CS170 Reflection: Final Project

Bashyam, Neha
Liu, Kevin Liu
Shen, Anita

Introduction:
The goal of this project was to find a routing that would minimize the average pairwise distance between a number of "towers" where each tower must either be in the routing itself or directly adjacent to a tower that is in the routing.

Methodology:
Our approach to solving this problem is 2 pronged:

Our first solution builds a tree from the ground up, starting with a node that has the highest degree relative to the graph. We believe that this node is the best candidate for the starting point, as a node having a higher degree would allow for better interconnectivity. We then began building a tree using a modified Prim's algorithm where we consider not only the edge weights, but also the degree. The exact formulation used was 1/inverse(degree) + edge weights in selecting the next node from the set of available neighbors. When adding new nodes to our tree, we make sure that they are not redundant (i.e already in tree or does not create any benefit in extending the range). However, if there are no good nodes to add, we will randomly visit a node and go from there. We believe that this modified approach is good because it takes into consideration not just the weights of the edges but also how viable a node is in terms of connectivity by looking at its degree.

This first approach did **not** perform well initially, so we implemented additional pruning and testing after the tree was made and removed additional nodes as we could. This allowed us to slightly (not a lot, but noticeably) reduce the average pairwise distance in some cases. There was also the risk of a infinite recursion loop, as our base case depended on randomness to a certain degree (stopping condition was when all nodes explored, we sometimes randomly choose which node to explore if "stuck" at a leaf). As such, as implemented a system to keep track of our recursive calls and terminate the program when our stack gets too large.

Our second solution was to create an MST from the nodes in the graph to identify a network with the least cost. This ensures that our network is connected and can be further optimized. We then removed leaf nodes from our MST because they were already adjacent to one other node in the graph, and would decrease the overall cost. However, we realized that actually adding leaves with light weights would decrease the average cost. So, we added leaves back as long as the initial cost without any leaves didn't increase. This approach was good because we were able to find a connected MST with a low cost, and further optimize it by including only the necessary leaves. For our algorithm, we just used functions provided in the networkx package, such as the MST function, deep copy, etc.

Combining both of these solutions and finding the T produced that minimized the average distance became our final output, this gave us our optimal_solution at the time.

CS170 Reflection: Final Project
Bashyam, Neha
Liu, Kevin Liu
Shen, Anita

Materials:
We used instructional machine (Hive 15) to run our code. We also used the NetworkX library to help us find the MST and other graph functions. Other imported libraries include Python libraries such as sys, os, operator, random, and copy.

Conclusion: While our solution did not rank objectively well, we learned a lot from these approaches and from reading about other algorithms (like Campo's alg). Given more time (or had we started earlier), we anticipate that further optimizations could be made.