Minimum Spanning Tree

This document contains summary of some of the items discussed as part of the COMS 3110 Lectures. This document **does not replace the lecture materials**. This document may contain some topics that are not covered as part of the lecture; you will not be tested on those parts, they are made available to you for gaining further knowledge on topics/concepts that are related to class-lecture.

Basic Definition & Background. As we have done in Dijkstra's, we will consider weighted graph, in particular, weighted, undirected, connected graph without any constraint on the valuation of the weights. A spanning tree T for a weighted undirected graph G is a tree over the vertices of the graph, such that all vertices are connected (i.e., there is a path between any pair of vertices). As this is a tree, therefore, there is unique path between the vertices (no cycle); and as there is a path between any pair of vertices, it contains $|V| - 1$ edges, where V is the set of vertices in the graph. We will represent a spanning tree in terms of the edges that are present in the tree, e.g., $T = \{e_1, e_2, \ldots, e_k\}$ denotes a spanning tree containing edges e_1, e_2, ..., e_k.

A spanning tree is associated with a weight, which is computed as the sum of the weights of the edges that form the spanning tree. For a graph, there can be multiple spanning trees. For a spanning tree T, we will use $wt(T)$ to denote its weight. The spanning tree with the minimal weight is referred to as minimum spanning tree (MST). The objective is to identify a MST for an undirected, weighted, connected graph.

We will consider Prim's algorithm, which is based on greedy algorithmic strategy. As we have done in the previous section, we will consider two properties that hold for the MST problem and will use them to realize the algorithm.

1. Optimal substructure property: Consider a graph G and its MST T^*. If $e = (u, v)$ be an edge in T^*, then we can construct a graph G/e by contracting the edge e. This construction involves, merging the vertices u and v to form a new vertex, and adding all edges associated with u and with v with the new vertex (if there is an edge from u to x and v to x, then the edge with the minimal weight is added between the new vertex and x).

   The graph G/e is smaller than G (has one less vertex and has at least one less edge). We claim that if T is a MST of G/e, then $T \cup \{e\}$ is a MST of G. Intuitively, the MST of G can be obtained from the MST of G/e and e.

   The proof of this claim relies on few observations. First note that, $T^* - \{e\}$ is a spanning tree of G/e. This is immediate because, any connectivity between vertex-pairs other than the pair u and v is maintained in $T^* - \{e\}$ and the graph G/e contains all pairs of vertices except the pair u and v (which are merged).

   As T is MST of G/e, we conclude that

   $$wt(T) \leq wt(T^* - \{e\}) = wt(T^*) - wt(e) \tag{1}$$

   Next, consider the tree $T \cup \{e\}$. This tree is a spanning tree for the graph G. This is because all connectivity between pairs of vertices (other than the pair u and v) are maintained in T and $T \cup \{e\}$ (in particular the edges in T creates a partition of graph G where one group are reachable to and from u and another group reachable to-and-from v) . The addition of e allows for connectivity between u and v in G.

   The weight of the spanning tree $T \cup \{e\}$ is $wt(T) + wt(e)$, which is $\leq wt(T^*)$ (from Equation 1). Therefore, $T \cup \{e\}$ is MST of G.

2. Greedy Choice Property: Partition the set of vertices in G into A and B. We refer to this partition as a cut. The edges that connect vertices in group A with vertices in group B are referred to as cross-edges. The claim is that for a cut, the minimal weight cross-edge is part of some MST.

Consider a cut A and B, and a minimal cost cross-edge $e = (u, v)$, where $u \in A$ and $v \in B$. Consider next a MST T^* that does not contain e. Our proof-strategy involves changing the edges in T^* to create a new MST that includes e.

By our consideration, in T^*, u and v are not directly connected. Therefore, there must be a different (and exactly one) path that connects u and v. As these vertices are in two different groups, such a path must include a cross-edge between A and B. Let us denote that cross-edge as e'.

Now, we argue (exchange argument) that we can create a spanning tree by replacing e' in T^* with e, i.e., $T = (T^* - \{e'\}) \cup \{e\}$. All the vertices that we connected (via some paths) due to the presence of e' are now connected (via possibly different paths) due to the inclusion of e. Note that, inclusion of e in place of e' does not result in any cycles in T. In short, T is a spanning tree.

The weight of T is $wt(T) = wt(T^*) - wt(e') + wt(e)$. As e is a minimal cost cross-edge, $wt(e) \leq wt(e')$, which, in turn, implies that $wt(T) \leq wt(T^*)$. Therefore, T is a MST and it contains e.

The validity of the above properties in the context of MST leads to the following algorithm.

1. Select an arbitrary vertex v in G.
2. Create a cut $A = \{v\}$ and $B = V/v$ (all vertices other than v).
3. By greedy choice property, we know that the minimal weight cross-edge is part of a MST. Let $e = (v, v')$ be one such edge. We add e to our partial result for computing MST.
4. Using Optimal substructure property, we contract G with respect to e to obtain G/e and we know the MST of G includes the MST of G/e. In G/e, vertices v and v' are merged into one (let us call the merged vertex as vv'). Now our objective is to find the MST for G/e.
5. We repeat the process of creating a cut, now with $A = \{vv'\}$ and B containing the rest of the vertices of G/e.
6. Again, by greedy choice property, we know that the minimal weight cross-edge must be part of some MST of G/e. We add such an edge to our partial result.
7. The process continues (till contraction leads to a graph containing just one vertex) by contracting with respect to the newly added cross-edge.

The algorithm is realized as follows using minheap implementation of priority queue, which keeps track of the minimal weight cross-edge that connects a vertex in group B with a vertex in group A.

```
Input G = (V, E, wt)

1. Initialize d(v) for all vertices to infinity, and v.parent = undefined;
2. d(s) = 0; s.parent = 0;  // s is some arbitrary vertex
3. add all vertices to a minheap H with key being d-value
4. Initialize v.inQ = true for all vertices.
5. while H != empty
6.   v = extractMin(H);   // this is a vertex getting added to group A
7.   v.inQ = false;
8.   for all (v, v') in E and v'.inQ == true
9.       if (d(v') >  wt(v, v') then
10.           d(v') = wt(v, v')
11.          v'.parent = v;
12.          updateHeap(H, v', d(v'))
```

Notice the difference between above algorithm (MST) and the single source shortest path algorithm. The only difference in lines 9 and 10, where the d-values are being updated. In case of MST algorithm, the d-values indicate the closeness of vertices to any vertex in the partial MST (any vertex in group A) and in case of single source shortest path algorithm d-value indicate the closeness of vertices to the source vertex. Hence, for MST, the d-value of a vertex is updated based on the weight of the edge that connects the vertex to some vertex in group A; while for single source shortest path algorithm d-value of a vertex is updated based on the sum of the weight of the edge that connects the vertex to some vertex v (whose shortest distance from source is already known) and the distance of v.