

UNIVERSITY OF ILLINOIS AT CHICAGO  
DEPARTMENT OF COMPUTER SCIENCE

# TAXI RIDESHARING PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF  
REQUIREMENTS FOR THE COURSE CS 581 DATABASE  
MANAGEMENT SYSTEMS

CHICAGO, ILLINOIS  
MAY, 2017

## Table of Contents

1. Acknowledgement.....	2
2. Motivation.....	2
3. Objective.....	3
4. Assumptions.....	3
5. Algorithm.....	4
6. Angle Calculation.....	5
7. System Architecture.....	6
8. Data Cleaning.....	7
9. Data Preprocessing.....	8
10.Implementation.....	10
11.Cost Calculation.....	13
12.Experimental Setup.....	15
13.Results.....	15
14.Github link.....	21
15.Softwares and Libraries.....	21
16.Conclusion.....	22
17.References.....	22

## **Acknowledgement**

The successful completion of this project has to be attributed to a few people, towards whom we like to extend our gratitude. We would like to show our immense gratitude to **Prof. Ouri Wolfson**, Course Instructor for CS 581 at University of Illinois at Chicago, who introduced us to the spatiotemporal aspects along with many other advanced database concepts, which was enlightening. The comments provided to us during our project presentations was very helpful and informative. We would like to thank the university of Illinois at Chicago for providing us the right resources (the ACM digital library) which was the reference point for many research papers we referred to while working on the project.

We would also like to thank **Aishwarya Vijayan**, the teaching assistant for the course, who always had the best approach to resolve our doubts and a provided a solution leading us to the way of progression. We would also take this opportunity to thank our fellow classmates for their presentations and the class discussions. Participations in such class activities helped us gain our knowledge which in turn was applied in our project.

## **2) Motivation**

Of Late, with the increase in the population, several concerns regarding transportation, resource utilization and environment has risen. To better manage this scenario, ride sharing is proposed as an effective approach for these challenges. Added to that is the availability of internet and smartphones, ride sharing has become an inevitable part of a common man's lifestyle. Additionally, ridesharing provides a wealth of benefits with reduction in travel costs, traffic congestion, energy consumption and pollution resulting from these vehicles servicing multiple trips. This report proposes an effective approach for static matching of trip requests, i.e a data analysis framework that achieves fast matching of prior known trips. The proposed alternative rests on the idea that multiple passengers can be picked up from the pickup location (hotspots) and transported to their different desired destinations. It increases the average occupancy of the taxis, and subsequently decreases the total travel distance to service these requests. This ridesharing approach is subject to capacity, time and monetary constraints. The monetary constraint provides incentives for both the passenger and the taxi drivers. Passengers will end up paying less as compared to no ridesharing scenarios and the drivers will earn more per mile in ride sharing scenarios. The report implements and evaluates the framework using data of New York City - Taxi and Limousine Commission. The results demonstrate the findings, effectiveness and the performance analysis of the proposed framework.

The rest of the report is divided as follows. Section 3 shows the objectives behind the project and approach, Section 4 gives the assumptions we made, Section 5 describes the algorithm used, Section 6 begins to explain the technical aspects of concepts used in the algorithm. Section 7 shows the high level system architecture, Section 8 & Section 9 show the data cleaning and

processing steps, Section 10 explains the three flavours of our project which matches  $k=2$  rides,  $k=3$  rides and  $k=2\&3$  rides. Section 11 onwards describes the gains in terms of cost savings for the driver and passenger, Section 12 shows our experimental setup and finally we showcase all results and conclusion in Section 13 & 14.

### **3) Objectives**

The objectives of the project are as follow:

- Reduce traffic congestion on the road by reducing the number of taxis per passenger amounting to roughly 40% of the total road traffic in NYC
- Reduce the Per Mile Riding Cost for each passenger
- Increase the Per Mile Earning of the driver when driving in ride-sharing
- Increase the profits earned by the company behind ride-sharing
- Compute the optimal match with minimum computational delays
- Reduce the total distances travelled , thereby also resulting in reduction in total time taken to travel

### **4) Assumptions**

- Single point of pick-up is assumed for the ease of computation : JFK Airport
- Capacity Restrictions:
  - Number of people Per Request upto 3 Passengers
  - Total Capacity of the car = 4 Passengers
  - Maximum number of trips that can be merged = 3 Trip Requests
- No passengers is willing to walk to or from their final destination or initial pick-up point
- It is assumed that all passengers are willing to rideshare within the Acceptable Delay Time. The Acceptable delay time is defined as - *“A percentage of the total Original Time (OT) it will take to go from the source to the destination directly (without ride-sharing).”*  
For this model the maximum acceptable delay time is as follows:
  - Travel Distance > 5 Miles, Maximum Acceptable Delay = 30% of the Original Travel Time  $t$
  - Travel Distance  $\leq$  5 Miles, Maximum Acceptable Delay = 50% of the Original Travel Time  $t$
  -
- The model also assumes that there is a 100% Taxi Availability for all the trip requests. This implies that for every ride there is always a taxi to service the ride.
- All weather and traffic condition are ignored

## **5) Proposed Solution(Algorithm):**

The proposed algorithm is the basis on which matching of trips occurs. In this approach we have used a heuristic approach to minimize the search space and achieve maximum matching and maximum monetary savings. The algorithm picks up candidates of matching greedily, as in, for each trip it finds the best possible match, and for this candidate pairing it proceeds to find the third possible match (if any). Hence, it grows greedily.

The below steps explain the algorithm in a brief manner. For manageability purposes the data was divided into weeks and the algorithm was run on each week, by considering pools of 5 minutes. Detailed explanation of the experiments is provided in section 12.

1. Time Window { 5 mins } pools
2. List all the trip requests for every pool of 5 mins.
3. Order the trip requests by the earliest arrival time and save it in a list ( $L_T$ )
4. For every Taxi request( $T_r$ ) in the ordered list:

### ***METHOD: FIND\_PAIR( $T_r$ ):***

- a. Calculate the angle with respect to  $T_r$  and all the other trip requests within the same pool
  - b. Form a subset of candidate pairings by selecting trips with angles  $\leq 30^\circ$  to  $T_r$
  - c. Constraint Checking:
    - i. C1 : Capacity constraint
    - ii. C2: Re-order the trips according to the nearest drop off distance from the pickup point (hotspot).
    - iii. C3: Delay Constraint
  - d. If satisfied for  $T_r$ , add the candidate  $T_r$  to a running list
    - i.  $T_r : \{\text{Candidate 1, Candidate 2, .....}\}$
5. From all possible candidates that fulfil constraints for  $T_r$ , Find candidate with minimum increase in total distance. This will be the main pairing
  6. Remove the match from the ordered list ( $L_T$ )
  7. Goto step 3 and repeat the algorithm for rest of the requests

## **6) Technical Calculations:**

### **a) Calculation of Angles:**

A major part of the core algorithm approach is to calculate the angles of Source- Destination 1 with all other destinations in the time pool. A major challenge in achieving this is our destination points are in the format of Latitude & Longitude, but, for efficient calculation of angles we require points in the coordinate system (x,y,z) and directed vectors. Hence, we first needed to

convert the latitude and longitude points into a 3-dimensional point system. The next step was to convert these points into directed vectors, this ensures similar angle calculations along different directions of the road. Technically, this was achieved using PYTHON and some in built libraries.

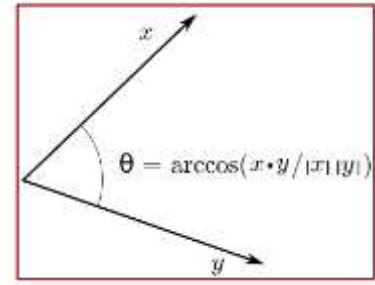


Fig: Depicting angle calculations

The steps below explain the steps taken to calculate angles between sets of points. The same method was used to calculate angles across all data points.

**METHOD: CALCULATE\_ANGLES(S-D1, D2)**

1. Convert the points to numpy latitude/longitude radians space  
 $a = np.radians(np.array(A))$
2. Convert the radian point to 3-dimensional point space using the formula:  
 $(x,y,z) : ( \cos(lat) * \cos(long), \cos(lat) * \sin(long), \sin(lat) )$
3. Form vectors in 3D space  
 $A = X - Y, \quad B = Y - Z$
4. Calculate Angle using the formula:

$$\cos \alpha = \frac{\overline{a} \cdot \overline{b}}{|\overline{a}| \cdot |\overline{b}|}$$

## 7) System Architecture:

The system architecture is explained in the diagram below:

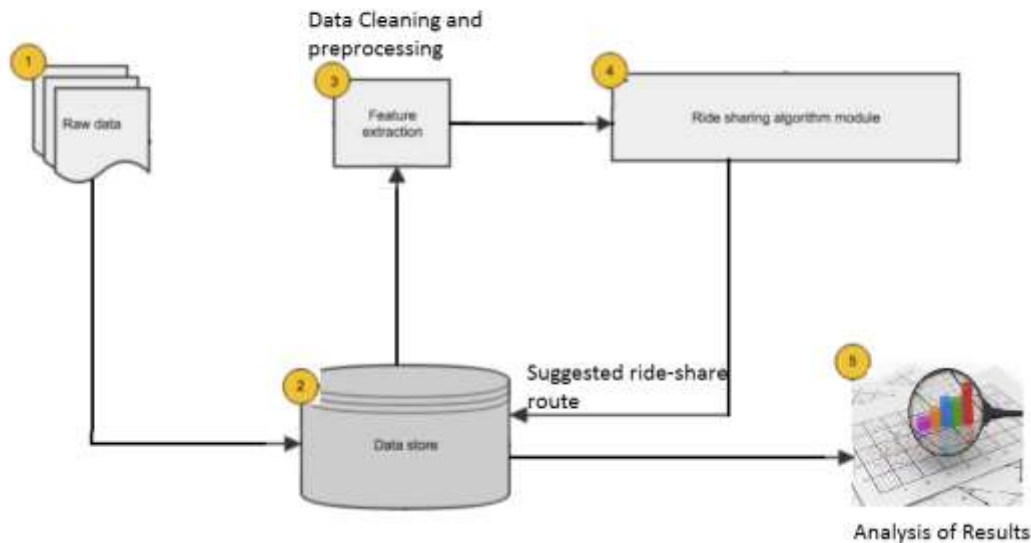


Fig 1: System Architecture of the Taxi Ridesharing System

The raw taxi data is imported into our database management system (POSTGRESQL). Further it is processed and cleaned using assumptions mentioned in Section 8. The next step is to begin creating trip requests from this cleaned data. For each of the requested trip in 5 minute window pools the matching algorithm runs and creates the taxis according to the matched trips. The last phase deals with analysis and outputs.

## 8) Data Cleaning

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
	Ven	tp	tp	pass	trip	pick	pick	Rate	store			pay							
1	ID	datetime	datetime	count	distance	longitude	latitude	ID	flag	longitude	latitude	type	amount	extra	tax	tip	tolls	improve	total
2	2	1/1/16 00:00	1/1/16 00:00	2	1.1	-73.99	40.735	1	N	-73.9818	40.7324	2	7.5	0.5	0.5	0	0	0.3	8.8
3	2	1/1/16 00:00	1/1/16 00:00	5	4.9	-73.981	40.73	1	N	-73.9445	40.7167	1	18	0.5	0.5	0	0	0.3	19.3
4	2	1/1/16 00:00	1/1/16 00:00	1	10.54	-73.985	40.68	1	N	-73.9503	40.7889	1	33	0.5	0.5	0	0	0.3	34.3
5	2	1/1/16 00:00	1/1/16 00:00	1	4.75	-73.993	40.719	1	N	-73.9622	40.6573	2	16.5	0	0.5	0	0	0.3	17.3
6	2	1/1/16 00:00	1/1/16 00:00	3	1.76	-73.961	40.781	1	N	-73.9773	40.7585	2	8	0	0.5	0	0	0.3	8.8
7	2	1/1/16 00:00	1/1/16 00:18	2	5.52	-73.98	40.743	1	N	-73.9135	40.7631	2	19	0.5	0.5	0	0	0.3	20.3
8	2	1/1/16 00:00	1/1/16 00:26	2	7.45	-73.994	40.72	1	N	-73.9664	40.7899	2	26	0.5	0.5	0	0	0.3	27.3
9	1	1/1/16 00:00	1/1/16 00:11	1	1.2	-73.979	40.745	1	N	-73.992	40.7539	2	9	0.5	0.5	0	0	0.3	10.3
10	1	1/1/16 00:00	1/1/16 00:11	1	6	-73.947	40.791	1	N	-73.9208	40.8656	2	18	0.5	0.5	0	0	0.3	19.3

Fig: Table showing the raw input data

The raw data (Fig. 1) used for this project was obtained from New York City Taxi and Limousine Commission <sup>[1]</sup>. The data selected was for the year 2016. A few attempts were made to import the data for the entire year into the database but the NYC Taxi and Limousine Commission altered the structure and attributes of the tables (Fig. 2) they used to stored their data. Hence, July onwards, a different structure is used to stored the taxi trips which is depicted the figure below and highlighted in red. The new structure has pick-up and drop-off ID's instead of latitude and longitude pairs. Further, the NYC TLC did not release an index to map those IDs to specific latitude and longitude pairs in and around the Manhattan area. Hence, we could only use data from January 2016 - June 2016.

Jan - June	Vendor ID	tpep pickup datetime	tpep dropoff datetime	passenger count	trip distance	pickup longitude	pickup latitude	Rate code ID	store and fwd flag	dropoff longitude	dropoff latitude
	2	1/1/16 00:00	1/1/16 00:00	2	1.1	-73.99	40.735	1 N		-73.9818	40.7324
	2	1/1/16 00:00	1/1/16 00:00	5	4.9	-73.981	40.73	1 N		-73.9445	40.7167
	2	1/1/16 00:00	1/1/16 00:00	1	10.54	-73.985	40.68	1 N		-73.9503	40.7889

July - Dec	Vendor ID	tpep pickup datetime	tpep dropoff datetime	passenger count	trip distance	Rate code ID	store and fwd flag	PU Location ID	DO Location ID
	1	7/10/16 06:56	7/10/16 06:59	1	0.5	1 N		263	236
	2	7/10/16 10:50	7/10/16 10:55	5	1.34	1 N		142	163
	2	7/10/16 10:50	7/10/16 11:08	1	9.48	1 N		74	66
	1	7/10/16 10:50	7/10/16 10:55	1	1	1 N		264	264

Fig. 2: Tables showing changes in Pickup points between months of Jan and July 2016

### Data Cleaning Constraints:

The following constraints were imposed to clean the data before the data was preprocessed as trips:

- **passenger\_count** : The per trip request passenger count was kept in the range of [1,3] as more than or equal to 4 passengers per trip request cannot be accounted for ride sharing
- **trip\_distance** : The trip distances in the data set that were less than zero, due to a faulty measuring device or some error in storing the data were removed. The records with trip distance more than 200 miles were also removed
- **dropoff\_longitude** : All records with the value as zero for this attribute were dropped from consideration
- **dropoff\_latitude** : All records with the value as zero for this attribute were dropped from consideration
- **payment\_type** : This attribute could only take the value of either 1 or 2. Any other value for this attribute rendered the corresponding record as worthless and was dropped
- **fare\_amount** : The fare amount of all the records had to be greater than zero and less than \$2000. All records corresponding to any other value for this attribute were removed
- **total\_amount** : The total amount, which is the sum of fare\_amount, tax, service charge, tip, toll etc had to meet the same constraint as fare\_amount. Also, all records with total\_amount less than fare\_amount were ignored
- **tpep\_pickup\_datetime** : All the records in which the pickup\_datetime was larger than the dropoff\_datetime were removed



## 9) Data Preprocessing

Selected Source: John F. Kennedy (JFK) International Airport

All rides originating from this source point were scheduled for ride sharing with the assumption that, all the rides that match the criteria are willing to ride share. Since, the area belonging to JFK cannot be realistically considered as a single point (would result in very less destination points) we used all points within a radial area to a center point of JFH, physically present on the road as source points. To select all the points within the parameter of the airport, Haversine formula<sup>[2]</sup> (Spherical Cosine Law formula) was used.

Having decided the single point of pick-up at JFK International Airport, the next challenge was to normalise all the pick-up point coordinates that lie inside the perimeter of the JFK International Airport to one single point. All the pick-up points were then normalised to the central point w.r.t which the Haversine formula was applied.

$LATITUDE\ BETWEEN = latpoint - (3.5 / 111.045)\ AND\ latpoint + (3.5 / 111.045)$
$LONGITUDE\ BETWEEN = longpoint - (3.5 / (111.045 * COS(RADIANS(latpoint))))\ AND\ longpoint + (3.5 / (111.045 * COS(RADIANS(latpoint))))$

Fig 3: Formula used for Haversine Distance Calculation

Following the point normalization, the next step was to re-calculate distances from the new source point to respective destination points. In order to perform this keeping in mind the road distances (without weather and traffic conditions) the service Open Source Routing Machine<sup>[3]</sup> (OSRM) , with API “route” was used. OSRM- ROUTE API takes a HTTP request with latitude and longitude pair of points and returns a JSON object which contains the road distance in meters and duration in seconds. Below image shows the HTTP request and returned JSON object.

<b>OSRM API Service:</b>	
--------------------------	--

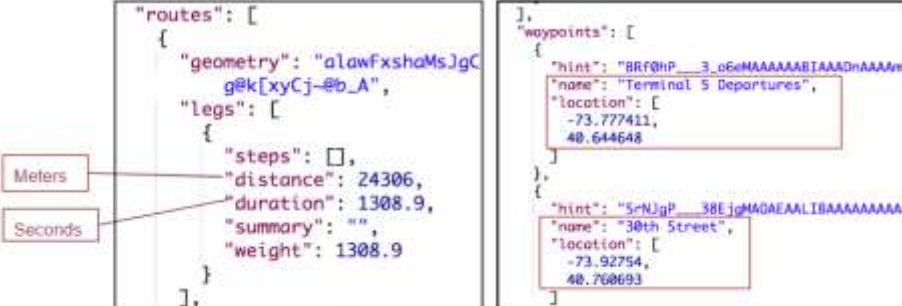
<b>JSON Response:</b>	
-----------------------	--

Fig 4: OSRM Request and JSON response format

Sending HTTP requests over the network incurs a lot of latency, hence, the OSRM service was set up to run locally using the road map of New York and compiling the OSRM backend service to run on it. All steps to run OSRM can be found in the github link of our code base.

Thus, for the normalized source point and all destination points the road distances and time were re-calculated and stored in our database.

## 10) Implementation:

Using our heuristic model approach, we developed 3 variants of the same algorithm. This enabled us to compare the matching of 2 trips vs 3 trips, and a new hybrid approach. Below briefly explains each flavor of the algorithm.

- **K=2:** (Matches 2 trips)

This variant attempts to match two trip requests only. Such that the deviation from the original route for the passenger dropped second will be within the acceptable delay time (The accepted delay time calculation is explained in section 2). If there are multiple candidate pairs for a particular trip request, the algorithm chooses the candidate which results in minimum increase in total distance travelled after ride-sharing. Once the final match is found, it is removed from the waiting queue and added to the final pairing list. The algorithm runs in a recursive manner until the waiting queue is empty for a particular pool and produces matchings of 2 trips. If for a trip the algorithm cannot find a feasible match, the trip is assigned to travel alone. The working of the algorithm is illustrated in the picture given below.

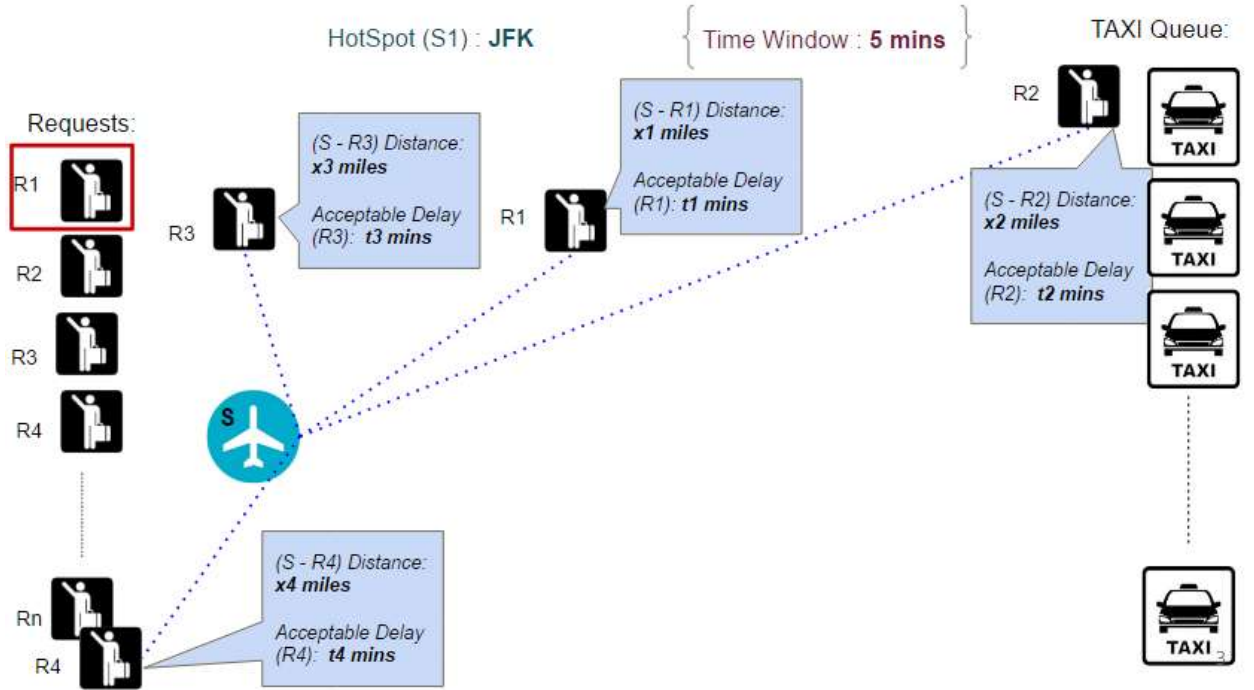


Fig 5: Image showing working of  $k=2$  flavor of the algorithm

The above figure shows wording of the  $K=2$  algorithm. Consider all the requests shown in the left side to be in a pool of 5 mins. The requests are ordered according to the earliest request time (pickup\_time). To start the matching process, all the angles w.r.t request R1 is calculated on the fly, with respect to all other requests in the pool. Further, the travel distance and time is calculated using the OSRM service as needed.

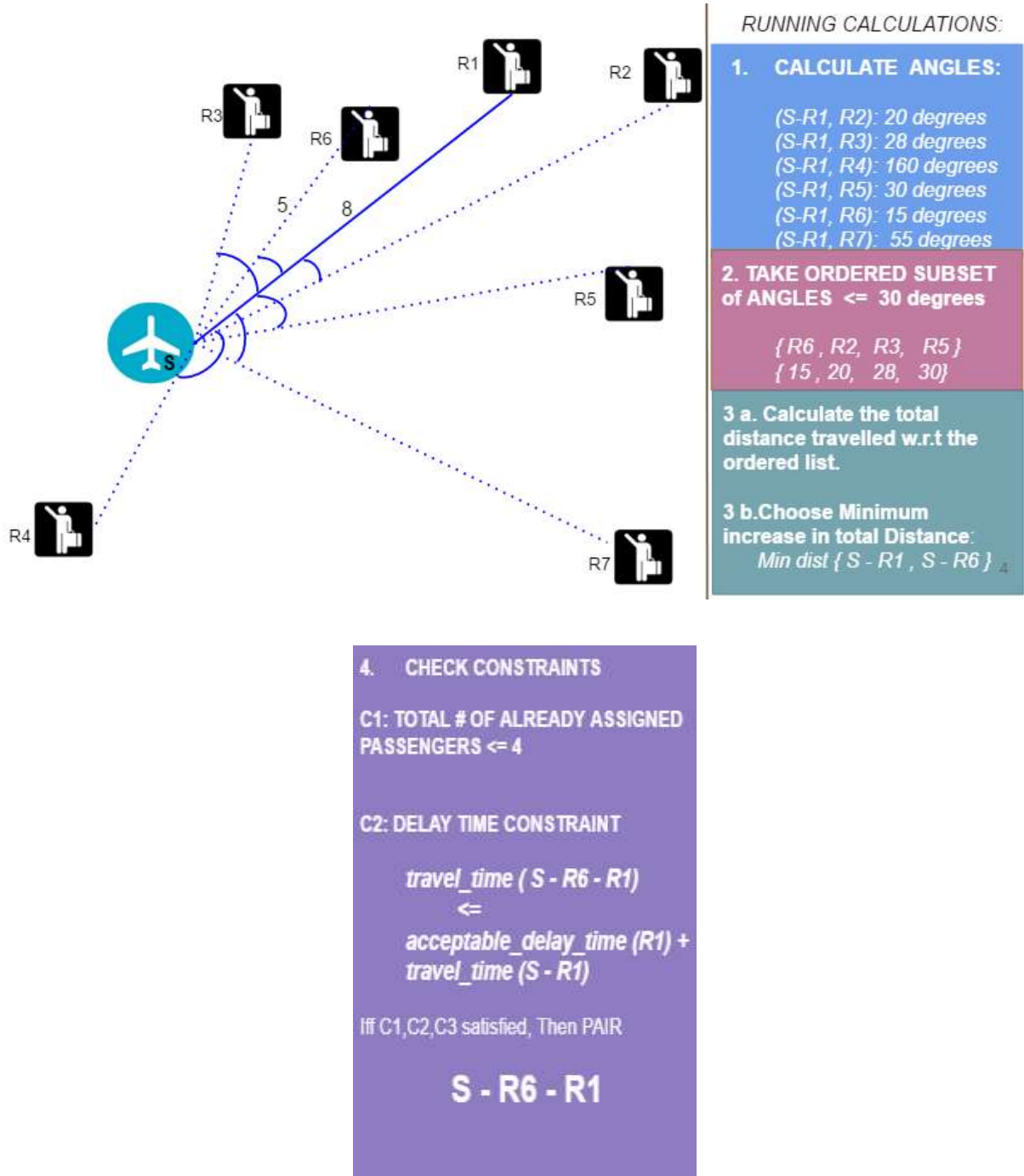


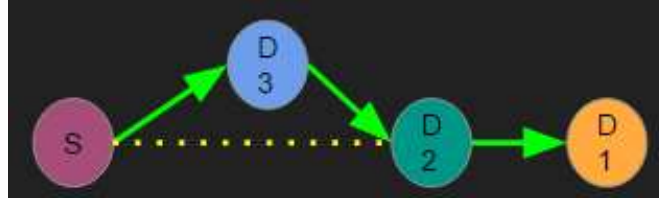
Fig 6 : Working of K=2 algorithm

Fig 6 shows all the candidate pairing for R1, which satisfies the capacity and delay time constraints. Once all the candidate pairs are found, the algorithm chooses the final pairing by selecting the one with minimum increase in total distance. If at the end there are requests for which the algorithm could not find a perfect pairing, it is assumed that these requests are given separate taxis. Such passengers are given a 30% discount on the taxi

fare as a concession for willing to rideshare. The cost metrics is explained in detail in the Cost Calculation section.

- **K=3:** (Match only 3 trips)

This variant is similar to K=2, the only difference is that the algorithm tries to match 3 requests only. If a match of 3 requests isn't found in the current pool, then the passengers go alone. There is no 2-trip pairing performed in this variation of the algorithm. Consider that the figure below is a possible pairing:



*Fig 7: Image showing merging of 3 trips*

In order to calculate the acceptable delay time after the addition of D3, we need to calculate the effect it has on all destination drop-offs post insertion. Firstly, the delay time for D2 is calculated and verified as follows:

$$\text{Eq. 1} = \text{Travel\_time}(S - D3 - D2) \leq \text{Travel\_time}(S - D2) + \text{Acceptable\_delay}(D2)$$

If Equation 1 is satisfied, the algorithm proceeds to calculate and verify the delay time for the passenger dropped third which in this case is D1. The equation is as follows:

$$\text{Eq. 2} = \text{Travel\_Time}(S - D3 - D2) + \text{Travel\_Time}(D2 - D1) \leq \text{travel\_time}(S - D1) + \text{acceptable\_delay}(D1)$$

If both above equations are satisfied, then D3-D2-D1 is considered a final match. The requests which cannot be matched in any 3 pairs possibility is given a separate taxi and considered a single trip. Mainly, this flavor of the algorithm performs a pairing of 3 trips or 1 trip. In the experimental result analysis section, we represent this variation as K=3. It is also worthy to mention that this flavor gives a much higher saving in terms of time saved, distance travelled, taxis used, etc. as compared to K=2.

- **K=Hybrid:** (Match 2 & Match 3)

The savings could be further enhanced, if K=2 and K=3 matching is combined. This results in maximum pairing possible with our angle approach. Hence, in the hybrid implementation, the algorithm first tries to find a possible 3 pairing greedily, if it fails it concludes in 2 pairs. All other trips are sent alone. The working of the hybrid implementation is like K=3, though there are little improvements to it. Taking the scenario given in Fig. 7, if any of the delay constraints fail (Eq.1 or Eq. 2), the algorithm attempts to find an alternative solution unlike K=3. It checks if a pairing of D3-D2 or D2-D1 satisfies all the constraints, which is like K=2. After all the layered verification and validation, if there are requests which cannot be paired to a match of 3 or 2 trips, it is

given a separate taxi. The results for hybrid approach gives the most savings from among the 3 variants and is explained in detail in the results section of this report.

## **11) Generalized Cost Savings (K=2)**

### **a) Driver's Earning:**

Cost Evaluation Parameters:

<i>Per - Request pickup (in \$)</i>	<i>Per - Mile(in \$)</i>	<i>Per - Minute of Journey(in \$)</i>
-------------------------------------	--------------------------	---------------------------------------

Earning Formula:

$$\begin{aligned}
 &(\$Per\text{-}Request\ pickup * n) && + \\
 &(\$Per\ Mile * total\_trip\_distance\_in\_miles ) && + \\
 &(\$Per\ Minute\ of\ journey * total\_trip\_time\_in\_minutes)
 \end{aligned}$$

Where,  $n=1$  for no ridesharing  
 $n=2$  for ridesharing

### **Per Mile Gain(c):**

$$\frac{(\text{Combined Original No Ridesharing Cost } (Q) - \text{Total Ridesharing Cost } (Q))}{\text{Total Ridesharing Distance}}$$

### **Passenger Fare:**

The rider pays per mile  $F(D)$  which is a transport authority set cost function.

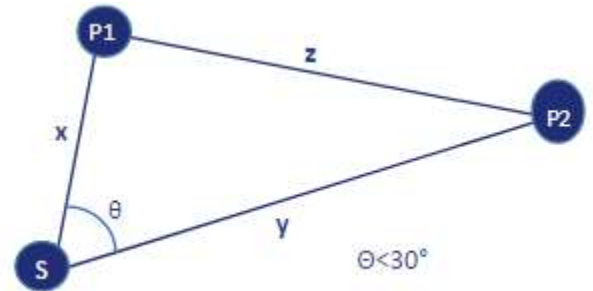


Fig 8: Diagram depicting cost savings of ride sharing vs non-ridesharing

Passenger 1:

<b><u>Alone</u></b>	<b><u>Ride Sharing</u></b>
$F(D) * x$	$x * (F(D) - c)$

Passenger 2:

<u>Alone</u>	<u>Ride Sharing</u>
$F(D) * y$	$y * (F(D) - c)$

$c$  = Gain per Mile (Calculated from the Driver Earnings above)

This is a sample of cost function for  $K=2$  approach. It can be further enhanced to fit into the other two variants of the algorithm.

**Note:** For all the rides that could not be paired, the rider gets 30% flat discount on the actual billed distance with no ride sharing.

## **12) Experimental Setup:**

This section explains the running of our algorithm and the data statistics used across runs.

<b>Year: 2016</b>	<b>#Total Trips</b>	<b>#Total Pools - Having Trips (Window - 5 mins)</b>	<b>Average pool size</b>
Week 1( <i>January W1 &amp; W4</i> )	25,794	1,029	~35 (34.59)
Week 2 ( <i>February W1</i> )	35,418	1,607	~29 (29.19)
Week 3 ( <i>February W2</i> )	25,166	1,081	~32 (31.57)
Week 4 ( <i>February W3</i> )	18,956	822	~31 (30.51)
Week 5 ( <i>February W4</i> )	48,562	1,946	~32 (32.36)
<b>Total:</b>	<b>153,896</b>	<b>6,485</b>	

*Fig 9: Table showing Data Statistics of all performed runs*

For the month of January, data was available for dates from 1st-4th and 29th-31st, So we considered it to be one week for the ease of calculation. The total number of pools column, in the above table, accounts only those pools which have some trips in them (count of trips  $> 0$ ). The empty pools are not considered, since it has zero computation time.

The algorithm was implemented for 5 weeks. Since 3 variants of the algorithms were applied on each of these weeks, it included running the algorithm for 15 rounds. The data was distributed among three team members for faster execution of the project and results were consolidated.



### 13) Experimental Results

The results of the experiments conducted are depicted in the form of graphs. Savings in terms of Distance, Time, Cost and Taxes will be discussed sequentially in this section. The time complexity of the algorithm will also be discussed in terms of efficiency of the algorithm.

#### a) Savings in terms of Distance Travelled

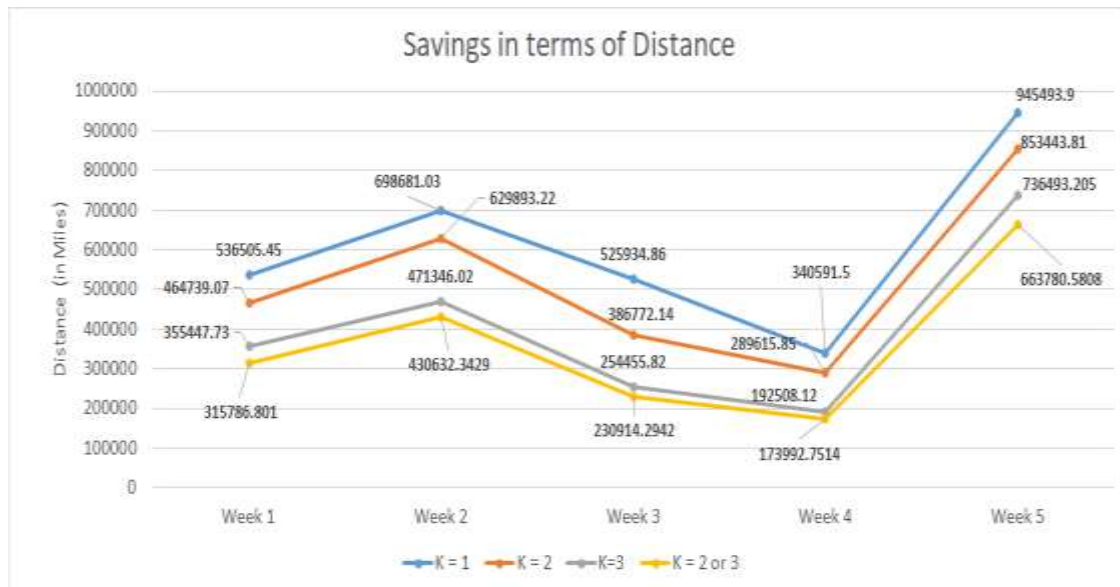


Fig 10: Graph showing distance saved in miles per week

The above graph represents the savings observed for the different variants of the proposed algorithm over the period of 5 weeks. The X -Axis shows the respective week, the Y-Axis shows distance travelled in miles. Each data point represents the number of miles travelled during a week in a fixed ride sharing scenario. To make the above graph more comprehensible, the same data is represented in the form of 'Percentage Savings per week in each variant of the algorithm in the graph below.



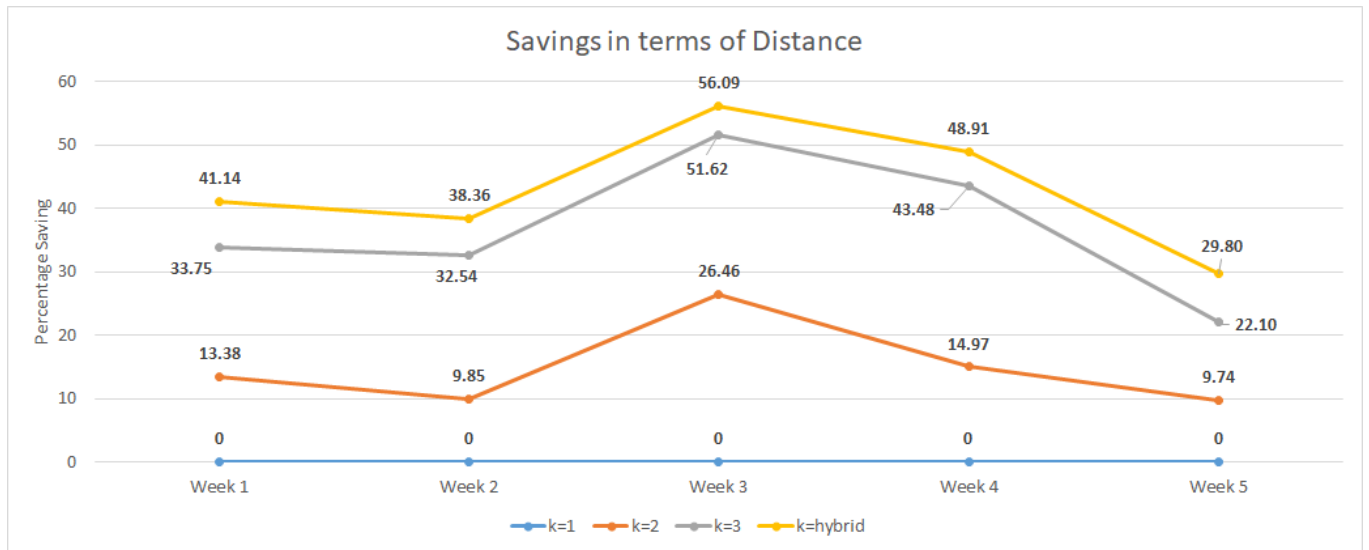


Fig 11: Graph showing distance saved in miles per week (In Percentages)

From the graph it is clear that the algorithm which aims to match only two trips achieves a lower saving than matching three trips. This is intuitive as when we merge three trips we are essentially reducing the overall distance greatly. Further, it can also be noted that the K=3, K=Hybrid algorithms performance almost overlaps with marginal changes. Overall, it can be observed in both the graphs, K= hybrid performs the best as compared to the other two variants, with a maximum saving achieved in the distance travelled.

### b) Savings in terms of Time

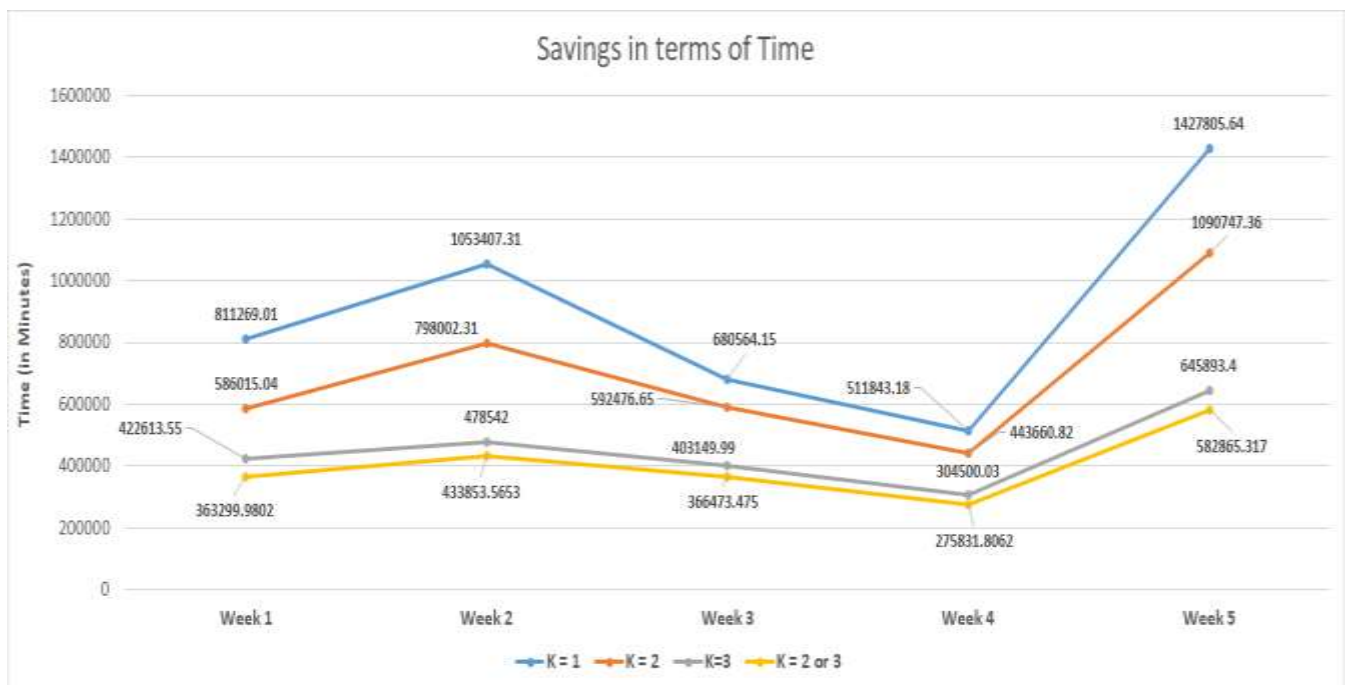


Fig 11: Graph showing Time saved in minutes per week

The graph shown above represents the savings in terms of time (in minutes). The proposed model exhibits results that reinforce the theoretical concept of proportionality between distance and time. This graph is proportional to the graph that represents savings in terms of distance (in miles) indicating that if the distance is saved so is the time. To better understand the above graph, another graph that depicts savings in percentage time saved is depicted below.

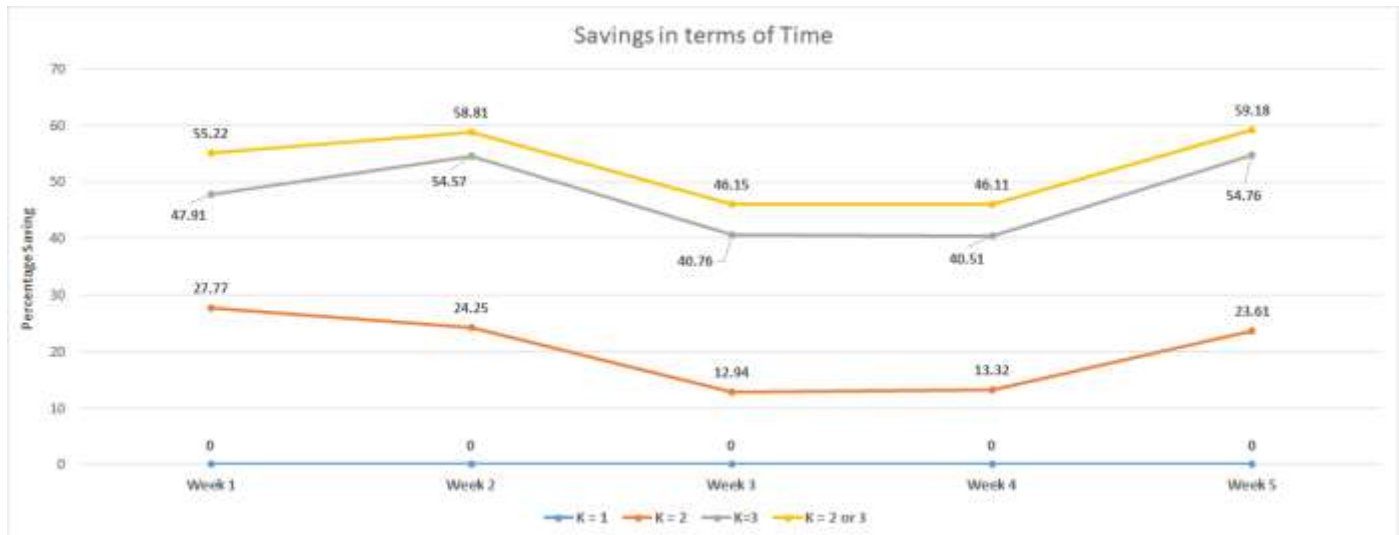
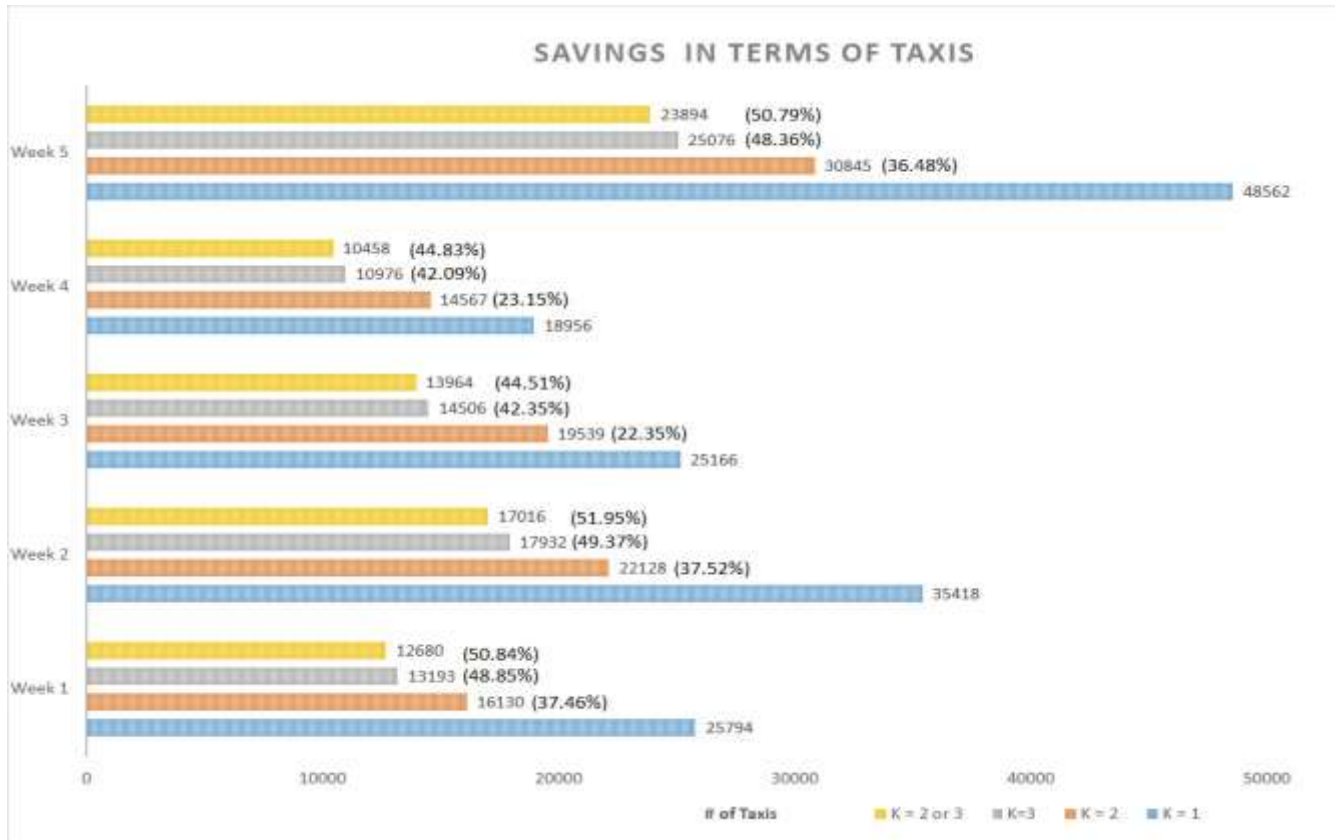


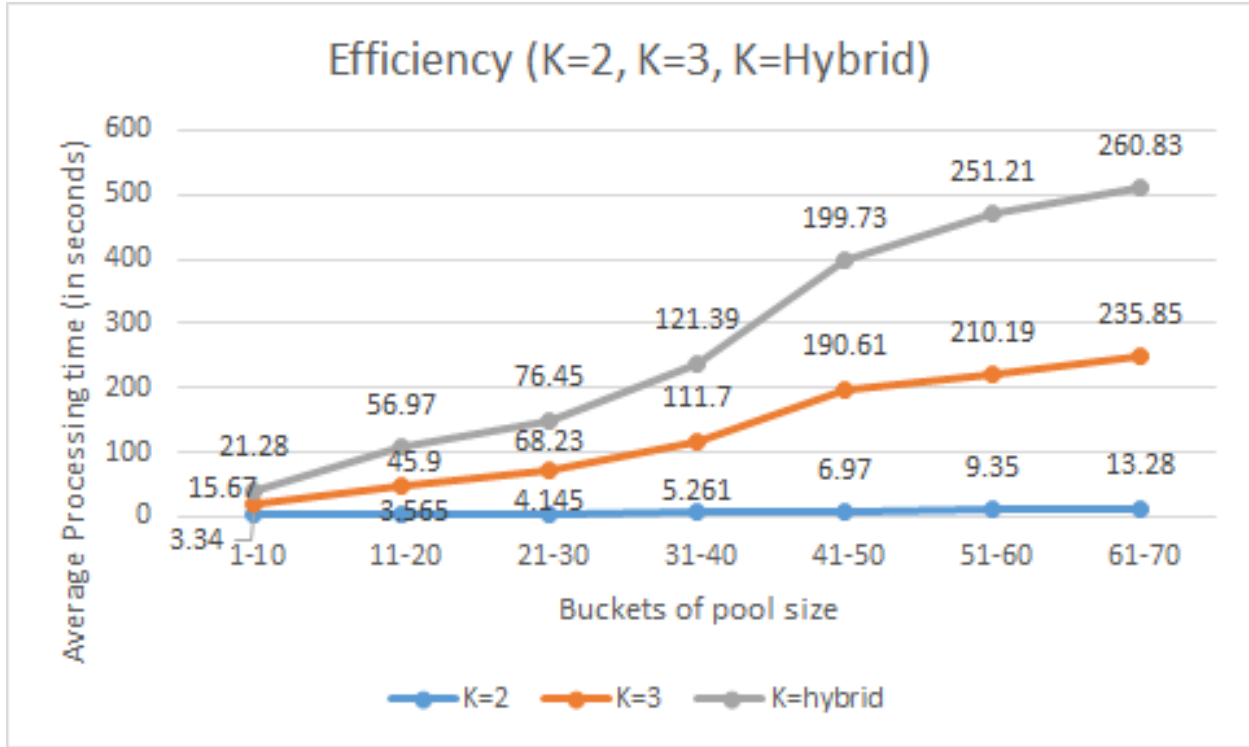
Fig 12: Graph showing time saved in minutes per week (Percentages)

### c) Savings in terms of Taxis



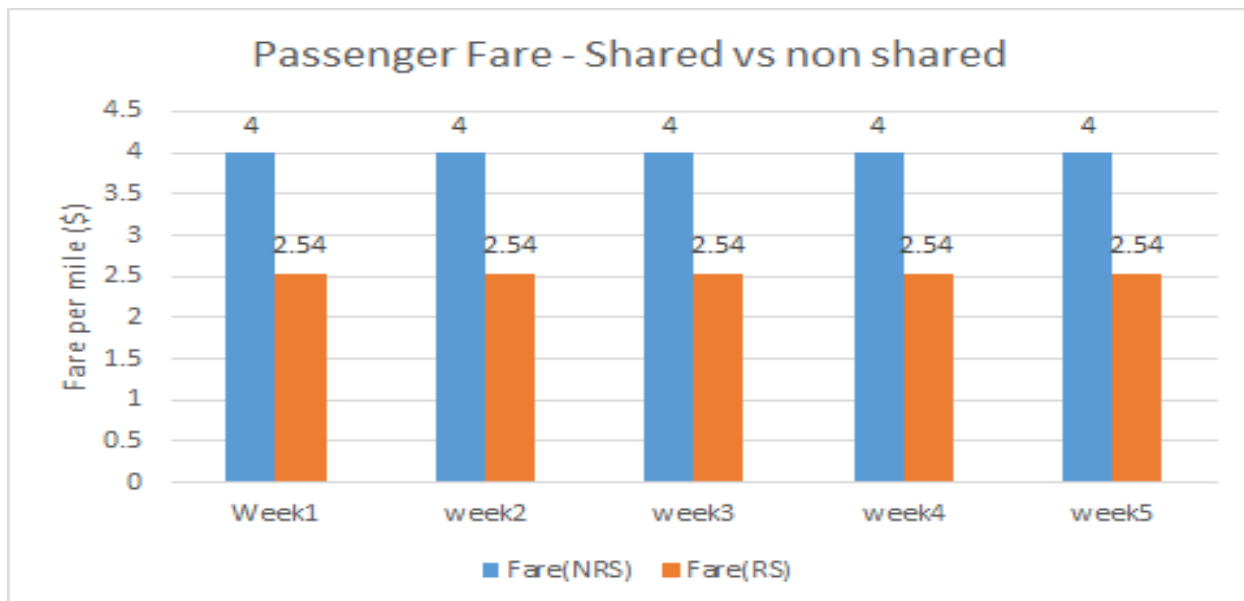
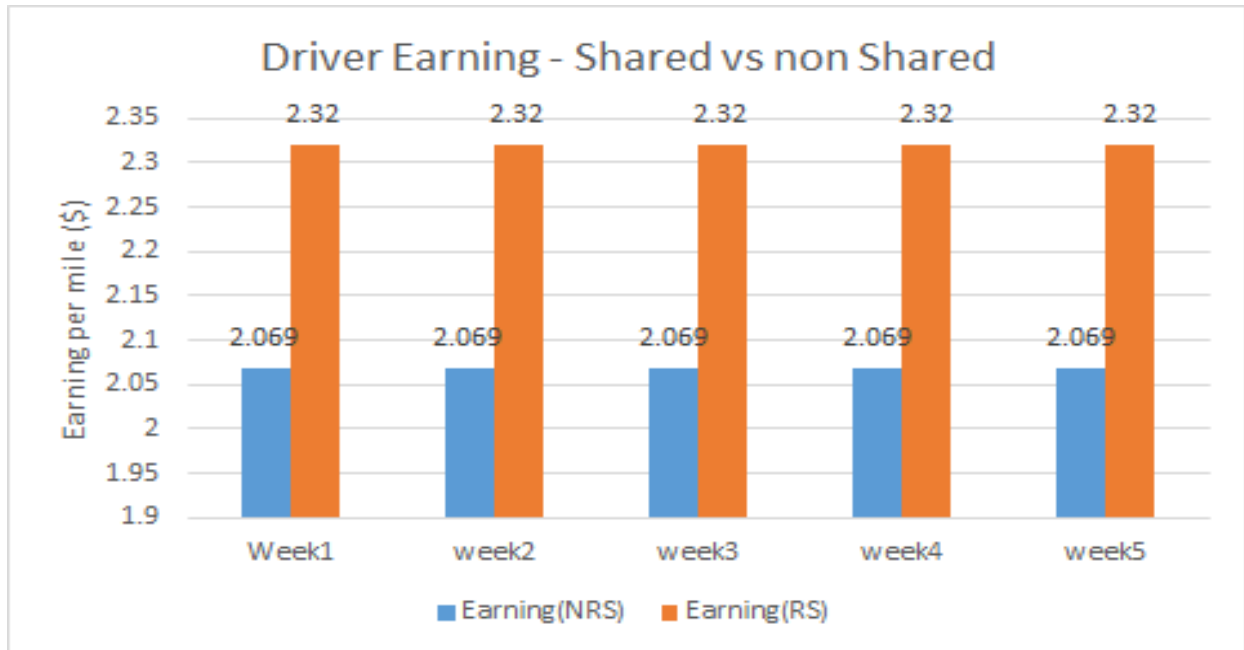
This analysis represents the number of taxi trips saved for each of the variant of the algorithm. Here, the blue bar represents the number of taxis required without ride sharing to service the requests. Since it represents a no ride sharing scenario, it also signifies the total number of trips in that week. The percentage against each bar represents the number of taxis saved. Now the assumption made here is that each taxi makes a single trip since the taxis are not being tracked in real time. Yet, the savings observed are impressive and go as high as almost 52%.

#### **d) Processing Time of the Algorithm**



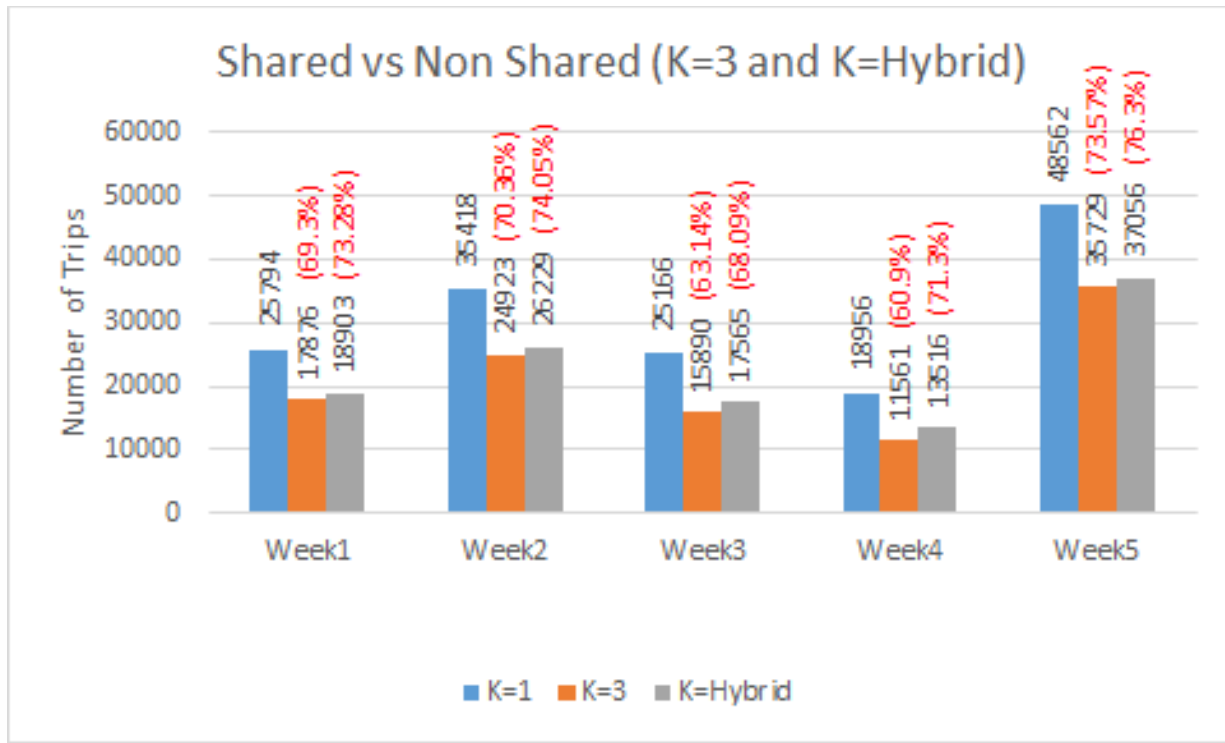
The above plot represents the time it takes to process all the trip requests and finalize the ones that get paired. As is seen from the plot, the processing time for pairing just two trips ( $k=2$ ) together is relatively constant w.r.t the curve for processing  $k=3$  and  $k=hybrid$ . Also, supporting the intuitive hypothesis, the processing time increase directly proportional to the pool size. Meaning, as the number of trips increase in the pool, it takes longer and longer to find the perfect match and maximize savings across all matrices. Comparing the curve of  $k=3$  and  $k=hybrid$ , it can be well observed that  $k=hybrid$  takes longer processing as the pairing of 3 is sought first and if not found pairing of 2 is done and the next iteration looks for another set of 3 to be paired and so on. This sums up to additional computations and hence increase in the processing time.

*e) Cost Saving of the algorithm*



The plots above show the driver earning per mile and the passenger fare per mile without ridesharing and with ride sharing for 5 weeks. The plot runs consistently for all the weeks. And it shows that the driver's earning per mile with ride sharing is more than without ridesharing. Similarly, passenger fare per mile with ride sharing is less than without ridesharing. The difference between the passenger fare and the driver earning ( $2.54 - 2.32 = .22$  cents) is retained by the taxi company as the profit. These plots of averaged values show the consistency of the cost function.

#### f) Shared vs Non-Shared Trips



This plot represents the percentage of total rides that could rideshare (in k=3 and k= Hybrid setting). K= hybrid could pair more rides than k=2 as indicated by the plot. This percentage averages around to be 73% for k=hybrid which is a clear indicator of the highest savings.

## 14) Project Scripts and Files

The script files and the preprocessed and cleaned data files have been posted online and are available on [GitHub](https://github.com/yogeetak/TaxiRideSharing). Please follow <https://github.com/yogeetak/TaxiRideSharing> to the files.

The project dependencies and setup & execution instructions are mentioned in the readme.md file as well as in this report.

## 15) Software and Libraries Used

The software and APIs used for this project are as follow:

1. PostgreSQL
2. Python 3.6
3. OSRM (setup locally to minimize the latency)

The python libraries used for this project are as follow:

1. csv
2. psycpg2
3. datetime & timedelta
4. sys & os
5. operator
6. time
7. json
8. numpy
9. math
10. urllib
11. urlopen
12. Request

## 16) Conclusion

Using a Greedy Algorithm combined with a Heuristic approach 3 variants of the same algorithm were devised namely  $K = 2$  (pairing at max 2 rides),  $K = 3$  (pairing at max 3 rides) and  $K = 2/3$  or the hybrid approach (pairing at max 3 else 2 rides).

The Maximum savings were achieved with  $K = 2/3$  (hybrid approach). The savings are as detailed below:

- 43 % of savings in terms of distance
- 53 % of savings in terms of time
- 53 % of savings in taxis used
- 73 % of total rides were shared

With these results, it is pretty much evident that the project could meet its objectives.

## 17) References

1. [http://www.nyc.gov/html/tlc/html/about/trip\\_record\\_data.shtml](http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml)
2. <http://www.plumislandmedia.net/mysql/haversine-mysql-nearest-loc/>
3. <http://project-osrm.org/docs/v5.5.1/api/#general-options>
4. <http://stackoverflow.com/questions/42584259/python-code-to-calculate-angle-between-three-points-lat-long-coordinates>