DATABASE MANAGEMENT SYSTEMS (CS581)

RIDESHARING

FINAL PROJECT REPORT

TEAM 4

Jennifer Jesuraj

Mahak Gupta

Shilpa Amarendrababu Sujatha

Viren Mody

UNIVERSITY OF ILLINOIS AT CHICAGO

SPRING 2017

Table of Contents

1. Introduction
2. Project Proposal(Objective and Assumptions)
    2.1. Objective
    2.2. Assumptions
3. Algorithm Description
    3.1. Shareability Definitions and other Terminology
    3.2. Constraints
    3.3. Pseudocode of the Algorithm
4. System Design
5. Technical Components
    5.1. Jupyter Notebook and Python 3
    5.2. Plotly
    5.3. GraphHopper
    5.4. Pandas Library in Python
6. Implementation Details and Instructions
    6.1. Dataset
    6.2. Instructions
7. Experimentational Details

8. Results

8.1. Graph 1

8.2. Graph 2

8.3. Graph 3

8.4. Graph 4

8.5. Graph 5

8.6. Graph 6

8.7. Graph 7

8.8. Graph 8

8.9. Graph 9

8.10. Graph 10 a , b

8.11. Graph 11 a, b

9. Conclusion

10. References

1. INTRODUCTION

With the increase in population, the demand for vehicles has also increased. Traffic jams at peak hours in busy cities is becoming a concern. This alarming situation has led to new and innovative alternatives such as ridesharing. Carpooling has become a popular solution to deal with the increasing traffic conditions and also to prevent the adverse effects on the environment caused by the excessive pollution. Studies have shown that taxis have become an alternative for travel for most people to beat the difficulties of commuting through congested areas. 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 different ones?

The above question is handled by this ridesharing project. Obtaining separate taxis from highly congested areas is both expensive and time consuming. With lots of people arriving to high-density hubs all through the day, many taxi companies have increased their car pools at such hubs. Busy hubs as these are potential places where taxi ridesharing can be implemented quickly and easily. Ridesharing provides the benefits of cost reduction for the passengers, minimization of the distance travelled by the cabs thereby reducing fuel consumption. This reduction leads to lesser pollution as well as less congestion on the road which improves traffic conditions.

2. PROJECT PROPOSAL

2.1. Objective

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

2.2 Assumptions

Here are a list of assumptions and restrictions that we will follow in performing our static data analysis.

1) Single source: JFK Airport
2) All passengers are willing to rideshare.
3) No passenger is willing to walk
4) Max Passenger Taxi Capacity is 4
5) Pool window: Every 2, 3, and 5 minutes.
6) Traffic conditions will **NOT** considered.
7) Static matching
8) Infinite taxis

We have designed a ridesharing algorithm which aims at optimizing distance. It performs **static matching** of the trips with John F Kennedy International Airport as the source. All the trip requests arriving from the users at this source is logged and the algorithm is run for every 2, 3 or 5 minute time interval. The time frame during which the requests are merged is known as the pool window. The trip merging algorithms would also consider all the constraints such as the passenger count limits, time delay constraints and the distance constraints before combining the trips. After the algorithm suggests the most optimal merging for a particular pool window, the cabs will be dispatched for all the merged trips who are part of that particular window and then the next pool window is processed. The number of trips after merging is guaranteed to be lesser than the actual number of trip requests obtained in that particular pool window.

The assumptions considered while running the entire algorithm is that, firstly, all passengers placing a request are willing to rideshare. Secondly, the maximum number of passengers that can travel in a single taxi is set to four. Thirdly, No passenger is willing to walk after drop off. Fourthly, There are infinite number of cabs for ridesharing and lastly, traffic conditions have not been considered.

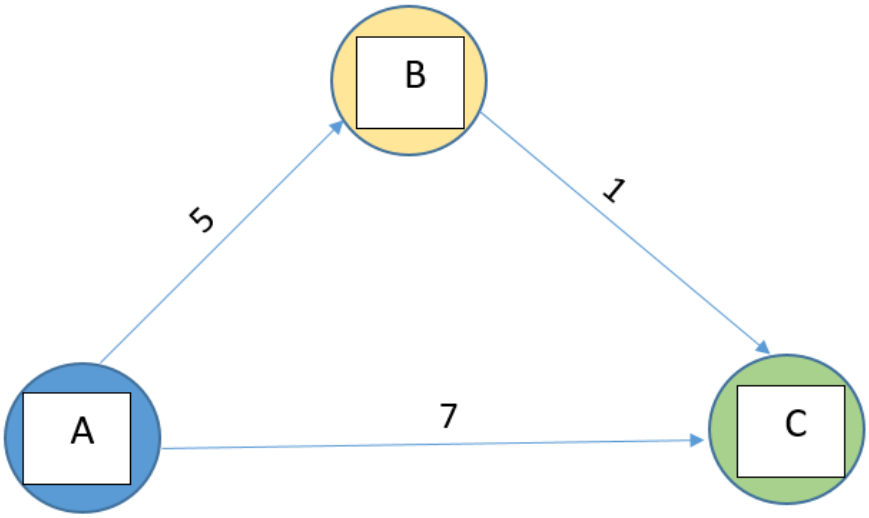
3. ALGORITHM DESIGN

The algorithm is a modified version of the solution mentioned in the paper “Quantifying the benefits of vehicle pooling with shareability networks”. The algorithms described in the paper makes use of the maximum weighted matching algorithm from graph theory to find the optimal matching for ridesharing. The paper’s first approach to the problem is to set ***k***, maximum shareable trips, to 2. The solution discussed employs a graph constructed by modeling trips as nodes, edges as the possibility of ride-sharing between trips, and edge weights as the amount of distance saved if those two trips (nodes) were combined to build a shareability network (discussed later). The maximum weighted matching algorithm the author uses for ***k = 2*** is ***optimal***.

The author then considers higher values of ***k***, maximum number of shareable trips, and claims that for ***k = 3***, there is a heuristic solution built on top of the ***k = 2*** optimal algorithm that is feasible, but for values ***k ≥ 4*** the solution is computationally difficult. When considering combining more than two trips for ridesharing, we cannot model the problem as a simple graph. The author then employs the concept of a hypergraph to model the problem, however since maximum weighted matching on a hypergraph is NP Hard, the author uses a greedy heuristic to deal with only triplets. Thereafter, the remaining trips are matched using the optimal algorithm of maximum weighted matching. The solution for k > 3 trips is not considered because it is computationally unfeasible.

With limited resources for hypergraphs and maximum weighted matching, we have implemented a variant of the algorithm discussed in the paper for dealing with the case where k>2. We used the concept of hypergraphs in the algorithm, but not their implementation as they become an NP Hard problem for k > 2. We have solved the problem using the same concepts as the paper offers, however we do not officially implement hypergraphs for triplets of trips, but we have used other data structures to maintain the concept of the shareability network or graph.

3.1. Shareability Definitions and Other Terminology



Consider the shareability graph shown above. The nodes are the trips, the edges represents the possibility of ride-sharing between the two considered trips, and the weights represent distance saved if those two trips are combined. The above graph consists of three requests A, B and C. Ridesharing is possible between the following pairs: AB, BC and AC for **k=2** and the possible triplet: ABC for **k=3**.

Pool Window: The time interval during which the requests can be merged is known as a pool window.

Delay time: Maximum amount of delay each passenger can bear due to ridesharing. It is considered as variants of multipliers (1.2, 1.3, 1.5 times) of the original expected arrival time to the destination which is specified in the historical data.

3.2. Constraints

Delay Time Constraint:

The total time taken to reach destination without ridesharing and a factor of this time that the passenger is ready to bear for ridesharing. The equation for the new expected delayed destination time is as shown below:
$DT = T_{orig} + xT_{orig}$
Where x = 0.2 , 0.3, 0.5

For example, in order to check this constraint for the triplet, ABC, with the path JFK to A, A to B, and then B to C, the following constraints have to be met:

A's destination time does not need to be checked since A is dropped off first and acts as an individual trip.

$DT_B \geq T_{JFK\ to\ A} + T_{A\ to\ B}$
The new expected delayed destination time of B has to be greater than the sum of the time from JFK to A and A to B.

$DT_C \geq T_{JFK\ to\ A} + T_{A\ to\ B} + T_{B\ to\ C}$
The new expected delayed destination time of C has to be greater than the sum of the time from JFK to A A to B, and B to C.

Distance constraint:
The total distance travelled by ridesharing should be less than the sum total of distance travelled by the individual rides to their respective destinations. The equations are as shown below:

For k=3, $D_{ridesharing} < D_{Dest1} + D_{Dest2} + D_{Dest3}$
For k=2, $D_{ridesharing} < D_{Dest1} + D_{Dest2}$

Passenger Count constraint:
To facilitate ridesharing using actual passenger count from historical data, a constraint is imposed that limits the sum total of all passengers in original rides is less than or equal to 4.
The equations are as shown below:
$P_{Dest1} + P_{Dest2} + P_{Dest3} \leq 4$

3.3. Pseudocode of the Algorithm

The following algorithm according to the paper is within a factor of ***k*** of the optimal solution. Recall, the reason for that is because the algorithm in this paper for ***k = 3*** is a greedy heuristic built on top of the ***optimal*** algorithm for ***k = 2***. First, triplets of trips are chosen greedily, and then we implement the optimal algorithm for k = 2 with the remaining trips.

Here is the pseudo code for the algorithm:
- For each pool window in dataset (pool window size: {2, 3, 5} minutes)
  - Find all combinations of triplets from trips in current pool window (i.e. Combinations for {A, B, C, D} are ABC, ABD, ACD, BCD)
  - For each triplet combination

- If passenger count constraint is met, then find all possible paths (permutations) of the triplet (i.e. for ABC, they are ABC, ACB, BAC, BCA, CAB, CBA).
- For each path/permutation
  - Query Graphhopper API for trip path (i.e. JFK to A to B to C)
  - Store the distance for entire path
- Of all the 6 paths/permutations, find the path/permutation (trip) with minimum distance, T_{min}
- If the distance constraint and delay time constraints are met, then add T_{min} to the shareability network, ***S***.

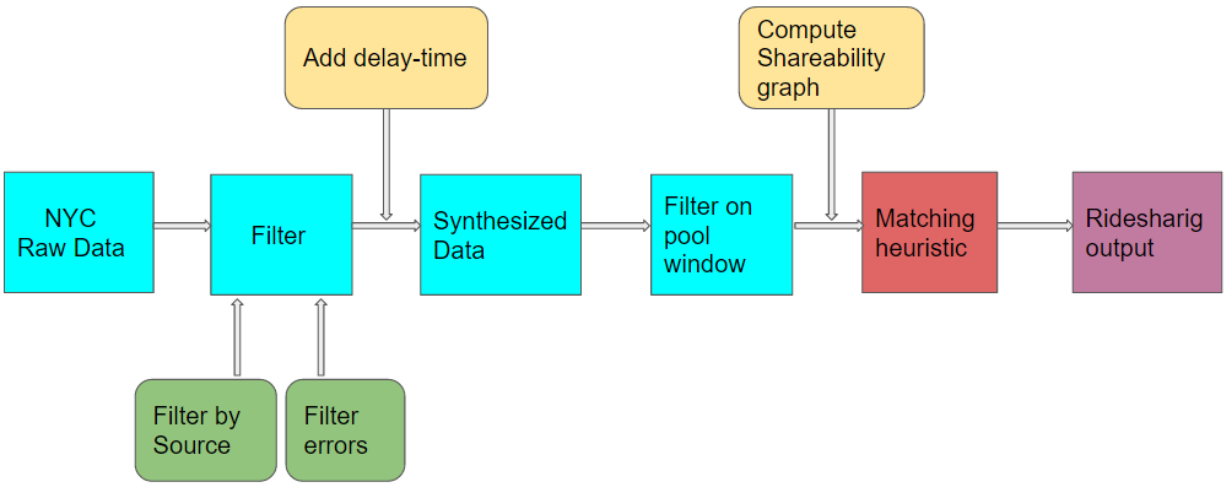
○ While ***S*** is not empty,
- Greedily pick the T_{min}, set of shareable trips, with the greatest distance saved.
- Remove all other shared trips in ***S*** that contain a node/trip that was just greedily picked.

○ With the remaining trips that have not been combined thus far, find all combinations of paired trips.

○ For each paired trip
- If passenger count constraint is met, query the Graphhopper API for the two possible paths (i.e. for AB, query JFK to A to B and JFK to B to A).
- Store T_{min}, the path and distance of the path with the minimum distance.
- If the distance constraint and delay time constraint are met, then add T_{min} to shareability network, ***S***: add an edge between those two nodes/trips and add the edge weight of distance saved.

○ Perform Maximum Weighted Matching on ***S*** for pairs to obtain all the combined trips.

4. SYSTEM DESIGN



Steps:

- The NYC Raw Data for the month of May was read from a csv and stored as a Data Frame in pandas.
- The raw data contained several errors in it. Errors varied from zero distance errors, time mismatch errors, etc. These erroneous data rows were removed from the database.

- The data was further filtered based on the source by considering a polygon area around JFK region. Only trips originating at John F Kennedy Airport were used for the ride-sharing algorithm.
- Additional columns were added to provide delay time values.
- The synthesized data was further filtered based on the pool window size.
- The algorithm was run on each pool window previously obtained.
- The algorithm provided the list of all merged trips as the output.

5. TECHNICAL COMPONENTS

The technical components that were used for building the complete ridesharing project are as follows:

5.1. Jupyter Notebook with Python 3

Python is an interpreted, object- oriented programming language. The entire algorithm was implemented in Python 3.

5.2. Plotly

Plotly is python's graphing library used for plotting graphs with options of visualising, manipulating and solving many problems related to graphs. It was used for visualising the results of ridesharing as a graph.

5.3. GraphHopper

Graphhopper is a routing engine built over the open street map. The service is capable of routing between any two or more locations on the map provided the mode of transportation. We used Graphhopper for finding the best route between n(maximum 4) points.

5.4. Pandas Library in Python

Pandas is an open source software library in python for data manipulation and analysis. It provides high performance, easy to use data structures and data analysis tools for python programming. It provides features such as a Data Frame object for data manipulation with integrated indexing, data set merging and joining, data structure column insertion and deletion, reshaping and pivoting data sets, tools for reading and writing data between in-memory data structures and different file formats. We have use pandas to hold the trip data and the associated attributes, to synthesize attributes such as delay time and filter data based on selected attributes.

6. IMPLEMENTATION DETAILS AND INSTRUCTIONS

6.1 Dataset

Due to the nature of our algorithm, computation times were very high limiting our ability to process are a large dataset. We processed only trip_data_5.zip from 2013 and stored them in a Pandas DataFrame.

Date: May 1st, 2013
Time Interval: 12:00pm - 11:59pm
Total Time Interval: 12 Hours

Each trip in the algorithm is represented by a row with the following attributes,

- Trip ID
- Source (Location)
- Destination (Location)
- Pickup Time
- Drop-off Time
- Distance
- Trip Time (from source to destination)
- Delay Time

As described in the project proposal, our concentration in this project is to implement Ride Sharing from John F. Kennedy International Airport, New York. We started off by filtering all records that started from John F. Kennedy by finding all the trip requests that originated within a polygon that covered the entire airport (see source code). As part of our algorithm, we made the assumption or restriction of having only one source at JFK airport, so we changed all pickup locations to one predefined set of latitude and longitude.

The data was cleaned and filtered of errors such as latitudes and longitudes that were all zero or invalid. We removed data that had travel times of zero or travel distances of zero.

In contrast to the paper we used as a guide, we did not use a static delay time such as 3 minutes. In our approach, delay time is a synthesized attribute that was used to reflect the total time a passenger is willing to spend while ridesharing that we impose on the rider with a dynamic factor. We experimented with 3 different factors of delay time DF = {1.2, 1.3, 1.5} which in other words mean an additional 20%, 30%, or 50% extra to the expected arrival time to the destination. We added this new delay time and created 3 separate data files for quicker processing.

6.2 Instructions

Please see ReadMe file on GitHub.

7. EXPERIMENTATIONAL DETAILS

All the experiments that we conducted were based on a combination of a few variables:

1) Maximum Shareable Trips (k = {2, 3})
2) Pool Window (PW = {2, 3, 5} minutes)
3) Delay Factor (DF = {1.2, 1.3, 1.5})

We ran results for all combinations of the above giving us 18 sets of data to work with. After getting data on those 18 different experiments, we created plots to see how the variables impacted the following:

7.1 Distance

One of the biggest saving by the implementation sharing is in the amount of distance travelled, which is measured as the sum of distance travelled by all the cabs. Experiments were conducted to find the amount of distance saved as a result of ride sharing. Walking parameter also contributes directly to the distance metric as the distance between the drop point and the actual destination is considered to be saved.

7.2 Computation Time

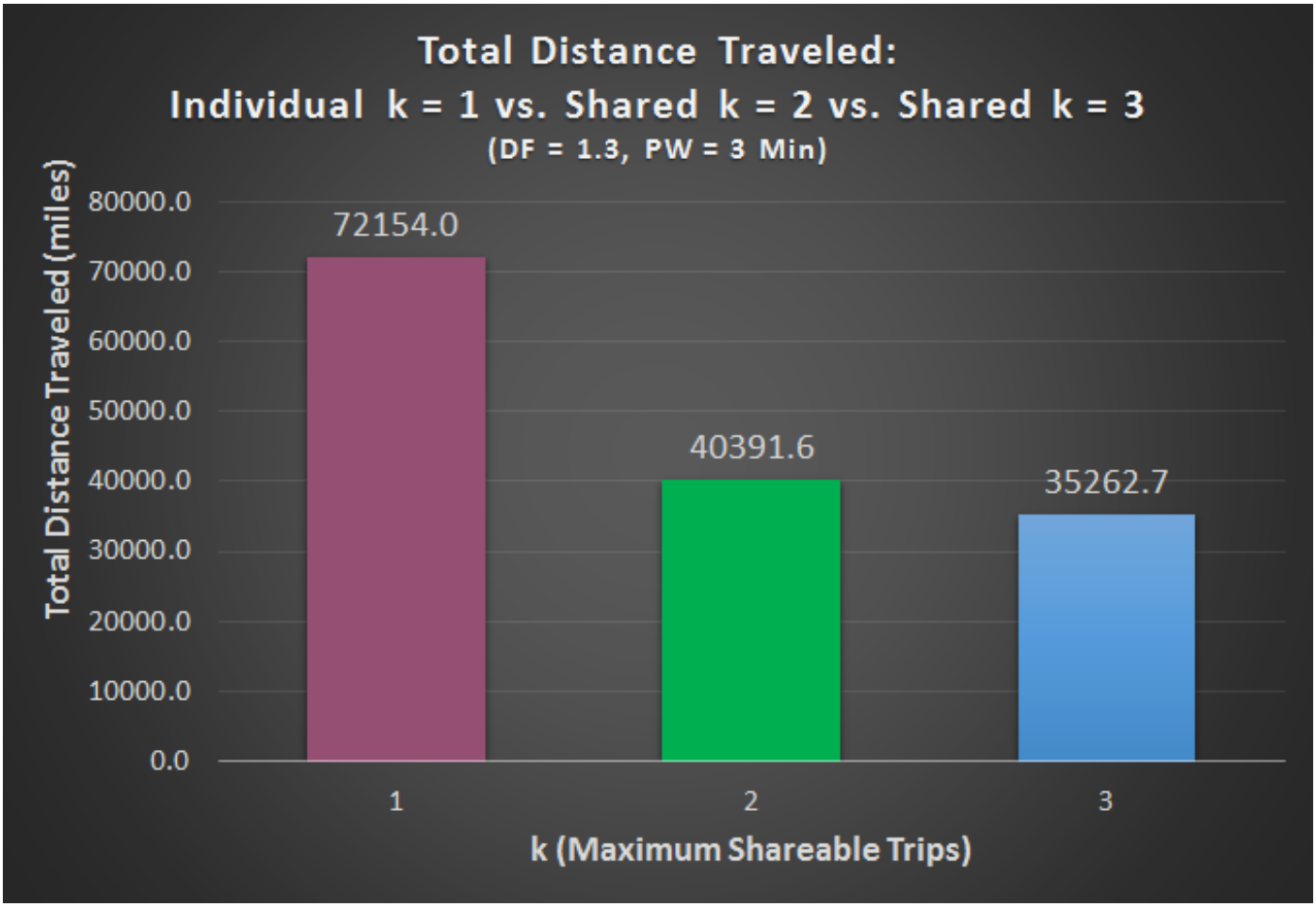
Alongside distance, computation time was the other significant result we looked into. Algorithms can save incredible amounts of distance, but practically, in the real world, if it can't compute the results within a require window of time it is futile. For example, when originally testing our algorithm on 10 minute pool windows, computation time always exceeded the 10 minute window time and probably would have taken hours or days to process which defeats the purpose of the ridesharing according to constraints.

7.3 Trip Count

The basic concept of trip sharing is to reduce the total number of individual trips that had to be taken. Experiments were conducted in this front to find the number of trips that was merged for any given time window. The effect of other parameters on the final trip count which includes both the merged trips and the lone trips.

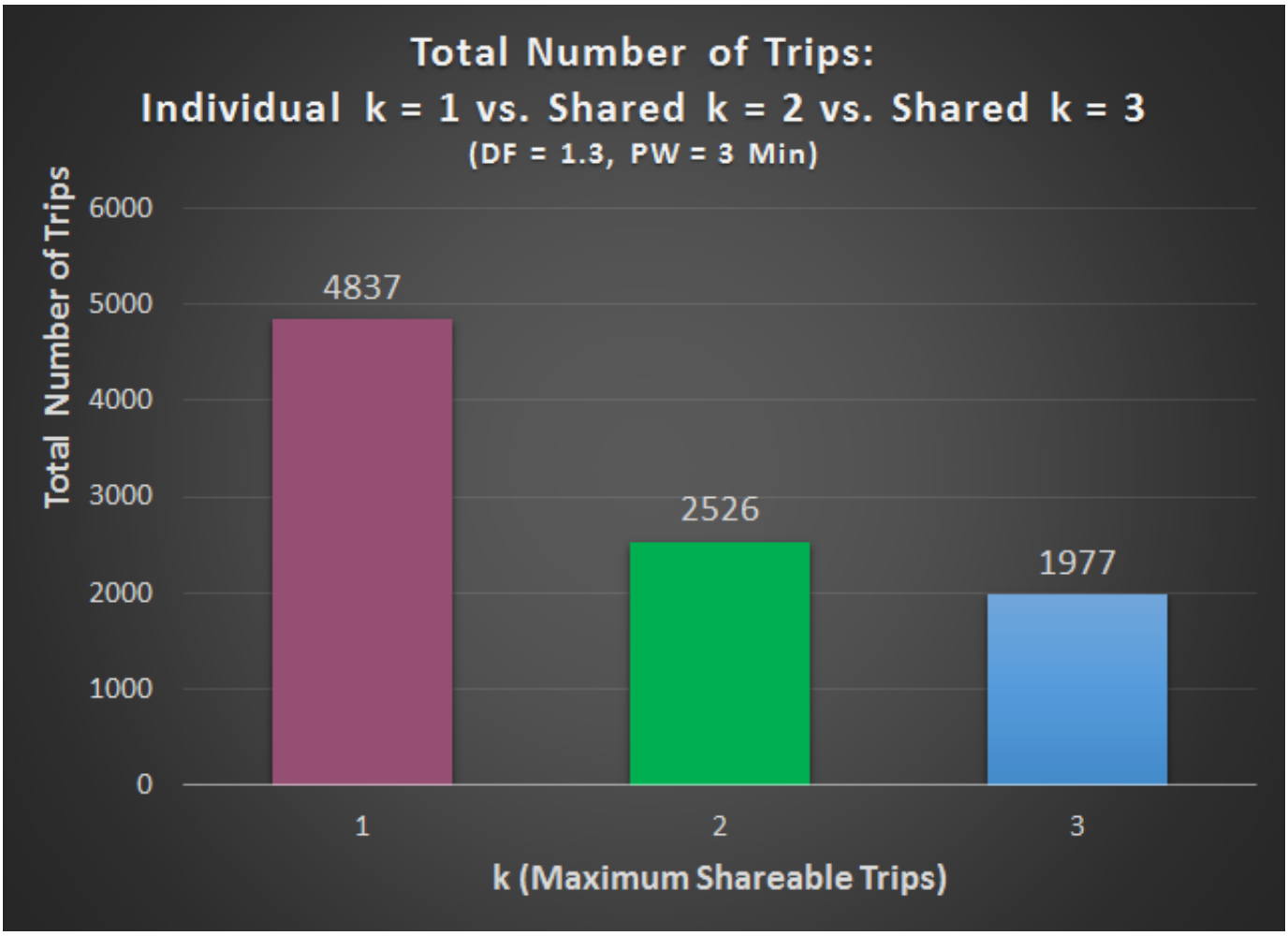
8. RESULTS

GRAPH 1:



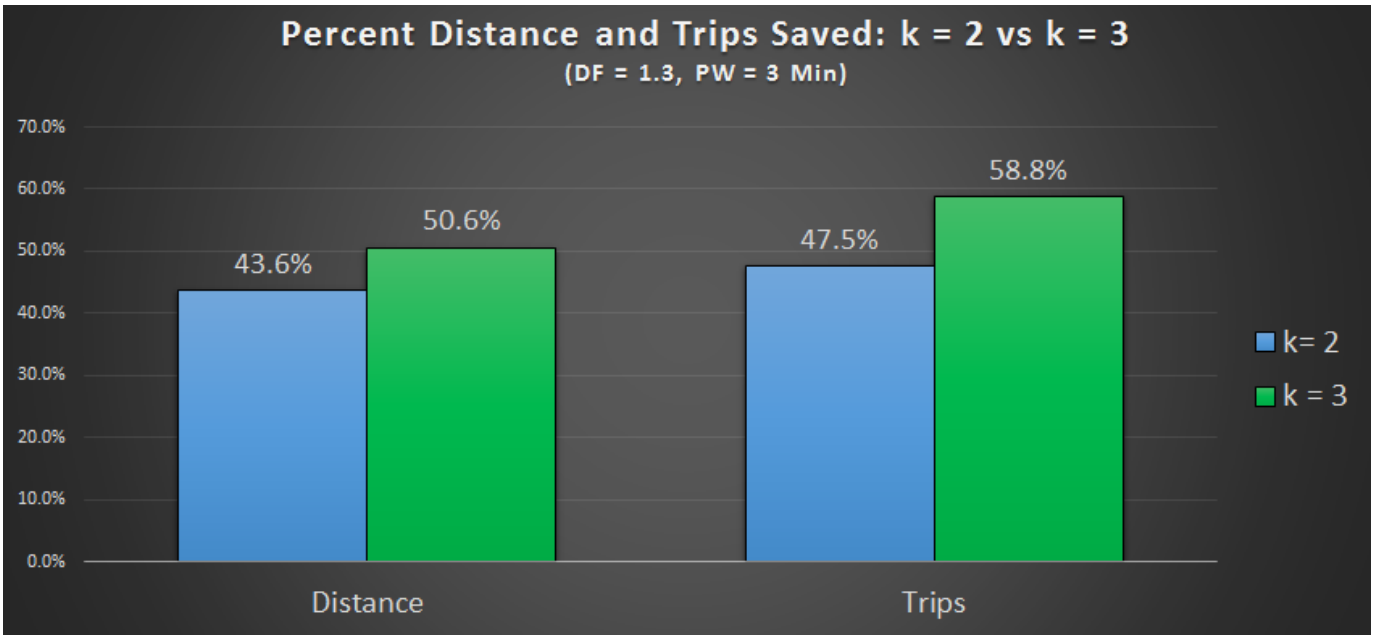
This graph shows how helpful ridesharing is in terms of reducing the distance travelled. This was one of the first requirements for the project which have been met with implementing ride-sharing for up to 3 trips.

GRAPH 2:



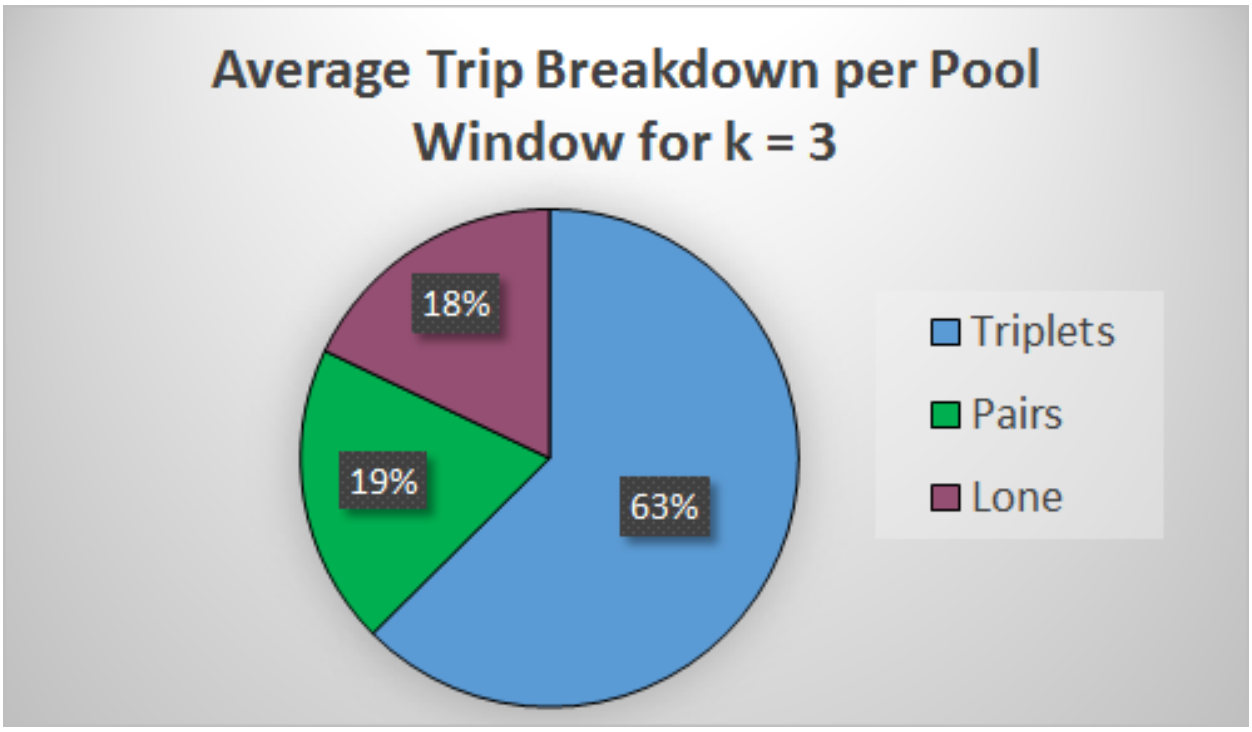
Intuitively, it goes to show the reduction in the number of trips the taxis have to make to drop passengers. Reduced number of trips helps reduce traffic on the roads and pollution, which was another one of the main requirements for the project.

GRAPH 3:



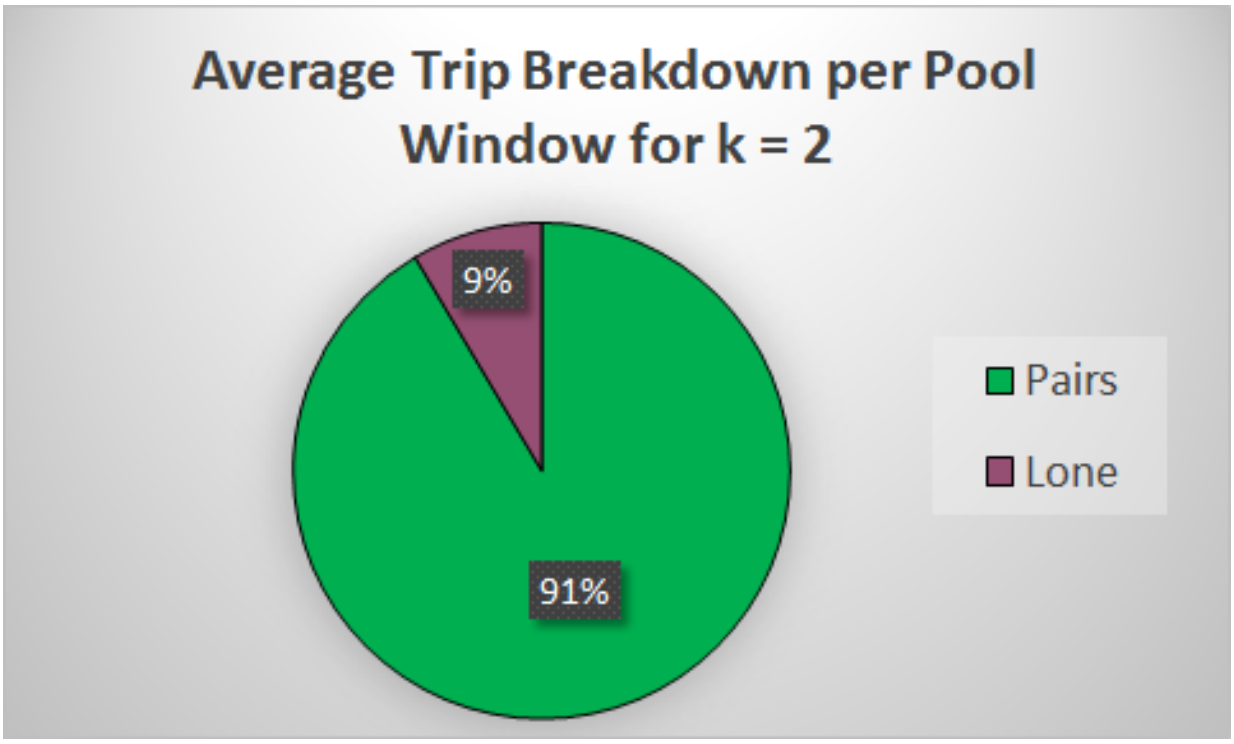
The paper we used to guide us presented an algorithm for both ***k = 2*** and ***k = 3*** maximum shareable trips. Intuitively, we should be able to save more distance traveled and trips when ***k = 3***, but the paper stated that the algorithm for ***k = 2*** is optimal, however the algorithm for ***k = 3*** was a greedy heuristic used with the optimal algorithm and therefore is just an approximation algorithm. We understand that we have only tested 12 hours worth of data on a given day, but our results show that the ***k = 3*** algorithm was able to perform, on average, better than ***k = 2*** optimal algorithm. Markedly though, the difference in percent distance saved and percent trips saved is not a vastly large difference.

GRAPH 4:



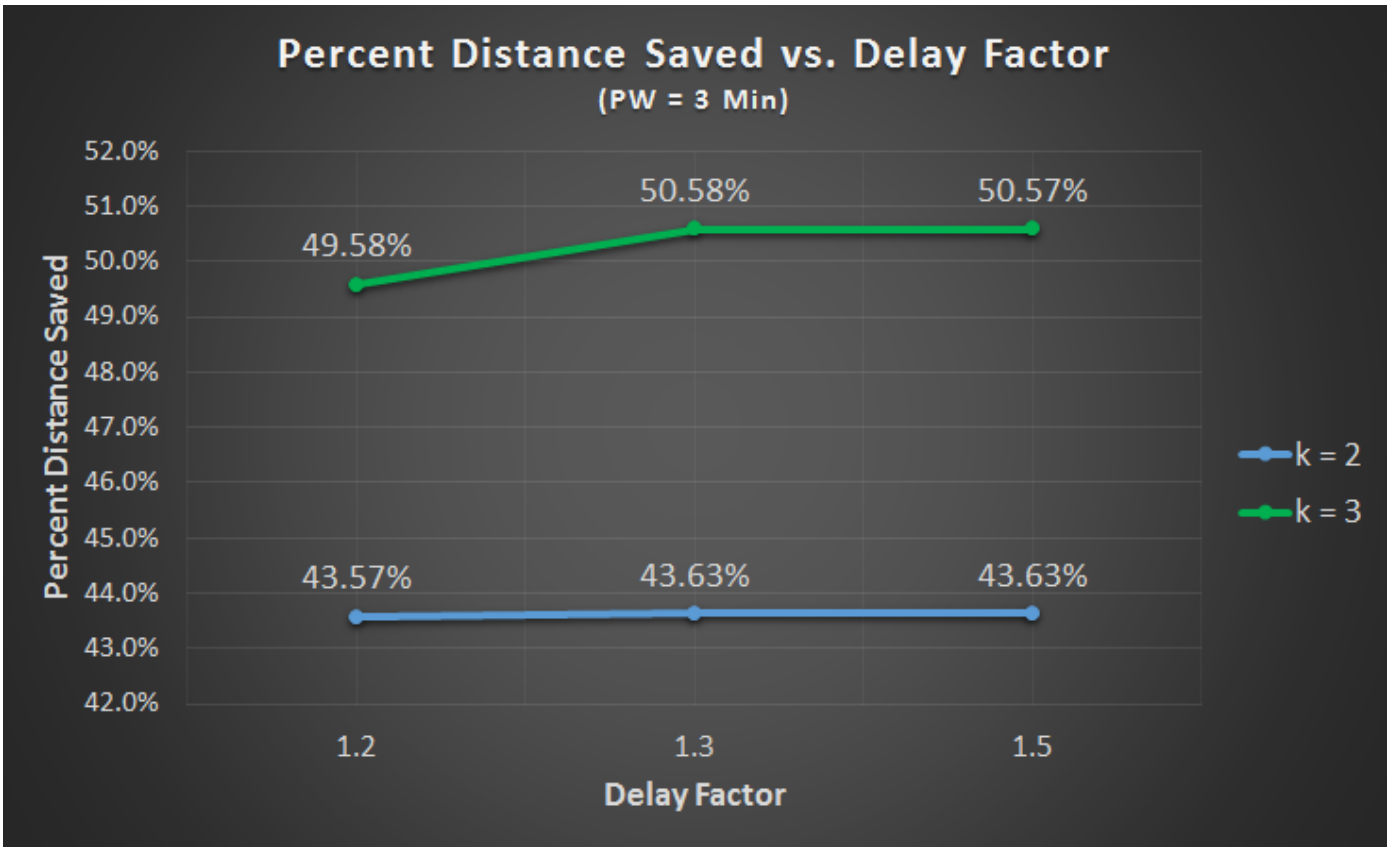
This graph is just a visual indication that the algorithm for ***k = 3*** maximum shareable trips is clearly a greedy heuristic in which as many shareable triplets are taken before pairs are even considered.

GRAPH 5:



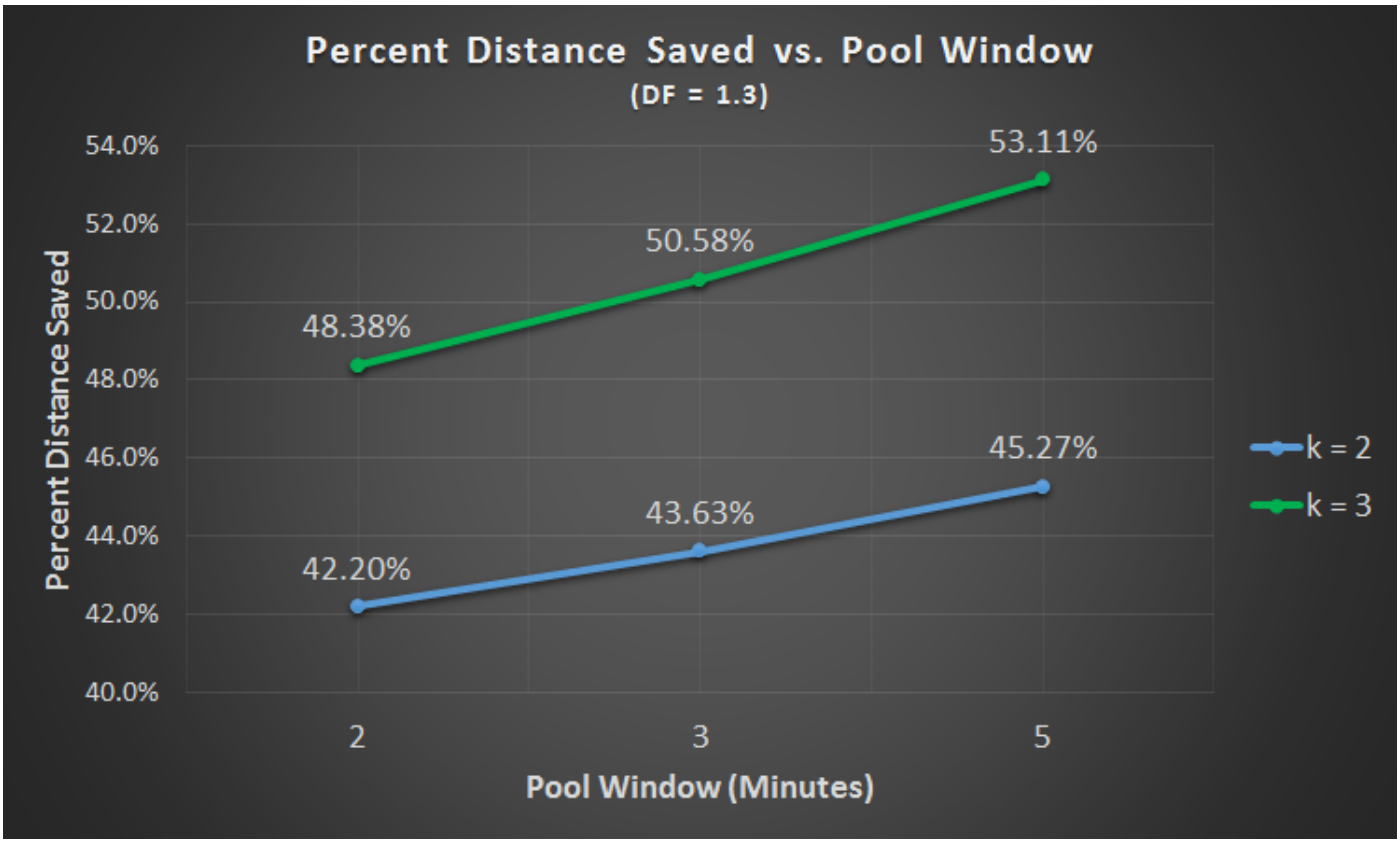
The purpose of this pie chart is just to show, in comparison, of the potential of rides that can be shared when using the ***k = 2*** optimal algorithm versus the ***k = 3*** greedy heuristic which is “close” to optimal. Similarly, the algorithm is able to combine more than 90% of the trips in each pool window as pairs and only 9% had to be travelled as lone trips.

GRAPH 6:



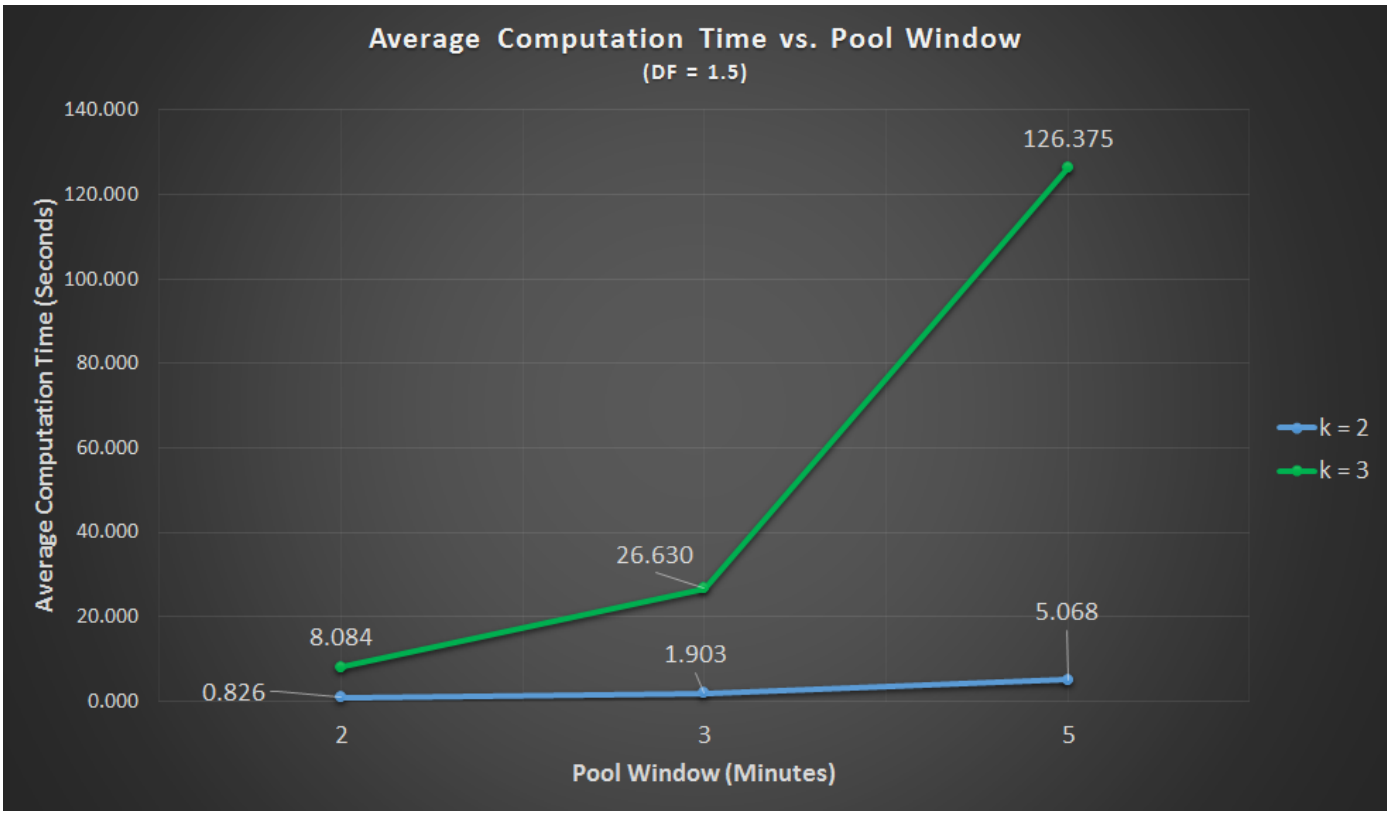
This graph shows how the delay factor affects the distance saved. For k = 2, delay factor doesn't seem to affect the distance saved for taxis but for k = 3, that is, merging 3 trips, the distance saved does go up marginally. This indicates that ride-sharing is more successful when people are willing to spend about 50% more time than they would without ridesharing.

GRAPH 7:



When working with a delay factor of 1.3, this graph shows that more distance can be saved when the pool window time is increased. This seems intuitive as well because in a smaller pool window, trips with nearby or on-the-way destinations are fewer because of fewer total trips. Secondly, pool window affects k=2 and k=3 in a similar pattern.

GRAPH 8:

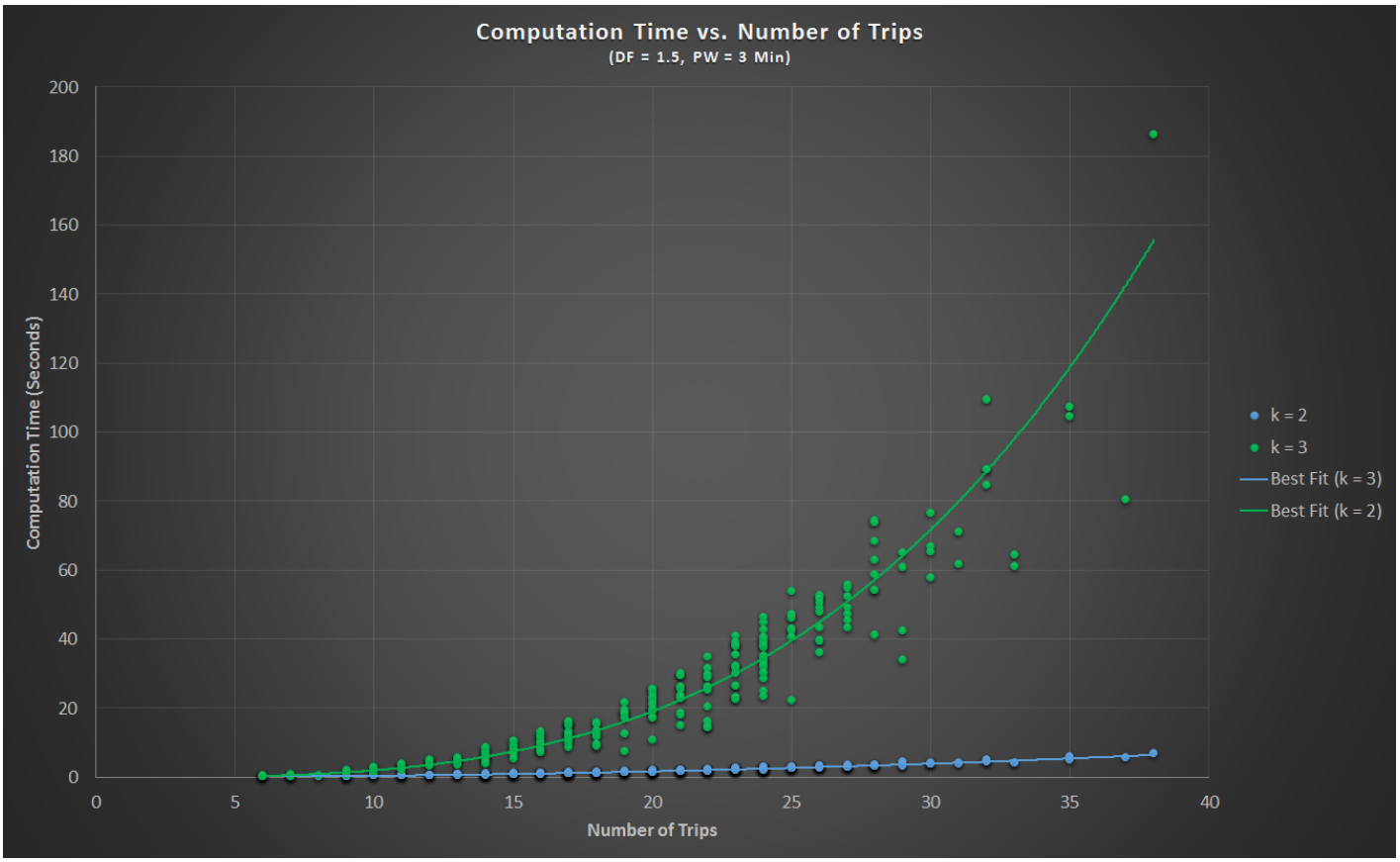


As most of the previous graphs were intuitive, so is this graph, but still is a very significant graph. Clearly the ***k = 3*** greedy approximation algorithm is a "good" algorithm based on our limited results, but one of the setbacks of working with 3 maximum shareable trips is that as the the pool window increases which implies a larger set of trip requests to check, the computation time grows exponentially. For this very reason, we were not able to process our original pool window of 10 minutes each. However, in stark contrast, the computation time for ***k = 2*** grows drastically slower than ***k = 3***. Practically, although ***k = 3*** may produce better results by a small margin, it is not a feasible solution for handling larger numbers of trips due to larger pool windows. However, it may be valuable in controlled environments with smaller pool windows where trip numbers are predictable and reasonable.

One important conclusion that we can make from this graph is about the rate at which computation time grows as the pool window increases. Notice that from PW = 2 to PW = 3, there's a small increasing gap between the graphs for k = 2 and k = 3, but then from PW = 3 to PW = 5, there's an immense gap between the two graphs. The reason is because as the pool window increases, the number of trips to process grows. The reason it grows at a faster rate (either best fit by an exponential or power function) is because of all the processing that is required to handle ALL combinations of triplets and then all permutations of valid triplets. With a trip count of 10 within a pool, we are dealing with the combination of 10 choose 3 just to find potential matchings. Then for each of those combinations, we have to find 6 permutations for each of which we make Graphhopper API calls.

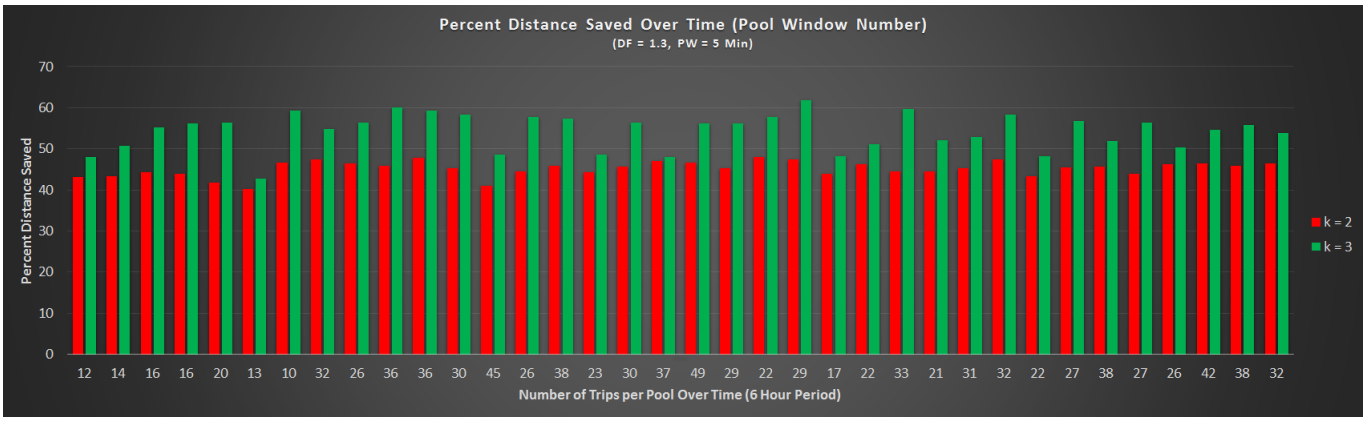
We do not have a graph for computation time versus delay factor because its impact on computation time is nominal and does not warrant discussion or any interesting insight.

GRAPH 9:

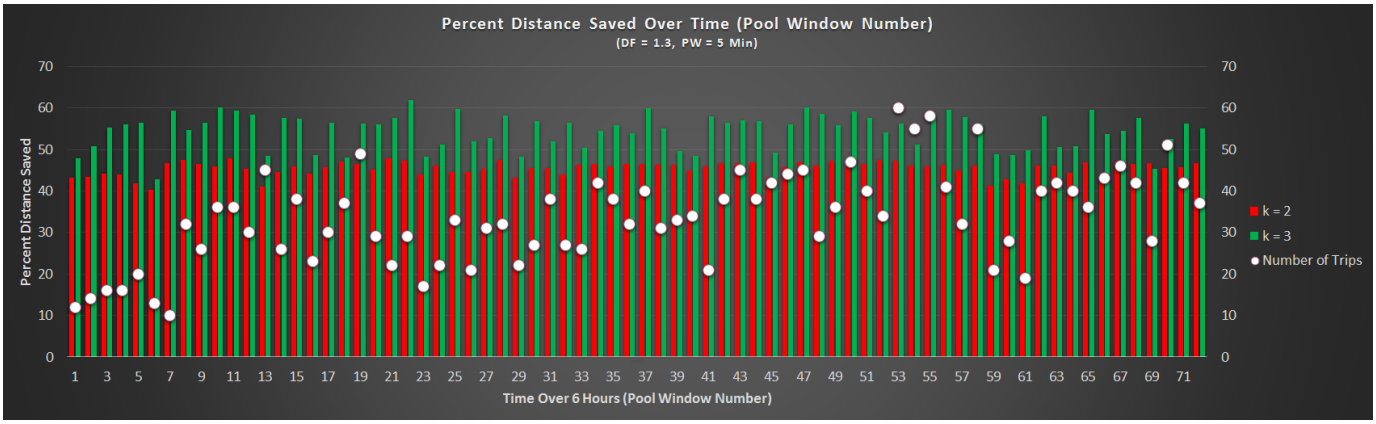


This graph shows how computation time is affected by the increase in the number of trips. When maximum shareable trips is 2 (k=2), computation time increases marginally, however when merging as many as three trips (k=3), we see an almost exponential growth in computation time with increasing number of trips. This is caused because for each triplet, 6 different distance permutations are calculated. As the number of trips increases, the combinations increase a swell and push the computation time higher.

GRAPH 10a:

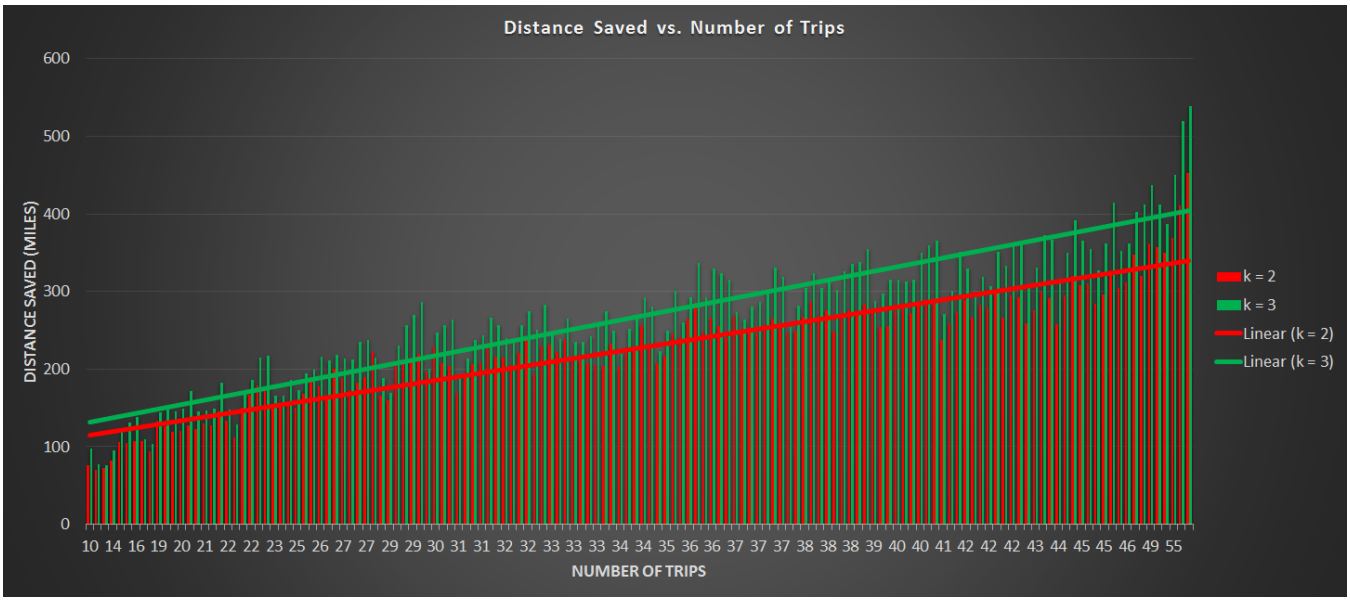


GRAPH 10b:

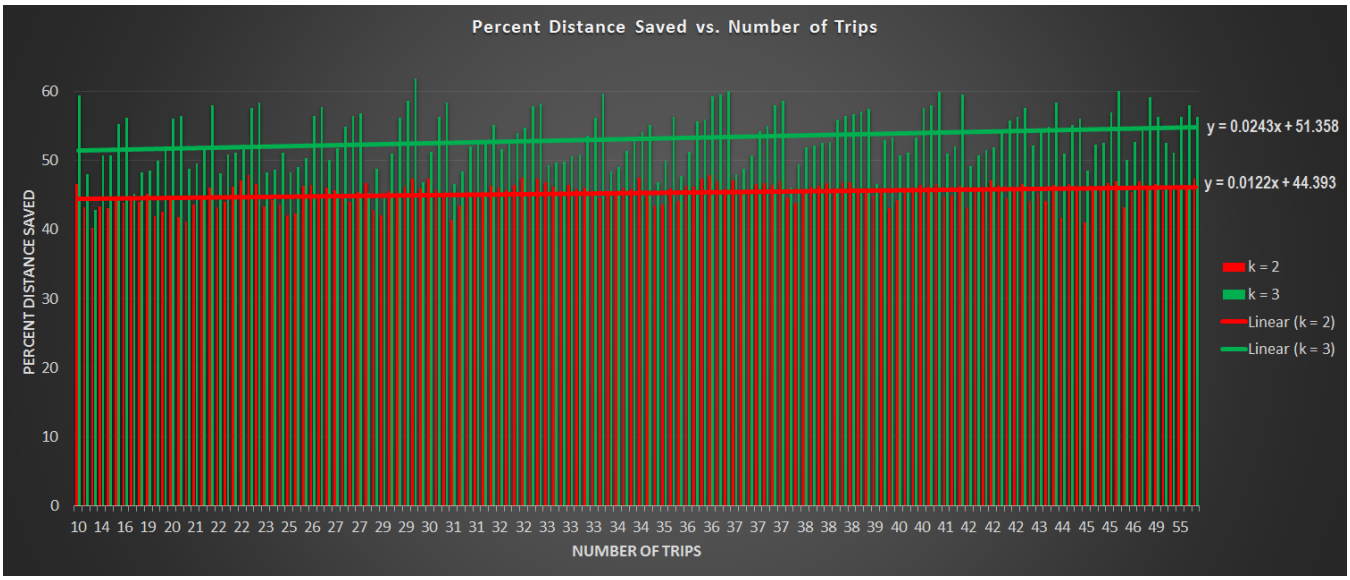


Graphs 10a and 10b above are substantially the same in that they both show that over a given 6 hour period, the majority of the time, more distance can be saved when setting the maximum shareable trips at 3 as opposed to 2. Graph 10b shows the number of trips for each of those given pools plotted. Intuitively, one would think that as the number of trips increase the percent of distance saved should also increase, however this is not the case. In actuality, as the number of trips increases, the amount of distance saved increases, not the percentage. The percentage has a tendency to float within a range of values. See the two graphs below:

GRAPH 11a:



GRAPH 11b:



As discussed under graph 10b, we suspected that as the number of trips increases the percent distance saved does not increase even though actual distance saved increases. Graph 11a clearly shows that as the number of trips increase, the amount of distance saved increases. On the other hand, graph 11b shows that percent distance saved does increase, however nominally and for k=2 and k=3 it is at a rate less than 0.03% per added trip. For the domain of 10 to 60 trips, our percent distance savings hovers around 44% for k = 2 maximum shareable trips and at about 51% savings for k = 3 maximum shareable trips.

9. CONCLUSION

In conclusion, we were able to achieve our initial goal of minimizing the distance traveled through ridesharing while maintaining passenger delay constraints. Our results have shown us that the vast majority of the time that setting the maximum shareable trips to ***k = 3*** produces better savings in distance and trips with respect to all experiments we performed on all variables (pool window and delay factor) which help contribute to a bigger long term solution of reducing congestion in New York and reducing fuel consumption, therefore reducing pollution. We were able to save more distance by increasing the delay factor, but practically that presents the issue of unhappy passengers who have to wait longer than they would like to. We were able to save more distance by increasing the pool window size from 2 to 3 and then to 5 minutes, but not to 10 minutes. This also presents an issue because practically our computation time has to be smaller than the pool window size. It is of no value to process a set of trips that takes longer than the time we have allotted to grab trip requests. Granted, the ***k = 2*** algorithm is optimal and is more powerful in that it can handle larger pool windows and has extremely fast computation times, but its setback is that it won't save as much distance or reduce as much congestion by reducing the number of trips. Therefore, ***k = 3*** is a plausible, feasible solution for ridesharing when the number of trip requests is low. We can control this to some extent by having practical but smaller pool windows like every 3 minutes to 5 minutes. We can even adjust the window sizes to be smaller during high traffic times like evenings and weekends when the airport is busiest, just as we would have trains more frequently during rush hour. With that said, we can experiment with a variable pool window size and a variable delay factor, even a variable ***k*** which changes between 2 and 3 for maximum shareable trips, but it would still be necessary to test this algorithm on a larger set of data to see the results of the bigger picture.

10. REFERENCES

Paolo Santi, et. al. (2014, September 16). Quantifying the Benefits of Vehicle Pooling with Shareability Networks. *Proceedings of the National Academy of Sciences, 111*(37), 13290-13294. Retrieved from http://www.pnas.org/content/111/37/13290