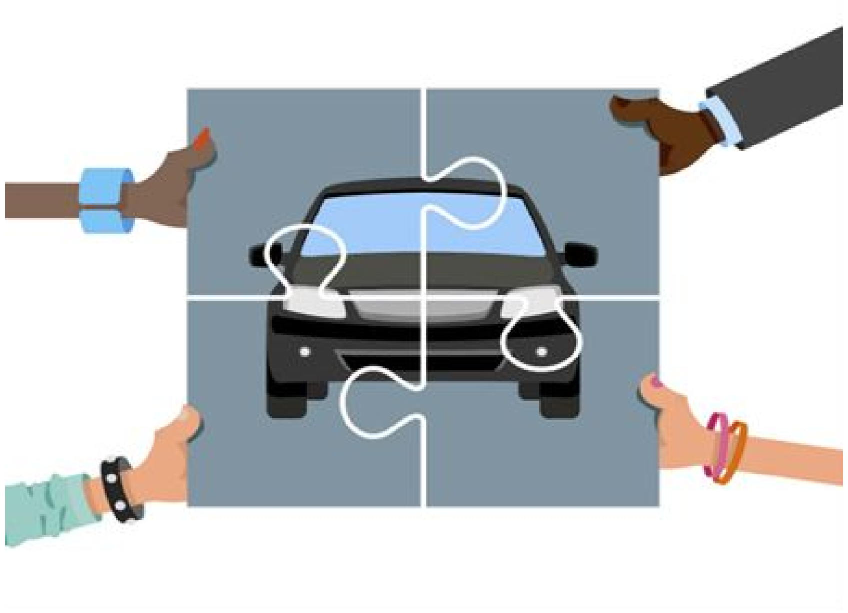
DATABASE MANAGEMENT SYSTEMS
CS 581



RIDE SHARING

Group 8

Yang Hao

Upasna Menon

Rashmitha Mary Allam

Nivetha Shanmuga Sundaram

University of Illinois at Chicago

Spring 2017

TABLE OF CONTENTS

INTRODUCTION

The potential benefits from increased ridesharing are substantial, and impact a wide range of stakeholders. In a properly applied rideshare scheme, drivers and passengers achieve cost savings, they potentially achieve travel time savings and also benefit from increased travel options.

Assigning drivers to riders in an efficient way is a crucial component of ridesharing system for matching rides.In this project report, we discuss the features of our proposed ridesharing system with an algorithm to save distance by reducing the total number of trips.

After exploring different alternate approaches for ride sharing, we came up with a clustering based approach. Finally, we present results with average distance savings of 49% increase with ride sharing when compared to without ridesharing.

OBJECTIVE :

- Devise a ride-sharing algorithm that enables real time taxi ride-sharing.
- To increase the vehicle utilization by matching the trips closeby and therefore reducing the total distance travelled by each taxi.
- Develop an automated matching algorithm to merge the trips.
- To test the algorithm's efficiency in terms of distance and cost using JFK dataset.
- Beneficial for driver as it results in distance savings and passenger as it results in fare savings.

ALGORITHM

Step 1: Clustering:

To reduce the search space and runtime of our algorithm, we have used a cluster based approach, after which ride matching is performed to find the potential rideshare matchings.

We have chosen K-Means algorithm to achieve clustering.

K Means Algorithm:

- Aims to partition n observations into k clusters.
- Given a set of observations ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_n$), k-means clustering aims to partition the n observations into k ($\leq n$) clusters.
- $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ where 'S' is the search space of the destination requests grouped into k clusters.
- In our ridesharing system, we group the dropoff-latitudes and longitudes (destination requests) into k clusters.
- K means is used to group the N trips into K clusters based on the trip's destination location by using drop-off latitudes and longitudes from the data set.
- Trips within a cluster are eligible for matching since destinations are closer to each other.

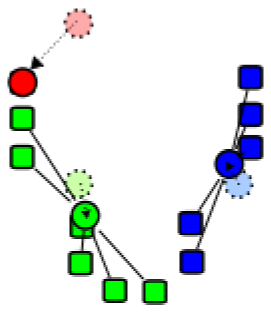


Figure 1 : Centroid of each cluster becoming new mean until convergence

- By itself K-means clustering doesn't guarantee an optimal result although it reaches a stable result quickly .Therefore , we can agree that K-means clustering is locally optimal.

Figure 2 : Visualization of Clustering

Sample K means plot of one pool:

Pool Size = 5 minutes from 2013 dataset.

No. of trips = 44

No. of clusters = 11

Step 2: Trip Matching / Merging Conditions :

- After running K means algorithm on the preprocessed data set, the trips in every cluster are sorted using **greedy approach** based on the distance from JFK airport.
- Every cluster is retrieved and trips in each cluster are merged into a ride by satisfying 2 constraints.

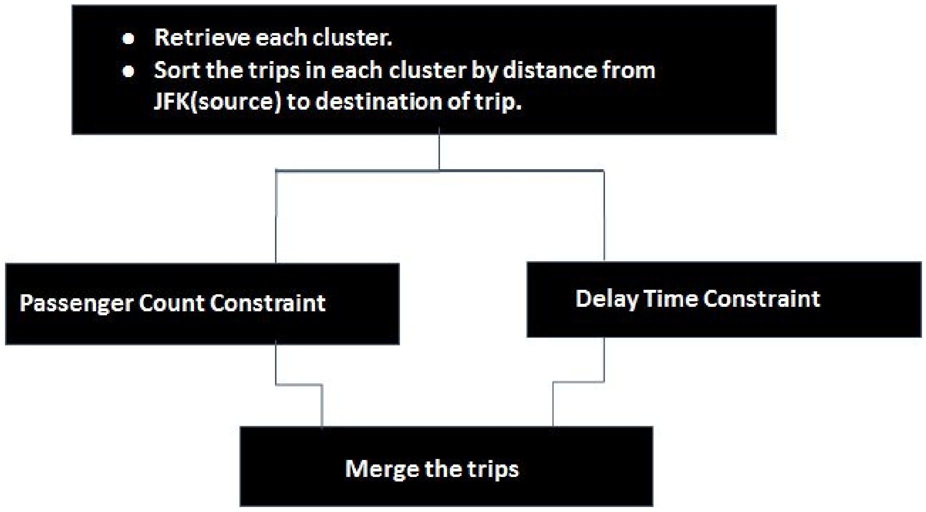

Retrieve each cluster.
Sort the trips in each cluster by distance from JFK(source) to destination of trip.
Passenger Count Constraint
Delay Time Constraint
Merge the trips

Figure 3 : Trip Matching

Passenger Count Constraint:

- ❖ Total number of passengers in each taxi should not exceed 4 after ridesharing.
- ❖ If total number of passengers exceed 4, assign passengers in the trip to the next taxi.

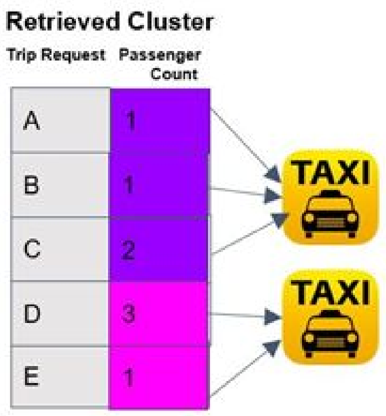


Figure 4 : Example of Passenger Count Constraint

Delay Time constraint:

- ❖ Delay time for each trip request should not exceed 25% of the travel time.

The trips retrieved in each cluster are sorted using the distance computed using GraphHopper API (Explained below) .

GraphHopper API to calculate Distance between any two locations :

- GraphHopper is an open source road routing library and server written in Java and provides a web interface called GraphHopper Maps.
- The Apache License allows everyone to customize and integrate GraphHopper in free or commercial products, and together with the query speed and OpenStreetMap data this makes GraphHopper a possible alternative for us to gain the real time distance and time costing as routing services and GPS navigation software.
- This project evaluates ride-sharing algorithms on spatio-temporal data.The data in this case represents nearly 700 million trips in New York City. The analysis was based on spatio data of New York OSM instead of the default instructions used in the Graphhopper Page. The New York OSM data can be gained via the link: http://download.geofabrik.de/north-america/us/new-york.html. And range of New York OSM is defined as the square in the following figure:

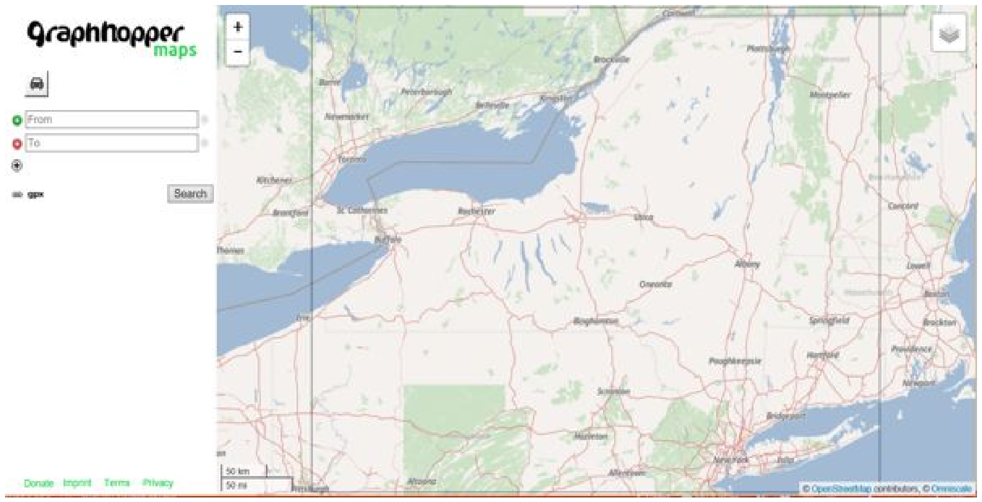


Figure 5 : Graph hopper API to Calculate Distance

- Steps to install Graphhopper API and build the function to calculate real time time and distance between two points :

1.The New York OSM file can be obtained from: http://download.geofabrik.de/north-america/us/new-york.html

2. Install the latest JRE and get GraphHopper Server as zip (~9MB)

3. Unzip the GraphHopper file and put the OSM file in the same dictionary.

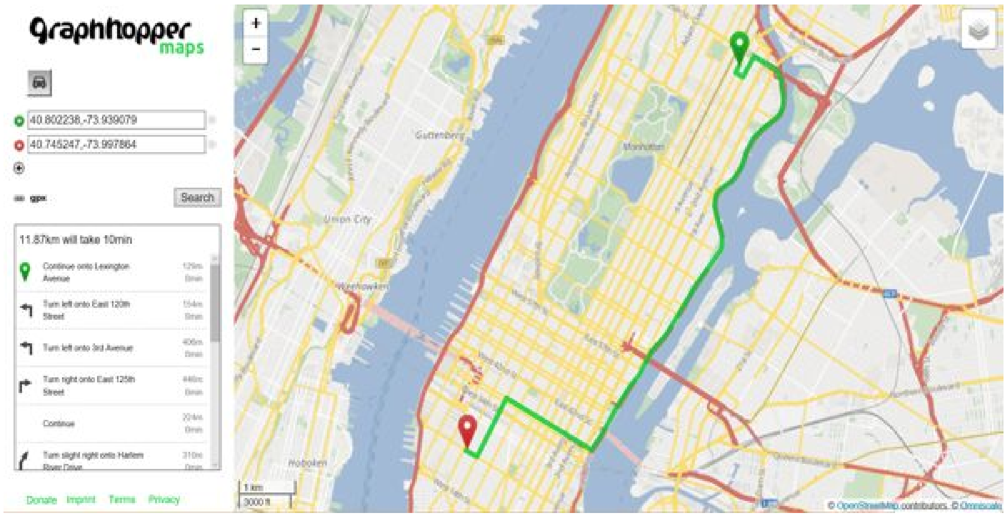
4. Run the command from window cmd under the dictionary: java -jar graphhopper-web-0.6.0-with-dep.jar jetty.resourcebase=webapp config=config-example.properties osmreader.osm=new-york-latest.osm.pbf.

- Function getDistance(plat,plong,dlat,dlong) is created to calculate the distance and time between two points by sending :

“requestString='http://localhost:8989/route?point='+str(plat)+'%2C'+str(plong)+'&point='+str(dlat)+'%2C'+str(dlong)+'&vehicle=car‘ “ to the Graphhopper API.

❖ Please note that Graphhopper must be running all the time while algorithm is running.

Routing is the process of finding the 'best' path(s) between two or more points, where best depends on the vehicle and use case. With GraphHopper API you have a fast and solid way to find this best path.

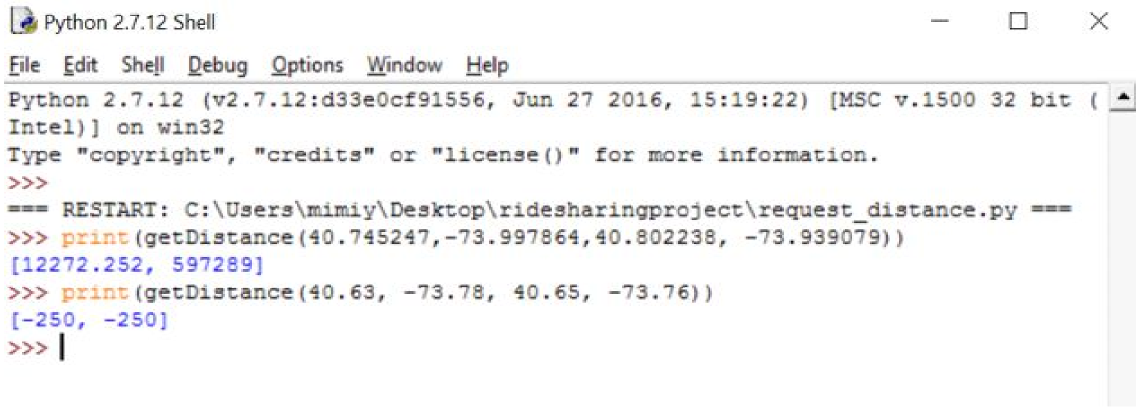


40.802238,-73.939079
40.745247,-73.997864

11.87km will take 10min

Continue onto Lexington Avenue	129m	0min
Turn left onto East 120th Street	154m	0min
Turn left onto 3rd Avenue	406m	0min
Turn right onto East 125th Street	446m	0min
Continue	224m	0min
Turn slight right onto Harlem River Drive	310m	0min

Figure 6 : Sample Example of Calculating Distance



```
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (
Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\Users\mimiy\Desktop\ridesharingproject\request_distance.py ===
>>> print(getDistance(40.745247,-73.997864,40.802238, -73.939079))
[12272.252, 597289]
>>> print(getDistance(40.63, -73.78, 40.65, -73.76))
[-250, -250]
>>> 
```

Figure 7 : Using GraphHopper API to get distance from Python Gradle

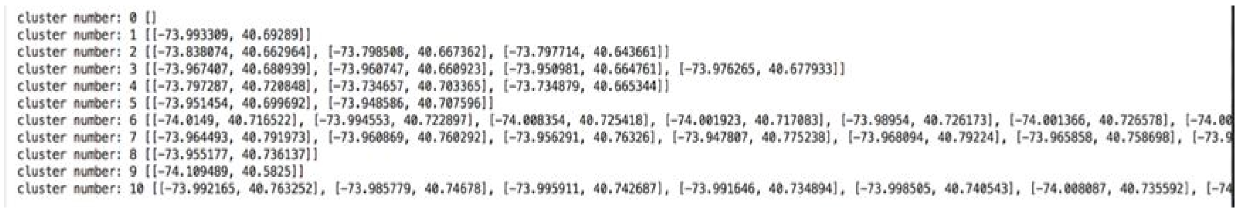
- Function getDistance(plat,plong,dlat,dlong) is created to calculate the distance and time between two points by sending :

“requestString='http://localhost:8989/route?point='+str(plat)+'%2C'+str(plong)+'&point='+str(dlat)+'%2C'+str(dlong)+'&vehicle=car‘ “ to the Graphhopper API.

Sample result of Algorithm for one pool :

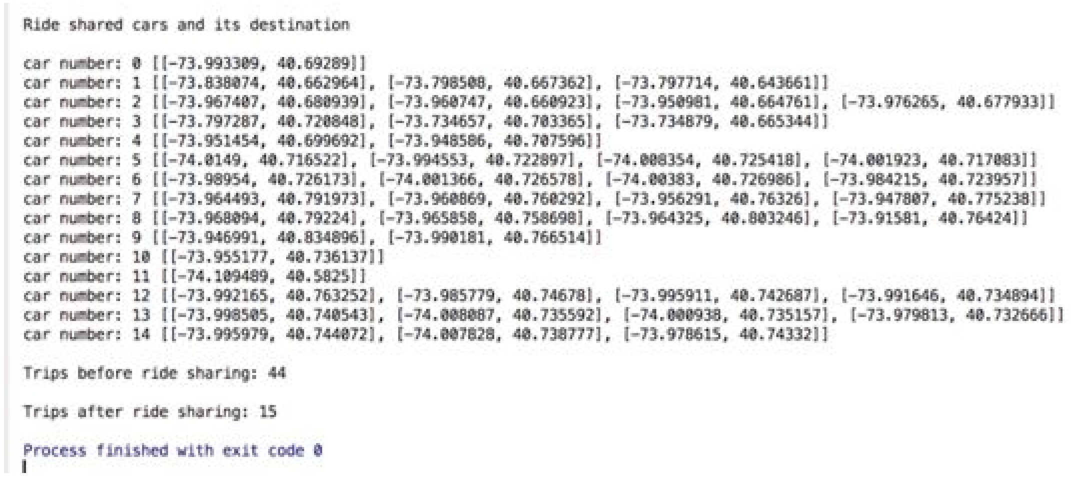
Pool Size = 5 minutes from 2013 dataset.
After running our algorithm for one pool these are the results we obtained.
No. of clusters formed = 11
No. of trips before ride sharing = 44
No. of trips after ride sharing= 15

CLUSTERING



```
cluster number: 0 []
cluster number: 1 [[-73.993309, 40.69289]]
cluster number: 2 [[-73.838074, 40.662964], [-73.798508, 40.667362], [-73.797714, 40.643661]]
cluster number: 3 [[-73.967407, 40.680939], [-73.960747, 40.660923], [-73.950981, 40.664761], [-73.976265, 40.677933]]
cluster number: 4 [[-73.797287, 40.720848], [-73.734657, 40.703365], [-73.734879, 40.665344]]
cluster number: 5 [[-73.951454, 40.699692], [-73.948586, 40.707596]]
cluster number: 6 [[-74.0149, 40.716522], [-73.994553, 40.722897], [-74.008354, 40.725418], [-74.001923, 40.717083], [-73.98954, 40.726173], [-74.001366, 40.726578], [-74.00
cluster number: 7 [[-73.964493, 40.791973], [-73.960869, 40.760292], [-73.956291, 40.76326], [-73.947807, 40.775238], [-73.968094, 40.79224], [-73.965858, 40.758698], [-73.9
cluster number: 8 [[-73.955177, 40.736137]]
cluster number: 9 [[-74.109489, 40.5825]]
cluster number: 10 [[-73.992165, 40.763252], [-73.985779, 40.74678], [-73.995911, 40.742687], [-73.991646, 40.734894], [-73.998505, 40.740543], [-74.008087, 40.735592], [-74
```

TRIP MATCHING



```
Ride shared cars and its destination

car number: 0 [[-73.993309, 40.69289]]
car number: 1 [[-73.838074, 40.662964], [-73.798508, 40.667362], [-73.797714, 40.643661]]
car number: 2 [[-73.967407, 40.680939], [-73.960747, 40.660923], [-73.950981, 40.664761], [-73.976265, 40.677933]]
car number: 3 [[-73.797287, 40.720848], [-73.734657, 40.703365], [-73.734879, 40.665344]]
car number: 4 [[-73.951454, 40.699692], [-73.948586, 40.707596]]
car number: 5 [[-74.0149, 40.716522], [-73.994553, 40.722897], [-74.008354, 40.725418], [-74.001923, 40.717083]]
car number: 6 [[-73.98954, 40.726173], [-74.001366, 40.726578], [-74.00383, 40.726986], [-73.984215, 40.723957]]
car number: 7 [[-73.964493, 40.791973], [-73.960869, 40.760292], [-73.956291, 40.76326], [-73.947807, 40.775238]]
car number: 8 [[-73.968094, 40.79224], [-73.965858, 40.758698], [-73.964325, 40.803246], [-73.91581, 40.76424]]
car number: 9 [[-73.946991, 40.834896], [-73.990181, 40.766514]]
car number: 10 [[-73.955177, 40.736137]]
car number: 11 [[-74.109489, 40.5825]]
car number: 12 [[-73.992165, 40.763252], [-73.985779, 40.74678], [-73.995911, 40.742687], [-73.991646, 40.734894]]
car number: 13 [[-73.998505, 40.740543], [-74.008087, 40.735592], [-74.000938, 40.735157], [-73.979813, 40.732666]]
car number: 14 [[-73.995979, 40.744072], [-74.007828, 40.738777], [-73.978615, 40.74332]]

Trips before ride sharing: 44

Trips after ride sharing: 15

Process finished with exit code 0
```

NOTE : The results shown above are example results for just one pool for visualization. All graphs presented in the report in Results section are averaged results of a pool after running our algorithm for 1 year.

Assumptions made for Algorithm execution and Experimental evaluation:

Some of the assumptions made for testing the data set with the algorithm are:

- JFK is the only source of pickup used for preprocessing data set.
- All the locations within 2-mile radius of JFK's location are filtered.
- Maximum number of passengers per trip is less than or equal to 4.
- Passengers are not willing to walk.
- Maximum delay tolerated by each passenger is 25% of travel time.
    - ❖ Example: If total travel time for a passenger is 10 minutes, then we consider matching trips only if the delay doesn't exceed 2.5 minutes
- Window size: Tested algorithm for matching trips in different time windows of 2 minutes, 3 minutes, 5 minutes and 10minutes
    - ❖ Example: 08/01/2013 12:00:00AM - 08/01/2013 12:02:00AM

      08/01/2013 12:00:00 AM - 08/01/2013 12:05:00AM

      08/01/2013 12:00:00 AM - 08/01/2013 12:07:00AM

      08/01/2013 12:00:00 AM - 08/01/2013 12:10:00AM

Software/ Technology used :

DBMS : *MySQL*

Programming Language : *Python*

Directions API : *GraphHopper API*

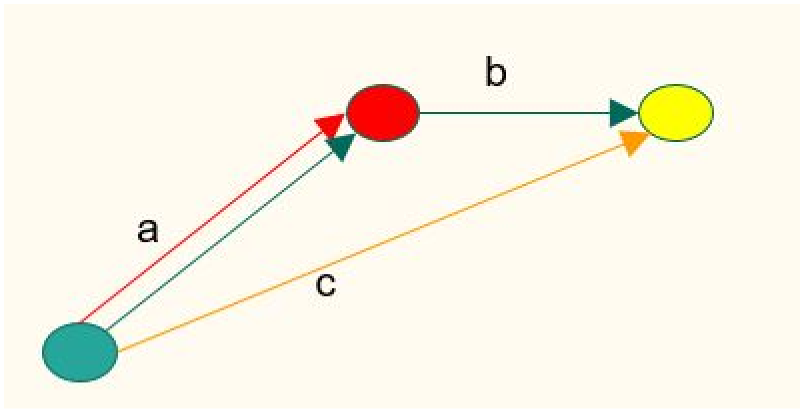
IDE : *PyCharm and Python IDLE*

Fare Saving model with Ridesharing

- On the personal or private level, ride-sharing involves creating one-time, non recurring trips between individuals who share geographically similar itineraries. It is distinct from traditional notions of carpooling in that there is no long-term agreement between passengers to travel together for any time or for any purpose.

- For example, if two riders realize just before hailing a taxi that they share a common destination and decide to split the fare. There is a deliberate, short-notice matching decision process between riders that is continuously being made throughout a trip.

- Here we proposed the fare saving model to give the rule for the splitting the fare for ridesharing customers. It is a simplified Fare model. We assume that Fare quote is proportional to the distance at the rate p. The equation is that: $F = p . D$. F is fare,and D is distance. The cost for each group of passengers is biased. For example, there are two grouped trips which agree with the ride sharing. Both of them have the same pick up location as JFK and share the ride for a specified distance. One with shorter trip will be dropped off in the middle of the trips. And the other group of passengers will continue the rest of the trip by themself. We have a rough fare estimation for shared distance based on the different groups. Here group red with shorter trips should pay the shared distance and a amount of money to make compensation for the other group which has waiting time. The following table shows how the model works for the two groups. The group red has shorter trips and will reach its destination in the middle of the whole trip.



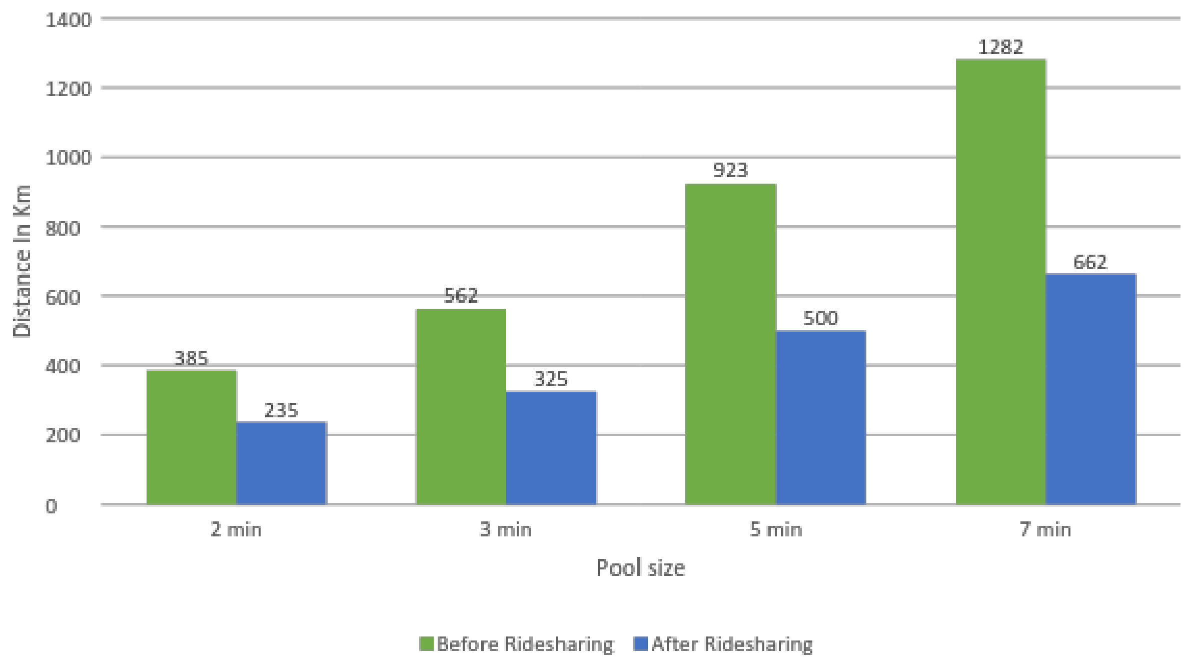
User	Individual trip	Trip with ridesharing	Fare saving
Red	p.a	p.(a/(a+c)·a+ φ)	p.(a - a/(a+c)·a-φ)
yellow	p.c	p.(c/(a+c)·a- φ +b)	p.(c - c/(a+c)·a+ φ -b)

Figure 8 : Table demonstrates the fare model for two groups which anticipate the ridesharing. And φ as compensation cost for user yellow who has waiting time.

RESULTS

Different experiments were conducted on 2013 dataset by testing performance on varying pool sizes . Results are shown in the following graphs for 1 year timeframe(2013 dataset).

<u>Average Distance Saved :</u>

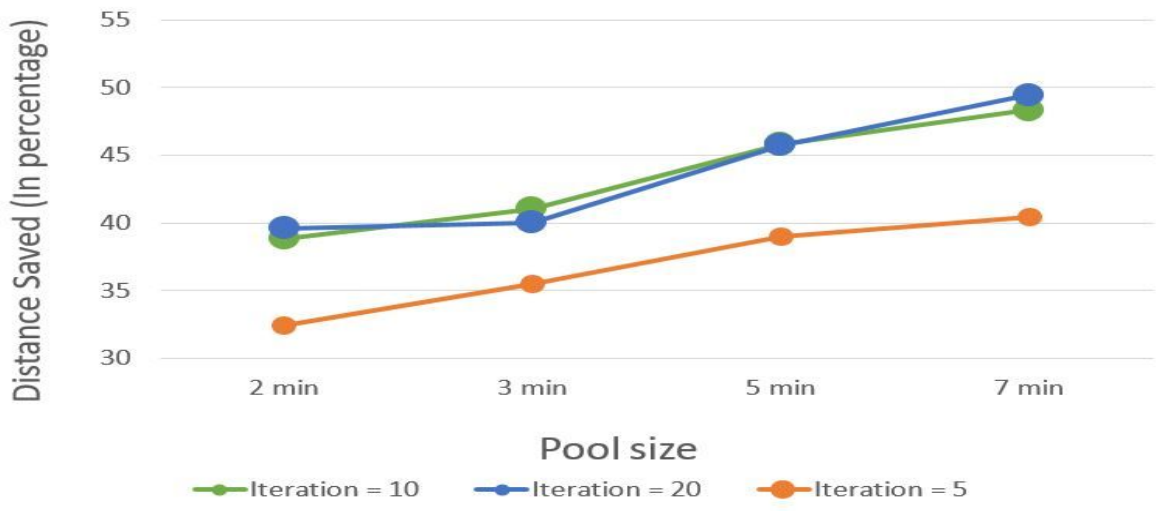


- This graph shows the average distance saved for a pool.
- As the pool size increases, the distance savings increases.

Explanation :

- As the pool size increases, the number of records(rows) in the pool increases.
- Therefore there will be more ride requests in each cluster, which increases the chances of a trip being paired with another more accurately.
- This will help in better trip matching . Hence, the distance saved will be better too.

Percentage of distance savings for different iteration :



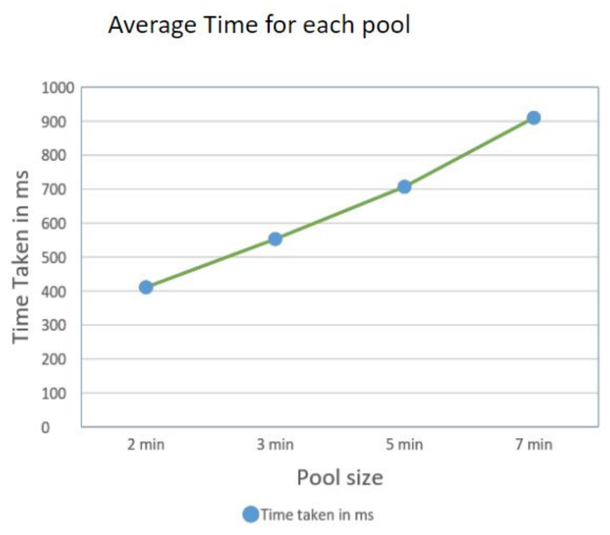
- In *k-means clustering* we initially pick k random centroids and assign the given data to one of these k centroids (which ever is nearest). After this we create new centroids by taking the mean of the assigned points.
- As the number of iterations increases the cluster centroids keep changing.
- The higher number of iterations provides the higher probability of finding the the centroids with the lowest Eout.

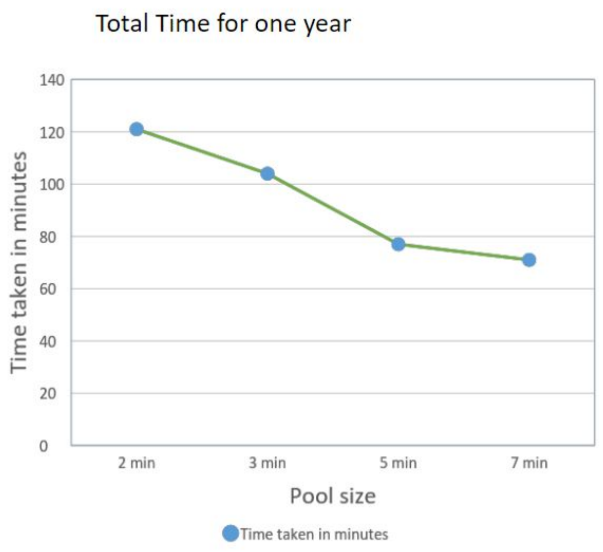
Observations :

Let , number of iterations be represented as I.

- Here , we noticed that when I = 5, there was a significant increase in distance savings.
- However, when we increased I to 10, the distance savings was much higher when compared to the savings for I =5.
- This is because as I increases from 5 to 10, centroids found provided more optimal results of ride matching.
- When I =20, convergence criterion is met and no significant change is observed.

Execution Time :





Observations :

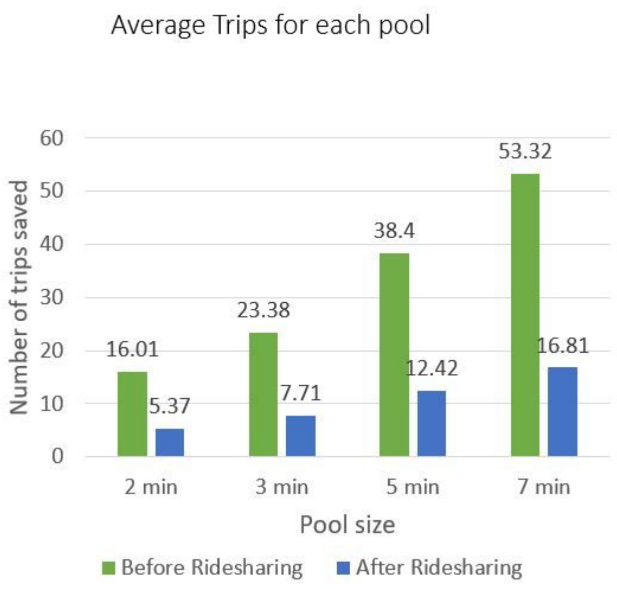
1. First graph shows the Average time for each pool.

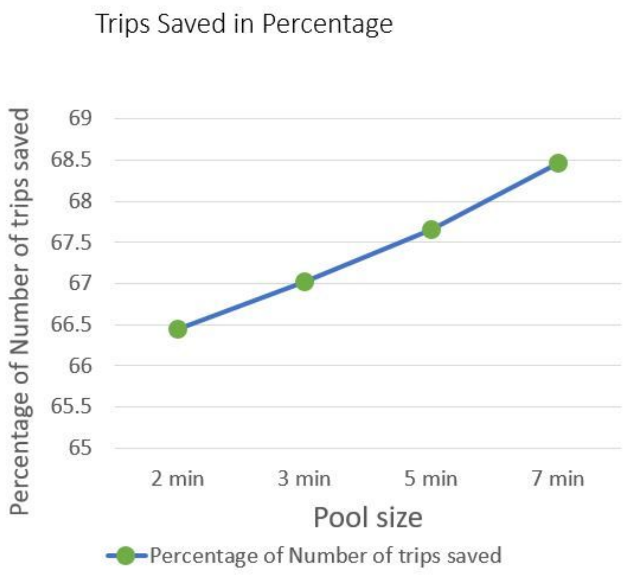
- For pool size = 2, the number of records or trips obtained will be few. Therefore, the execution time was less.
- For pool size =5, the number of trips obtained increases and so does the execution time.
- Similarly, for pool size 7 the execution time is highest. Therefore execution time is highest.
- Hence, there is a steady increase in the curve.

2. Second graph shows the Total time for 1 year.

- For 1 year, the number of 2 minute pools will be more than the number of 5 minute and 7 minute pools.
- Therefore, 2 minute pools take more total execution time in a period of a year.
- Similarly, the number of 5 minute and & 7 minute pools are lesser . Therefore, they took less total execution time.

Average Trips Saved with Ride Sharing :





- These graphs show the Number of trips that were saved due to our ride sharing scheme.
- As we can see from graph 1, the Average Number of trips after ridesharing is always lesser than the that before ridesharing, for all pool sizes.
- This shows that our algorithm works correctly and successfully implements ride sharing or merging of trips.
- In graph 2, we can see the percentage of savings as pool size increases.
- We notice that as pool size increases, savings of trips increases.
- This is because , as pool size increases more trips are obtained and therefore more trips are merged.

REFERENCES

https://www.packtpub.com/books/content/working-data-%E2%80%93-exploratory-data-analysis

https://en.wikipedia.org/wiki/K-means_clustering

https://en.wikipedia.org/wiki/GraphHopper

https://github.com/kkvamsi/NewYork-Taxi-Ridesharing

http://download.geofabrik.de/north-america/us/new-york.html

https://github.com/graphhopper/graphhopper/blob/0.6/docs/core/quickstart-from-source.md