Ride Sharing Project Report

Team - 1

Benito Alvares
balvar30@uic.edu

Harish Venkataraman
hvenka8@uic.edu

Karan Davanam
kdavan2@uic.edu

Prajwal Kammardi
pkamma2@uic.edu

Spring 2021

1 Introduction

1.1 Problem Statement

Taxis have become a huge part of a city ecosystem. Metropolitan cities like New York suffer huge traffic congestion. New York ranks 4^{th} in the most congested cities in the US[3]. One of the factors for congestion is car ownership. It becomes imperative with companies that offer taxi services to address this issue by establishing carpooling programs to reduce the number of vehicles on the roads. Ride sharing has numerous other benefits such as reducing the parking demands, eliminate vehicular emissions and creates less stressful commutes. Therefore ride sharing companies such as Uber and Lyft are quickly becoming the favoured transportation option over taxis in population dense cities[4].

1.2 Objective

The project aims to develop a real-time ride sharing system to effectively develop an algorithm to optimally share rides between users. The secondary aims are to reduce the fuel consumption for the companies offering rides by reducing distance per customer and also increase revenue per ride for the companies.

2 Data

2.1 About the Dataset

1. The dataset used for the project is taken from the NYC taxi & Limousine Commission.
2. The yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.
3. For the project, the Yellow taxi trip data was chosen and for the year 2019.
4. The NYC TLC also provides shapefile of the New York city. This shapefile is used to map zone geometry to the Location IDs specified in the dataset. The shapefile format is a geospatial vector data format for geographic information system (GIS) software.

2.2 Preprocessing

The data is cleaned, dates are parsed, invalid values are removed, such as,

1. Only the records with the condition $0 < passengers < 3$ are retained. The rest, discarded.
2. $trip\ distance <= 0.1$

3. Pickup and Drop-off zones with same IDs are discarded
4. Zone IDs with 264 and 265 are discarded as they are mapped to N/A
5. Pickup times and drop-off times with equal values

This preprocessed data is then inserted into the PostgreSQL database. The data is indexed by the pickup-times to have faster queries.

2.3 Working with the Zone IDs

1. PostGIS is used to load the shapefile into the database using the appropriate projections.
2. Each zone is associated with a geometry. To map the zones to latitude and longitude, the centroid of the geometry is taken, which represents the location of that zone
3. The centroids are then used to compute the interzonal distance between different zones

2.4 Assumptions

1. The maximum occupancy in a ride is **3**.
2. The maximum number of rides merged is **2**.
3. The passengers do not have any social scores associated with them.
4. The request time of a ride is the same as its existing pickup time.
5. For a request, all the possible rides for merging arrive after it in the pool.
6. Each passenger in the pool can wait a limited amount of time to be matched. This is called delay time. This assumption was simplified in the implementation by making, *delay time = remaining time left in the pool window*
7. The pooling window times considered are **5, 7, 10 minutes**.
8. There is an infinite amount of taxis to be merged at the source of the requests.

2.5 Assumptions Discarded

1. *Only x% of riders in a pool are ready to share per simulation* - This was discarded as the number of requests in a pool were very low for the time windows that were used.
2. *The matching is done on a first come first serve basis* - This was a redundant assumption as the graph based approach inherently handled it

2.6 Technical Components

1. The application was built using Python 3.6, Pandas & Numpy.
2. The data was persisted on PostgreSQL database. PostGIS extension was used to store the zones and their geometry.
3. **Open Source Routing Machine** (OSRM) [2] along with OpenStreetMap data was used to compute the distance from the source to destination. As the API online offers limited amount of queries per session, a Docker instance of the server was downloaded on the local machine. An adjacency matrix containing the distance between the different zones was pre-constructed and cached for higher throughput.
4. NetworkX [1] is a Python library that provides graph based abstractions for applications in Python. The library provides APIs for multiple graph algorithms. The main algorithm of interest is the **maximum weighted matching** algorithm which is used for merging the rides.

3 Implementation Details

3.1 Approach

A very intuitive way of thinking about ridesharing between rides in a pool is by using graphs. A graph is able to model relationships between rides, with each node being a single ride. The favorability of merging two rides is modelled by using the edges between two rides and their corresponding edge weights. This allows to encode the assumptions and merging priorities numerically in the edge weights such that candidate rides that are more favourable for matching have a higher edge weight than those that are less favourable. The approach first considers all the rides in a particular pool and constructs a weighted, undirected graph between them such that each node in the graph is a single ride. Then, a custom logic is used to define edges and their corresponding weights between the nodes in such a way that they respect the assumptions and priorities. Finally, to retrieve the merged rides, **maximum weighted matching** algorithm is used. It is a bipartite matching algorithm that outputs pairs of rides that have the maximum weights between them, giving the rides capable of being merged.

3.1.1 Edge Weight

For computing the edge weights between any two rides, a custom formula is used which is capable of prioritizing whatever is deemed necessary for ridesharing. The rides are to be matched such that the overall distance traversed is less and the waiting time between pickups is less. So the edge weight is defined as -

$$W = W_{distance} + W_{time}$$

Distance Weight As mentioned above, the rides are merged together such that the distance traversed by the taxi when ridesharing is much less than the distance traversed when using individual rides. Let D_A and D_B be the distances traversed by two individual rides A and B. Let D_{AB} be the distance from the pickup location of A to pickup location of B and D_{BA} be the reverse of that. Note that D_{AB} is not always equal to D_{BA} because the on-ground route may differ based on the direction of travel.

The total distance traversed without ridesharing is -

$$D_{total} = D_A + D_B$$

The distance traversed when ridesharing is -

$$D_{min} = min(D_A + D_{AB}, D_B + D_{BA})$$

The distance saved when ridesharing is -

$$D_{saved} = D_A + D_B - D_{min}$$

The above quantity is normalized to get the final distance weight -

$$W_{distance} = \frac{D_{saved}}{D_{total}}$$

$$-0.5 <= W_{distance} <= +0.5$$

$W_{distance}$ is maximum when the rides are picked up from the same location (to LaGuardia) or dropped off at the same location (from LaGuardia). A negative value indicates cases where ridesharing is worse than taking individual rides, for example, when the pickup location is on the opposite sides of the dropoff location.

Time Weight Rides are merged with higher priority if the difference between their request times is less. Essentially, rides should not be merged if the passenger for the first ride has to wait a long time before being picked up. Let T_A and T_B be the request time of two rides A and B. Let $pool\_size$ be the size of the current pool in seconds. Then -

$$T_{AB} = abs(T_A - T_B)$$

Normalizing the above quantity to obtain the final weight -

$$W_{time} = \frac{T_{AB}}{pool\_size}$$

$$0 <= W_{time} <= 1$$

The maximum value of T_{AB} indicates that the rides are requested at the exact same time.

3.1.2 Additional Considerations

- The costs hold true for both LaGuardia as pickup and LaGuardia as dropoff
- The rides in a pool are sorted in the order of their request times. Each ride forms a node in the graph and the edges are constructed between combinations of the nodes
- For example, if the rides arriving in order are A, B, and C, the edges are constructed between (A, B), (A, C) and (B, C)
- In the pre-processing step, the rides where passenger count was greater than or equal to 3, were discarded. But it is still possible to overshoot the maximum occupancy by merging two rides with passenger count equal to 2. So in this case, an edge between these two nodes is not constructed so that the algorithm does not mistakenly merge them
- Maximum weighted edge algorithm ensures that all the rides are merged in the case with even number of nodes, and leaves out one node in the case of odd number of nodes

3.2 Simulation Workflow

Algorithm 1 is a pseudocode for running the simulation. The **stored_procedure** returns rows in the times specified from the database which are filtered with trips that originate and end at LaGuardia. This is done for months Jan to June.

Algorithm 1 Simulation of Taxi-rideshare

```
dataframe ← STORED_PROCEDURE                    ▷ returns the ride details in a specified timeframe
data_split_pu ← FILTER(dataframe[pickup = 'LGD'])    ▷ rides that originate at LaGuardia
data_split_do ← FILTER(dataframe[dropoff = 'LGD')    ▷ rides that end at LaGuardia
for each split do
    for each poolsize do
        Initialize graph G(V, E)                ▷ V are the rides and E are potential matches
        Calculate edge weights using the weight equations
        MAXIMUM_WEIGHTED_EDGE(G)
    end for
end for
return set of edges and unmatched nodes
```

Figure 1 demonstrates the workflow of the simulation. The adjacency matrix derived from OSRM is cached in the main memory and is run only once.

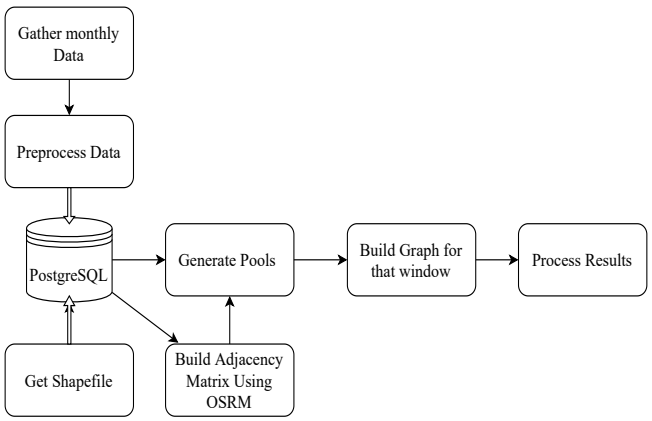


Figure 1: Workflow Diagram

4 Results

4.1 Evaluation Metrics

The project showcases a number of visualizations that focus on the algorithm utilization, savings, computation times along with a breakdown for specific days of the week. The project uses variable pool window sizes of 5, 7 and 10 minutes to evaluate the algorithm's performance across the board. The computation times incurred by the pooling algorithm as it processes data over Jan 2019 to June 2019 are also analyzed .

4.2 Visualization: Distance & Trip Savings

The plots of the total individual & pooled distances across different months in intervals of 500 million kilometers are displayed below. On the whole, similar trends of distance savings across different pool sizes are seen. It is also observed that increasing the pool size saves around 25 million kms on average per month.

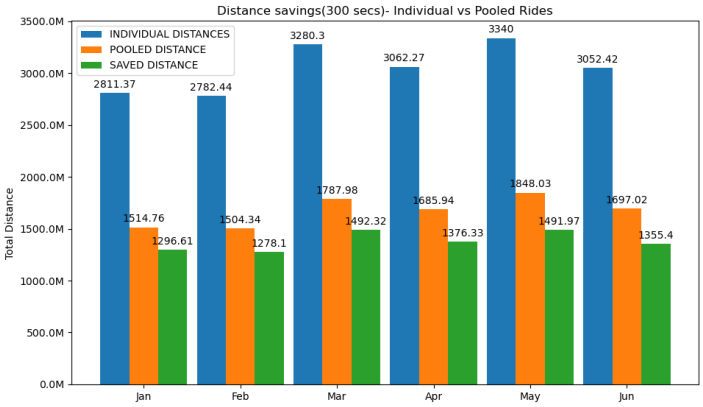
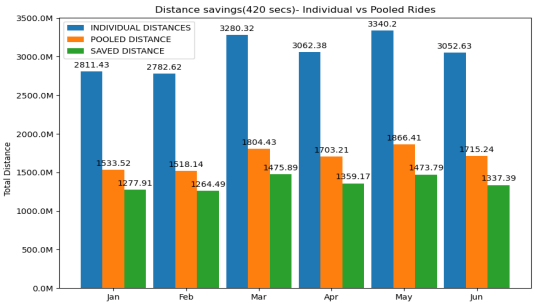
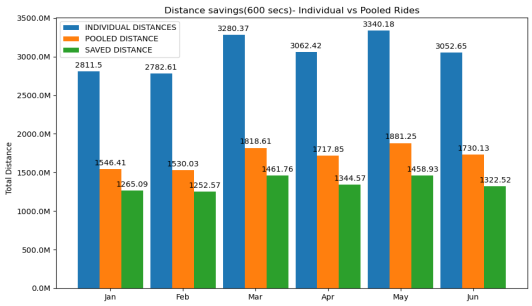


Figure 2: Pool Size: 5 Minutes



(a) Distance Savings — 7 Minutes



(b) Distance Savings — 10 Minutes

4.3 Visualization: Utilization

This project defines utilization as the percentage of distance and trips saved when considering rides that are pooled against individual rides. The algorithm is observed to perform best when considering pool sizes are larger and more number of favorable pairs become available.

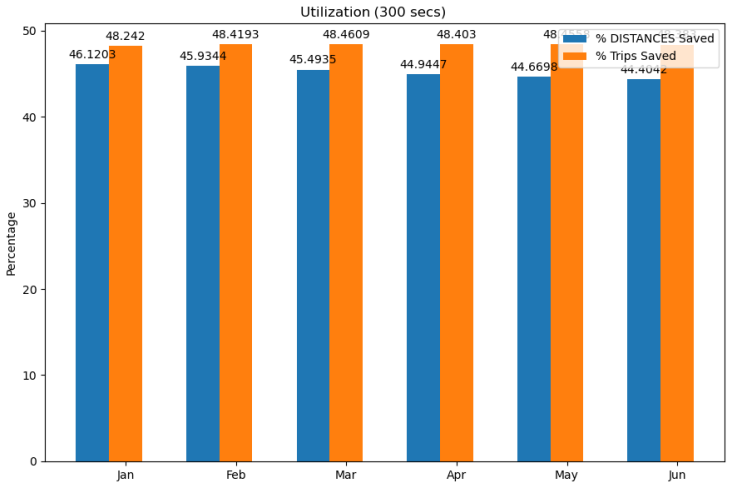
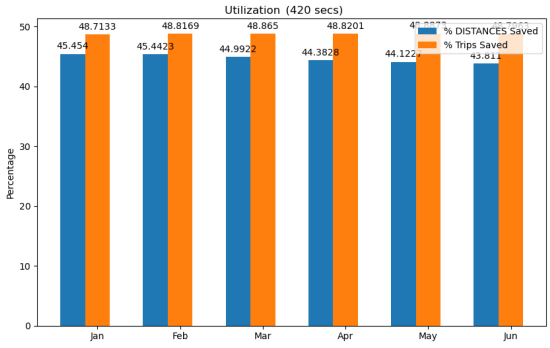
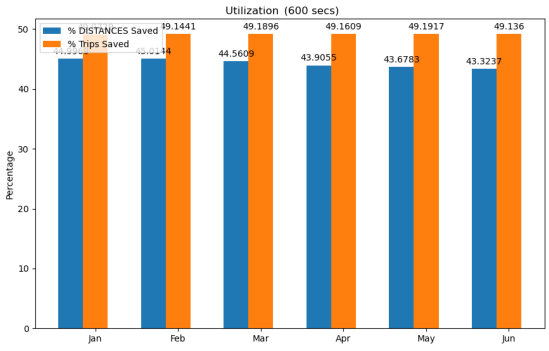


Figure 4: Pool Size: 5 Minutes



(a) Utilization — 7 Minutes



(b) Utilization — 10 Minutes

4.4 Observations: Utilization Breakdown by Days of Week

This project also performs day-wise comparisons for a specific month to learn how the days of the week may impact the pooling strategy. It breaks down the monthly utilization and savings to plot it against the specific days of the week. The plots below are for the month of May 2019 the algorithm is observed to perform especially well for Sundays and Mondays in this case.

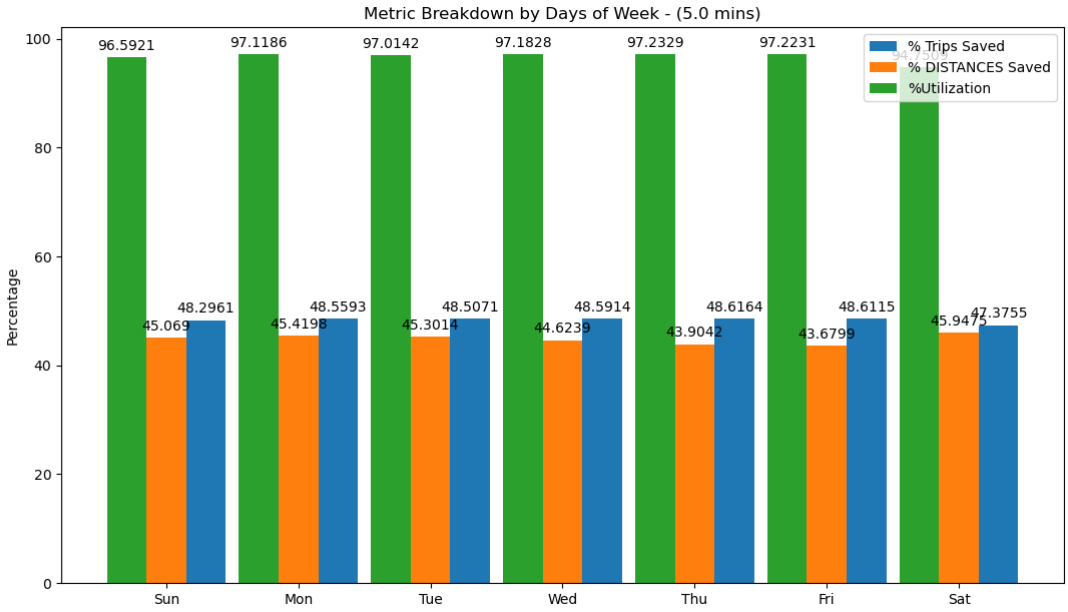


Figure 6: May 2019 — Pool Size: 5 Minutes

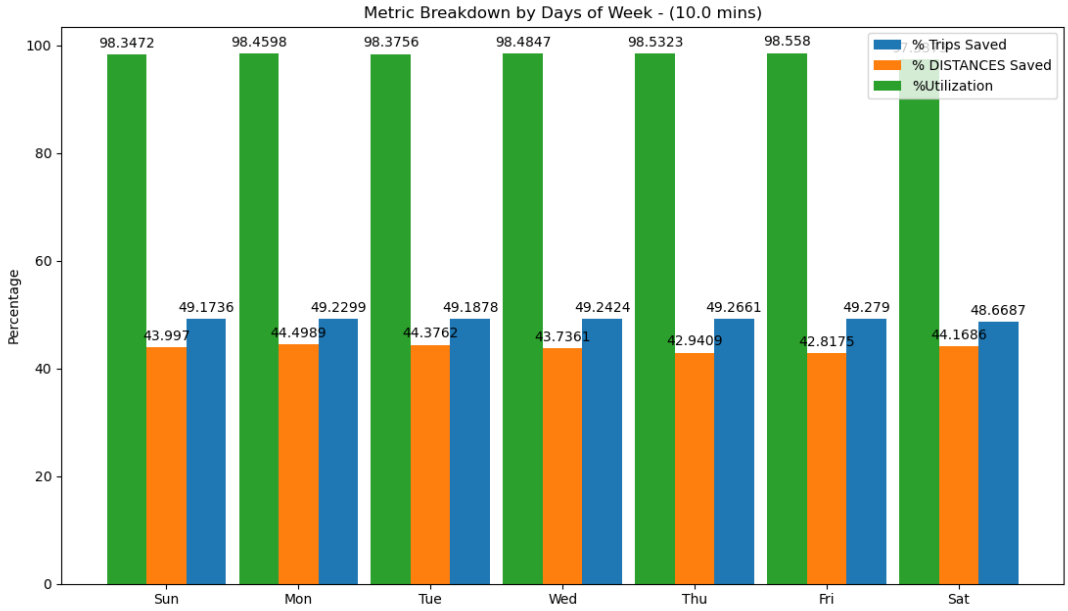


Figure 7: May 2019 — Pool Size: 10 Minutes

4.5 Observations: Cost Savings by Days of Week

The day-wise comparison is used to further understand how the days of the week is impacting the pooling algorithm in terms of cost to customer. It can be observed that there is an overall savings of 42% for users that pool day time rides and approximately 48% in rides pooled at night.

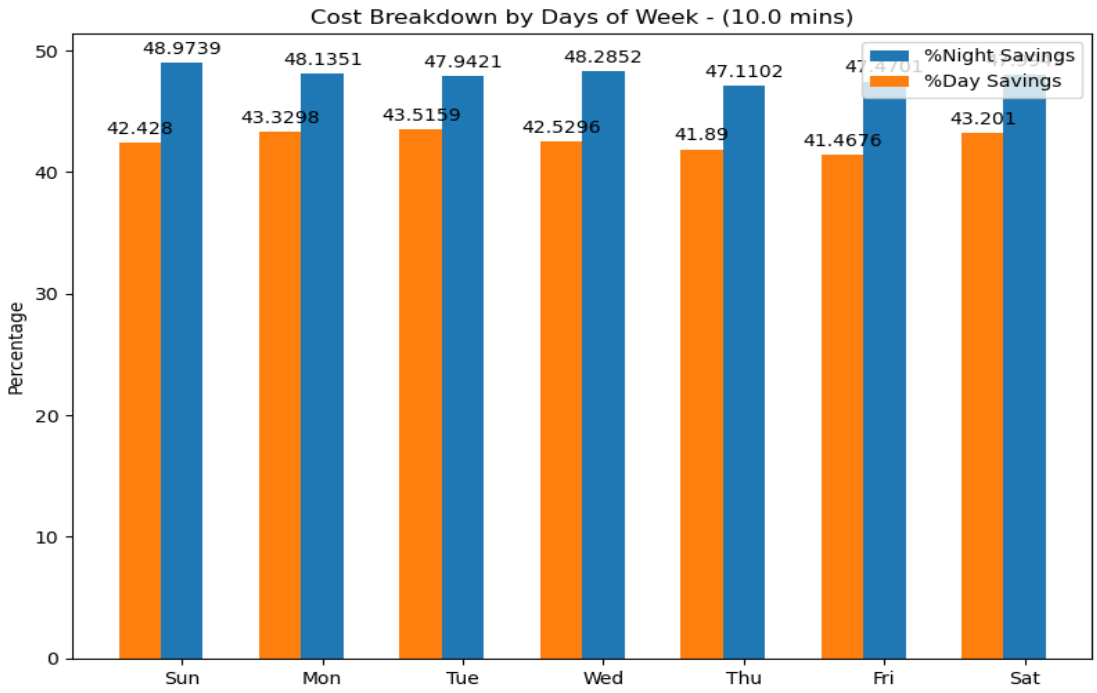


Figure 8: Pool Size: 10 Minutes

4.6 Observations: Computation Times

Computation time for the project was found to be in direct correlation with the pool size. Increasing the pool window size would significantly increase the observed computation times

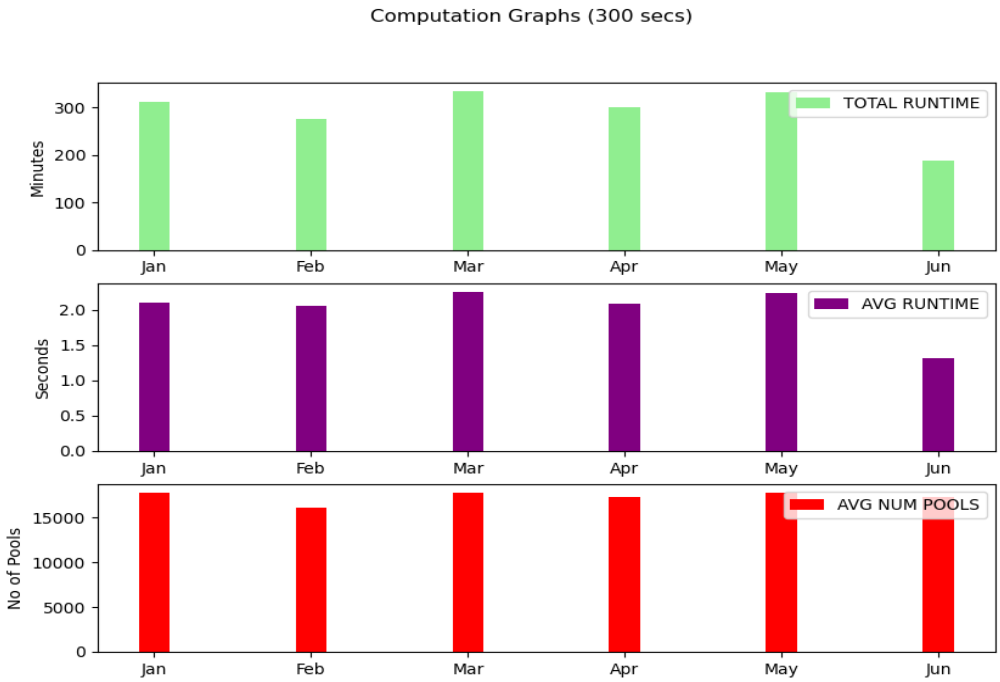


Figure 9: Pool Size: 5 Minutes

5 Conclusion

A simulation of ridesharing system was developed to study the effectiveness of the idea. The project also studied the performance of the graph based approach used to develop the system. The NYC TLC Data subset from Jan to June 2019 was sourced, preprocessed and then stored in the Postgres database. Next, OSRM was used to compute and cache the adjacency matrix for interzonal distances. Then, the required ride requests were fetched from the DB and the ride sharing graph was generated using NetworkX. This process was repeated for each pool and the results were aggregated to be visualized. Overall, based on the results, the number of trips are greatly reduced thereby, increasing the revenue per trip for the taxi companies. The reduction in trips also saved the distance travelled for the companies without affecting a lot of customers. Further optimization is needed in the merging phase of the algorithm and that will be left for future work.

The project helped to learn the nuances of ride sharing algorithms and gain insights into tuning and optimizing pooling simulations.

6 Future Work

1. The current implementation can be optimized for faster execution or better hardware can be utilized to improve computation times.
2. There can be many weights added that can contribute to a better ride merging experience for the customer since they can just be plugged in to the weight calculating function
3. Integration of social scoring data or matching higher rated drivers and passengers.
4. Adding more features to simulate real-world scenarios like limited number of taxis in zones, fluctuating demand, walking distances etc.
5. Another area of interest could be to accommodate passengers that want to modify their destination once the ride has started.

References

[1] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[2] Dennis Luxen and Christian Vetter. "Real-time routing with OpenStreetMap data". In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '11. Chicago, Illinois: ACM, 2011, pp. 513–516. ISBN: 978-1-4503-1031-4. DOI: 10.1145/2093973.2094062. URL: http://doi.acm.org/10.1145/2093973.2094062.

[3] Jordan Friedman. *The 10 Most Congested Cities in the U.S.* URL: https://www.usnews.com/news/cities/articles/10-cities-with-the-worst-traffic-in-the-us. (accessed: 10.30.2020).

[4] *The advantags and Disavantages of Ridesharing.* URL: https://www.cooneyconway.com/blog/advantages-and-disadvantages-ridesharing.