Imam Widodo
Brian Chin

Phase 1 Design Doc

The barebones algorithm is the brute force algorithm, which we use as a baseline model for comparing optimal distances. From the starting node, we go to every neighboring edge-sharing node as well as dropping off m <= n, where n is the number of TAs still in the car and m is a number from 1 to n and determines how many and which TAs. In the next iteration, we go to every neighboring node that shares an edge with the node we picked. Note that the neighboring node includes nodes that we have gone through and with dynamic programming we can solve subproblems given the node we are currently on. As the number of nodes in a graph increases, brute forcing takes more time and is unable to find the result in a sensible time frame. An even more simplified approach would be to randomly select a constant number of nodes from the pool of nodes every iteration in the graph.

Our next algorithm will be based on a reduction to the traveling salesman problem. Using the knowledge that driving back and forth to drop off single TAs is generally inefficient, we will start our algorithm by first clustering homes that are close to each other into nodes using the distance matrices given in our inputs. Based on a cut off parameter, TAs within a distance of each other will be added to their own “node.” Once all TAs are added to a node, create a new graph consisting of our starting location and vertices representing our new nodes. Within this graph, calculate the shortest distances between all nodes through vertices that are NOT homes. Here, we would treat all edges going through a non-home vertex like part of a direct node to node path (i.e. a vertex with 4 incident edges is equivalent to 3 different paths). Finally, once we’ve constructed a graph of JUST nodes and the shortest distances between each, we have generated an instance of the Traveling Salesman Problem.

From here we can basically try running different Traveling Salesman Problem algorithms like Greedy/GRASP, Simulated Annealing, etc. to generate an accurate approximation of the path traveled by the car.

Finally once we have created a route between all of the nodes, we can set about optimizing paths within every node. This can be done using a brute force method (assuming that the number of homes in a node will be small). For nodes of one TA, we can compare traveling to these nodes using our car or just dropping off the TA at the next closest node to walk home.

Reasoning:

Once we have a TSP, we can generate solutions using traditional TSP algorithms. Thus in order to get to this point we can consolidate homes into nodes to reduce our graph’s complexity and also take advantage of dropping off TAs. We subsequently convert all non-home vertices into paths, bypassing the need to visit non-home vertices.