CS 170 Design Doc

Jasper Hu, Timothy Kha, Shangen Lu

Naive:

Initially, we attempted to come up with a brute force approach. At each vertex we wish to consider two things: how many TA's to drop off, and which vertex to go to next. For each TA dropped off, we must consider their shortest path home (from Dijkstra's), and we also must determine which combination of TA's would be optimal for that specific vertex. The runtime for this approach would be at least O(n!), if not more, as we have to try all combinations of vertices and TA's. From here, we considered making optimizations but ultimately decided against this extremely convoluted approach.

Dynamic Programming and TSP:

We were told from homework 11 that a reduction exists between Drop the TA's Off and TSP if the car's energy expenditure was equal to 0.5; thus it would be natural to see if a reduction existed between car's energy expenditure variant with ⅔ and TSP. We tried combining vertices, adding fake edges, and other graph mutating techniques but none yielded any results. Nonetheless, it was promising and we may return to this technique.

Greedy Algorithms:

We also considered an algorithm that simply takes the nearest drop off after each vertex, which apparently gets screwed quite spectacularly when the graph is in a diamond arrangement.

Approximation Algorithms and Christofides' Algorithm:

This approach yielded the fastest results and seems relatively easy to code; however, we are never guaranteed the correct optimal result, and we are bounded 1.5 * opt. We would want to change the parameters to have edge weights of 4/3 so that it encapsulates the driver going to and from. Furthermore, we found out that there are some instances where approximation algorithms perform extremely poorly It is curious though, if we could get a brute force solution or dynamic programming solution that guarantee optimality to run within the allotted time frame, it would be more advantageous to do exactly that rather than be bounded.