1 Algorithm Considerations

Pre-Processing vs. Post-Processing

Originally, the algorithms in our design doc depended on pre-processing the graph to simplify it before running TSP. We wanted to prune unneeded and unlikely-to-be-traversed nodes by assigning heuristic values based on the number of homes "further down" the path. This was to help us decide how worthwhile driving to that node would be. During implementation, we found that pre-processing nodes, although easy for us to do visually, was a very difficult algorithm to implement.

On the other hand, post-processing of our return values would mean a heavier load on our TSP algorithm because there would be many more nodes that we would need to travel to. However, we found that in practice, the difference was negligible. Thus, we decided to use post-processing in our algorithm since it would be easier to implement.

Clustering

While looking at the input graphs, we thought that grouping homes with similar traversals would allow us to drop off multiple TAs at once, minimizing energy expenditure. So, we attempted to develop an algorithm that incorporated node clustering.

Initially, we used a k-means clustering algorithm to group nodes. The idea was to start with all clusters of size 1, then slowly increase cluster size and use hill climbing to find the number of clusters with minimum energy expenditure. Then, for each cluster, we would pick the drop-off point by selecting the node with the best shortest path distance to the start and run TSP on each cluster's drop-off location to determine the optimal tour. However, since we found that increasing the size of the clusters only ever increased the cost of the solution, we did not proceed with this approach.

We then tried another node clustering method, which consisted of expanding clusters around homes by greedily adding neighbors based on distance. We noticed that as we expanded the cluster sets, nodes that appeared in multiple clusters were good drop-off nodes for homes that only had one edge. However, this algorithm encouraged walking rather than driving because drop-off locations often ended up very close to the starting node. Further, this algorithm had issues on homes with multiple edges. Thus, we also decided to scrap this idea.

In general, we found that clustering too often encouraged walking over driving, increasing the cost of solutions.

2 Final Algorithm Design Process

Phase 1: Baseline TSP solver

Since driving is usually better than walking (except in the case of a spindly path) our baseline algorithm was to drive to each TA's home and drop them off, deciding the order with the TSP solver from Google's `ortools` library. We first constructed a new adjacency matrix of only the houses and the start node by using `networkx`'s `shortest_path` function, which was then plugged into the TSP solver.

Phase 2: Post-Processing: Removing A → B → A Patterns

We established in most cases that it is efficient to drive the TAs home directly, except when there is a "one-way" path that leads to the home of only 1 TA.

Thus, our first modification to TSP was that when the tour outputted by TSP contains a pattern like A→B→A, it would more optimal to drop the TA who lives at node B at node A instead, letting them walk the rest of the way.

However, this takes care of some spindly paths, but not all, as shown in figure 1. Thus, we then attempted to remove all one-way paths.

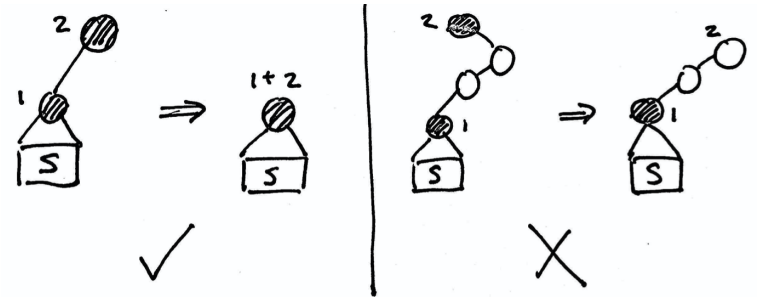

Figure 1: *Here (and in all figures), S denotes the start node and shaded nodes are TA homes. We see that in the first case, the pruning gives the optimal path, but in the second case, the pruning fails.*

Phase 3: Removing All One-Way Paths

In the last phase, we removed all one-way paths in our initial tour, thus accounting for the cases where phase 2 failed to find the optimal solution on a spindly graph. However, we took care to not remove a spindly path if multiple TAs were going to walk down the same path, as we realized that would be less efficient than driving them. This is illustrated in figure 2.

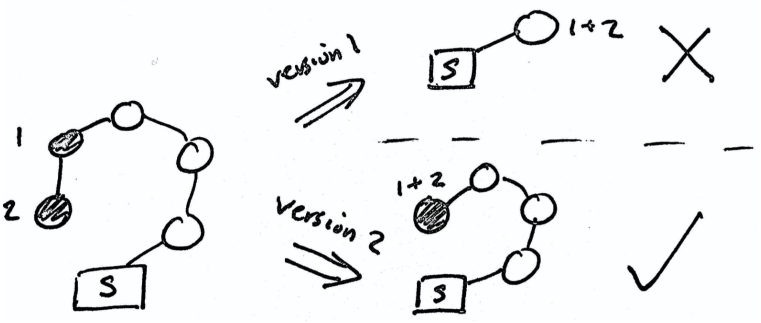

Figure 2: *If we assume uniform edge weights of 1, the cost of version 1 is* $7\frac{2}{3}$ *while the cost of version 2 is only* $3\frac{1}{3}$.

Extra Optimizations:

While inspecting the results for the solver, we focused on outputs that spent drastically more energy on walking than driving and noticed that some solutions involved walking when driving to an (initially further) location and dropping TAs off there was cheaper overall. One example of this is group 184's inputs, which we solved manually.

3 Final Thoughts and Future Improvements

Overall, while the project was NP-hard, we really enjoyed being able to think about different possible solutions. Something that we noticed when drawing out smaller graphs was that our eyes would almost always recognize an optimal or close to optimal solution because we could physically "look ahead", while our algorithm could not, forcing it to take less optimal paths.

In the future, one possible improvement is having our solver look at possible moves up to a certain number of steps ahead, deciding which path our final path should take based on which "future move" uses the least amount of total energy, similar to a game tree. We also only had time to optimize for "spindly" paths, but there are other circumstances at which it would be optimal to walk rather than drive, such as if many TAs are grouped together strongly (see figure 3). Since the tour may not contain a spindle, we do not optimize for this case currently, but an improved algorithm would be able to.

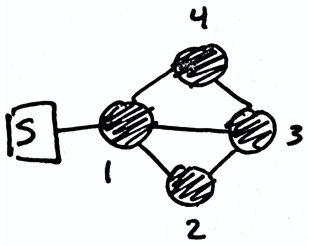

Figure 3: *Here, our final solver returns $S \to 1 \to 2 \to 3 \to 4 \to 1 \to S$, dropping each TA off at their home. However, the optimal solution is $S \to 1 \to S$, where TAs 2, 3, and 4 walk from node 1.*