CS 581 - Database Management Systems

University of Illinois at Chicago

(Spring 2018)

RIDESHARING

PROJECT REPORT

Team - 01

Arjan Singh Mundy (mundy3@uic.edu)

Mark Hallenbeck (mhalle5@uic.edu)

Pratyush Bagaria (pbagar2@uic.edu)

Sachin Mathew (smathe37@uic.edu)

Shreyash Bali (sbali3@uic.edu)

Contents

1. Introduction & Project Proposal

Providing on-demand transportation service to the people is one of the successful concepts and is expanding its reach every day. With many companies operating in the public domain and providing almost same experience to their rider's it hardly differentiates them. Among the various type of services, these companies provide one such is where they give an option to its users/riders to share their ride with other users/riders with whom they have never met. There are many advantages of having ridesharing service in place including monetary savings to the customers, increased revenue to the company and being eco-friendly with more number of customers opting for this service than to take a personal vehicle. Somewhere in between these advantages and ease of commutation the riders have rarely questioned about the quality of the rides that they have taken or they are about to take and neither did these companies have made a system available to the users which consider the preferences of their riders.

Through this project, we are trying to address that lacking concept of giving importance to rider's social preferences by proposing a model which allows riders to specify their expectation and characteristics about fellow riders with whom he/she shares a ride. We believe this will enhance the user experience and will encourage them to choose ride sharing more often. On implementing the proposed model, we have also analysed and compared it with a similar model which does not consider the rider's social preferences to draw an analogy and see whether it is feasible to have such a model wherein social preference of riders can be considered.

1.1. Objective

Our objective is to maximize the distance saved using rideshare while keeping the riders social preference vis-à-vis his/her idea of a quality ride share in consideration.

1.2. Assumptions List

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

- Pooling model to process rideshare trip requests; experiments conducted with Pool Window of 1 min & 2 mins with JFK Airport as the single source point.
- The number of trips which we can be merged is capped to 4; this aligns with the fact that most of the vehicles can accommodate 4 passengers at max.
- The maximum slack time (de-tour time) which any passenger can bear is not more than 20% of the total estimated travel time.
- Since the system is based on pooling model, we leave the user with an option to decide the number of subsequent pools in which he/she wishes to get considered into to find a rideshare partner.
- Additionally, we assume that an infinite number of taxis are available at the source to service merged and unmerged requests.

2. Implementation Details

2.1. Dataset Considered

The dataset was provided by the NYC Taxi and Limousine Commission. To be precise, the project was implemented using the May 2015 Yellow taxi trip data. 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. However, the fields pick-up location, itemized fares, rate types, and payments were not considered for the implementation of the project.

2.2. Technical Components

The entire project was developed using the Python 3.4 on Jupyter Notebook. The data was stored in MySQL Workbench Community for Windows version 6.3.10.

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 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. System Formulation

Before we move directly to algorithms we would like to explain interested readers a bit about our approach on how we are fairly considering de-tour time for each passenger, leaving an option with the riders to choose the maximum number of pools in which their requests should be considered and their social preferences which go together with what attributes for an ideal cab ride, in below sub-sections -

3.1. Passenger's de-tour time & Pool Forwarding

When passengers are willing to ride share they may experience some delay compared to individual trip rides. We are assuming that each passenger is willing to be delayed at most 20% distance of their individual trip. Suppose it takes 10 miles for a passenger from source to destination, then he will be willing to travel another, at most, 2 miles i.e. a total of 12 miles.

3.2. Social Preferences

Every rider while creating a profile in the system will be asked to select their preference to each of the questions on a range of 0 to 10, some of these questions could be:

a) how likely you want your co-rider to talk with you or vice-versa?
- 0 indicates that the rider doesn't want a conversation to happen at all
- 10 indicates that rider is really looking forward to having a conversation

b) how comfortable you are with music in the cab?
- 0 indicates that the rider doesn't want music inside the cab at all
- 10 indicates that rider is comfortable with loud music as well

c) how do you feel when your co-rider eats food sitting next to you?
- 0 indicates that the rider dislikes when co-rider sits next & eats his/her meal
- 10 indicates that he/she is comfortable if his/her co-rider is having a meal

d) are you comfortable in sharing a ride if your co-rider smokes in cab while on trip?
- 0 indicates that the person is highly allergic to smoke smell
- 10 indicates that he/she is totally comfortable if his/her co-rider smokes

Now to check if two riders are socially compatible, we calculate the dissimilarity score between them using the formula:

$$dissimilarity\ score = \frac{\sum_{i=0}^{n} |X_{i1} - X_{i2}|}{n * 10}$$

where 'i' is one of those questions which can be asked while there are 'n' such questions. In our system, we have three such questions for which randomly values are assigned.

To illustrate this, we have following two cases -

Case #1 where two riders are socially compatible

Scenario 1	how likely you want to talk with your co-rider?	how comfortable are you with music in the cab?	how do you feel when your co-rider eats food sitting next to you?
R1	3	5	7
R2	4	6	8
|diff|	1	1	1
dissimilarity score sum of |diff| / 30	***0.1 (=3/30)***		

Case #2 where two riders are socially incompatible

Scenario 2	how likely you want your co-rider to talk with you?	how comfortable are you with music in the cab?	how do you feel when your co-rider eats food sitting next to you?
R1	10	0	10
R2	0	10	0
|diff|	10	10	10
dissimilarity score sum of |diff| / 30	***1 (=30/30)***		

Once dissimilarity score is calculated, we filter out edges which were initially made in the share-ability graph without considering social scoring which has high dissimilarity score, decided by the distance they are together. The distance was considered here because differences between people will play a major role if they are traveling together for more time compared to shorter distance. If distance they are together is more than 25 miles their dissimilarity score must be less than 0.3. If distance they are together is more than 15 miles their dissimilarity score must be less than 0.5. If distance they are together is less than 15 miles their dissimilarity score must be less than 0.8

3.3. Algorithm Description

Step 1: Specify the start date, end date and the pool window size to run the algorithm.
Step 2: Update the distance and time for all requests in the database using GraphHopper API
Step 3: Fix source as JFK (or any other desired source, coordinates to be set)
Step 4: PoolData <= Get all the requests received in a pool window using the get_data() function
Step 5: For all requests in PoolData:

a. Get all combinations of requests.
b. Get trip details from the database for each request
c. T_distance, T_time <= calculate the distance and time between source and drop-off locations for each request
d. Dist_between <= calculate the distance between their destinations using graphhopper API
e. Delay <= calculate the allowed delayed time for both the trips (1.2 * T_time)
f. depending on the closer dropoff location of both trips, decide which trip gets dropped first
g. Shared distance <= T_distance of request getting dropped first
h. Check for social score compatibility of both the trips
i. Check if total time of shared travel is less than the delay for each trip
j. Check if any distance is saved by merging the rides,
k. Graph1 <= Add nodes and an edge between them in the graph
l. Merged1 <= Using max weight matching graph algorithm obtain the best possible pairs of rides from graph1
m. Graph2 <= Delete all the edges between the pair of nodes present in Merged1 from the graph.
n. Merged2 <= Using max weight matching graph algorithm obtain the best possible pair of rides from graph2.
o. Check if either of the rides from the pairs in Merged2 is present in Merged1. If both rides are present, then add them to the pair they are connected to in Merged1 and check for delay constraints. Else, only one of them is connected to a pair in Merged1 add it to the pair in Merged1. Delete the respective pairs from Merged2.
p. For all the merged trips in Merged1, calculate the distance saved and the cost saved by each merge.
q. For all the single trips that did not get merged in the present pool, check if they have opted for suspension to the next pool. If they have opted to get suspended to the next pool, add them to the next pool window until the number becomes zero.
r. For all other single rides, output them as non-shareable rides

Pseudo Code to check Social Compatibility

Step 1: Get the request ids of both the trips and the shared distance between them.
Step 2: Range <= Depending on the shared distance find the distance_range that it falls in
Step 3: Threshold <= For the respective Range find the threshold value
Step 4: Get the social preferences for each of the request id's
Step 5: Difference <= find the sum of individual differences of social preferences
Step 6: Normalize the Difference to lie between 0 and 1
Step 7: If Difference < Threshold:
        Return as True: Socially Compatible
    Else:
        Return as False: Socially Incompatible

Pseudo Code to Compute Distance and Cost Savings

Set base rate: 1$ per mile do following for all the trips merged

a. Normal distance <= Calculate the sum of all the individual distances from source to each drop-off locations.
b. Normal cost <= Normal Distance
c. Shared distance <= Get the sum of distances of the first trip to be dropped from source and subsequent distances between their drop-off locations in the order of dropping them.
d. Depending on the number of trips merged a discount of 20%, 30% or 40% is given.
e. Distance saved <= Normal Distance - Shared Distance
f. Cost saved <= Normal Cost - Shared Cost
g. Total Distance saved <= Sum of all the distances saved in that pool window
h. Total cost saved <= Sum of all the costs saved in that pool window

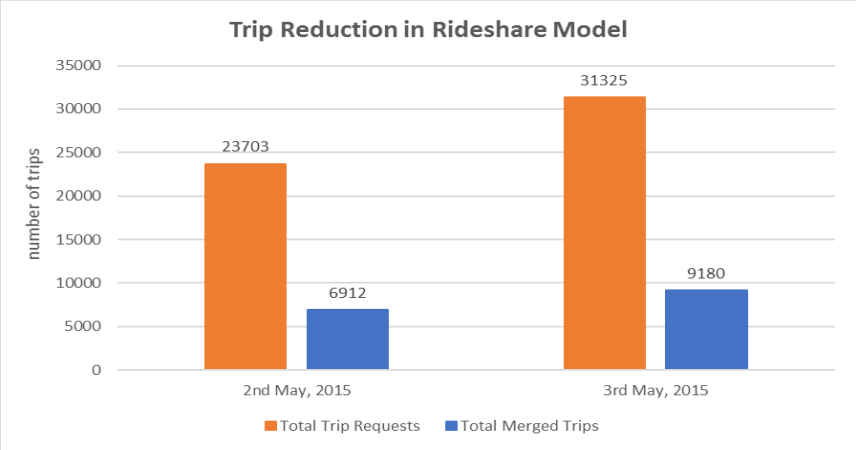
4. Execution Steps

Please follow the below GitHub link where ReadMe.md is available which comprises the instructions to install necessary software, API's and steps to execute –

https://github.com/shreyash1505/RideSharing_CS581

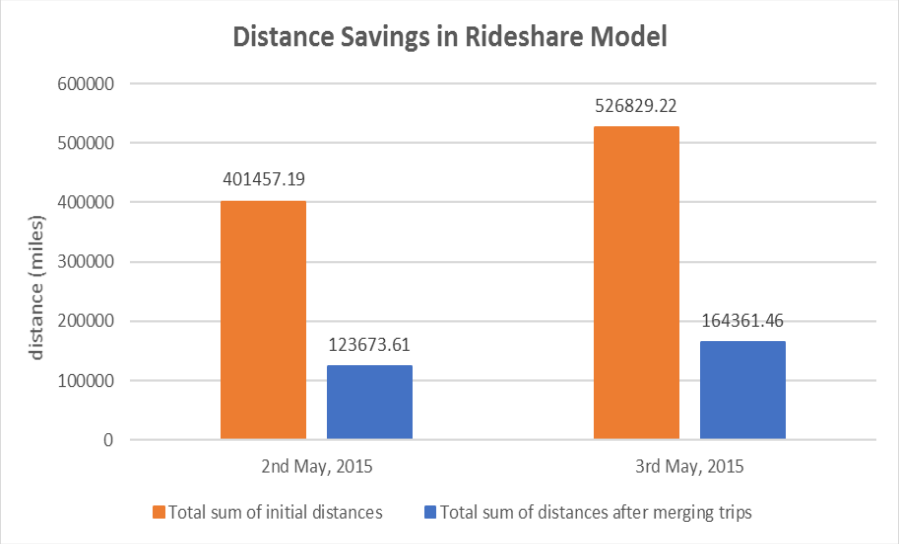
5. Graphs

5.1. Trips reduction in Rideshare Model



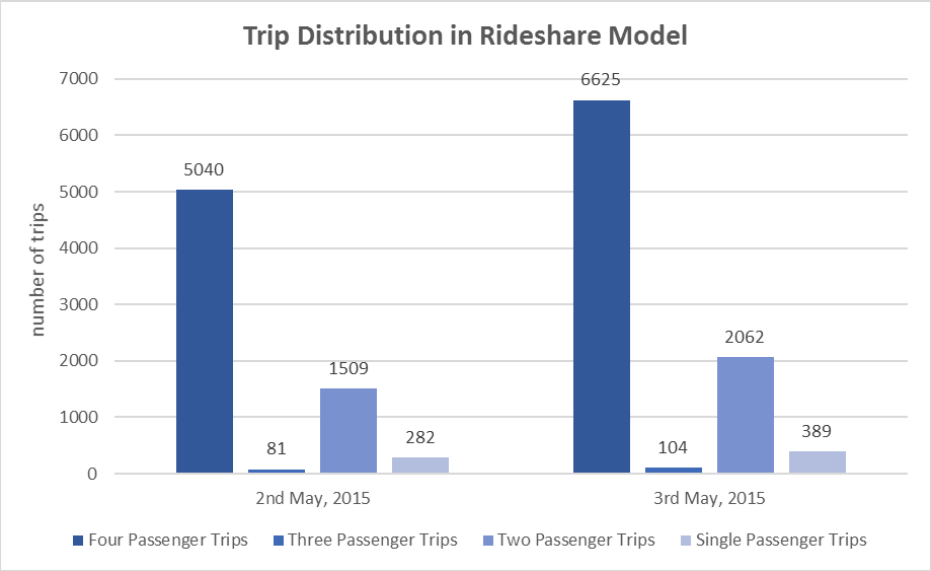
From the above graph, which is plotted based on the outputs generated by our algorithm, we can observe that the final number of trips in rideshare model will be 6912 which includes 4-Passenger trips, 3-Passenger trips, 2-Passenger trips and the single passenger trips for which an ideal rideshare match couldn't be found on 2nd May 2015 data wherein the original trip requests were 23703. Similarly, the results for 3rd May has been plotted alongside. These new numbers may also intuitively mean that those many vehicles would be required to service all the trip requests.

5.2. Distance Savings in Rideshare Model



In the second graph, we represent from the outputs obtained, the total savings in terms of miles for both the days of the dataset considered. This is in accordance with the previous graph. Basically, when we obtained 6912 trips (shown in the previous graph) after performing ridesharing then the corresponding total distance traveled would be 123673.61 miles. Similarly, the data for the second day is also plotted. One can also calculate the savings in terms of percentages as well using this data.

5.3. Trip Distribution

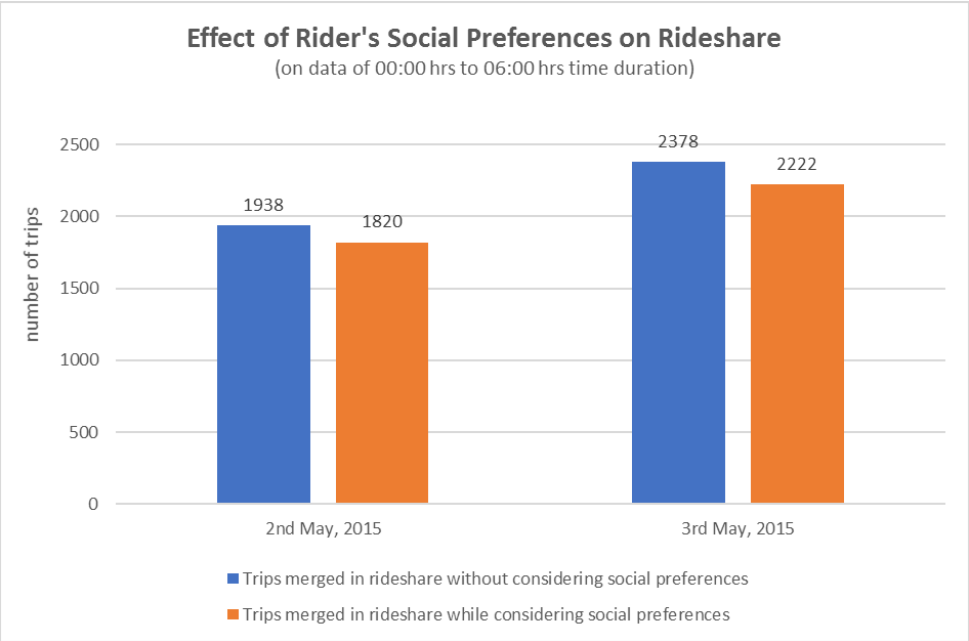


In the left half of the above graph, 6912 resultant (merged) trips for 2nd May 2015 dataset is broken down to 4 Passenger, 3 Passenger, 2 Passenger & Single Passenger (no-match found) trips respectively to 5040, 81, 1509, 282. Similarly, the right half of the graph shows corresponding bars for trips which originated on 3rd May 2015

5.4. Execution Time

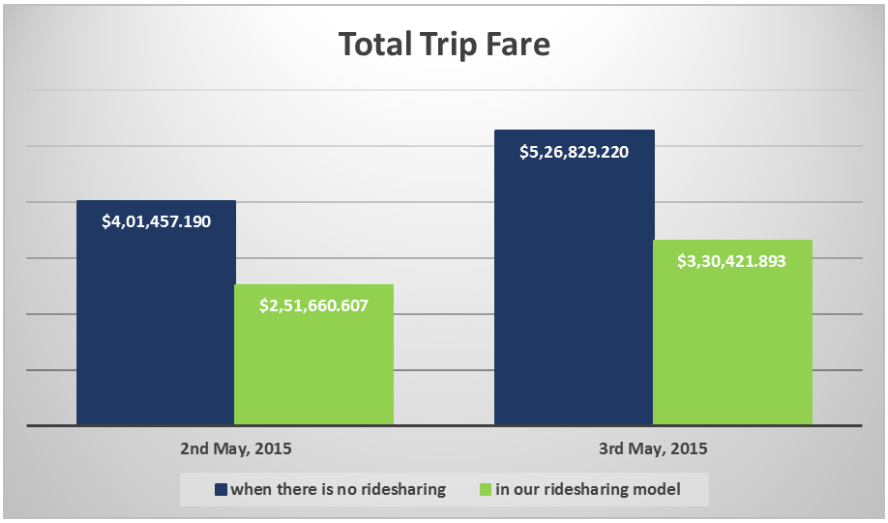
Number of requests in each pool versus Time in seconds graph above indicates the average execution time taken by algorithm combined with the time taken by GraphHopper API to compute distances between the pairs of drop-off points. Here we see an exponential increase in time with an increase in the number of requests in the pool. It was observed during execution that most of the time in the above graph was being added by GraphHopper API.

5.5. Effect of Rider's Social Preferences



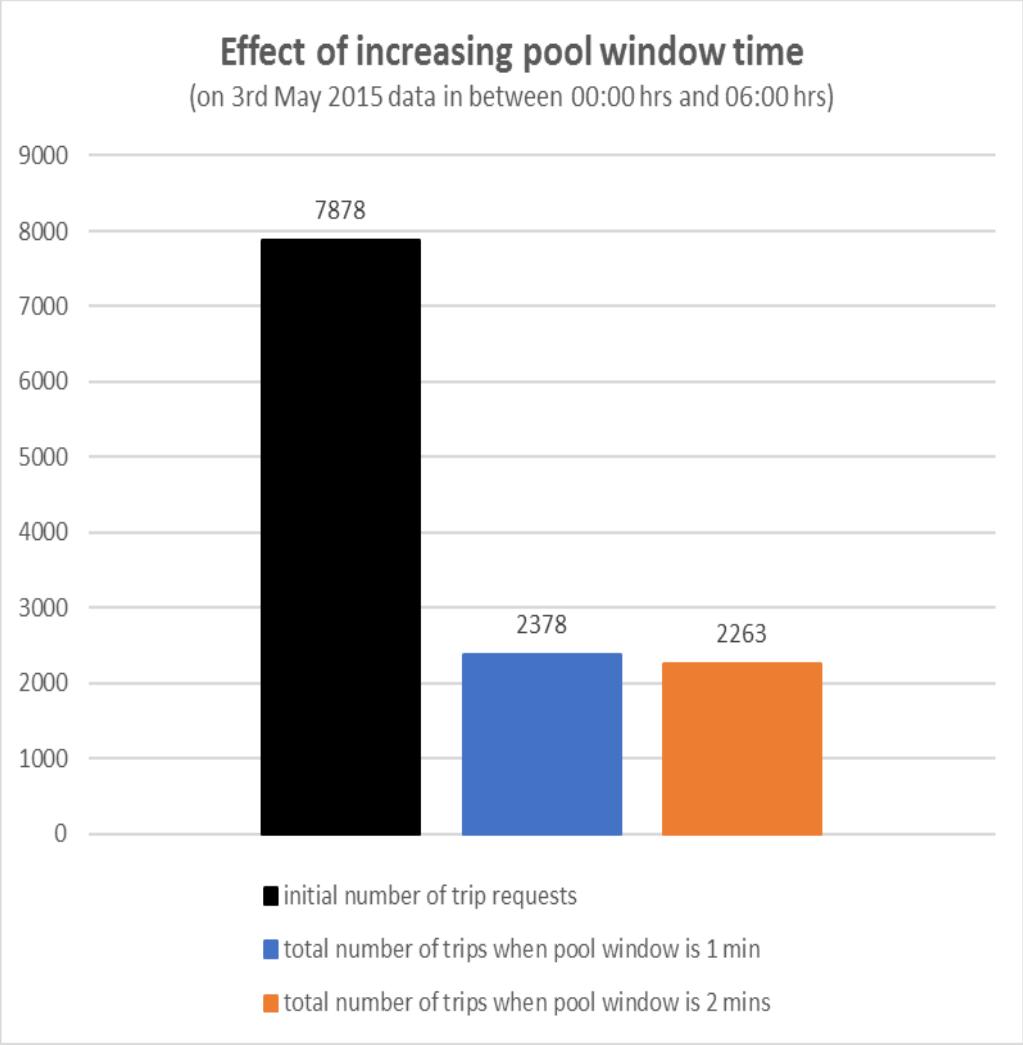
We observe from the above plot that when rider's social preferences were considered while merging their trip requests, the number of trips merged goes down by 6-6.5% when compared with the number of trips merged when their preferences were not considered and were solely merged with an aim to maximize the miles savings. We believe the percentage may variate based on how trips are spread throughout the day and depend on the social scores. This plot was obtained on a limited time window of 6 hours for both days.

5.6. Trip Fares (based on our pricing model)



Here we basically try to show the total savings that could be made by the customers after opting for ridesharing. It is estimated that around 37% savings are made in terms of dollar amount.

5.7. Effect of increasing pool window



The three vertical bars in the above draws a comparison between the initial number of trip requests initially was there in the dataset for the stated time period, the number of trips merged when pool window was kept 1 minutes and the number of trips merged when pool window was kept 2 minutes. It is intuitive that when pool window is increased to 2 minutes then there will be more number of trips in that pool and chances of finding a ride-share partner for any trip request would be more which can remain unmatched in the pool of 1-minutes. Additionally, the trips with 3-Passengers could be formed to a 4-Passenger trip and so on. Thereby the total number of trips will be less when pool window time is increased.

6. Conclusion

The following conclusions can be drawn based on the results generated by our experiments and execution over the specific dataset -

1. The number of trips in the ridesharing model reduces drastically as compared to the total trips running on a non-rideshare model. Based on the experiment results, the total number of trips using ridesharing model dropped by 71% when compared to the total number of trips without ridesharing.

2. Because of reduction in the number of trips, there were significant savings in terms of the distance traveled as well. The total trip distance was reduced by approximately 69% by the ride-sharing model.

3. The social preferences of the users only affected 6% of the total shared trips. Which means that a ride-sharing company can provide an ideal ridesharing experience to their customers without taking much of a hit on their revenue. However, it remains to be tested how the social score could affect rides for multiple pick-up locations.

4. It was observed that the forwarding of unmatched trip requests to the next pool did not help the customer much. This was because the dataset did not contain a steady record of the trip requests for every minute of the day.

5. The ridesharing model is also friendly to the pockets of its customers. The total cost incurred by the passengers in the ridesharing model reduced by 37%.

6. Most of the running time of the algorithm was consumed by calls to the GraphHopper API.

7. References

[1] “Quantifying the benefits of vehicle pooling with shareability networks” - Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H. Strogatz, and Carlo Ratti

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

[3] GraphHopper Directions API [https://graphhopper.com/api/1/docs/routing]

[4] MySQL Connector [https://dev.mysql.com/doc/connector-python/en/]

[5] NetworkX [http://networkx.github.io/documentation/networkx-1.10/overview.html]

[6] Jupyter Notebook [http://jupyter.org/index.html]

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