each step connects two components with the globally lightest edge between components. This is typically programmed as "lightest edge which does not create a cycle", which can be done using union-find and initially sorting the edges. A simple implementation runs in $O(m \log n)$ time.

2.2 Prim's Algorithm

Prim's algorithm is similar to Kruskal's, instead maintaining a single large component in a forest of singletons. Each iteration adds the lightest edge leaving the large component. An implementation using binary

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

2.3 Boruvka's Algorithm

In a similar way to Kruskal's, we join connected components of a forest at each iteration. Every component tracks the minimum cost edge leaving it. At each iteration, we add these edges to the forest and update the tracked edges.

It's easy to prove that the edges in each iteration must be acyclic between components. If there is a cycle, let the tracked cycle edges be $e_1, \ldots, e_k$, where e_1 has the minimum weight of the cycle. By choice of tracked edges, it must be that

$$w(e_1) < w(e_2) < \cdots < w(e_k) < w(e_1)$$

which is a contradiction.

Let the number of components be k. At each step, Boruvka's algorithm at least halves the number of components. In the worst case, there are $k/2$ unique edges tracked and $k/2$ components after joining. In the best case, there are $k-1$ unique edges, and only one component remains after joining. Boruvka's algorithm therefore runs in $O(m \log n)$ time ($\log n$ iterations, check all edges per iteration).

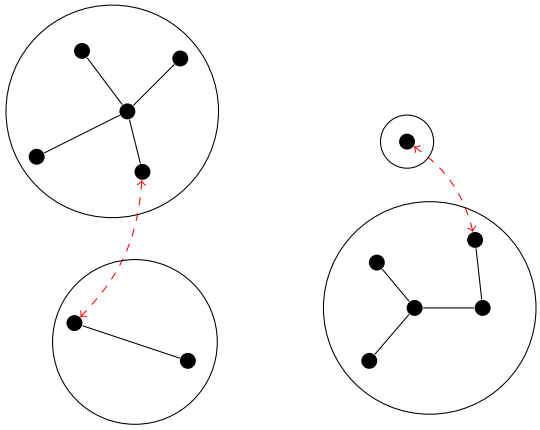


Figure 2: Boruvka's algorithm. The red edges are added next. Notice that it's possible for multiple components to have the same minimum weight outgoing edge.

Boruvka's algorithm actually predates both Kruskal's and Prim's by about 30 years, but received a resurgence of interest after the publication of Karger-Klein-Tarjan. The algorithm (in a serial setting) is no faster than Kruskal's or Prim's, but as we will see the fast reduction of number of components (per iteration) inspires the much faster Karger-Klein-Tarjan algorithm.

2.4 Correctness for Previous Algorithms

Correctness for all these algorithms exploits the following property.

Property 1. *For any cut $(S, \overline{S})$ the minimum weight edge in the cut must be in the MST.*

Given Property 1, both Kruskal's algorithm and Prim's algorithm are correct, as each algorithm considers sets of cuts and chooses the minimum edge in every choice until the graph is spanning.

Proof. Assume for contradiction that in some cut $(S, \overline{S})$, the MST T does not choose the minimum weight edge e^*. If we add e^* to T we get a cycle C. Obviously, some other edge of the tree (also in the cycle) crosses the cut. For all other edges $e \in C \cap (S, \overline{S}), e \neq e^*$, we have that

$$w(e) > w(e^*)$$

by our assumption on uniqueness of weights. It follows that the tree formed by replacing any such e and with e^* has strictly lower weight, but remains spanning and connected. This contradicts our assumption that T was an MST, so for all such cuts T must include the minimum weight edge. □

Another property we'll be using later in the analysis of the Karger-Klein-Tarjan algorithm is is on the maximum weight edge in cycles.

Property 2. *For any cycle C in the graph, the maximum weight edge in C cannot be in any MST.*

Proof. Consider any cycle C with maximum weight edge e, and assume for contradiction that e is in MST T. Consider the cut of T, $(S, \overline{S}) = \{e\}$. Both sides of the cut are connected. Since there are no cycles in T, at least one other edge $e' \in C$ spans $(S, \overline{S})$ and $e' \notin T$. By choice of e:

$$w(e) > w(e')$$

Replacing e with e' in T therefore gives a connected spanning tree with strictly lower cost, a contradiction to our choice of T. It follows that the maximum weight edge of any cycle is in no MST. □

3 Karger-Klein-Tarjan

The algorithm is as follows:

1. Run two steps of Boruvka's algorithm on the input graph, contract the resulting spanning forest as G. If it's connected, output G as the MST.
2. Sample edges in G independently with probability 1/2 to form sampled graph H. Recursively compute the minimum spanning forest of H[1]as F.
3. Discard all *F-heavy* vertices in G, then recrusively find the MST in G. Output the resulting MST of G.

Where F-heavy is defined as follows:

Definition 1 (F-heavy)**.** *Edge $e \in G$ is F-*heavy *for a forest F if and only if e is the maximum weight edge of a cycle in $F \cup \{e\}$.*

By definition, edges of F and edges between components of F are always F-light. By Property 2, these discarded edges are never in the MST of G. Combined with Property 1 (the correctness of Boruvka's), this algorithm is correct.

[1]MSF rather than MST because H may be disconnected. The same algorithms still work.

3.1 Runtime Analysis

On each step, per level of recursion (with m edges input):

1. $O(m)$, as we run Boruvka's for a small constant number of iterations. As this suggests, we need not use exactly 2 iterations of Boruvka's – any small, constant number of iterations works.
2. $O(m)$ from sampling the edges, then some T_1 for the first recursive call on H.
3. $O(m)$ for filtering out F-heavy edges, then some T_2 for the second recursive call on a smaller G.

For the $O(m)$ in Step 3 we state without proof a theorem which gives the time to perform the F-heavy edge removal step. This theorem is a result from Dixon *et al* [DRT92].

Theorem 1. *An MST can be verified in $O(m)$ time. In other words, we can decide which edges are F-heavy in $O(m)$ time.*

The total running time is therefore $O(m) + T_1 + T_2$, where T_1, T_2 are random variables depending on the sampling process for H. We can think of this recursive call structure as a binary tree (two children for two recursive calls). Each left child is a recursive call on H from Step 2, whereas each right child is a recursive call on the F-light subgraph of G from Step 3.

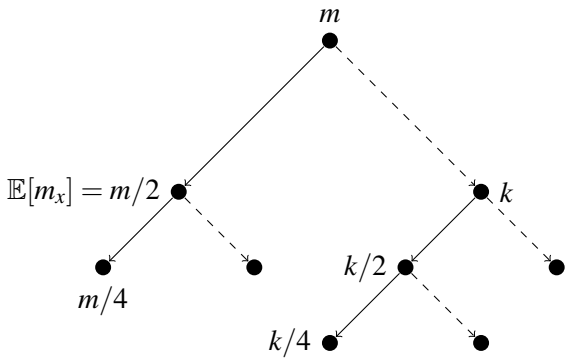


Figure 3: The recursive call structure of Karger-Klein-Tarjan, marked with the expected number of edges (i.e. input size) at each subproblem. Solid edges (resp. dashed) indicate recursive subproblems spawned in Step 2 (resp. Step 3).

Let X be a random variable for the number of edges of some recursive call, and Y be the number of edges in its left child. We have (by nature of the sampling process):

$$\mathbb{E}[Y \mid X = k] = \frac{k}{2}$$
$$\mathbb{E}[Y] = \mathbb{E}[X]/2$$

Now we can decompose the binary tree into left paths, where each begins at some right child.

Let m_x (resp. n_x) be the number of edges (resp. vertices) in recursive call x, and let

$$R(x) = \{x \mid x \text{ is a right child}\}$$

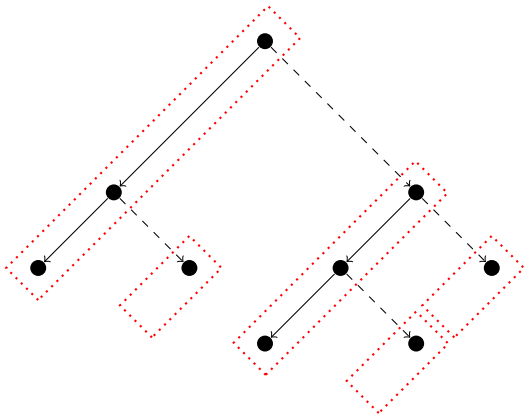


Figure 4: Decomposition of the recursion tree into left paths. Each left path starts at some right child, and is marked by a dotted red rectangle.

The expected number of edges on each left path is at most $2\times$ the number of edges at the root/head. Since the work per x is linear in m_x, this gives an expression for the recursive calls $T_1 + T_2$.

$$T_1 + T_2 = O\left(2 \cdot \sum_{x \in R(x)} m_x\right)$$

To bound this expression, we use the following lemma.

Lemma 2. *Suppose H is an iid sample of edges in G with probability p. Let F be a MSF of H. Then the expected number of F-light edges is at most n/p.*

If we apply Lemma 2, so the number of edges at each instance is at most:

$$m_x \leq \frac{n_x}{1/2} = 2n_x$$

Recall that each iteration of Boruvka's algorithm reduces the number of vertices by at least 1/2.

$$\sum_x n_x \leq 2n$$

Finally, we have the value of $T_1 + T_2$.

$$\begin{aligned} T_1 + T_2 &= O\left(2 \cdot \sum_{x \in R(x)} m_x\right) \\ &= O\left(4 \cdot \sum_{x \in R(x)} n_x\right) \\ &= O(n) \end{aligned}$$

Giving us a runtime of:

$$O(m+n)$$

It remains to prove Lemma 2.

Proof. The procedure in the algorithm samples G, and then runs an MST algorithm. Swapping these steps gives an equivalent process:

1. Run an MST algorithm on G (e.g. Kruskal's).
2. Sample edges of G. Discard non-MST edges and add MST edges with probability p. Call the resulting forest F.

To analyze the number of F-light edges, we consider F a sample of the total set of F-light edges generated through a series of coin flips with probability of heads p. At the end, we know F has at most $n-1$ edges (size of the source MST).

Let Z be a random variable giving the number of coin flips before the first head. Z has a geometric distribution with parameter p, and the expected number of coin tosses before a single edge is added is $1/p$. By linearity of expectation, the expected total number of coin tosses (tested F-light edges) before n edges are added to F is therefore at most n/p.

We conclude that the number of F-light edges in G is at most n/p. □

4 Summary

In this lecture, we reviewed the minimum spanning tree algorithms by Kruskal [Kru56], Prim [Pri57], and Boruvka [Bor26] before introducing the randomized linear time algorithm by Karger, Klein, and Tarjan [KKT95].

References

[Bor26] Otakar Borůvka. O jistém problému minimálním. 1926.

[DRT92] Brandon Dixon, Monika Rauch, and Robert E Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992.

[KKT95] David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)*, 42(2):321–328, 1995.

[Kru56] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[Pri57] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957.