Ride Sharing

Team 4: Ashwin Krithekavasan, Nivetha Babu, Parinitha Ganapathisubramaniam Kannan, Raghavendra Somannavar

1 Introduction

This project implements and evaluates a ride sharing algorithm on spatio-temporal data. NYC taxi cab 2016 data has been used for evaluations.

1.1 Constraints

The following are the constraints of our implementation:

- Only trips originating from JFK and ending at Manhattan are considered.
- Taxi is always available at JFK.
- The maximum occupancy of the taxi is 4.
- Maximum number of mergeable trips is 2.
- No walking.
- Real-time traffic considered
- Pooling windows of different sizes 5 min, 10 min were considered.
- Delay times of 20% and 30% were considered

2 Algorithms Evaluated

2.1 Overview

- Selecting all rides within the pooling window - 5 mins, 10 mins
- Dividing Manhattan into grids based on Lat-Long Approximation - Each grid of size ~**0.7 miles**
- Merging rides with destinations **on the same grid in Manhattan**
- Merging rides wherein **one ride is en-route to another ride** based on grids in Manhattan and route direction from Google Maps API
- For rides **which were merged** from the above two steps, check if the merge doesn't violate delay time constraint
- For rides **which cannot be merged** in the previous two steps, the following are used:
    - **Euclidean Elimination**
    - **Fair Matching Algorithm**

2.2 Selecting Rides

- All rides within the mentioned pooling window and source as JFK and destination as Manhattan are selected.

2.3 Approximation of Latitude and Longitude - Creating Grids in Manhattan

- **Variation of Distance with Latitude and Longitude**
    - Each degree of latitude is approximately 69 miles.
    - Each degree of longitude is at the max 69 miles.
- **Latitude & Longitude variation of Manhattan**
    - Latitude -> 40.700262 to 40.884967 => 0.19 degree
    - Longitude -> -74.019088 to -73.890887 => 0.13 degree

- Total Area of Manhattan = 22.8 sq. miles
- Divide map of Manhattan into set of points.
- Consider Latitude & Longitude in steps of 0.01 degrees
- Approximately, 250 points cover entire Manhattan.
- Distance between pair of points is approximately 0.7 miles.
- Assign Unique Ids to each point/tuple.

2.4 Checking if destinations of any 2 rides are similar based on computed grids in Manhattan

- Round/Approximate the destination, latitude & longitude to 2 decimal places.
- The resulting approximated latitude & longitude will be 1 of the 250 points within Manhattan
- Create a bucket for approximated destination latitude & longitude.
    - **Key** - tuple of approximated (latitude & longitude)
    - **Value** - set of rides which have this point as destination
- Merge the rides in pairs which fall in same bucket as possible combinations

2.5 Merging rides wherein one ride is en-route to another ride based on computed grids and route direction from Google Maps API

- Sort the rides based on longest trip distance.
- Shortest route from each grid to every other grid is pre-computed using Google Directions API and stored offline
- Route consist of intermediate set of latitudes & longitudes.
- Round/Approximate all the intermediate route entries in the set to 2 decimal places (to get approximate grids in which the trip route goes through)
- Check for the ride which is in drop-off bucket corresponding to any of the above rounded latitude & longitudes.
- Merge the rides in pairs as possible combinations

2.6 For rides which were merged from the above two steps, check if the merge doesn't violate delay time constraint

- Get the list of rides merged together from the above two steps
- Calculate the actual distance it would take if the trips were merged and the accurate time including traffic from the Google Maps API
- Check if the merged trips violate the delay-time constraint for either of the trips
    - If **delay-constraint not violated -** Add the rides as merged into the final table
    - If **delay-constraint violated -** Add the rides back to the original pool to check if they can be merged in the following steps to some-other trip

2.7 Euclidean Elimination

Highly improbable ride merges are eliminated based on Euclidean distance between the ride destinations.

- Calculate Euclidean Distance b/w destinations of every possible ride within the merging pool
- Based on the following formula, eliminate rides which cannot be merged

$$SP(A) + Tdest\ A, dest(B) < SP(B) + D\text{elay}(B)$$

- This can further reduce the computational time spent in the next step wherein the rides are merged based on fair matching algorithm

2.8 Fair Matching Algorithm

- Create a shareability graph with rides as nodes and benefit of sharing in terms of mileage saved as edges
    - For calculating the edge-weights (mileage saved in merging rides A, B), 4 different values are calculated:
        - Distance from Option:1 - Pickup A -> Pickup B -> Drop A -> Drop B
        - Distance from Option:2 - Pickup A -> Pickup B -> Drop B -> Drop A
        - Distance from Option:3 - Pickup B -> Pickup A -> Drop A -> Drop B
        - Distance from Option:4 - Pickup B -> Pickup A -> Drop B -> Drop A
        - For each option check time constraints (i.e) ride doesn't exceed original ride time * (1 + delay time %)
        - Consider option with least distance and store new_distance (merged ride) / (sum of original distance of two rides) as edgeweight
- Combine rides with highest benefit of merging, proceed merging rides which haven't been merged already and with highest benefit
    - Consider nodes which have the highest edge-weight and merge them as one ride
    - Remove the nodes which have been merged from the graph and select from the rest the nodes which have highest edge weight
    - Continue till there are edges left in the graph, remaining nodes are left as single rides

3 Experiments Conducted

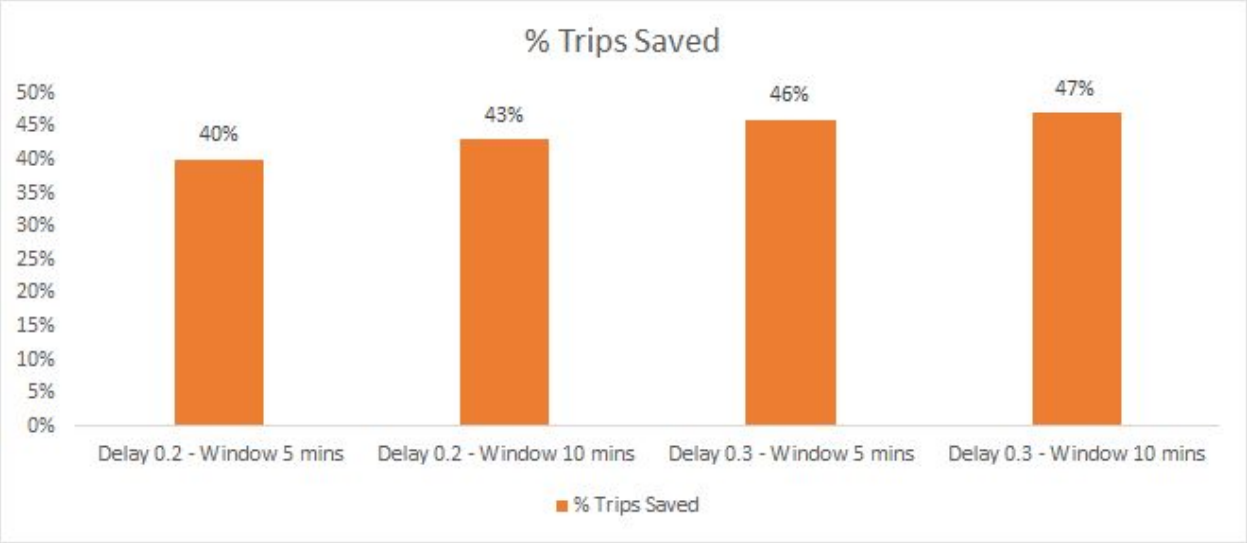
- Different combinations of pooling windows and delay times were considered
    - Pooling windows of 5 min and 10 mins
    - Delay times of 20% and 30%
    - January - 2016 data was considered for the experiment
- To reduce computation time -
    - Offline data of routes was used to merge trips where one trip is en-route to another or if destinations are closer based on grids
- For rides not merged via Grids
    - Google Maps API was used to obtain real-time route, distance and time considering traffic
    - Fair-share Algorithm was adopted to merge rides
- MySQL database was used to store processed data

4 Results

4.1 Test Results

Below are the results for January 2016 data.

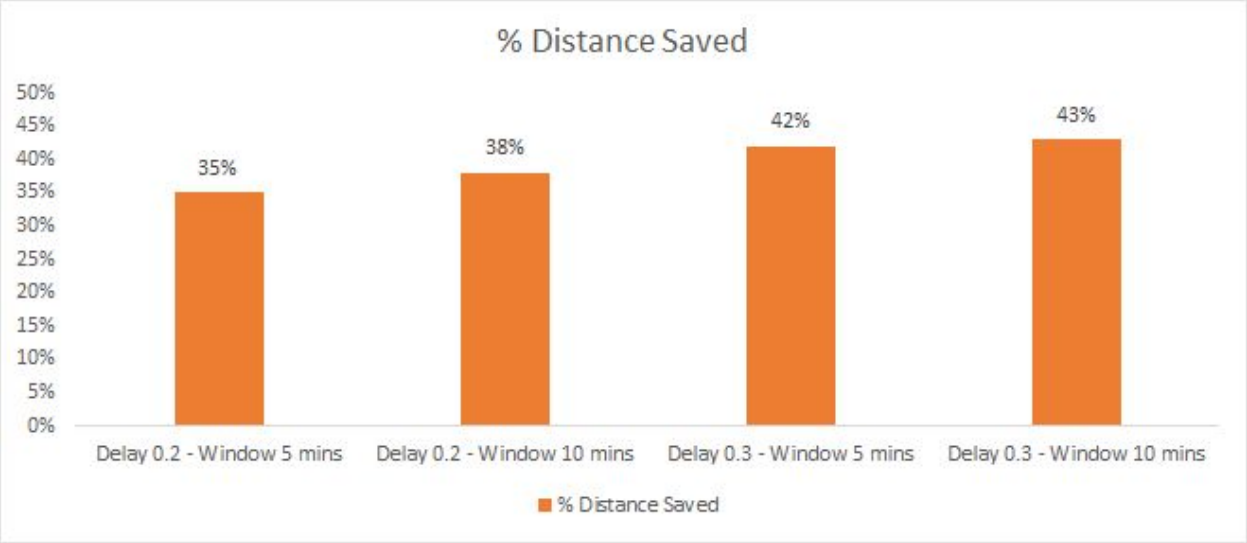
4.1.1 Trips Saved



The above graph depicts the results for different combinations of delay time and pooling window size

- We observe that increasing the delay time had the biggest impact with almost 46-47% of trips being saved with 5 min and 10 min size pooling window

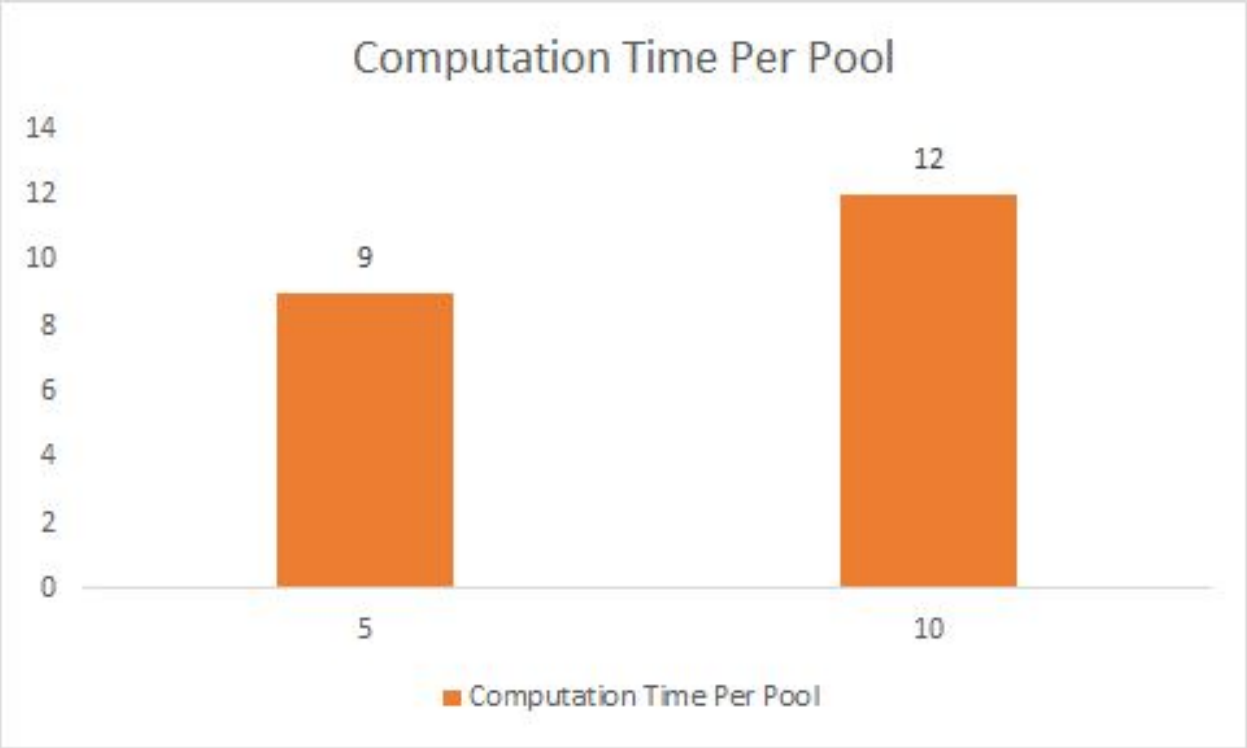
4.1.2 Distance Saved



Same as the previous graph, the above graph shows distance saved results for different combinations of delay time and pooling window size

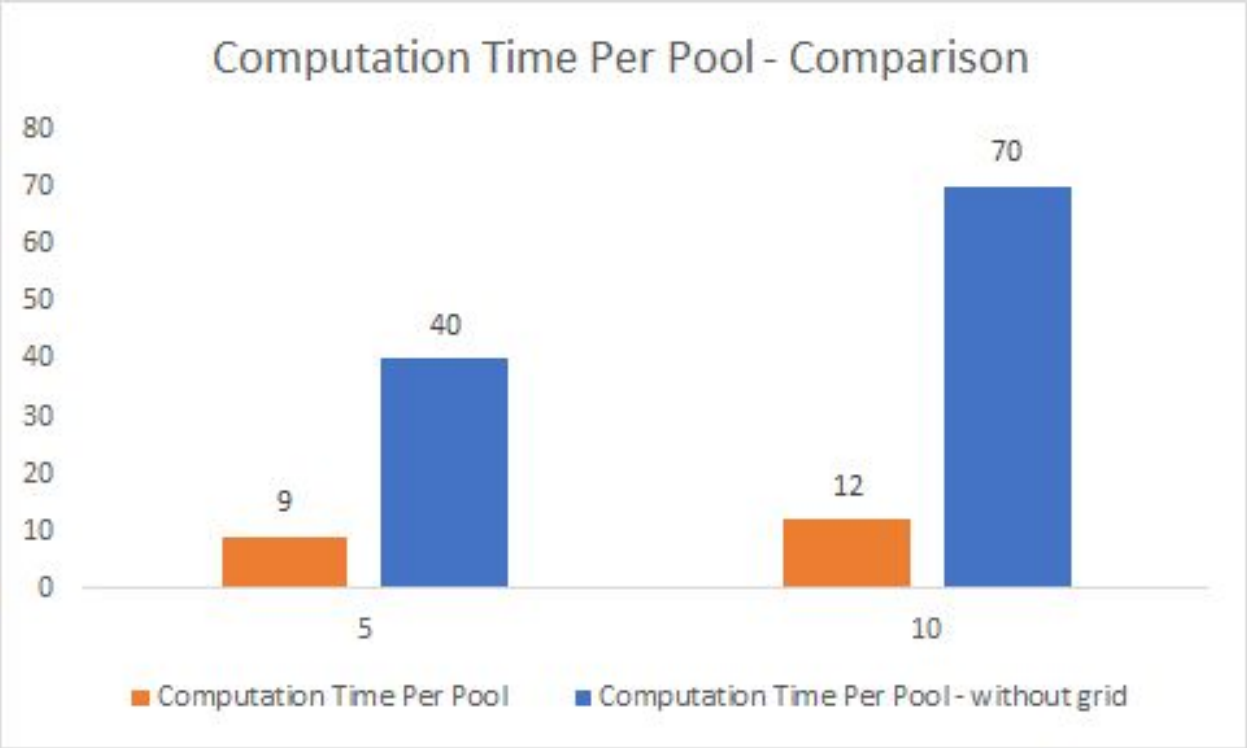
- We observe that increasing the delay time had the biggest impact with almost 42-43% of distance being saved with 5 min and 10 min size pooling window

4.1.3 Computation Time per pool



This graph depicts the computation time taken to process per pool in seconds

- It took ~9 seconds on an average to merge rides in each pool for 5 mins pooling window
- It took ~12 seconds on an average to merge rides in each pool for 10 mins pooling window
- The above data is as expected since we are accessing the Google API to get real-time distance and duration of the rides along with traffic



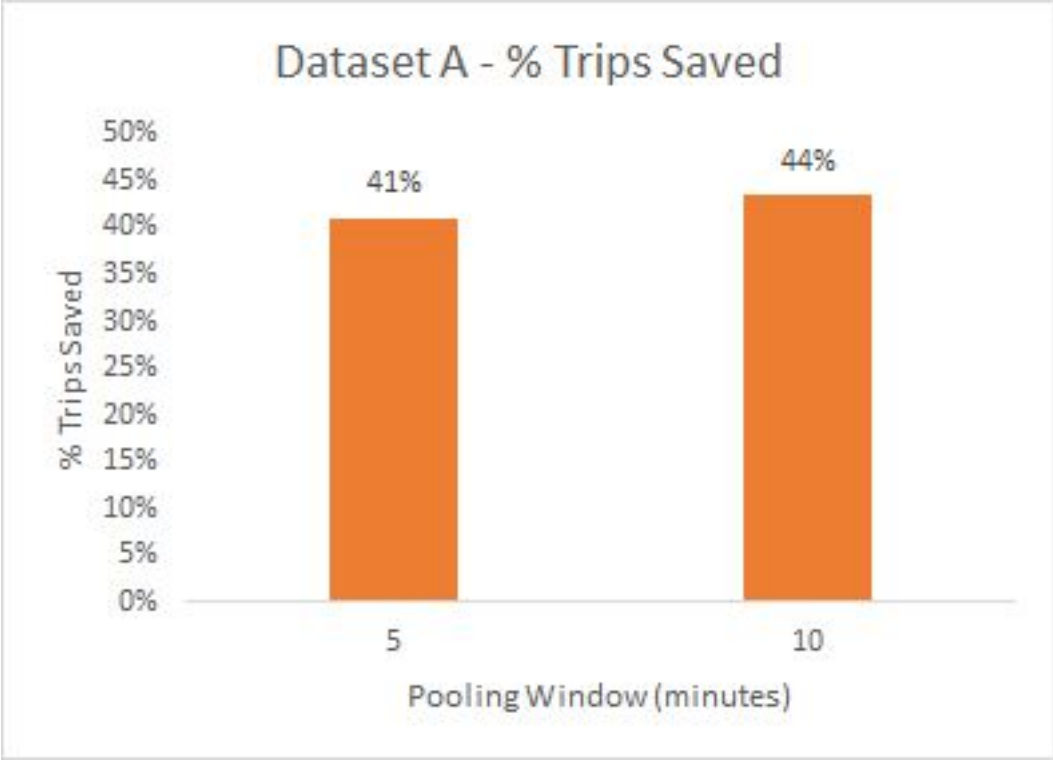
This graph depicts the comparison of computation time taken to process per pool in seconds between our final algorithm and if we had not used grids to pre-merge rides before fair-sharing

- **For 5 minute pooling window:**
    - The orange bar represents the average computation time per pool for our final algorithm - 9 seconds
    - The blue bar represents the average computation time per pool if we did not use grids and ran the fair-share algorithm to merge every possible ride - 40 seconds
    - **Observation:**
        - This is expected since pre-merging rides based on similar grid destination and en-route grid makes the computation graph in the fair-share algorithm sparse
        - Thus, when we ran the code without grids - each pool itself took several seconds(close to a min) to process as for every possible combination of ride that can be merged, **the Google API has to be accessed which takes the maximum computation time**
- **For 10 minute pooling window:**
    - Similar data observed for 10 minute pooling window as well

4.2 Demo Results - Dataset A

- Data considered: 10th June 2016, 8 AM to 10 PM
- Number of pools obtained for:
    - 5-minute window = 168
    - 10-minute window = 84

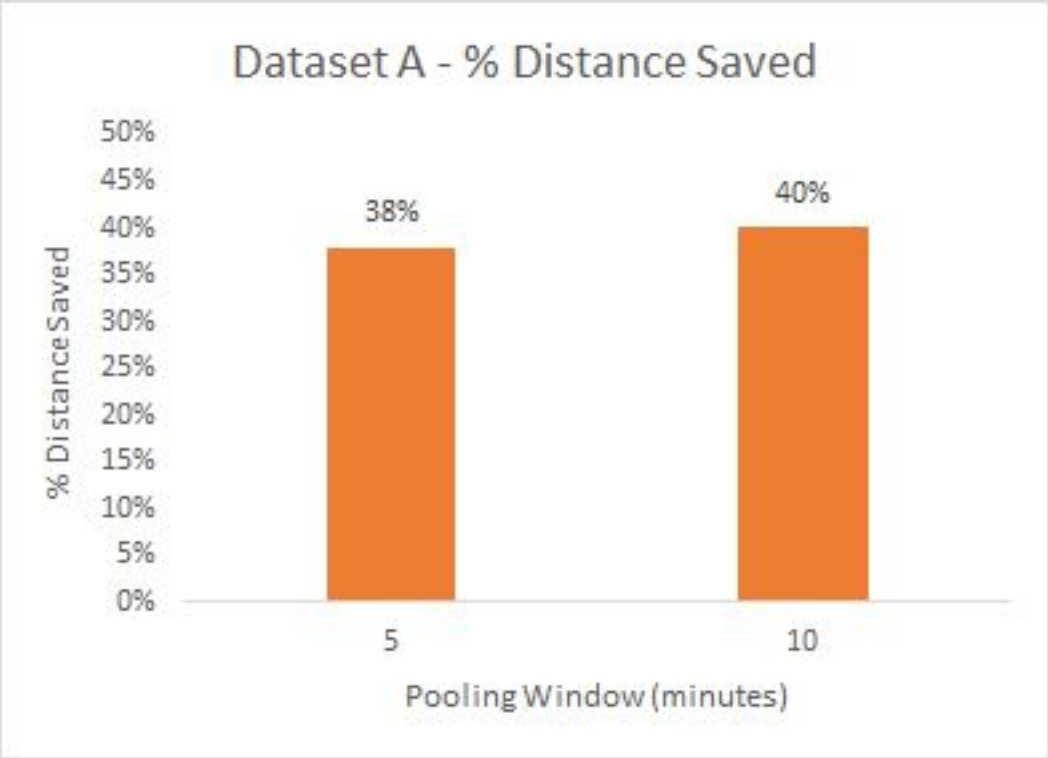
4.2.1 Trips Saved



Above graph depicts the % of trips saved with different pooling window size and delay time 20% for the test dataset A

- Similar results observed but slightly higher trips saved compared to overall month data which could be because the entire month considers trips at night time as well whereas test data A was all only during the day

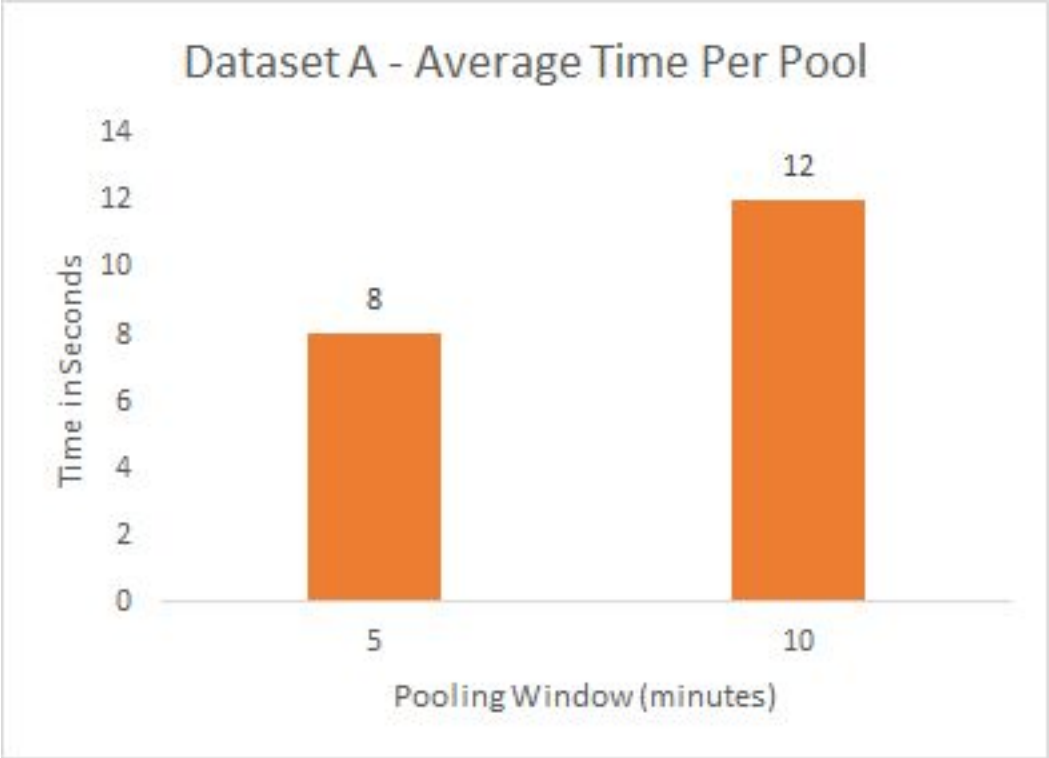
4.2.2 Distance Saved



Above graph depicts the % of distance saved with different pooling window size and delay time 20% for the test dataset A

- Similar results observed but slightly higher distance saved compared to overall month data which could be because the entire month considers trips at night time as well whereas test data A was all only during the day

4.2.3 Computation Time per pool

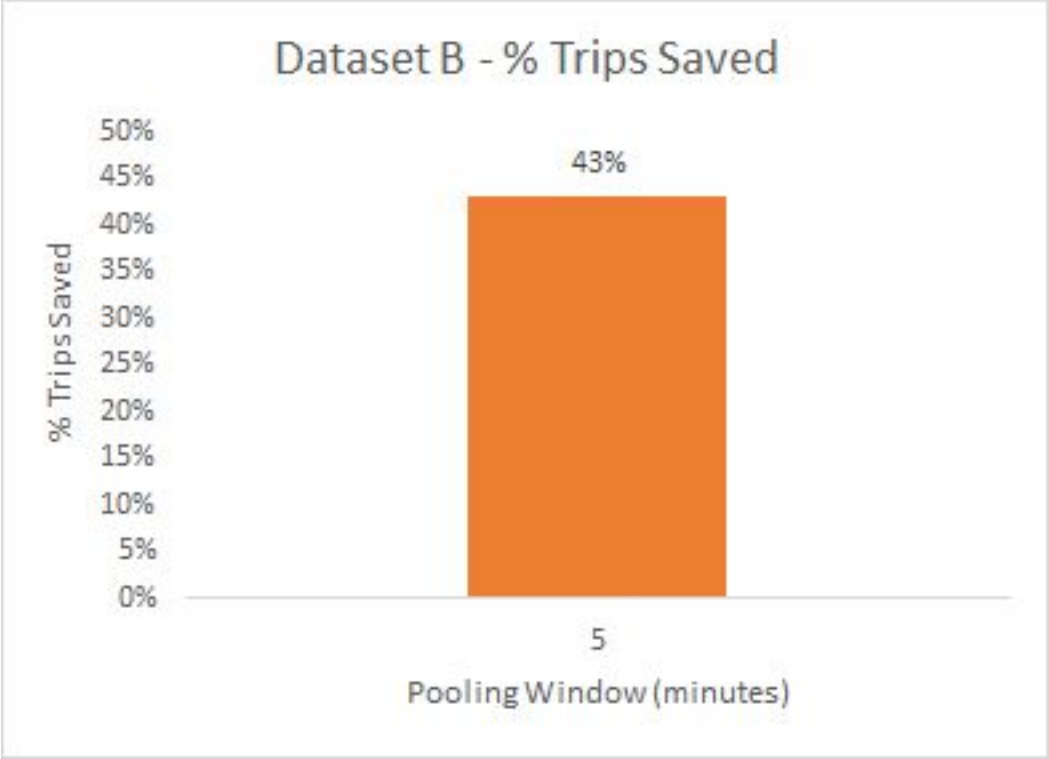


Similar average computation time per pool compared to the overall month's dataset

4.3 Demo Results - Dataset B

- Data considered: 10th March 2016, 2 PM to 4:30 PM (5 min pooling window)
- Number of pools processed: 30

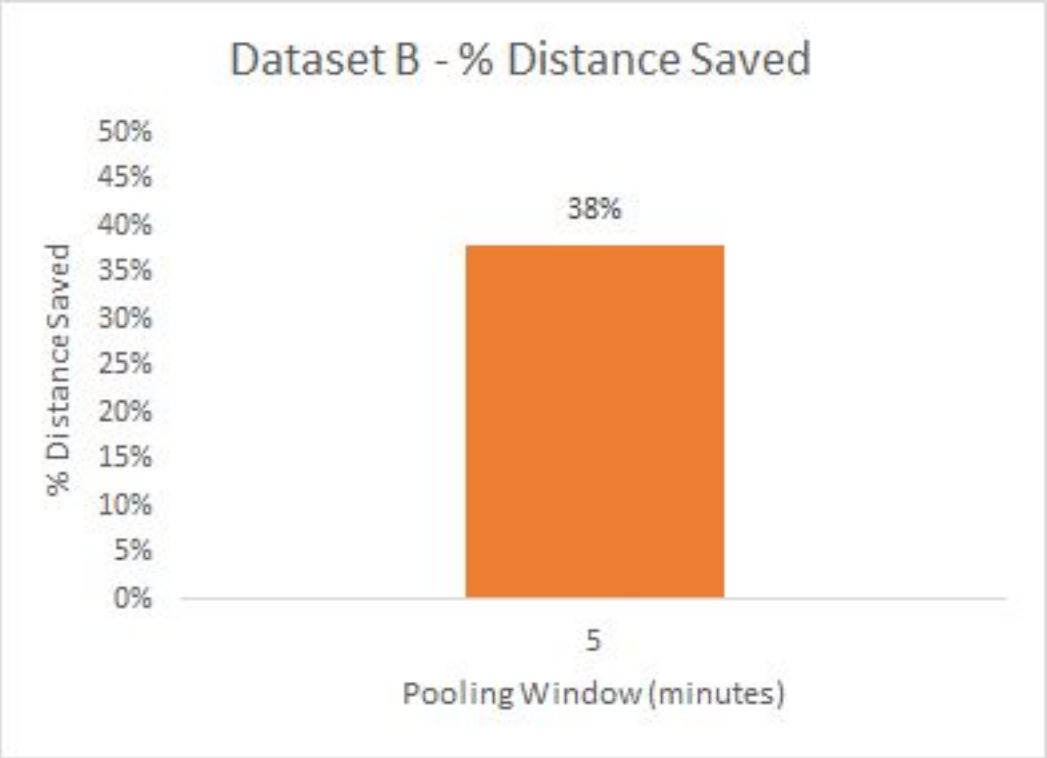
4.3.1 Trips Saved



Above graph depicts the % of trips saved with 5 min pooling window size and delay time 20% for the test dataset B

- Similar results observed but slightly higher trips saved compared to overall month data which could be because the entire month considers trips at night time as well whereas test data B was all only during the day

4.3.2 Distance Saved



Above graph depicts the % of distance saved with 5 min pooling window size and delay time 20% for the test dataset B

- Similar results observed but slightly higher distance saved compared to overall month data which could be because the entire month considers trips at night time as well whereas test data B was all only during the day

5 Software/Technology used

5.1 Project Tools

- Python
- Jupyter Notebook
- MySQL DB

5.2 Collaboration Tools

- GitHub - for code sharing
- Slack - for resource sharing

6 Libraries and dependencies

6.1 Python Libraries

- pandas
- numpy
- googlemaps
- math, datetime
- pyodbc
- json, requests
- sqlalchemy

6.2 Offline Grid Route Data

- Pre-computed offline routes between grids and saved as JSON file

7 References

Piazza course resources: https://piazza.com/uic/spring2019/cs581/resources
Google Maps API - https://developers.google.com/maps/documentation/