DATABASE MANAGEMENT SYSTEMS (CS581)

# TAXI RIDESHARING

## Big Data Analysis Project - Final Report


## Team:

Sneha Dattatraya Shet
Ankita Acharya
Yukthi Papanna Suresh
Amit Panthi
{sshet6,aachar9,ypapan2,apanth2}@uic.edu

UNIVERSITY OF ILLINOIS AT CHICAGO
(SPRING 2020)

# 1.Introduction

## 1.1 Problem Statement

**"Necessity is the mother of invention"**, but sometimes these inventions, though comforting at the start, turn back to the inventor. The same thing happened with motor vehicles. They started with this huge luxury they gave to mankind by letting them travel fast and safe, but nowadays they are the source of one of the major problems the big cities are facing " The Traffic". New York drivers spent 13 percent of their time sitting in congestion, with 11 percent of that being attributed to daytime traffic. New York ranked as the fourth-most congested urban area in the U.S. on the 2018 INRIX Global Traffic Scorecard examining more than 200 cities across the world. The annual report, released in February 2019, is based on data from 300 million sources covering more than 500 miles of the road [1].

But we have also developed some solutions to fight these problems. One of these solutions is **Ridesharing.** According to studies taxis are successful in cutting emissions and reducing air pollution. Between 2009 and 2015, the legislation more than doubled the fuel efficiency of the fleet of 13,500 yellow taxis, leading to estimated declines in air pollution emissions [2].

Taxi requests from crowded areas such as airports, train stations, bus hubs, etc. have also drastically increased for single riders. Wouldn't it be smarter to group people traveling to nearby places together into a single taxi rather than in separate ones?.

We have addressed this problem in our ridesharing project. We believe our project will help with the issues mentioned above by reducing the number of vehicles on the road and the pollution caused by them. The project can also help with the problem of high fuel consumption. Addressing these issues is of grave importance because if we don't, we sure are moving towards our slow but steady impending doom.

## 1.2 Objective

The goal of our project is to minimize distance traveled through ridesharing while meeting passenger delay time constraints.

# 2.Assumptions and Restrictions

## 2.1 Assumptions

To implement the system within the time frame and analyze the results, we limited our approach by keeping certain assumptions as listed below:

- LaGuardia airport as the origin and as the destination
- No walking allowance by passengers
- Pool Window - 2, 5 and 7 minutes
- No social preferences are considered
- Traffic conditions  are not considered
- Taxis are always available at source
- All passengers are willing to rideshare

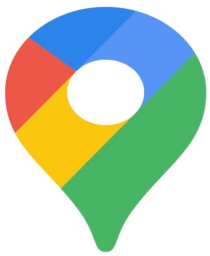
## 2.2 Restrictions

- At most 2 trips are pooled together
- Maximum taxi passenger capacity considered is 4 (so trips with passenger count more than 3 are not considered because they cannot be merged with any other trip)
- Delay time allowed for each passenger is 20% of the original trip time
- For **Demo** purpose - experimented dates are
    - **May 17th 2016 ( entire day : 12am to 11:59pm)**
    - **May 14th 2016 ( algorithm run for 5 mins starting from 7pm)**
    - **June 6th 2015 (peak time -8am, algorithm run for 5 mins)**
- For plotting **yearly averages** - data considered is **January 2015** to **December 2015**.

## 2.3 Dataset Considered


The dataset we used was provided by the NYC Taxi and Limousine Commission. To be precise, the project was implemented using the Yellow taxi trip data [3] from **January 2015** to **December 2015**. The official yellow 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.

## 2.4 Maps and Locations



We used the Open Street Map data for this project. We filtered our data according to the co-ordinates OSM and categorized them as **"to LaGuardia"** and **"from LaGuardia"**. The New York State map data (.osm.pbf file) is loaded onto the GraphHopper which later helped us in finding the routes between two or more points on the map.

# 3. Algorithms evaluated

## 3.1 Previous work

We have referred to some previous work to incorporate in this project the basic idea being carpooling.

### 3.1.1 UberPool

**Uber Technologies Inc**. offers a peer-to-peer ride-sharing service called UberPool which matches riders heading in the same direction so that users can share the ride and cost. Uber restricts users to have only 2 passengers per request [4]. It does not provide a feature for the users to enter a delay or a walking constraint.

Lyft Inc. also offers a similar peer-to-peer ride-sharing service called Lyft Shared. Lyft, like Uber, has the same restriction and does not provide a feature for the users to enter delay or walking constraints.

### 3.1.2 Uber H3 (Map indexing)

H3 is a **geospatial indexing** system using a hexagonal grid that can be subdivided into finer and finer hexagonal grids. H3 is developed by Uber and is used in this project to represent the trip pickup and dropoff locations as a hexagonal grid.The basic functions of the H3 library are for indexing locations, which transforms latitude and longitude pairs to a 64-bit H3 index, identifying a grid cell.

The project uses the function *geoToH3* that takes in latitude, longitude, and resolution (between 0 and 15, with 0 being coarsest and 15 being finest), and returns an index. A resolution of 15 and 10 is used to represent LaGuardia airport for trips corresponding to *From LaGuardia* and *To LaGuardia* respectively.

In case of new york city locations, a resolution of 8,15 is used for trips corresponding to *From LaGuardia* and *To LaGuardia* respectively.

Hence every ride request's pickup location and drop-off location are mapped to the nearest hexagon using the H3 library.

Next we precompute the shortest path distance and time for all possible pairs of the hexagons of New York City using the Graphhopper API (explained in the next section).These distances are indexed by their corresponding H3 indices.

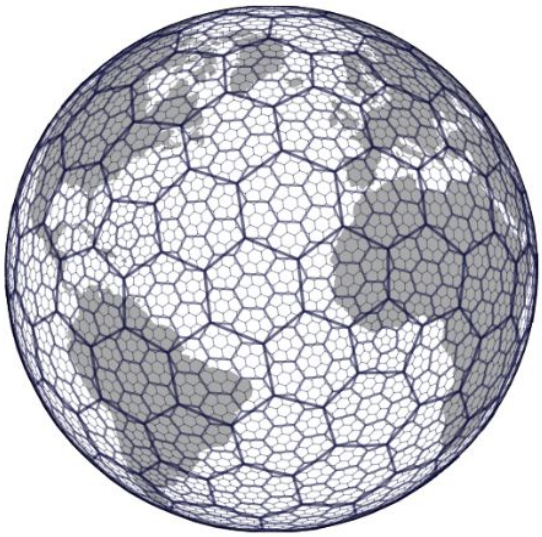


*Figure 1. H3 enables users to partition the globe into hexagons for more accurate analysis.*

## 3.1.3 GraphHopper:

GraphHopper is a fast and memory efficient Java routing engine.We are using the free version of the web API ***0.10.0*** to get distance (in miles) and time (in minutes) between trip pickup and dropoff locations.

## 3.1.4 NetworkX:

NetworkX python package provides a **maximum weight matching** algorithm.
The NetworkX API takes a set of vertices and edges as an input and returns pairs of nodes that are matched based on the maximum weight.

# 3.2 Algorithm:

## 3.2.1 Ride Sharing Graph

To know how we consider rides and potential shares, the concept of Ride Sharing Graphs (RSG) is introduced. In a RSG, our nodes are individual trips (trips that are not yet shared but have individual trip details) while our edges are potential shares. The graph is represented using the networkx library in Python. In later sections, we define how we compute the edge weights for our RSG.

## 3.2.2 Precomputation of Distances

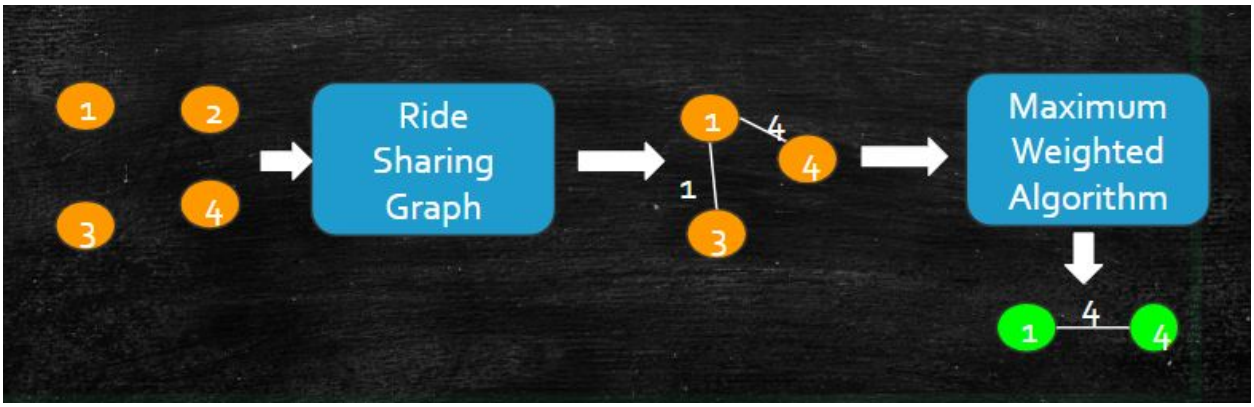
The Ride Sharing Algorithm expects distance,duration between trip's pickup and dropoff locations.To reduce the execution time of the algorithm,we precompute the distance and duration between every possible combination of trips within each pool window using Graphhopper API.

These distances and durations are then saved into csv file and are indexed by their corresponding H3 index.For example , if we have calculated the distance between $trip_a$ and $trip_b$ using their respective latitude and longitude, then this distance and duration is saved as ($H3index_a$, $H3index_b$, distance, duration) where $H3index_a$ and $H3index_b$ is obtained from H3 library using $trip_a$'s and $trip_b$'s latitude and longitude respectively.These csv rows are then imported into Pandas dataframe and H3 indices are used as index which then provides faster search time. A snapshot of the distance saved in pandas DataFrame can be seen below.

| | | distance | duration |
|---|---|---|---|
| pickup_h3 | dropoff_h3 | | |
| 8f2a1000a64d309 | 8f2a100e4932d8e | 0.006273 | 0.01211 |
| 8f2a1000a66bba1 | 8f2a100e4005c42 | 1.138614 | 2.58453 |

## 3.2.3 Merging Conditions/Assumptions/Constraints(Edge Weight Calculation)

Merging conditions refer to the specific variables that determine if two rides can be pooled together or merged together into one ride.




In this experiment, we employ distance saved and delay factors that determine if two rides can be merged.

In theory, some possible factors can be distance, time for each passenger, time for the whole shared ride, the amount of delay each passenger is willing to bear, social factors to better match passengers, and more.

Before we go over the specific factors we consider in this project, it is vital to define the trip sequences we consider.

Following the template defined by Santi et. al. [5], given two individual trips a and b, we consider the following possible shared trip sequences for each trip type ie From LaGuardia and To LaGuardia.

From LaGuardia:

- $D_{LGA \rightarrow d(a)} + D_{d(a) \rightarrow d(b)}$
- $D_{LGA \rightarrow d(b)} + D_{d(b) \rightarrow d(a)}$

Note : $D_{p(a)}$ is pickup of trip a

$D_{d(a)}$ is destination of trip b

Similarly for To LaGuardia:

- $D_{p(a) \rightarrow p(b)} + D_{p(b) \rightarrow LGA}$
- $D_{p(b) \rightarrow p(a)} + D_{p(a) \rightarrow LGA}$

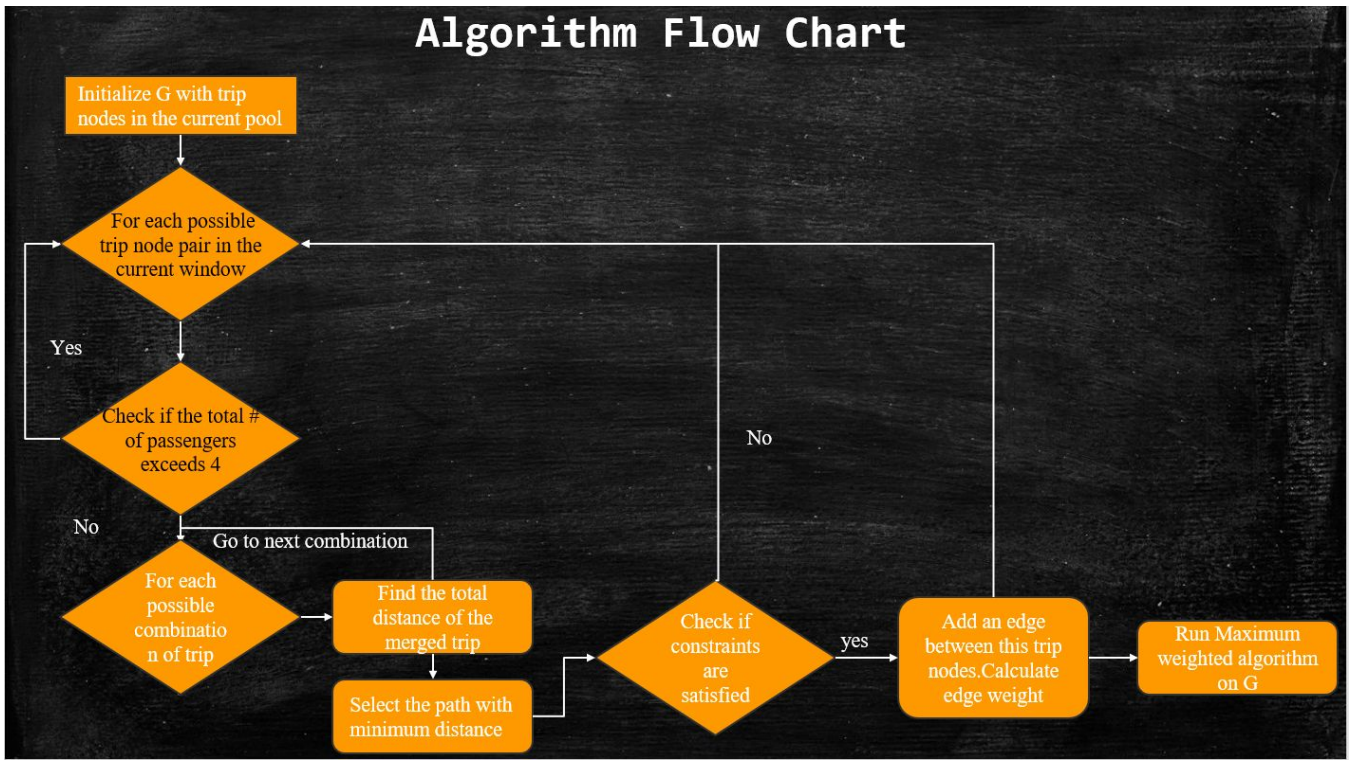
These four are the possible sequences that we can pool two rides together. In this project, we consider the following factors to determine if two trips can be merged:

1. **Distance saved for merged trip**: The overall trip distance saved for the merged trip

2. **Delay constraint for each passenger**: Depending on the sequence of pickup and drop-off, the time delay each passenger is willing to take for a shared ride vs individual rides.

As mentioned, these distances are pre-computed and stored in-memory for fast lookup.

**Distance :** We consider the distances between each hexagon's central point as our distance from any point x to y.




Distance comparisons are easy as we sum up all the possible distances for each of possible sequence and select the one that satisfies the following conditions:

$$D \leq D_A, D_B$$

$$D \leq D_{min}$$

where D is the possible shared distance in consideration, $D_A$ and $D_B$ are individual distances of trips A and B respectively, and $D_{min}$ is the current minimum distance encountered.

**Time :** It is obvious that sharing rides may increase the total ride time of each passenger involved when compared to individual ride times.

The question becomes how we can minimize this extra time with respect to the delay the passengers are willing to take. In our formulation, we consider the following cases for incorporating time in our calculations:

$$T_{s\,A} \leq T_A$$

$$T_{s\,B} \leq T_B$$

$$T \leq T_{min}$$

where $T_{s\,A}$ , $T_{s\,B}$ are new shared trip times for passenger A, B respectively, $T_A$, $T_B$ are original trip times for A, B and $T_{min}$ is the current minimum time encountered.

One thing to notice is that $T_{s\,A}$ and $T_{s\,B}$ are calculated based on the proposed sequence. For example, if A is picked up first, then B is picked up, then A is dropped off, then B is dropped off, the time for A is only till he gets dropped off. B's time is the whole trip as he waits for the taxi to come to him and then eventually gets dropped off. These times are variable depending on the order of pickup and dropoff based on the four possibilities listed above..

### 3.2.4 Ride Matching Algorithm

Once our ride sharing graph (RSG) has been constructed with appropriate edge weights, we can now construct matching sets. Our matching sets contain nodes of individual trips and edges with maximum weights that maximizes distance. We use the Maximum Weighted Matching algorithm to get such matching sets.

The algorithm is a means of finding the maximum matching in a graph.This polynomial time algorithm is used in several applications including the assignment problem, the marriage problem, and the Hamiltonian cycle and path problems (i.e., Traveling Salesman Problem). The algorithm takes $O(n^3)$ time complexity.

We will not go over the exact algorithm as it is quite verbose to explain and detail here. Instead, we shall go over some key concepts used in the algorithm.

The algorithm takes a general graph G = (V, E) and finds a maximum matching M.The algorithm starts with an empty matching and then iteratively improves it by adding edges, one at a time, to build augmenting paths in the matching M. Adding to an augmenting path can grow a matching since every other edge in an augmenting path is an edge in the

matching; as more edges are added to the augmenting path, more matching edges are discovered. The blossom algorithm has three possible results after each iteration. Either the algorithm finds no augmenting paths, in which case it has found a maximum matching; an augmenting path can be found in order to improve the outcome; or a blossom can be found in order to manipulate it and discover a new augmenting path.

# 4. How experiments were conducted?

## 4.1 Technical Components

**Jupyter Notebook:**

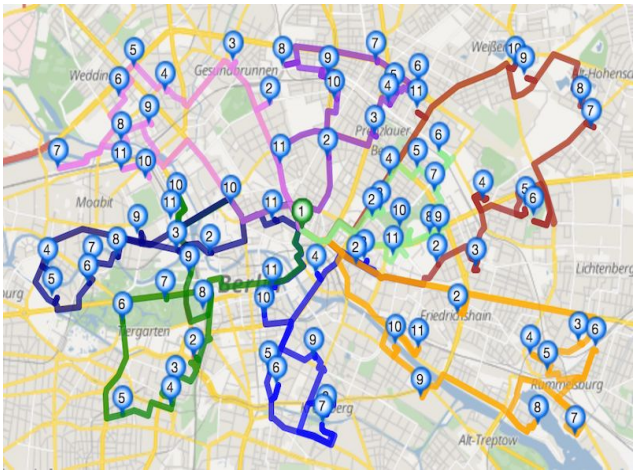

The entire project was developed using Python 3.7 on Jupyter Notebook. The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Includes data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

**Data Storage:**

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series**.** It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

**GraphHopper API:**

GraphHopper API was used to determine the distances from the source to the destination. Since GraphHopper API has become a paid service, a decision to install an old but free version of GraphHopper API was installed on the system locally to provide the source to destination routes and distances.




NetworkX python package provides a maximum weight matching algorithm. The NetworkX API takes a set of vertices and edges as an input and returns pairs of nodes that are matched based on the maximum weight.

# 4.2 Libraries Used

Below are a few Python libraries used in the project -

- **requests**
- **pandas**
- **numpy**

- **h3 (Uber H3)**
- **matplotlib**
- **networkx**
- **json**
- **datetime**

External modules used -

- **Open Source Routing Machine**
- **Graphhopper API**
- **Jupyter Notebook**
- **Collaboration Platform: Github**

More details on dependencies and instructions to run the code can be found in the Readme file submitted.

# 5.Results

## 5.1 Evaluation Metrics

The following metrics have been considered to evaluate our system.

• **Pool Window Time**: A variable pool window time has been used to analyze the performance of the algorithm. Pool windows of 2, 5, 7 and 10 minutes have been evaluated.

• **Distance Saved (as a percentage)**: This metric gives us the distance saved by combining the trips as compared to the original individual trip distance.

$$\%\text{distance}_{\text{saved}} = \left((\text{distance}_{\text{original}} - \text{distance}_{\text{pooled}}) / \text{distance}_{\text{original}}\right) * 100$$

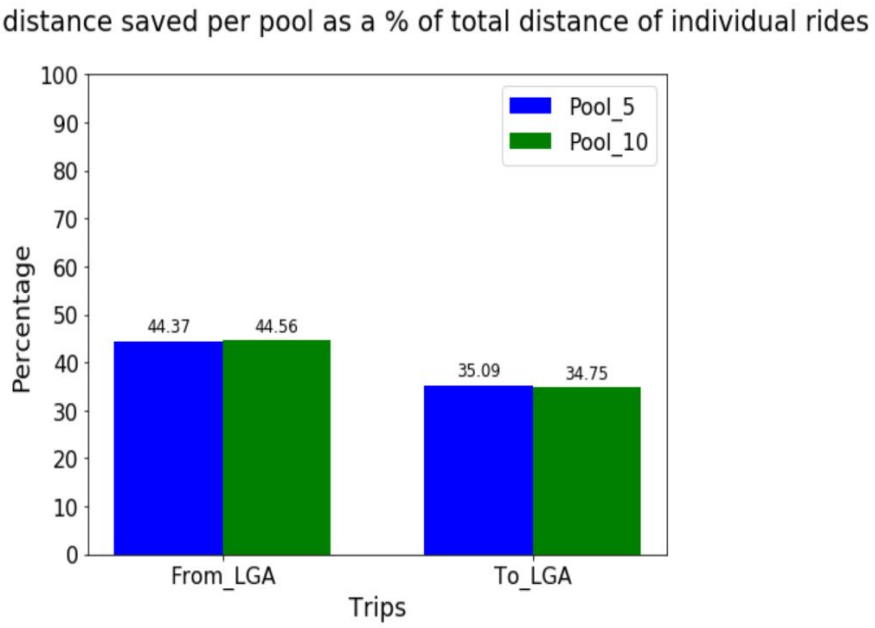
• **Trips Saved (as a percentage)**: We define trips saved as the number of trips pooled together divided by the original number of trips. This value is multiplied by hundred and is represented in the form of a percentage.

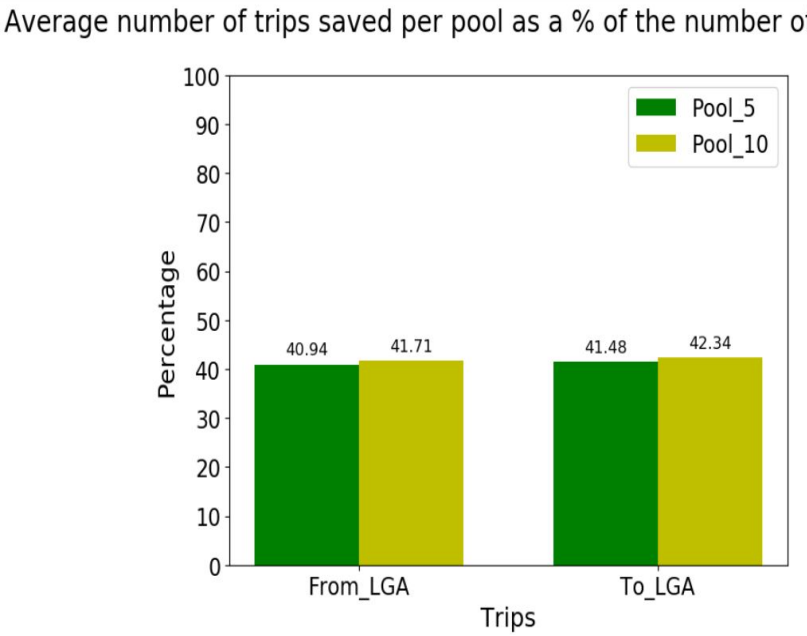
• **Computational Time**: The time taken by the system to process and generate pools for the given pool window time. As mentioned above, the code is executed on a cluster of CPUs.

# 5.2 Output Plots

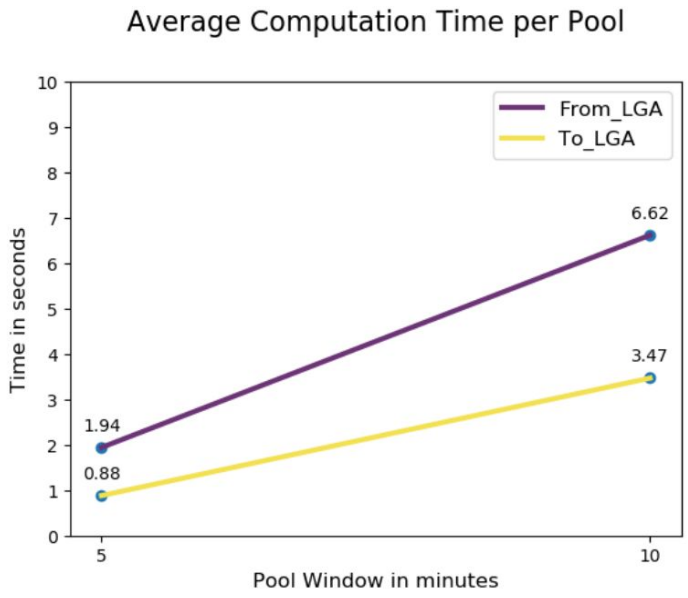
## 5.2.1 Demo results

**Test Data A : May 17th 2016 ( entire day : 12am to 11:59pm)**



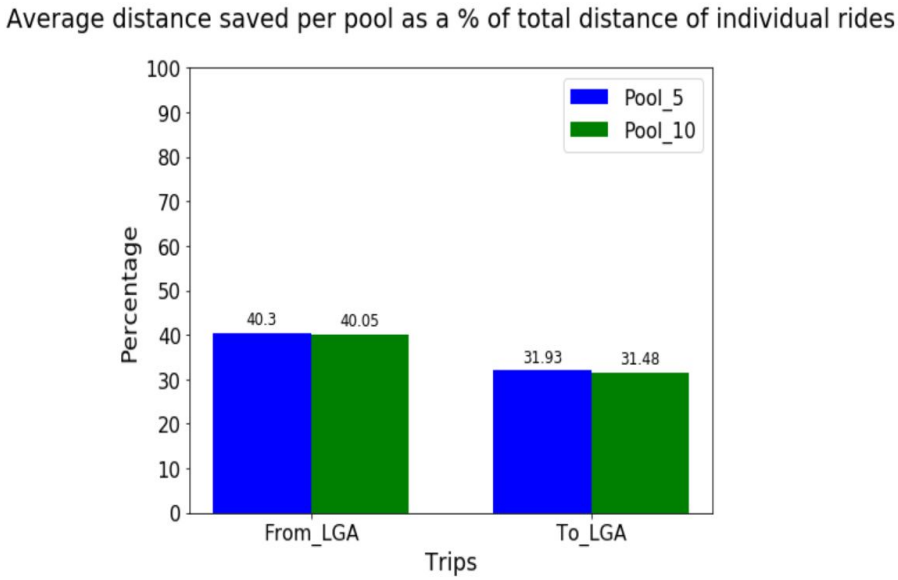


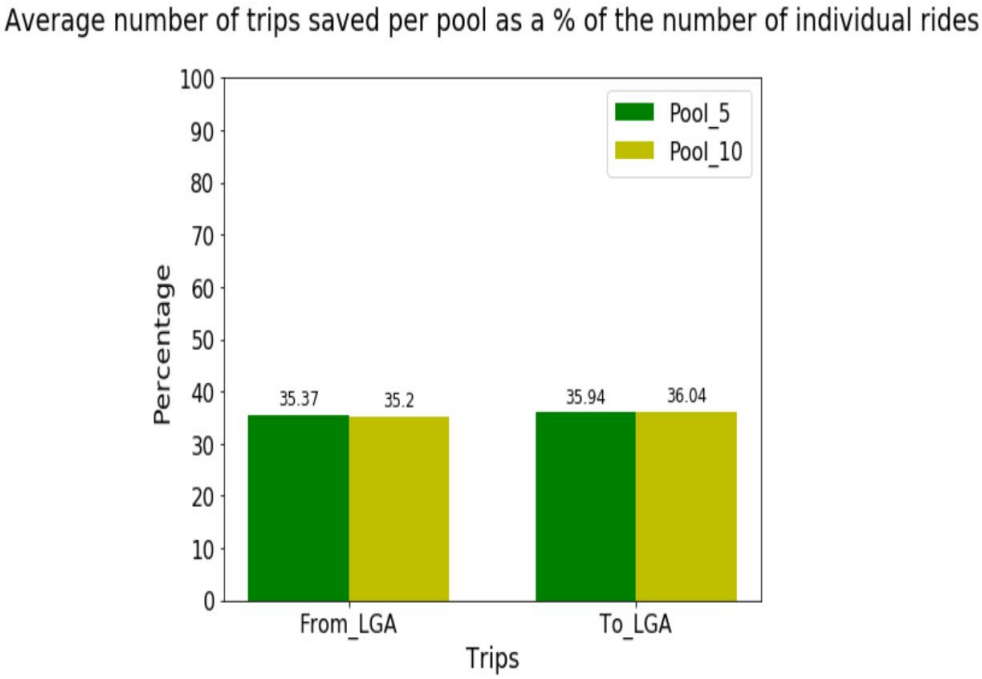

For Test data A with the increase in the pool window size, we can see an increase in the average distance saved and average trips saved. However the increase is not as significant as it just represents only one day's data

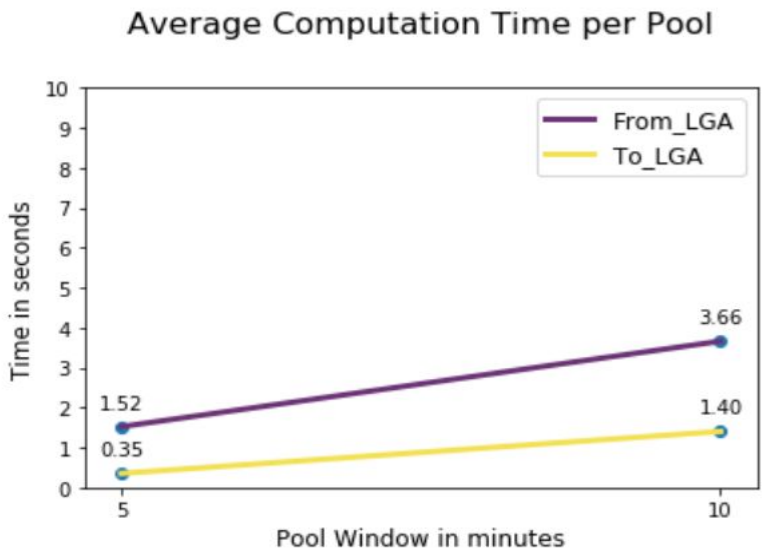



With the increase in the pool window size, we can see a drastic increase in the average computation time. This is because the number of trips in the individual pool increases with the increase in the pool window size.

## Test Data B : May 14th 2016 ( algorithm run for 5 mins starting from 7pm)

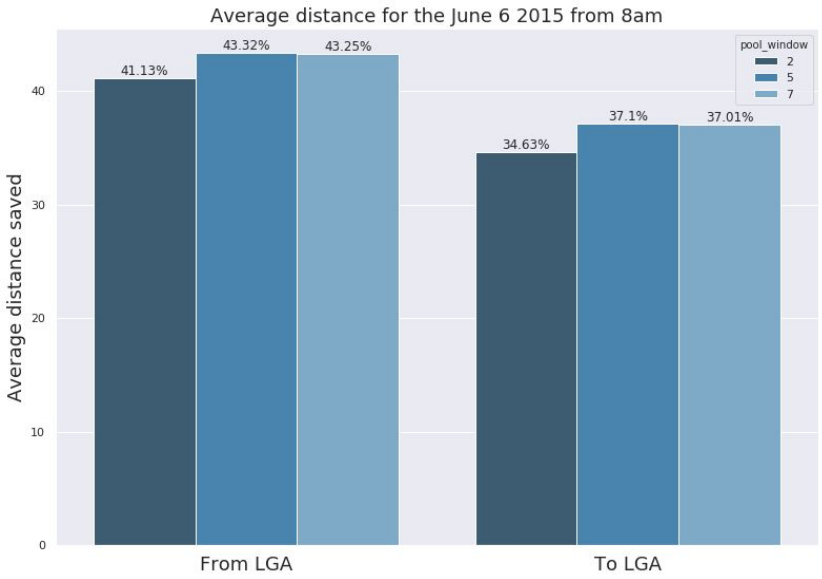


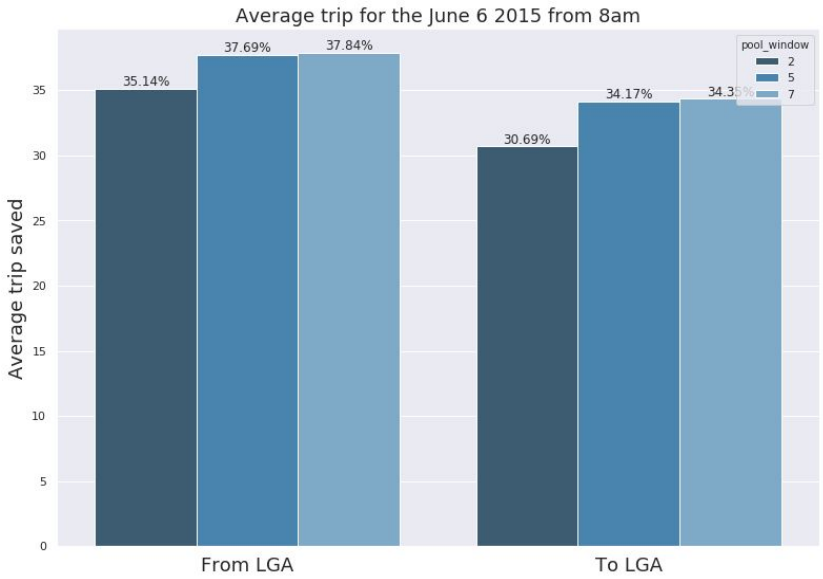






The results for Test data B seems to follow the same trend as Test data A as there is no significant change with increase in the pool window size.

## June 6th 2015 (peak time -8am ,algorithm run for 5 mins)

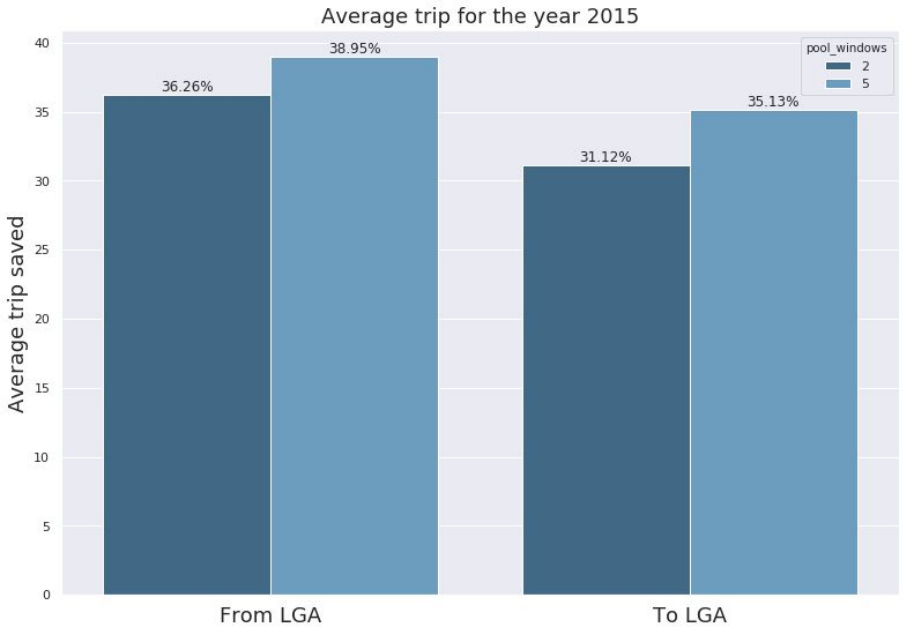


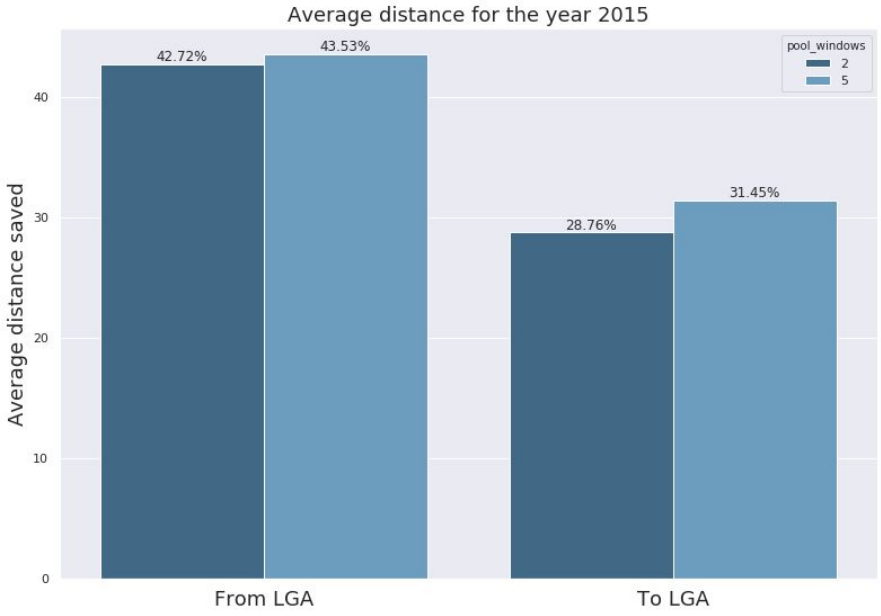



We experimented with different pool windows for a single day's data.As seen from the graph there is a visible increase in the average distance saved and average trips saved from pool window size - 2 mins to pool window size - 7 mins.

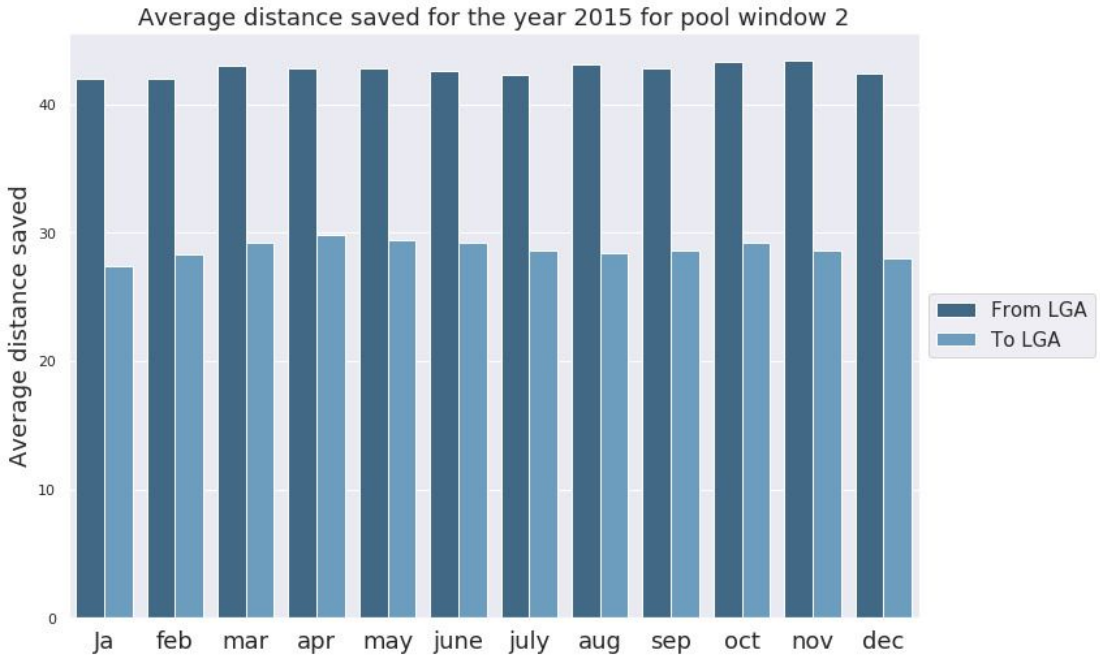
## 5.2.2 Yearly Averages

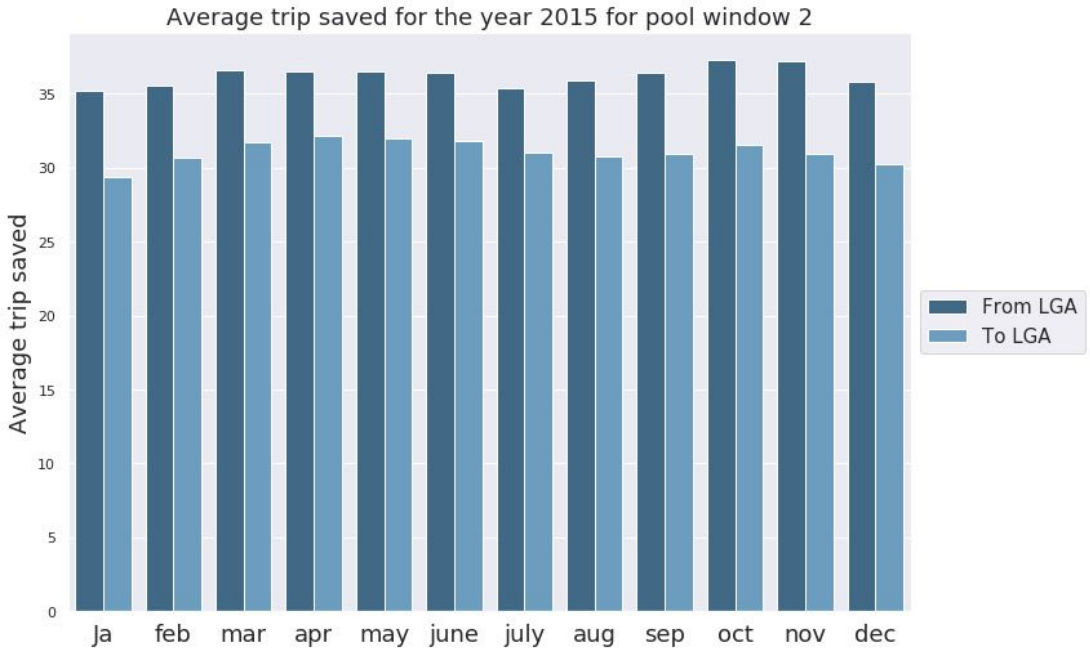
For plotting **yearly averages** - data considered is **Jan 2015 - Dec 2015**




# 5.3 Observations and Trends




# 5.4 Summary on the Plots

- The model gives higher saving in general on *From LaGuardia* Data than *To LaGuardia*. This is because the number of trips for the former is much higher than the later.
- The model showed a hike of ≃ 3 % - 5 % in average trip saved and average distance saved when run on entire 2015 year data.This trend could not be captured by running the model on just 1 day's data.This directs to the conclusion that increase in pool window increases the savings on distance and number of trips.
- Analysis of distance and trip saved on a monthly basis for the year 2015 shows that the savings are higher during summer and fall months while lower during winter (jan and dec).This is because people tend to travel more during summer (vacation seasons) and also Nov and Oct are festive seasons ie Halloween and Thanksgiving. However the month of December seems to record lower savings since it is the beginning of winter season.

# 6.Issues raised during final presentation

Though our demo results looked consistent, we could not conclude something concrete because the results were for a single day. We had not looked over large data i.e. at a month's data or a year's data which would allow proper data analysis.

As seen from the previous section, we have addressed this issue by doing analysis on the 2015 year's data.

# 7.Collaboration and project management

## 7.1 Collaboration and Management

The team member responsibilities are as follows:

1. Data Gathering, Data Cleaning, Data preprocessing such as converting date and time to required format : Amit Panthi
2. Distance Precomputation : Sneha Dattatraya Shet
3. Design, Implementation and testing of the algorithm : Divided into 2 parts
    a. From LGA : Sneha Dattatraya Shet, Yukthi Papanna Suresh
    b. To LGA : Ankita Acharya, Amit Panthi

4. Experiments conducted : Discussions and suggestions were made by all team members. Team members conducted the experiments for various configurations of the model such as running the algorithm for various times/days/months/years
5. Data Visualizations: Yukthi Papanna Suresh, Sneha Dattatraya Shet
6. Project presentations: Sneha Dattatraya Shet, Yukthi Papanna Suresh, Ankita Acharya, Amit Panthi
7. Project Report: Sneha Dattatraya Shet, Yukthi Papanna Suresh, Ankita Acharya, Amit Panthi.

## 7.2 External Factors that affected & Issues faced

- Computation of distances using Graphhopper API was a little troublesome as it was not compatible with few of our systems.Hence precomputation was carried out on 2 systems as opposed to utilising all 4 systems.
- Team work involves a lot of effective communication. Due to COVID 19 lockdown, we could not meet physically(may be more impactful in putting ahead different perspectives) but had to shift to virtual calls. Collaboration tool that helped for this purpose was Google Hangout.
- Time management : Due to extended spring break, few of our team members had rescheduled paper presentations and final project presentations in the same week which called for better time management during that week.
- Online presentation : During our final project presentation, we faced some technical issues like audio problems while shifting from one team member to another and because of limited time had to run through certain slides.

## 7.3 Conclusion

In conclusion, we created and simulated a ride pooling system to study the performance of our graph algorithms and their trade-offs. First, we sourced the data from NYC Taxi Data set for the months of January to December 2015. We used the geoToH3 function to index the data. We then used Graphhopper API to procure distances and duration as necessary. Next, we generated the ride sharing graph using NetworkX library. We then used our custom logic for edge weight calculations. Finally, we used NetworkX's max_weight_matching algorithm for generating maximum matching sets. Running the

model for different pool window times showed the savings on distance and trip are higher for higher pool window times.However it also added overhead in terms of drastic increase in algorithm run time.Hence we conclude that selecting an optimum pool window is very essential for success of ride sharing criteria.With this project, we were able to discover nuances of ride sharing and gather more insight into metrics for better performance and optimization

## 7.4 Future Work

For future work, we can incorporate the following considerations into our system:

- Consider fare estimation as criteria for matching rides
- Consider road architecture (one-way, blocked, etc)
- Consider dynamic matching/routing
- Consider more than k=2 people per ride
- Allow passengers to change destinations dynamically.

# 8.References


[1] https://patch.com/new-york/new-york-city/only-3-u-s-cities-have-worse-traffic-nyc

[2] https://phys.org/news/2019-05-air-taxis-pollution-york-city.html

[3] https://phys.org/news/2019-05-air-taxis-pollution-york-city.html

[4] https://www.uber.com/us/en/ride/uberpool/

[5] https://www.pnas.org/content/111/37/13290

[6] “Real-Time City-Scale Taxi Ridesharing” - Shuo Ma, Yu Zheng, and Ouri Wolfson

[7] OpenStreetMap [https://www.openstreetmap.org/]

[8] Pandas [https://pandas.pydata.org/]

[9] Uber H3 [https://eng.uber.com/h3/]

[10] Graphhopper API [https://www.graphhopper.com/]

[11] OSM file  New York state map
[https://download.geofabrik.de/north-america/us/new-york-latest.osm.pbf]