CIS 320: Algorithms October 18, 2021

Lecture 12

Lecturer: Aaron Roth *Scribe: Aaron Roth*

Greedy Algorithms III: Minimum Cost Spanning Trees

Suppose we have a set of n computers that we would like to connect in a network. We could think of the computers as representing the vertices $V = \{v_1, , \ldots, v_n\}$ of a graph. We could build a connection between any pair of vertices (v_i, v_j) at some cost $c(v_i, v_j) > 0$, which could depend on things like the distance between the nodes, or the degree of existing infrastructure in the area already (e.g. it might be cheaper to connect far away nodes in large cities than to connect to nodes in remote areas). This suggests a computational problem: given a graph $G = (V, E)$ with nonnegative costs c_e associated with each edge $e = (u, v)$, find a subset of edges $T \subseteq E$ so that:

1. The induced graph $G' = (V, T)$ is connected, and
2. The total cost $c(T) = \sum_{e \in T} c_E$ is minimized subject to the connectivity constraint.

An initial observation is that without loss of generality:

Claim 1 *There exists a solution to the above problem $G' = (V, T)$ such that G' is a* tree *— i.e. a connected graph that contains no cycles.*

Proof Suppose otherwise — that every optimal solution $G' = (V, T)$ is a connected graph with a cycle. Consider a minimal optimal solution G', that cannot have any edges removed while remaining optimal and feasible. Let C be a cycle in G' containing edge $e = (u, v)$. Observe that $G'' = (V, T \setminus \{e\})$ will also be a connected graph, with no larger cost, contradicting the minimal optimality of G'. To see why, note that any path that previously traversed edge e can be modified to traverse the remainder of cycle C, maintaining connectivity of every pair of vertices. ■

This motivates the minimum spanning tree problem.

Definition 2 *An instance of the minimum spanning tree (MST) problem is given by a weighted graph $G = (V, E, c)$. A solution to the MST problem is a subset of edges $T \subseteq E$ such that the subgraph $G' = (V, T, c)$ is a connected tree such that $c(T) = \sum_{e \in T} c_e$ is minimized.*

It turns out that for this problem, a variety of greedy approaches work. We will study two popular ones: "Kruskal's Algorithm" and "Prim's Algorithm". Both are simple. Kruskal's algorithm simply considers edges in increasing order of their cost, and adds them to the solution T so long as they do not induce a cycle. Prim's algorithm "grows" a connected set S, and at every step until $S = V$, adds to T the cheapest edge crossing between S and $V \setminus S$. (This is very similar to Djikstra's algorithm for computing shortest paths that you should have encountered before). We will assume for simplicity that all edge costs c_e are distinct, but this is only for convenience: all that we need for all of our arguments to go through is a consistent tie breaking rule in the case that there are multiple edges with the same cost.

Algorithm 1 Kruskal(G)

Initialize $T = \emptyset$
Sort the edges E in increasing order of costs so that $c_1 < c_2 < \ldots < c_m$
for $i = 1$ to m **do**
  if $T \cup \{e_i\}$ contains no cycles **then**
    Let $T = T \cup \{e_i\}$.
  end if
end for
Return T.

Algorithm 2 Prim(G)

Initialize $T = \emptyset$, $S = \{v_1\}$ for an arbitrary $v_1 \in V$.
while $S \neq V$ **do**
  Select $e = (u, v)$ such that $e = \arg\min_{u \in S, v \in V \setminus S} c_{(u,v)}$.
  Let $S = S \cup \{v\}$ and $T = T \cup \{e\}$.
end while
Return T.

Both algorithms grow their set of edges T in a different greedy manner, but both can be analyzed by understanding the same structural fact about the set of solutions to the MST problem:

If we partition the vertex set into two parts, then the cheapest edge crossing the partition *must* be in *every* minimum spanning tree.

Formally:

Lemma 3 *Let S be any subset of nodes such that $S \notin \{\emptyset, V\}$. Let $e = (u, v)$ such that $e = \arg\min_{u \in S, v \in V \setminus S} c_{(u,v)}$. Then* every *optimal MST solution T has $e \in T$.*

The idea for the proof is again an "exchange" or "local surgery" argument. If there were to exist some MST T that failed to include e, we will show that we could modify T by exchanging e for another edge in T to produce another solution T' with strictly lower cost, contradicting the optimality of T.

Proof Fix a subset of the vertices S such that neither S nor $V \setminus S$ is nonempty, and let $e = (u, v)$ be the minimum cost edge crossing from S to $V \setminus S$. Suppose for point of contradiction that there is a minimum spanning tree T such that $e \notin T$.

T is a spanning tree, so there exists a path in T from u to v. Consider walking along this path. Since $u \in S$ and $v \in V \setminus S$, this path must contain some edge $e' = (u', v')$ crossing between S and $V \setminus S$. Now consider the alternative solution $T' = T \setminus \{e'\} \cup \{e\}$. We make two claims:

1. $c(T') < c(T)$: Since we have exchanged e' for e, we have that:

   $$c(T') - c(T) = c_{e'} - c_e < 0$$

   Here the inequality follows from the fact that both c_e and $c_{e'}$ are edges crossing between S and $V \setminus S$, and by assumption, e is the minimum cost such edge, and costs are distinct.

2. T' is also a spanning tree: We need to show two things. First, that T' is connected. To see this, note that T is connected, and $e' = (u', v')$ is on a path between u and v. Hence any path in T that used edge $e' = (u', v')$ can be rerouted along the path $u' \to u$, along the new edge $e = (u, v)$, and finally along the path $v \to v'$.

   Second, we must show that T' is acyclic. Since T is acyclic, the only cycle in $T \cup \{e\}$ is the cycle including edge e together with the (unique) path from u to v in T. But this cycle is broken by removing e'.

■

With Lemma 3, we can prove the optimality of both Prim's and Kruskal's algorithms.

Theorem 4 *Prim's algorithm always returns a minimum spanning tree given a connected graph G.*

Proof By construction, Prim's algorithm maintains a set S of vertices of size $1 \leq |S| \leq |V| - 1$, and at each step, adds the cheapest edge crossing from S to $V \setminus S$. By Lemma 3, such edges must be part of every minimum spanning tree. At completion, we have a connected graph, so it must be a minimum spanning tree. ■

Theorem 5 *Kruskal's algorithm always returns a minimum spanning tree given a connected graph G.*

Proof Consider any edge $e = (u, v)$ that Kruskal's algorithm adds to its solution T, and let S be the set of vertices to which u has a path to in T at the moment just before e is added. We have that $u \in S$ by definition, but $v \notin S$, or else adding e would have created a cycle. Hence e crosses from S to $V \setminus S$. Moreover, since we are considering edges in sorted order of their costs, e must be the minimum cost such edge. Hence by Lemma 3, e must be contained in every minimum spanning tree. Hence Kruskal's algorithm only adds edges that are contained in every minimum spanning tree. Moreover, when Kruskal's algorithm terminates, it outputs a connected graph (and hence a minimum spanning tree, since it contains only edges in every minimum spanning tree). To see this, note that if it had two disconnected components, there would be some edge e in G crossing between these components that Kruskal's algorithm would have considered but declined to add to T. But this would only happen if adding e would create a cycle, which it could not if these were disconnected components. ■

Finally, a remark on our assumption that all edge costs are unique. This was only for convenience. In any graph, the set of spanning trees is finite. Consider the minimum nonzero difference δ between the cost of any two spanning trees with distinct costs, which is well defined since there are only finitely many such trees. Consider perturbing each edge cost by a uniformly random quantity in $[0, \delta/n]$. Now edge costs are all unique, and so our arguments apply: but this perturbation has changed the cost of each spanning tree by at most $deltan/(n-1) < \delta$, since spanning trees contain only $n-1$ edges. Hence any tree that is an MST under the perturbation was also an MST before the perturbation. The perturbation here is simply simulating any arbitrary (but consistent) tie breaking rule.