Group (Teamhard): Ziyue Zhou, Thomas Hsiung, Yunhao Luo

Project Reflection

Our algorithm works by starting with a Dominating Set (DS), ensuring a set of vertices that could be connected to all other vertices, while generally being a smaller solution than Vertex Cover, and then looked to try to minimize the cost of connecting the DS. We ranked each node of the DS by the number of neighbors it had in descending order, and then used A* search to find paths between the nodes of the DS by ranked order. We approached the network connection in this manner because maximizing the major nodes with the most neighboring connections seemed like the best way to keep the tree shallow and minimize the average cost between nodes. The methods for DS and A* were the included networkx implementations. We only used home desktop computers for running the computations. Laptops for testing smaller inputs.

When we initially approached the problem, the conditions given by the network connectivity and tree structure made it seem less like a Traveling Salesman Problem. Instead, our first intuition was that it seemed more like a Vertex Cover problem. We did some research into current implementations and approximations for solutions to each problem and found that TSP algorithms seemed largely to hinge on massive sets of Linear Programming, and we were unsure how to tackle that in code, whereas Vertex Cover tended to already have polynomial-time approximations. Those discoveries served to push us toward looking for Vertex Cover-type solutions.

Then we looked for the easy-to-implement/worst-case naive solution as a starting point. Minimum Spanning Tree is a solved problem and guarantees the lowest-weight tree. If we then pruned off the leaves, it would give us the upper-bound on our average distance cost. We did and submitted it out of interest to see where it would land on the Leaderboard, where it ranked 232/291 at the time of uploading. Using that as the starting point, we then tried to optimize by looking for built-in methods of the Python networkx library and any special cases to consider.

We considered clustering nodes together to determine the lowest average cost of each cluster, then merging them until it included the whole graph, but the best way to initially split the clusters seemed arbitrary and hard to optimize, so we abandoned that strategy. Then we looked for special cases to maybe find a base case and begin building up from there. In the unique situation of a supernode connected to all other nodes, the best answer would be that node. So we considered ways to subdivide the problem to look for supernodes of subsets of the graph, but like the clustering issue, the subdividing also seemed arbitrary and could easily miss out on

optimal solutions. We also considered attempting some kind of randomized algorithm, but couldn't think of any methods to reduce the potential pool of random trees or subtrees to make the randomization more valuable without sheer dumb luck.

Then we found the Dominating Set, which was similar to Vertex Cover, but fit the demands of the situation better by ensuring that all nodes would be accounted for, with fewer extraneous nodes than Vertex Cover solutions. Using Dominating Set as a springboard, we started looking for ways to organize the data to minimize the cost of connecting the nodes, and came up with the algorithm we used.