General Design

The Problem and Approach Taken by Code

**The Problem**

In this project, I am solving the problem of finding an optimal round-trip path through a series of addresses/locations inputted by the user according to the traveling salesman greedy algorithm.

**Approach Taken by Code**

*Address Order*

My code takes the following approach to solve it. Once the user enters a city, the program will use Google Maps API to calculate the distance from the new city to every previously entered city. Hence, a dictionary containing tuples of cities as keys mapping to the distance between the pair of cities is how I store the “graph” with vertices and edges. Next, the program will apply the greedy algorithm to the cities to determine the optimal order in which to visit them.

*Driving Directions*

The program retrieves the driving directions of how to go through the intermittent steps from Google Maps API as well. However, since the directions are in a HTML format, I used a function (taken from http://stackoverflow.com/questions/753052/

from by stackoverflow user Eloff) that parses a string of HTML and removes the tags to display the text in a cleaner format.

*Map*

The program also displays a visual map of the route in the web browser. To do this, I found sample code from the following website: <http://aspforums.net/Threads/151361/Google-Maps-V3-How-to-draw-route-between-two-locations-points/>.

With some modification to the above code, I managed to get the code to draw the route through a series of locations on a map that opens in a web browser. The route also closely matches the actual roads, a feature that I was not able to get working in pygmaps (python wrapper for displaying google maps).

When the user clicks the map button, the html file containing the code in the website is modified – the locations the user entered are entered in the list of “marker dictionaries” in the code. Each marker dictionary contains keys: title, lat, lng, and description. The values correspond to the address, lat, and lng of the actual locations input by the user. Then, the html file is opened in a web browser.

The program uses Google Places API to find nearby gas stations to any location on the user’s route.

*Traveling Salesman Algorithm Demo*

To make the Traveling Salesman Algorithm Demo, first, I used motionless.py (creates static google maps) to display a map containing markers representing various addresses the user entered in.

Then, I needed to draw nodes over the map corresponding to the locations of the markers on the map. To do this, I looped through every pixel in the map and identified those that were “red” (high r value, low g, b values). The points that were “red” were then stored as a list of possible marker coordinates.

Then, I “grouped” these points in successive lists, where each list was to represent an individual marker. To do this, I had to group together “red” pixels that were reasonably “close” to each other (within 1 marker length away from each other). One clear limitation of this approach is that when markers overlap on the static map, the algorithm breaks down. As a way of at least partially dealing with this, I decided to treat markers that overlap or are at least very close to each other as one collective marker (since if two locations are very close to each other, the distance between them is negligible compared to the distance of the overall route, and it is logical to treat them as one location together).

From there, I would pick a “representative point” in each group (marker) to represent the location of the marker. I drew blue ovals at these representative locations.

Finally, I applied the TSP greedy algorithm to the blue ovals and used the Euclidean distance between blue ovals as the distance metric to create the simulation.

*Gas Stations*

I used Google Static Maps API to find a list of gas stations around a specific city selected by the user. I used motionless.py to display a static map depicting this information.