Class Notes on Minimum Spanning Trees

Let $G$ be a weighted network (recall a network is just a connected graph, and that weighted means there is a cost to each edge.) We would like to find a subgraph that connects all the vertices with the least cost. Notice to connect all vertices, the fewest edges are used if we have a spanning tree. We will call the cheapest tree a **MINIMUM SPANNING TREE** (MST for short).

Consider the graph below. How could we find the MST of this graph?

One way is to "break up" every circuit by removing the most expensive edge. I call this Circuit Busting. Let’s go! We have a circuit of ABH. The most expensive edge is HB (7). So throw it out. The next circuit is HABCD and we throw out the most expensive edge, HD (10). Then GFD is a circuit, most expensive edge is CD (10) and FED is a circuit so throw out FE(11). We’re left with the following tree, of weight 42.

There are two algorithms we can use that are both based on the idea we just used—getting rid of circuits. One is like a nearest neighbor algorithm and one is like the cheapest link algorithm.

**Prim’s Algorithm**

Let $G$ be a network with $N$ vertices.

Step 1) Pick a starting vertex $u_1$. Pick the cheapest edge attached to $u_1$. Call that edge $e_1$. Let $T_1$ be the tree with one edge, $e_1$. 
Step 2). Pick the cheapest edge in the graph that connects to $T_1$ at only one vertex. Call that edge $e_2$ and the tree $T_2 = T_1 \cup e_2$.

Steps 3-N-2) Continue picking the cheapest edge $e_{i+1}$ in the graph that is connected to $T_i$ at only one vertex. Call the tree $T_{i+1} = T_i \cup e_{i+1}$.

Final step) Stop when you have $N-1$ edges.

Example of Prim’s Algorithm with graph above. Start at vertex $B$. The cheapest edge coming off of $B$ is $AB$ (3). Next I can pick an edge coming off of just A or just B. I pick AH(5) as it is the next cheapest edge attached to AB. Next I can pick the cheapest edge that comes off of $(AH \cup AB)$ which is BC (I can’t pick BH because it connects to my current graph at two vertices, B and H). Next I pick CD because it is the cheapest edge connected to my tree (AD is 10 and therefore more expensive). Then I pick DE because it’s the cheapest connected to the tree (the only other choice would be DG but it costs 10). Then pick DF because it’s cheaper than DG or EF. Then pick FG because it’s still cheaper than DG or EF. I have 7 edges now so I stop. My tree is the same one we got by busting circuits.

Kruskal’s Algorithm

Step 1) Pick absolute cheapest edge of the graph to be in the MST
Step 2) Pick next cheapest edge
Step 3-N-1) Continue picking and marking the cheapest remaining edge that does not create a circuit. Stop when you have $N-1$ edges.

Example with graph from above: AB is the cheapest. Then DE is the next cheapest. Then CD. Then DF (I can’t pick AH which is also 7 because that would create a circuit). Then FG then BC. I have 7 edges now so I stop. Notice I get the same tree I did using Prim’s.

Proof that Kruskal’s algorithm gives an MST:
1.) Show it gives you a spanning tree:
Because it prevents circuits from forming, it has to give you trees but possibly not just one. Suppose that it gives you two disconnected trees, \( Y_1 \) and \( Y_2 \). Suppose that \( Y_1 \) has \( k \) vertices. Then it must have \( k - 1 \) edges. That means \( Y_2 \) has at most \( N - k \) vertices, and \( N - k - 1 \) edges.

So we’ve got a total of \( k - 1 + (N - k - 1) = N - 2 \) edges. But we weren’t supposed to stop till we had \( N - 1 \) edges. So we must have one connected tree on \( N \) vertices.

2.) Show that the cost of the tree is minimum.
Let \( W(A) \) mean the weight of graph \( A \) (i.e., add up the weights of all the edges in \( A \)). Let’s suppose \( A \) is the tree we obtain from Kruskal’s algorithm. Then it has edges \( a_1, a_2, a_3, \ldots a_{N-1} \) where the index on the edge tells you the order in which it was added to \( A \). So since we add in order of cheapest to most expensive, we know that \( W(a_1) \leq W(a_2) \leq W(a_3) \leq \ldots \leq W(a_n) \). Suppose \( T_1 \) is any other spanning tree in the graph and that we order its edges from cheapest to most expensive as \( t_1, t_2, \ldots, t_{N-1} \).

Since the trees are different, there is some smallest \( k \) such that \( a_k \) is among the edges of \( A \) but not among the edges of \( T_1 \), but \( a_{k-1} \) was in \( T_1 \). So then \( T_1 \cup a_k \) has a circuit not entirely in \( A \). Call the edge not in \( A \) by the label \( e_1 \). Then if we take out edge \( e_1 \) from \( T_1 \) and add edge \( a_k \) instead, we’ll have a new tree, \( T_2 \) that has one more edge in common with \( A \) than \( T_1 \) did but we have \( W(T_2) \leq W(T_1) \). Now either \( T_2 = A \) find the cheapest edge of \( A \) that is not in \( T_2 \), call it \( a_{k_2} \). \( T_2 \cup a_{k_2} \) has a circuit not entirely in \( A \).

Let’s say the edge not in \( A \) is \( e_2 \). We take out \( e_2 \) from \( T_2 \) and put in \( a_{k_2} \) instead to get a new tree, \( T_3 \), that has one more edge in \( A \) than \( T_1 \) did, and the edge we replaced is cheaper than what was in \( T_2 \) so we have a cheaper tree, \( W(T_3) \leq W(T_2) \leq W(T_1) \).

Keep going with this process, taking out a more expensive edge and adding an edge from \( A \) until we have \( A \) itself. By the system therefore, \( W(A) \leq W(T_1) \) so we’ve proved that for any tree \( T_1 \), \( A \) has lower weight. Thus \( A \) is a minimum spanning tree.