CS5110 - Final Project

Hermès Henry & Daniel Murrow

Fall 2024

1 Prim's Algorithm

Prim's Algorithm is one for finding the Minimum Spanning Tree (MST) of a connected graph.

High-Level Pseudocode

1. Initialize the MST based on an arbitrarily chosen Node in the graph
2. Consider all edges from the graph **not** in the MST that connect to the Nodes in the MST
   - ⋆ To avoid cycles, we make sure to check if the edge we are considering connects to a Node that is not already connected to by another edge; i.e., only connect to a Node that is in the graph but not in the MST.
3. Select the edge with the lowest weight and add the connected Node to the MST
4. Repeat Step 2. and Step 3. until the MST contains every node from the original graph

Time Complexity

A simple implementation of this would initialize lists of edges and Nodes; For each of the nodes N, we look at each edge connected to it to find the minimum, leading to a time complexity of $O(N^2)$. We can improve this by using a priority queue instead of a list to keep track of our edges E and Nodes N. In a priority queue, adding and removing edges takes $O(\log N)$ and each edge is only ever processed once which leads to a total time complexity of $O(E \log E)$.

Low-Level Pseudocode

Algorithm 1: Prims(G)

Input: Undirected Connected Graph G
Output: Minimum Spanning Tree

```
// Step 1.
start ← randomNode(G)
MST ← new Graph(start)
// Initialize the PriorityQueue
pq ← new PriorityQueue()
for each edge e in start.edges do
    pq.push(e)
// Repeat Steps 2 and 3 until Nodes in MST are the same Nodes as in graph
while mst.Nodes != graph.Nodes do
    edge = pq.pop()
    // Make sure edge is not connected to a Node alreading in MST
    if edge.link_node not in mst.Nodes then
        mst.add_Node(edge.link_node)
        mst.add_edge(edge)
        // Add every edge of this new node to the priority queue
        for edge_new in edge.link_node.edges do
            if edge_new not in mst.Nodes then
                pq.push(edge_new)
// Finally, return the MST once the while loop breaks
return MST
```

Testing

In order to test my algorithm, I set a few simple criteria:

1. Does the MST contain the same number of nodes as the graph?
2. Does the MST contain the minimum possible number $(V - 1)$ of edges?
3. Is the MST a connected graph?
4. Is the sum of all the weights in the MST equal to the sum of all the weights in networkx's built-in MST function?

2 Maximum Clique Approximation

3 Girvan-Newman

4 Bellman-Ford