Alex Yang
Rodrigo Palmaka
Bilal Syed

CS 170 Project Reflection

In this project, we were tasked with discovering a tree with small average pairwise distance between its vertices under the constraint that: for all vertices "V" in the overarching graph G, either that vertex is in the tree or is adjacent to (a neighbor of) a vertex in the tree.

Our first approach to this which we knew was NP-complete was to run a minimum spanning tree algorithm on the graph which by definition would encapsulate all the vertices in the graph. However, this algorithm didn't take advantage of the condition that vertices in the graph can be neighbors to (rather than actually in the tree) nodes in the tree, which therefore increased our average pairwise distance by including every vertex in the graph. And so, we sifted through homeworks to try to look for something more intuitive, and we came across the minimum dominating set problem in Homework 12. From the homework solution, we knew this approach was NP-Complete. And so we ran Networkx's embedded method for a minimum dominating set (MDS) by passing in the graph, which would return the set of vertices that captures an appropriate minimum dominating set of the graph. Following this, we would have to create a subgraph of these vertices to include pathways and edges connecting the vertices from the overarching graph. And so we chose one of the vertices in this MDS vertex set as the source node to run Djisktra's on in order to connect the MDS vertices using the least graph weight possible. And so, we ran a loop which returned the shortest pathways from this source node to every other node in the MDS vertex set. This would ultimately yield us our "shortest paths" subgraph with optimal intermediate nodes from the overall graph to connect the vertices in the MDS set. Lastly, we would run a minimum spanning tree on this subgraph to determine the final minimum dominating set. However, we noticed that we could do better. We noticed that the order in which the dominating set algorithm we used chose vertices to check determined the final minimum dominating set it would return (since there can be multiple dominating sets). And so we wanted to instantiate an ordering of the vertices based on a degree heuristic we came up with.

Due to this notion of ordering, we wanted to calculate a weight for each vertex in the graph which would determine the order by which vertices were checked to be included in the dominating set or not. And so we did this with our method "deg_heuristics(G, v)" in our utils.py file. This method would take in a vertex, and calculate its weight based on this formula: (sum of edges adjacent to v) / (degree of vertex v). With a loop, we ran this method on every vertex in the graph, assigning a weight to that vertex based on the heuristic's calculation. Essentially, the vertices with the smaller pairwise distance with their neighbors and greater number of vertices adjacent to them received smaller weights and would be checked sooner for the MDS than vertices with greater pairwise distance with neighbors and less neighbors. This is intuitive

because naturally we want to check vertices with greater bandwidth and less cost for the MDS tree first.

So to summarize, we ran this degree heuristic method on every vertex on the graph to determine a sense of ordering for the MDS algorithm, then we ran the MDS algorithm on the graph with this ordering in mind. This algorithm would then return a set of vertices in the overarching graph which would be included in our MDS. We then ran Dijkstra's from a specified source node in this MDS set to every other node in the vertex set (with a loop) in order to determine the pathways that would make up the subgraph which would connect all the MDS vertices. Lastly, we ran the minimum spanning tree algorithm on this subgraph to achieve a connected tree with the smallest average pairwise distance among the MDS vertex set. Our initial approach of running just an MST on the overarching graph put us at near last place on the leaderboard, but with our new approach of determining an optimal ordering for MDS, running MDS, and then returning an MST on a subgraph connecting the MDS vertex set put as position #152 out of 277 submissions as of 4/3/20: 4:39 PM.