CO 353: Final Exam Proofs

A. Shortest Paths

In this problem, we are given a directed graph $G = (V, E)$ and edge lengths $\ell_e \geq 0$ for all $e \in E$. Given a start vertex $s \in V$, the goal is to find the shortest path from s to v for all vertices $v \in V$.

Dijkstra's algorithm. Let $d(v)$ denote the length of a shortest path from s to v. Let $d'(v)$ be upper bounds on the length of a shortest path from s to v, and let A be the set of visited vertices.

1. Initialize $A = \{s\}$ and $d(s) = 0$. For all $v \in V \setminus A$, let $d'(v) = \infty$.
2. While $A \neq V$, do the following:
   (a) For all $v \in V \setminus A$, set
   $$d'(v) = \min_{\substack{u \in A \\ (u,v) \in E}} \left\{ d'(v), d(u) + \ell_{(u,v)} \right\}.$$
   (b) Let $w = \operatorname{argmin}_{v \in V \setminus A} d'(v)$. Set $A = A \cup \{w\}$ and $d(w) = d'(w)$.

Proof of correctness. It is enough to show that the length of a shortest path from s to v is correctly computed for all $v \in V$. We proceed by induction on $|A|$. For $|A| = 1$, we have $d(s) = 0$ as expected.

Suppose that the result holds for $|A|$ at a given point in the algorithm, and that w is the vertex that is being added to A. Let $u \in A$ be the vertex that is determining the upper bound for w; that is,

$$d'(w) = d(u) + \ell_{(u,w)}.$$

We know that $d(u)$ is correctly computed by induction since $u \in A$. Suppose towards a contradiction that $d'(w)$ is not correctly computed. Let P_u be a shortest path from s to u, and let P' be a shortest path from s to w. Then we have that

$$\ell(P') < \ell(P_u) + \ell_{(u,w)} = d(u) + \ell_{(u,w)} = d'(w).$$

Note that a shortest path from s to w contains an edge (x, y) such that $x \in A$ and $y \in V \setminus A$. Then $d(x)$ is correctly computed, and we obtain

$$d'(y) \leq d(x) + \ell_{(x,y)} \leq \ell(P') < d'(w),$$

where the second inequality uses the fact that a shortest path from s to w containing (x, y) must use a shortest path from s to x followed by (x, y). In particular, since $d'(y) < d'(w)$, this means that Dijkstra's algorithm should have chosen y instead of w, which is a contradiction. □

B. Minimum Spanning Trees

A **tree** is a connected acyclic graph. A **spanning tree** of a graph $G = (V, E)$ is a subgraph $T = (V, F)$ of G such that $F \subseteq E$ and T is a tree.

In the minimum spanning tree problem, we are given a connected graph $G = (V, E)$ and costs c_e for all $e \in E$. We want to find a spanning tree in G of minimum cost.

First, we state the fundamental theorem of trees from MATH 239.

Fundamental theorem of trees. Let $T = (V, F)$ be a graph. The following are equivalent:

(i) T is a tree.
(ii) T is connected and $|F| = |V| - 1$.
(iii) T is acyclic and $|F| = |V| - 1$.

Consider the following three properties: (1) T is connected; (2) T is acyclic; (3) $|F| = |V| - 1$. Then by the fundamental theorem of trees, knowing that two of the three properties hold guarantees that the remaining one holds as well.

The following are two useful properties for determining if an edge is contained in a minimum spanning tree or not. We will make extensive use of the cut property in our algorithms.

Cut property. Suppose that the costs c_e are distinct. Let $S \subseteq V$ such that $S \neq \varnothing$ and $S \neq V$, and let

$$e = \operatorname*{argmin}_{f \in \delta(S)} c_f.$$

Then e is contained in every minimum spanning tree of G.

Proof. We proceed by contradiction. Suppose that $T = (V, F)$ is a minimum spanning tree of G such that $e \notin F$. Consider the graph $(V, F \cup \{e\})$. Then $|F \cup \{e\}| = |V|$ and this graph is connected, so it cannot be a tree by the fundamental theorem of trees. In particular, it must contain some cycle C, and we have $e \in C$ since $T = (V, F)$ was acyclic.

Note that $|C \cap \delta(S)|$ must be even because for every edge leaving the cut $\delta(S)$, there must be some edge coming back into the cut. Moreover, since $e \in C \cap \delta(S)$, this implies that $|C \cap \delta(S)| \geq 2$. Then there must be some other edge $e' \in C \cap \delta(S)$ distinct from e.

Now, consider $T' = (V, (F \cup \{e\}) \setminus \{e'\})$. We see that from the fundamental theorem of trees that T' is a tree. Indeed, we have that $|(F \cup \{e\}) \setminus \{e'\}| = |V| - 1$, and T' is still connected because if a path between two vertices used the edge e', then we can go along the edges in $C \setminus \{e'\}$ instead. Finally, notice that

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

since the edge costs are distinct with $e = \operatorname{argmin}_{f \in \delta(S)} c_f$ and $e' \in \delta(S)$. This implies that T is not a minimum spanning tree, which is a contradiction. □

Cycle property. Suppose that the costs c_e are distinct. Let C be a cycle in G and let

$$e = \operatorname*{argmax}_{f \in C} c_f.$$

Then e is not contained in any minimum spanning tree of G.

Proof. Let $T = (V, F)$ be a spanning tree containing e, and suppose that the endpoints of e are v and w. We show that T does not have the smallest possible cost. Similar to the proof of the cut property, we do this by exchanging e with an edge e' of cheaper cost in a way that we still obtain a spanning tree.

By deleting e from T, we obtain a graph with two components: S containing v and $V \setminus S$ containing w. The edge we are replacing e with should have one end in S and the other in $V \setminus S$ to connect the components back together.

To find such an edge, we follow the cycle C. Note that the edges in $C \setminus \{e\}$ together form a path P from v to w. If we follow P from v to w, we will begin at S and end in $V \setminus S$, so there is some edge e' in P that crosses from S to $V \setminus S$.

Let $T' = (V, (F \cup \{e'\}) \setminus \{e\})$. Note that T' has $|V| - 1$ edges and is connected by our above argument, so it is a tree by the fundamental theorem of trees. Moreover, we have $c_{e'} < c_e$ since e is the edge of maximum cost in C with $e' \in C$ so that

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

Then $c(T) > c(T')$, so T' is a spanning tree of G with smaller cost than T. It follows that T is not a minimum spanning tree. □

Now, we discuss two efficient algorithms that solve the minimum spanning tree problem.

Prim's algorithm. The main idea is to use the cut property to construct a minimum spanning tree starting from an arbitrary vertex $s \in V$. At each iteration, we keep track of a partial tree construction T and a set of vertices $A \subseteq V$ that are connected to s in T. We stop the algorithm once we have reached all the vertices in G.

1. Let $s \in V$ be an arbitrary vertex. Set $A = \{s\}$ and $T = \varnothing$.
2. While $A \neq V$, do the following:
   (a) Set $e = \operatorname{argmin}_{f \in \delta(A)} c_f$ where $e = uv$ with $u \in A$ and $v \notin A$.
   (b) Set $A = A \cup \{v\}$ and $T = T \cup \{e\}$.

Proof of correctness. We assume that the edge costs c_e are distinct. At each iteration of Step 2, we add one edge to T, so $|T| = |V| - 1$ since there are $|V| - 1$ iterations. Moreover, T is connected because an invariant of the algorithm is that all vertices are connected to s. It follows by the fundamental theorem of trees that T is a tree. Finally, by the cut property and our assumption that the edge costs are distinct, we see that T contains only the edges that are contained in every minimum spanning tree, so T itself is a minimum spanning tree. □

A consequence is that a graph with distinct edge costs c_e has a unique minimum spanning tree.

Kruskal's algorithm. This is a greedy algorithm that first sorts the edges by ascending edge costs and continually adds an edge to a partial tree construction T as long as no cycle is induced by that edge.

1. Sort the edges such that $c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m}$. Set $T = \varnothing$ and $j = 1$.
2. While $j < m + 1$, do the following:
   (a) If $T \cup \{e_j\}$ is acyclic, then set $T = T \cup \{e_j\}$.
   (b) Increment $j = j + 1$.

Proof of correctness. We again assume that the edge costs are distinct. Let T be the output set consisting of edges. We know that T is acyclic by construction as Step 2 of the algorithm rules out all edges that create a cycle. We show that (V, T) is connected by way of contradiction so that it is a spanning tree by the fundamental theorem of trees.

Suppose that (V, T) is not connected. Then there is a nontrivial cut $\delta(S)$ for some $\varnothing \subsetneq S \subsetneq V$ such that $\delta(S) \cap T = \varnothing$. Since G is connected, we have $\delta(S) \neq \varnothing$, so there exists some edge $e \in \delta(S)$. We look at the moment in time that e was considered in the algorithm. Since e was rejected, it must be that $T \cup \{e\}$ contains a cycle with $e \in C$. But $|C \cap \delta(S)|$ is even and $e \in C \cap \delta(S)$, so $|C \cap \delta(S)| \geq 2$. Then $|(C \setminus \{e\}) \cap \delta(S)| \geq 1$, contradicting the fact that $\delta(S) \cap T = \varnothing$ since $C \setminus \{e\} \subseteq T$.

Next, we show that we have a minimum spanning tree. Consider the moment an arbitrary edge $e = uv$ was added to T, and let T' be the set of edges in T just before the addition of e to T. Note that T' has no u, v-path, for otherwise such a path together with $e = uv$ would form a cycle. Hence, there exists

$S \subseteq V$ such that $u \in S$, $v \notin S$, and $\delta(S) \cap T' = \varnothing$. Due to the way that the edges are sorted in Step 1, we have $e = \operatorname{argmin}_{f \in \delta(S)} c_f$. By the cut property, e is contained a minimum spanning tree. Then this holds for all edges $e \in T$, implying that T is itself a minimum spanning tree. □

Clusterings of maximum spacing. We are given a set $U = \{p_1, \ldots, p_n\}$ of n objects and a distance $d(p_i, p_j)$ between points. We require that $d(p_i, p_i) = 0$ and $d(p_i, p_j) = d(p_j, p_i) > 0$ for distinct p_i and p_j. A **k-clustering** of U is a partition of U into k nonempty sets $C_1, \ldots, C_k$. The **spacing** of a k-clustering is the minimum distance between any pair of points lying in different clusters. We want points in different clusters to be far apart from each other, so a natural goal is to seek the k-clustering with the maximum possible spacing. That is, over all k-clusterings $C_1, \ldots, C_k$ of U, we want to maximize

$$\min_{1 \le i < j \le k} \left\{ \min_{p \in C_i, q \in C_j} d(p, q) \right\}.$$

Single linkage clustering algorithm. Consider the complete graph K_n on the vertex set U with the edge costs given by the distances. The connected components will be the clusters. We want to bring nearby points together into the same cluster as rapidly as possible so that they don't end up in different clusters that are close together. We see that if we sort by ascending distances and skip over edges where both endpoints are already in the same cluster, then we avoid cycles and end up with a union of trees.

Note that this procedure is precisely Kruskal's algorithm. The only difference is that we are looking for a k-clustering, so we stop once we hit k connected components. That is, we are running Kruskal's algorithm but stopping it before it adds the last $k-1$ edges. This is equivalent to taking the full minimum spanning tree and deleting the $k-1$ most expensive edges and defining the k-clustering to be the resulting connected components $C_1, \ldots, C_k$. We show that this is a k-clustering of maximum spacing.

Proof of correctness. Let $\mathcal{C}$ denote the clustering $C_1, \ldots, C_k$. The spacing of $\mathcal{C}$ is the length d^* of the $(k-1)$-th most expensive edge in the minimum spanning tree; this is the length of the edge that Kruskal's algorithm would have added next at the moment we stopped it.

Let $\mathcal{C}'$ be another k-clustering which partitions U into nonempty sets $C'_1, \ldots, C'_k$. We show that the spacing of $\mathcal{C}'$ is at most d^*. Since $\mathcal{C}$ and $\mathcal{C}'$ are not equal, it must be that one of the clusters C_r is not a subset of any of the k sets C'_s in $\mathcal{C}'$. Hence, there are points $p_i, p_j \in C_r$ that belong to different clusters in $\mathcal{C}'$, say $p_i \in C'_s$ and $p_j \in C'_t$ with $C'_s \neq C'_t$.

Since p_i and p_j are in the same component C_r, it must be that Kruskal's algorithm added all the edges of a p_i, p_j-path P before we stopped it. In particular, each edge in P has length at most d^*. But $p_i \in C'_s$ and $p_j \notin C'_t$, so let p' be the first node on P that does not belong to C'_s and let p be the node on P that comes just before p'. Then $d(p, p') \le d^*$ since the edge from p to p' was added by Kruskal's algorithm. But p and p' belong to different clusters in $\mathcal{C}'$, so the spacing of $\mathcal{C}'$ is at most d^*, as desired. □

C. Minimum Cost Arborescences

Let $G = (V, E)$ be a directed graph and let $r \in V$ be a distinguished node which we call a **root**. An **arborescence** rooted at r is a subgraph $T = (V, F)$ such that:

(i) T is a spanning tree of G if we ignore the direction of edges;

(ii) there is a path in T from r to every other node $v \in V$ if we take the direction of edges into account.

In other words, an arborescence rooted at r is essentially a directed spanning tree rooted at r.

Characterization of arborescences. Let $G = (V, E)$ be a directed graph and let $r \in V$. A subgraph $T = (V, F)$ of G is an arborescence rooted at r if and only if

(1) T has no cycles; and

(2) every node $v \neq r$ has exactly one edge entering it.

Proof. ($\Rightarrow$) Let T be an arborescence rooted at r. Then after ignoring all directions, T is a spanning tree. In particular, T has no cycles. Moreover, there is a unique path from r to v, and the last edge on this path must be incoming for v.

($\Leftarrow$) Suppose that T has no cycles and every node $v \neq r$ has exactly one entering edge. To show that T is an arborescence rooted at r, we first verify that for all $v \neq r$, there is a directed path from r to v in T. Consider the unique edge (v_1, v) incoming to v, and repeatedly follow edges backwards in this way.

Note that r is the only vertex that does not have any incoming edges. Indeed, if it did have an incoming edge, say (v, r), then we could continually follow the edges backwards as above and eventually obtain a cycle, which is a contradiction. Therefore, the above process must terminate at r since T has no cycles and r is the only vertex without an incoming edge. This gives us a directed path from r to v.

We now verify the other condition that T is a spanning tree of G ignoring directions. Since we can get from r to any other vertex $v \neq r$, we see that T is connected when ignoring directions. Moreover, r has no incoming edges. Since every edge is incoming for exactly one of its endpoints, the total number of edges in T is $|V| - 1$, implying that T is a spanning tree of G after ignoring directions by the fundamental theorem of trees. □

Characterization of graphs with arborescences. A directed graph $G = (V, E)$ has an arborescence rooted at $r \in V$ if and only if there is a directed path from r to every other node.

Proof. ($\Rightarrow$) Suppose that for some $v \neq r$, there is no directed path from r to v in G. Then no subgraph could possibly have a directed path from r to v; in particular, no arborescence rooted at r exists.

($\Leftarrow$) For each $v \neq r$, suppose that there is a directed path P_v from r to v. Then a breadth-first search tree rooted at r forms an arborescence rooted at r. □

In the minimum cost arborescence problem, we are given a directed graph $G = (V, E)$, a distinguished root node $r \in V$, and edge costs $c_e \geq 0$ for all $e \in E$. The goal is to find an arborescence rooted at r so that the total cost is minimized. We assume throughout that $G = (V, E)$ has an arborescence rooted at r as this can be easily verified using the above characterization.

We make the observation that every arborescence contains exactly one edge entering $v \neq r$, so if we pick some node v and subtract a uniform quantity from the cost of every edge entering v, then the total cost of every arborescence changes by the exact same amount. Therefore, the actual cost of the cheapest edge entering v is not important; what matters is the cost of all other edges entering v *relative* to this.

Due to this reasoning, we will define y_v to be the minimum cost of an edge entering v. Let f_v be an arbitrary edge achieving this minimum, and let F^* be this set of $n - 1$ edges. For each edge $(u, v) \in E$, we define its modified cost to be $c'_{(u,v)} = c_{(u,v)} - y_v$. Since $c_{(u,v)} \geq y_v$ for all $(u, v) \in E$, it follows that all the modified costs are still nonnegative.

It is possible that (V, F^*) is not an arborescence. However, since F^* has an edge incoming to every node $v \neq r$, the arborescence characterization implies that (V, F^*) contains a cycle C. The above discussion motivates the following crucial fact.

Relationship between original and modified costs. An arborescence T rooted at r is of minimum cost subject to edge costs c_e if and only if it is of minimum cost subject to the modified edge costs c'_e.

Proof. Let T be an arbitrary arborescence rooted at r. Note that the difference between its cost with respect to c_e and its cost with respect to c'_e is exactly

$$\sum_{e \in T} c_e - \sum_{e \in T} c'_e = \sum_{v \neq r} y_v$$

since an arborescence has exactly one edge entering $v \neq r$ in the sum. But this value is independent of the choice of arborescence T, implying that T has minimum cost with respect to c_e if and only if it has minimum cost with respect to c'_e. □

We now consider the problem with respect to the modified costs c'_e. All the edges in the set F^* now have cost 0 under the modified costs, so if (V, F^*) contains a cycle C, then we know that all the edges in C have cost 0. Then we can afford to use as many edges from C as we want since including edges from C doesn't raise the cost.

Therefore, our approach will be to contract C into a *supernode* to obtain a smaller graph $G' = (V', E')$. Note that V' contains the nodes of $V \setminus C$ together with a single node c^* representing C. We transform each edge $e \in E$ to an edge $e' \in E'$ by replacing each end of e that belongs to C with the new node c^*. This can result in G' having parallel edges (i.e. edges with the same endpoints), but we delete all self-loops, namely edges that have c^* as both its endpoints. We recursively find an arborescence of minimum cost in this smaller graph G' subject to the costs c'_e. The arborescence returned by this recursive call can be converted to an arborescence of G by including all but one edge in the cycle C.

Edmonds' algorithm. The above procedure is precisely Edmonds' algorithm. We summarize it below.

1. For all $v \neq r$:
   (a) Let y_v be the minimum cost of an edge entering node v.
   (b) Modify the costs of all edges $(u, v) \in E$ to be $c'_{(u,v)} = c_{(u,v)} - y_v$.
   (c) Let f_v be an arbitrary edge entering v of modified cost 0.

   Let $F^* = \{f_v : v \neq r\}$ be the set of chosen edges of modified cost 0.
2. If F^* is an arborescence rooted at r, then return it.
3. Otherwise, there is a directed cycle $C \subseteq F^*$.
   (a) Contract C to a single supernode to obtain a graph $G' = (V', E')$.
   (b) Recursively find an arborescence (V', F') of minimum cost in G' with respect to costs c'_e.
   (c) Extend (V', F') to an arborescence (V, F) in G by adding all but one edge of C.

Does this lead to an arborescence of minimum cost? One thing we need to worry about is that not every arborescence in G corresponds to an arborescence in the contracted graph G'. Could we "miss" the arborescence of minimum cost in G by focusing on G'?

What we do know is that the arborescences of G' are in one-to-one correspondence with the arborescences of G that have exactly one edge entering the cycle C. (By an edge entering the cycle, we mean an edge $e = (u, v)$ such that $v \in C$ but $u \notin C$.) These corresponding arborescences have the same cost with respect to c'_e since C consists of edges of cost 0. Therefore, to prove the correctness of the algorithm, we need to show that G has an arborescence of minimum cost with exactly one edge entering C. This is what we'll do next.

Existence of minimum cost arborescence with exactly one edge entering the 0-cost cycle. Let C be a cycle in G consisting of edges of cost 0 such that $r \notin C$. Then there is an arborescence rooted at r of minimum cost that has exactly one edge entering C.

Proof. Consider an arborescence T of minimum cost in G. Since r has a path in T to every node, there is at least one edge that enters C. If T enters exactly once, we are done.

Suppose now that there are at least two edges entering C. We will construct another minimum cost arborescence T' that has one edge entering C. Let (u, v) be an edge in T entering C that lies on a shortest path from r to C. Note that this path from r to C uses only one vertex in C, namely the last one. Delete all the edges that enter a vertex in C except for (u, v) from T, and add in edges of C except for the one that enters v.

We claim that T' is a minimum cost arborescence. In our construction, we are only adding edges with cost $c'_e = 0$, so $c'(T) \geq c'(T')$. Moreover, T' has no cycles because T previously had no cycles and now only (u, v) enters C. Moreover, there is exactly one edge entering each vertex $v \neq r$ because for each vertex on C, we either did nothing or added and removed an edge entering v. By the characterization of arborescences, it follows that T' is a minimum cost arborescence with exactly one edge in C'. □

Putting everything together, we now argue the correctness of Edmonds' algorithm.

Proof of correctness of Edmonds' algorithm. We proceed by induction on the number of nodes in G. If the edges of F^* form an arborescence, then the algorithm returns a minimum cost arborescence since F^* represents the cheapest way of having one edge enter each node $v \neq r$. Otherwise, we consider the problem with the modified costs c'_e, which is equivalent by the relationship we discovered between the original costs c_e and the modified costs c'_e. After contracting the 0-cost cycle C to obtain the smaller graph G', the algorithm produces a minimum cost arborescence for G' by the inductive hypothesis. Then by the previous result, there is an minimum cost arborescence in G that corresponds to the minimum cost arborescence computed for G'. □