Navigating Urban Environments Using Rideshare Services

Gerald Wang, Mark Estiller, Calvin Ji, Dharma Ong

Tuesday, March 30, 2023

Problem Description and Research Question

We originally planned to investigate the problem domain of sentiment analysis, but we decided to change our problem domain to urban transportation. **Our goal is to develop an algorithm that optimizes the path to take from one neighborhood to another neighborhood, depending on whether the user wants to minimize the *time*, *distance*, or *cost* of the rides**. We also wanted to see how much information we could extrapolate, i.e. predict the sizes of the neighborhoods in the data.

Our chosen problem of finding the best path to take to go from a starting neighborhood to an ending neighborhood is important for several reasons. Firstly, it can help in improving the efficiency of urban transportation, reducing traffic congestion, and minimizing the time and cost of travel. Secondly, it can aid in identifying areas of the city that are poorly connected or under-served by transportation, allowing for better planning and resource allocation. Finally, our chosen problem is a challenging and interesting one, which requires the use of advanced algorithms and techniques from the field of graph theory and optimization.

We chose our goal because we believe that it has the potential to make a significant impact on the lives of people living in urban areas, especially in large and densely populated cities. Additionally, we were drawn to the technical and intellectual challenge that this problem presents, and we believe that by working on this project, we will be able to develop our skills and knowledge in several areas, including data analysis, algorithm design, and optimization.

Data Set

For our project, we used ”My Uber Drives” from user Zeeshan-Ul-Hassan Usmani. The data encompassed his Uber drives in 2016 (1,175 drives total), and it was presented as a csv with the following columns going from left to right: start date, end date, category, start, stop, number of miles, and purpose. ”Start” and ”stop” referred to the pickup and drop-off neighborhoods for the Uber ride. ”category” referred to the category of the trip, for example, business, personal, etc., and ”purpose” referred to the purpose of the trip, for example, meals, errands, etc.

Computational Overview

As per our goal, we wanted to see if we could optimize for certain key factors of an Uber ride from one destination to another, such as time, distance, and cost.

To start, we needed to represent our data as a graph. Graphs play a central role in our project because the states that the Uber ride data encompasses are inherently graphs - each neighborhood connects to another, and we have the distance and time it took for each ride to connect them. Thus, we represented each unique neighborhood as a node in the graph, and represented the aggregate of rides between two neighborhoods as a weighted link by looking at factors such as the time, distance, and cost between the start and stop neighborhoods. We created the graph from our data set, and we did so through the following process: we passed the data set into read_csv, passed that result into get_avg_times_and_miles, passed the result from get_avg_times_and_miles into get_avg_costs. We then combine those 2 dictionaries using combine_dict_times_miles_cost, and finally passed the resulting dict into our create_graph function.

To get accurate and necessary data for our project, we filtered out the ”category” and ”purpose” columns from our data set, and this can be clearly seen in the read_csv function. We also noticed that some pickup and drop-off locations were unknown, so we filtered those out too.

Since there may be multiple trips with the same endpoints, we computed the average time, average cost, and average distance for the trip, and stored them in their respective links so that every link in our graph that connects 2 neighborhoods is weighted by these averages.

Next, we wanted to optimize paths between 2 neighborhoods. We used Dijkstra's algorithm, which was one of the harder algorithms that we implemented. Dijkstra's algorithm is a popular algorithm in computer science used to solve the shortest path problem in a graph. The goal of the algorithm is to find the shortest path from a starting node to all other nodes in the graph.

The algorithm works by maintaining a set of unvisited nodes, a set of visited nodes, and a set of tentative distances to each node. Initially, all nodes are unvisited and the distance to each node is set to infinity, except for the starting node, which has a distance of 0.

At each step of the algorithm, the node with the smallest tentative distance is selected and marked as visited. Then, for each of its unvisited neighbors, the algorithm calculates a tentative distance by adding the length of the edge connecting the current node and its neighbor to the current node's path length so far. If this tentative distance is less than the current recorded distance to the neighbor, the neighbor's tentative distance is updated.

The algorithm repeats this process until all nodes have been visited, and the tentative distances for all nodes have been calculated. The resulting set of tentative distances is the shortest path from the starting node to all other nodes in the graph. Dijkstra's algorithm can be applied to minimize other attributes as well, and in our case, our `find_best_path_dijsktras` method can additionally minimize path based on cost or time.

We also created the 3 functions below: `run_find_best_path_for_time`, `run_find_best_path_for_distance`, and `run_find_best_path_for_cost`. These 3 functions are similar to Dijkstra in function (i.e. finds the shortest path), but we implemented it anyway because we wanted to try multiple coding approaches to solve the same problem. Doing so allowed us to compare the functions as well as help us in case we ran into problems implementing the functionality we wanted using an approach.

Additionally, we wondered if we could extrapolate any conclusions about the sizes of the neighborhoods from our ride data (i.e. predict the sizes of neighborhoods). We implemented the estimate_neighborhood_size method and assigned each neighborhood's size to their respective size attributes.

In order to visualize our results, we used the **networkx** and **matplotlib** libraries. We did this by first converting our graph to a **networkx** graph in the `convert_to_nx()` function. This creates networkx nodes (`{networkx obj}.add_node`) of the neighborhood names and assigns them a size attribute based on 50 times the natural logarithm of respective size attributes + 1. The 50 scales the nodes to make sure that they are visible, and the log keeps large neighborhoods from completely covering the screen. If a path is passed in, we convert it to a list of sets of endpoints, and then use `{networkx obj}.add_edge` to add the edge with a color attribute of red and a weight attribute of the corresponding link's cost. If not, we render all edges as black with the respective link costs attached. Then, by using networkx's `spring_layout` and `draw` function, we found the positions of each node and plotted them. To visualize the graph more clearly, we changed the parameters in the `draw` functions to customize node sizes, node colors, font sizes, font colors, and edge widths. Finally, we display the resulting networkx graph using matplotlib.pyplot's `show()` method.

Instructions for Running the Project

1. Download all files from our MarkUs submission.
2. Install all required libraries in `requirements.txt`
3. Install the required data set zip file, extract all, and create a folder called 'data' to store it in.
4. Make sure that the 'data' folder is at the same depth as the other python files downloaded from the submission on MarkUs.

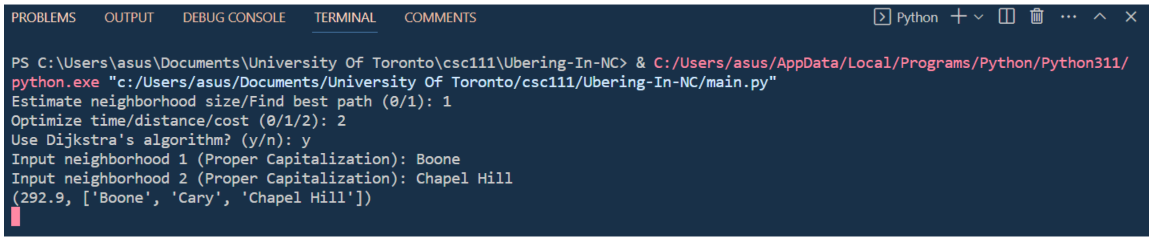
5. Run `main.py`

Features: ALL_NEIGHBORHOODS is a global function in main.py that contains a set of all of the neighborhood names in our chosen dataset. These neighborhood names will be useful for running many of the functions in the main.py file such as `run_find_best_path_for_time`, `run_find_best_path_for_distance`, and `run_find_best_path_for_cost`.

All the names in ALL_NEIGHBORHOODS can be used in the functions in main.py when appropriate. The most important function in main.py is `runner`, because it is an interactive function that allows the TA to run whatever function they want with whatever inputs they want.

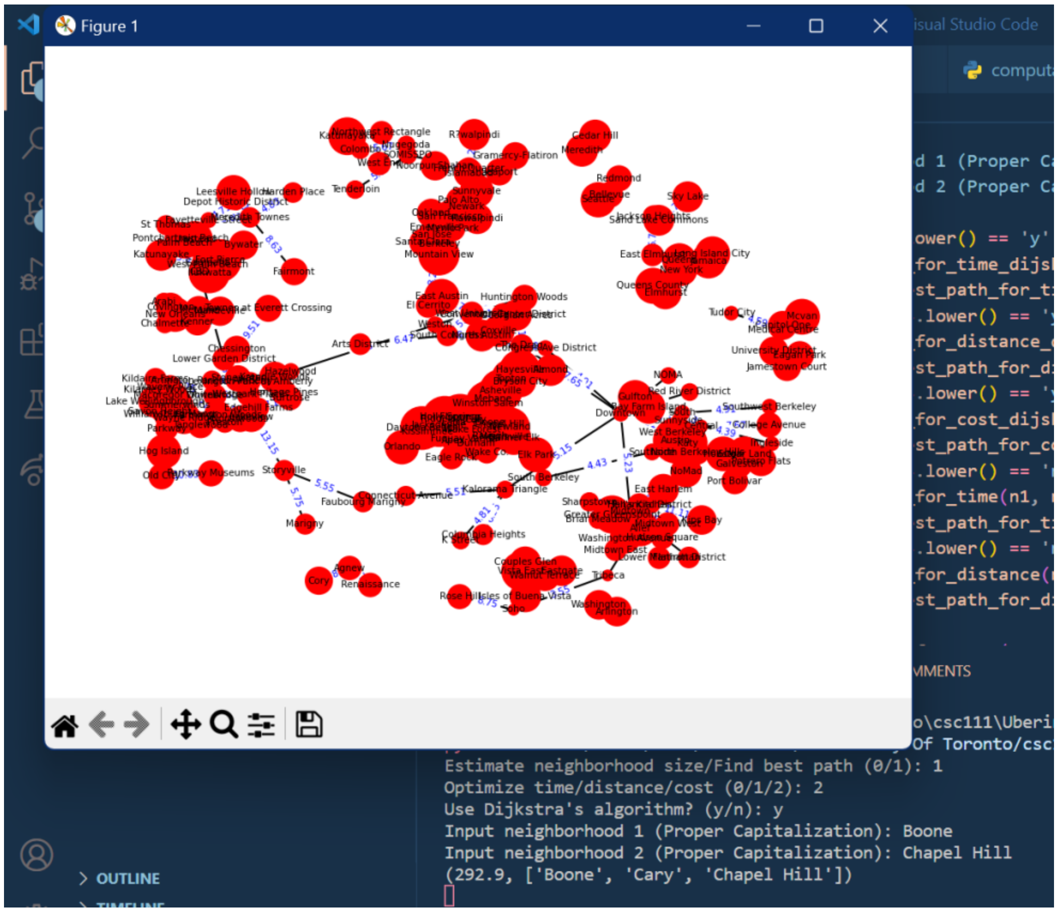
The file is interactive, and the user will be asked to fill in the prompts. The file being interactive allows the user to have greater control over what functions they run and the parameters they pass into the function.

Here are some pictures that illustrate the process of running main:



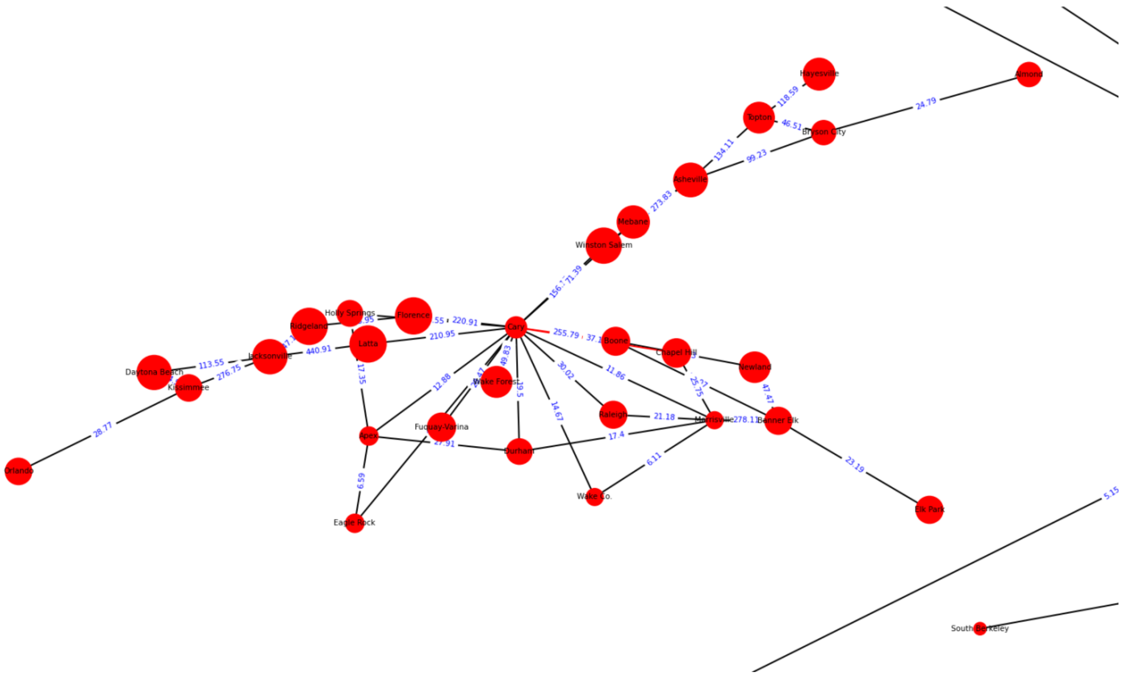
```
PS C:\Users\asus\Documents\University Of Toronto\csc111\Ubering-In-NC> & C:/Users/asus/AppData/Local/Programs/Python/Python311/
python.exe "c:/Users/asus/Documents/University Of Toronto/csc111/Ubering-In-NC/main.py"
Estimate neighborhood size/Find best path (0/1): 1
Optimize time/distance/cost (0/1/2): 2
Use Dijkstra's algorithm? (y/n): y
Input neighborhood 1 (Proper Capitalization): Boone
Input neighborhood 2 (Proper Capitalization): Chapel Hill
(292.9, ['Boone', 'Cary', 'Chapel Hill'])
```

Start by answering the prompts.

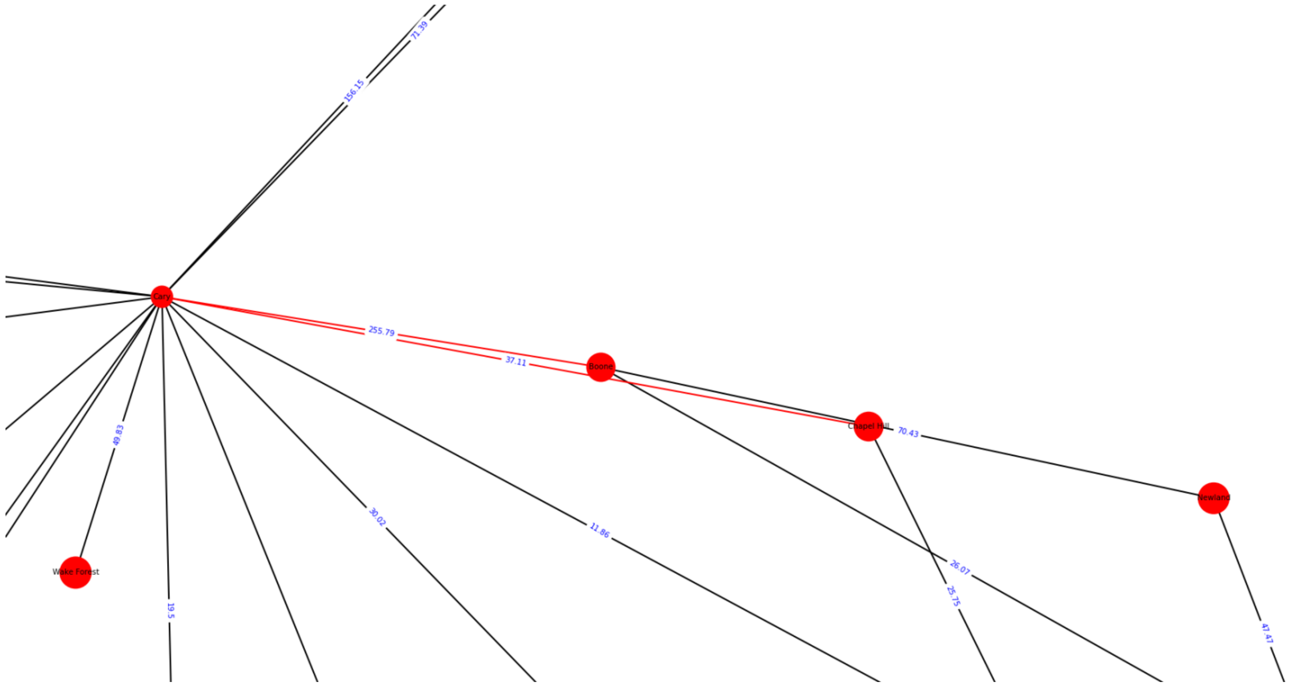


```
Estimate neighborhood size/Find best path (0/1): 1
Optimize time/distance/cost (0/1/2): 2
Use Dijkstra's algorithm? (y/n): y
Input neighborhood 1 (Proper Capitalization): Boone
Input neighborhood 2 (Proper Capitalization): Chapel Hill
(292.9, ['Boone', 'Cary', 'Chapel Hill'])
```

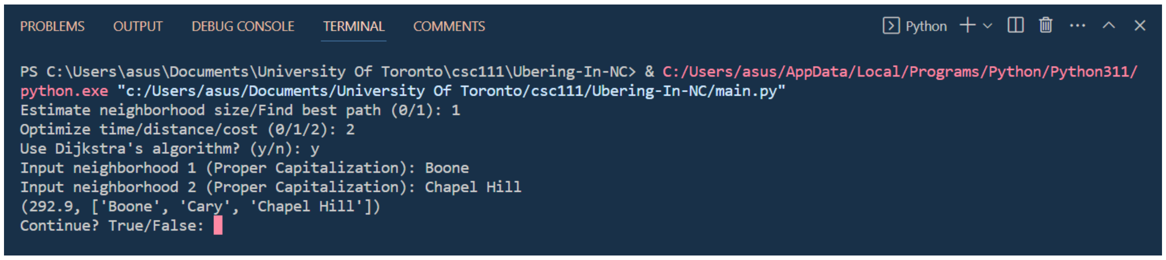
Click the magnifying glass on the bottom left to zoom in.



The user could select the area of the network to analyze by clicking and dragging across the screen. In this case, we decided to zoom in somewhere in the middle, where the start and end neighborhoods are located.



We zoomed in even more to analyze the behavior of this network. Notice that some of the links are highlighted in red. This represents the optimized path.



```
PS C:\Users\asus\Documents\University Of Toronto\csc111\Ubering-In-NC> & C:/Users/asus/AppData/Local/Programs/Python/Python311/
python.exe "c:/Users/asus/Documents/University Of Toronto/csc111/Ubering-In-NC/main.py"
Estimate neighborhood size/Find best path (0/1): 1
Optimize time/distance/cost (0/1/2): 2
Use Dijkstra's algorithm? (y/n): y
Input neighborhood 1 (Proper Capitalization): Boone
Input neighborhood 2 (Proper Capitalization): Chapel Hill
(292.9, ['Boone', 'Cary', 'Chapel Hill'])
Continue? True/False:
```

Once the display is closed, the user will be prompted and asked whether or not they want to continue performing calculations.

At the bottom of our main.py file, we also left a few print statements with neighborhood size estimates and their actual sizes commented out.

Changes to Our Project Plan

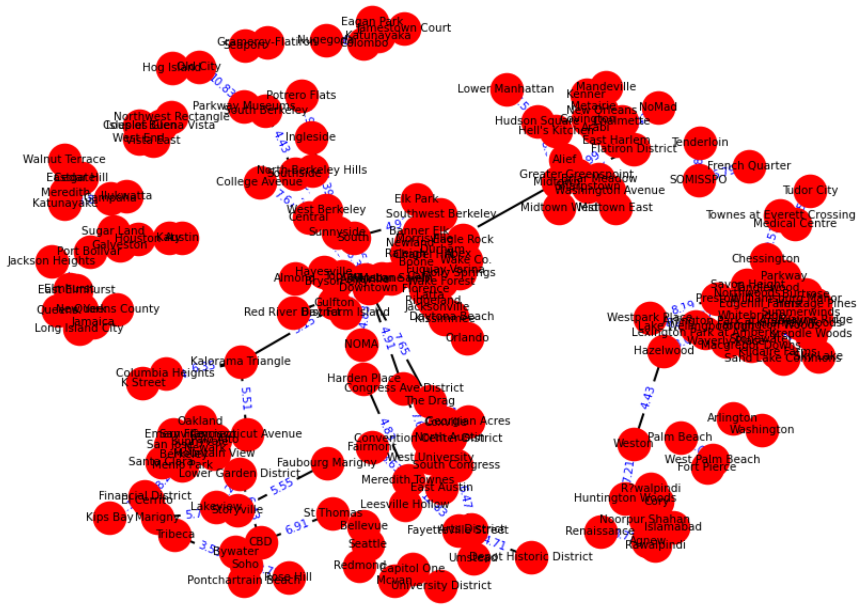
Based on TA feedback, our original idea of sentiment analysis of movie reviews did not involve trees or graphs enough. While the TA suggested we created a review generator, we realized that we wanted something more meaningful that could be applied to the real world. After much discussion within the group, we decided to analyze Uber rides in many different regions to see if we could optimize for certain key factors of an Uber ride from one destination to another, such as time, distance, and cost, as well as if we could try to estimate neighborhood sizes.

Discussion

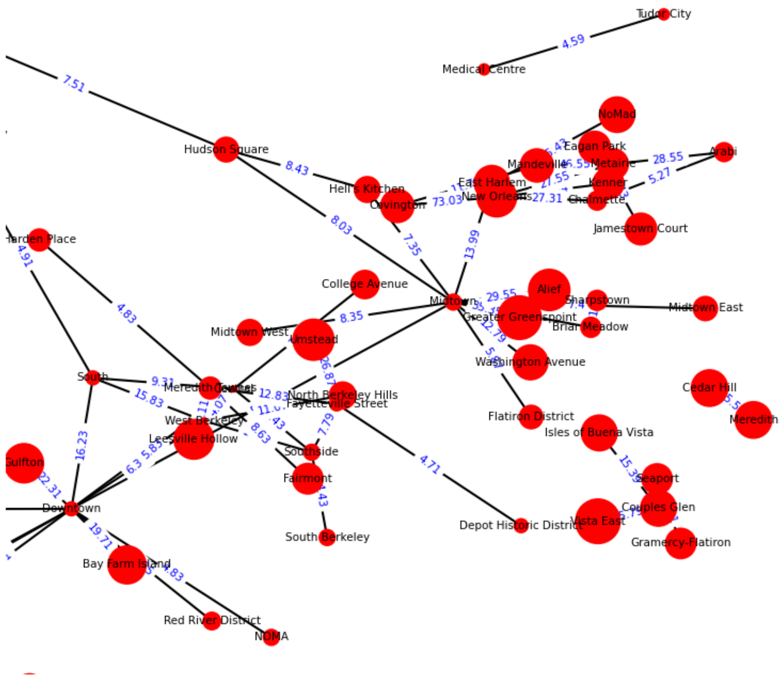
Going back to our research question, we can see that the functions `run_find_best_path_for_time`, `run_find_best_path_for_distance`, and `run_find_best_path_for_cost` can help optimize the path we should take if we want to minimize the quantities mentioned in the function header. This is useful for customers that use ride-share apps such as Uber, because it may allow them to save money and time, depending on what their needs are for any particular travel. These algorithms may be especially useful for someone who is trying to save money or trying to make it to an important meeting on time. These may also be very useful for vacation planning, as people who are on vacation may want to see as many sights as possible, which may be spread out over a large distance. People who are on vacation may also have a budget they want to stay under and may want to save money, and they may also have limited time at their vacation destination, so they may also want to use their time efficiently to make the most of the time they have. In these situations, it is clear how the aforementioned functions above could be of great help.

Some limitations we encountered while trying to display the graph was networkx's implementation for `spring_layout`. According to its documentation, `spring_layout` ”Position[s] nodes using Fruchterman-Reingold force-directed algorithm.” In other words, it ”[treats] edges as springs holding nodes close, while treating nodes as repelling objects.” When we used this function, networkx would output a valid graph. However, many of the nodes were overlapping, and the text we used to name the nodes would overlap and block each other out.

After changing `spring_layout` and `draw`'s parameters, we were able to customize the colors and font sizes, which would help create a more aesthetically pleasing graph.



Another limitation we found was when trying display the neighborhood size; to fix this we changed the size of the neighborhood in the display by using the functions we made to estimate neighborhood size. Furthermore, since the visualization is interactive, and one can zoom in and out of the graph to see certain features more clearly, which helped to negate many of the problems mentioned earlier.



As for further exploration, we considered implementing a function that would plot the coordinates of the neighborhoods, allowing us to map out the structure of a region's neighborhoods based on just a dataset of Uber rides in the area. One library that would help with this calculation would be the scikit library, and the `MDS` (Multi-Dimensional Scaling), which would find the coordinates of the nodes based on the distances between one nod and all other nodes. However, figuring out the structure of the neighborhoods using only the Uber data in the dataset was much harder than anticipated, partially because we did not have enough data, and we ended up scrapping the idea because we would have needed much more time in order to complete it. However, there is definitely room for further exploration, and we may come back to this idea in the future for our own personal exploration.

One other fix we could have done would be to add a function that easily finds the selected neighborhoods. In our current implementation, after displaying the graph, it is quite difficult to locate the neighborhoods we've chosen, especially if its size was smaller than the others. Even though the graph is interactive and allows the user to zoom in, it is difficult to see where exactly the neighborhoods we chose are.

In our estimation of neighborhood sizes, our estimate for 'Midtown' is 2.544 while the actual size is 2.254 mi^2 according to Wikipedia. This is a 12.0884% difference, which is pretty accurate considering our computation strategy of taking the averages of all of the links. Other neighborhoods that had less links, however, were very inaccurate.

However, despite the issues and setbacks discussed above, we were able to complete the project in a timely manner and achieve most of our initial objectives, and we believe as a group that the project went fairly well. We think that although there is room for improvement and future further exploration, the project was a success.

References

Usmani, Zeeshan-ul-hassan. \My Uber Drives." Kaggle, 23 Mar. 2017, https://www.kaggle.com/datasets/zusmani/uberdrives.

Ohnesorge, Lauren. "Uber Defends Price Surge That Charged Durham Man $455 on Halloween." Bizjournals.com, 4 Nov. 2014, https://www.bizjournals.com/triangle/blog/techflash/2014/11/uber-defends-price-surge-durham-man-455-halloween.html.

Felix, director. Dijkstras Shortest Path Algorithm Explained | With Example | Graph Theory. YouTube, YouTube, 26 Sept. 2020, https://www.youtube.com/watch?v=bZkzH5x0SKU&amp;ab_channel=FelixTechTips. Accessed 2 Apr. 2023.