Final Report

Ramiro de Rojas | Calvin Wang | Felicia Trinh

Our first serious attempt at a solver utilized Dijkstra's algorithm to find the shortest path between each home for every permutation of homes possible. In this method, we dropped off every TA directly at their home. From the starting vertex, we ran Dijkstra's algorithm to find the shortest path to the first home. From this home, we ran Dijkstra's algorithm to find the shortest path to the next home and so on. We continued to do this until we reached the last home. Doing this process for every permutation of the homes list ensured that we picked the shortest path between each home and the next one. It also ensured that every TA was dropped off on this path. However, this method didn't take advantage of dropping off TA's to reduce cycle times, so there was still room for improvement.

We combined the Dijkstra method above with a trial-and-error method that considers the three closest unvisited homes to the current vertex. According to a set of probabilities, one of the homes was chosen at random to be the next destination. Initially, each of these three homes were considered equally, but we recognized that this might lead to our solver picking unreasonably long routes more often than not. In the case of the largest graphs, it was in fact almost guaranteed that the solver would eventually choose an undesired path. So instead of equal weights, the three candidates were assigned a probability according to their distance from the current vertex. We ran this trial-and-error function for a certain number of iterations and picked the result with the minimum cost. Even if this method didn't improve our total cost in every instance, in the limit of a large dataset (approx 1000 graphs), it was bound to improve our solution somewhat. And if it didn't, we could always keep the dijkstra solution.

We then took advantage of the walking mechanic by taking into account the existence of "choke points," or subgraphs that only connect to the main graph via one edge. This method resulted in nonzero walking costs, but better total costs. If there was one home in the subgraph, we would drop off the TA at the vertex incident to the edge on the "main" graph. If more than one TA lived in a home in the subgraph, we traversed into the subgraph until we reached a vertex where the TAs' homes are down opposite edges, then drop them off. We recursively applied this algorithm with consideration for multiple connecting edges (comparing edge weights) and cycles in the subgraph.

In our code, we considered the shortest path to the next unvisited home from either the start vertex or the home that was just visited (let's call it "previous path"). For every intermediate vertex between the previous home and the next home, we calculated the shortest path to the next unvisited home and checked if the shortest path to the next unvisited home contains the previous path as a subpath. If it does, then this intermediate node is a "choke point" and we drop these two TAs here instead of at their individual homes. We then move on to the next unvisited home.

Our algorithm would have given better results if we were allowed to run more iterations of it in our solver. So in the end, time was a highly limiting factor in our solutions. Additionally, we did not take into account that other group's graphs were much more rudimentary than ours (i.e. complete graphs) so our solution didn't work as well on their graphs.

A computing resource we used was AWS EC2 instances, though most inputs were still processed locally. We used around 20 hours of the AWS micro servers, but all of the results were discarded since it was processing the results too slowly. Instead we used our local machines to run the solver on all instances.

Figure 1: A nicely drawn graph of our 50.in input file with the path taken by our solver outlined.
Legend:

- *Nodes:* Magenta(outer color): starting location // Blue (outer): nodes visited by car // Yellow (inner): Dropoff Locations without homes // Orange (inner): Home locations
- *Edges*: Red: traversed by car // Cyan: Traversed by walking TA