**Ride Analytics on New York City Taxi data**

Graduate Group # 5

Sai Duth Deekshit G, 810895764, sgarimel@kent.edu

Rohit Reddy G, 810894809, rgunna@kent.edu

Rohith Varma J, 810883584, rjampana@kent.edu

Sumanth D, 810882836, sdoddi1@kent.edu

**1. Introduction**

Transportation plays a vital role in large cities. There are several different transportations available to the people. Among them taxi mode of transportation has become a key player in large cities of united states and other countries. People are using taxis to get their necessities. From [1], as per NYC taxi and Limousine commission it is known that in New York city there are approximately 50,000 vehicles and 1,00,000 drivers exist as of today. There are variety of businesses like Uber, Yellow Taxi, Green taxi etc., that provide taxi services. Also, the data with lots of information in it about the taxi trip details and other attributes are made available online by the government. Using this data, we try to perform analytics on ride details which would help taxi business to improve their business and how to overcome the needs of the public. Also, would help government in planning effective transportation system. Along with this we would like to find top K nearest and most visited business places between pickup and drop off location of the ride with help of Yelp API to find the business with specific location coordinates.

In this project, we have considered the datasets of yellow and green taxi records which contain attributes like pickup and drop-off time and date, pick-up and drop-off locations, trip distances, fares, rate types, payment method, passenger count. The data is in CSV format for taxi rides month wise during the years 2009-16. The total size of the available data set is ~50GB. We plan to include part of the data available to fulfill our goals. The project is planned to be designed to scale to process more amount of data is needed and resources for it are available.

**2. Project Description**

This project is about analysis on taxi data available from the NYC government. This analysis is used to find solutions to the queries like which are the most common pickup and drop off locations, which are busiest routes for taxis, what are the most revenue generated areas for cabs in NYC, Trip duration and analysis on fares based on date and time etc. This would help cab drivers to find places that make them find enough rides and business firms to find the better areas to improve their business. We also provide data of most popular places that people frequently take ride in and around pickup and drop off location at specific time in a day which would help the new people to plan their schedule accordingly and could help potential businesses find most profitable areas to do business.

Since there is lot of data in the dataset, the initial challenge would be to eliminate the unwanted stuff from the dataset by cleaning the dataset. In the data cleaning process columns that are not relevant to the data analysis are discarded. As the data is large it difficult to perform operations efficiently using normal query languages. So, we use Apache Spark which provide an efficient way to perform operations on the data in terms of space and time. We execute the queries on the platform provided by [www.databricks.com](http://www.databricks.com). It provides a Virtual Analytics platform on top of Apache Spark used to execute advanced analytic solutions. The next challenge is to find K nearest neighbors between two points. In [2] distance used in continuous nearest neighbor was given based on the Euclidian distance between two points which may not be applicable for road networks as road networks are graphs that contain various edges. So, instead of finding the Euclidian distance we use Google Matrix distance API which provides the road network distance between two points and then find the continues nearest neighbor.

Workload distribution:

Data cleaning and analytics part was distributed among Sumanth D and Rohith Varma J and finding the popular restaurants based on ride evaluation and finding nearest popular restaurants between two points was distributed among Sai Duth Deekshit G and Rohit Reddy G.

**3. Background**

* In [2] Yufei Tao, Dimitris and Shen defined a method for retrieving nearest neighbor between every point of a line segment. The result contains a set of <point, interval> tuple in which point is the neighbor value and interval is the time span for which point is the nearest neighbor between two points. This is done by using Euclidian distance between the points. But, in road maps as it is a network using Euclidian distance to find the result may not be an efficient method as the actual network distance may vary from the Euclidian distance. So, to obtain the actual distance we use google matrix distance API. Combining this with CNN method could give us better result when compared to using Euclidian distance in CNN.
* Apache Spark is an engine for processing big data in fast and efficient manner. It contains several built-in modules for streaming, SQL, machine learning and graph processing. It provides an API named resilient distributed dataset (RDD). RDD allows us to develop both iterative algorithms which visit the dataset several times in a loop and exploratory data analysis (repeated database style querying of data) [Wikipedia]. It can execute batch processing jobs much better and faster than MapReduce (10 to 100 times). It contains a GraphX which provide a distributed graph system. Spark can run on Hadoop along with other tools like Hive and Pig which come under Hadoop ecosystem.
* Yelp API provide us the businesses name and address around the given location (latitude and longitude).
* Google Matrix Distance API provide us the distance between two locations (latitude and longitude) by taking the coordinates of two locations as input.
* Required software tools:

Query Language DBMS, SQL

programming languages: Scala, Python

Online tools: [www.databricks.com](http://www.databricks.com) (for running Scala or Python cells and storing dataset)

API’s: YELP API for getting list of business firms like hotels, restaurants etc. at a given location or coordinates. Google Distance Matrix API to get the road distance between two location coordinates.

* Required hardware:

Windows 10 or MAC OS, RAM: 4GB or more, Minimum HDD:50GB, Internet connection.

**4. Problem Definition**

Formal Problem definitions:

1. Analysis on Ride data:

* **Finding most frequent drop off which is used to find the popularity of a place:**

We are analyzing the most frequent pickup and drop off locations in NY city using MapReduce programming.

Most common pickup locations: Here, we are performing analysis on a region by considering the longitude and latitude of that pickup location. pickup\_longitude and pickup\_latitude are combined to form pickup\_location.

MapReduce Function:

Mapper:

map(Pickup\_location) = Emit(inter\_Pickup\_location,1)

Reducer:

reduce(inter\_Pickup\_location, 1) = Emit(Pickup\_location, Sum).

Most common drop off locations: Here, we are performing analysis on a region by considering the longitude and latitude of that drop off location. dropoff\_longitude and dropoff\_latitude are combined to form dropoff\_location

MapReduce Function:

Mapper:

map(Dropoff\_location) = Emit(inter\_ Dropoff\_location,1)

Reducer:

reduce(inter\_ Dropoff\_location, 1) = Emit(Dropoff\_location, Sum).

* **Finding most revenue generated areas in New York City:**

It describes which locations are most frequently visited i.e. regions which have more number of pickups. This is done using word count in MapReduce paradigm. Results are ordered based on the count of the pickups. Based on the results generated, we can know the most revenue generated areas for cabs.

* **Finding the average distance, average fare, payment methods:**

|  |  |
| --- | --- |
| Statistic | Value |
| max(total amount) | 185.75 |
| avg(total amount) | 15.31 |
| min(total amount) | 0.0 |
| max(passenger count) | 6 |
| avg(passenger count) | 2 |
| min(passenger count) | 1 |
| max(trip distance) | 42 |
| avg(trip distance) | 2.6 |
| min(trip distance) | 0 |

* **Finding whether driver took actual route to destination or not by comparing the actual distance:**

We can determine whether driver took the correct route or not by comparing the total trip distance with the distance given the Google API. If both the distances are same, then driver took the correct route. If both the distances are different, then driver took the wrong route.

1. Finding popular places between pick up and drop off locations:

* Using Continuous nearest neighbor along with google matrix distance API for finding the nearest places.
* Comparing them with our analyzed data in 1 find the popular place and output them.

Challenges:

* Eliminating unnecessary data.
* Handling huge data with efficient time complexity.
* Defining a method for finding the popular places between two locations.

Solutions:

* Using apache spark for performing the queries which deals with time and space problem for huge amount of data.
* Using nearest neighbor algorithm along with Google matrix distance API to find the K (any number)-famous places between pickup and drop-off location.

**5. The Proposed Techniques**

**Increasing the Speed of Execution of Analytics Using Apache Spark Framework on DataBricks:**

As defined in the above section, In the first phase we perform analysis on ride data of NYC. We write code in Scala using databrics which is built on Apache spark platform to run the code which has many advantages when compared to rest of the MapReduce technologies like Hadoop and storm. We choose spark because, Hadoop is not efficient when dealing with use cases that require multipass computations and algorithms. It is because, it requires frequent conversion of uses case into MapReduce pattern which makes the process slow due to replication and disk storage. Also, Hadoop clusters are difficult to set up and manage. So, we have used apache spark as a platform to perform our analytics because it involves less expensive shuffles in the data processing. Spark provides in-memory data storage which makes it easy when working with same dataset multiple times.

While processing the data in first phase spark process the data in different stages given in below figure.

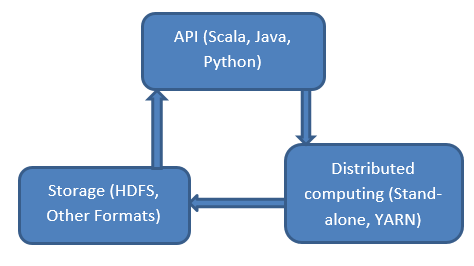


Figure Apache spark architecture model

The framework of Spark contains Resilient Distributed Dataset (RDD) which is like a table in a database. We input the data set to this RDD. This stores the data in different partitions. This mainly works on two operations. One is Transformation and other is Action. When a transformation function is called on a RDD nothing is evaluated it takes the RDD and returns a new RDD. Some example functions of transformation are filter, groupByKey, reduceByKey, map, aggregateByKey etc. When an Action function is called on a RDD object, data processing queries are computed at the time and the result value is returned. Example operations of action are reduce, collect, count, first, foreach etc.

We use Apache Spark with Scala to perform analytics on a part of the dataset. For performing analytics, the data is first read into a Dataframe and its API helps us perform different queries on the dataset. Parts of the dataframe are exported to create another dataset to perform analytics on small set of fields.

var coordinatesDf = inputDataframe

. select(”pickup longitude”, ”pickup latitude”,

”dropoff longitude”, ”dropoff latitude ”,

”trip distance”)

The following code snippet shows the kind of queries performed to obtain these results

inputDataframe

. select(max($”dropoff longitude”))

.show

We run the similar queries as above to compute results for following scenario (The code for this is given in attached file in file **“RideAnalytics.html”**):

* Finding most frequent drop off which is used to find the popularity of a place.
* Finding most revenue generated areas in New York City.
* Finding trip duration and analysis on fares based on date and time.
* Finding the average distance, average fare, payment methods.

Each query is converted into a spark job and is executed based on type of function Action or Transformation.

The converted spark jobs similar to that of mentioned below in figure2.

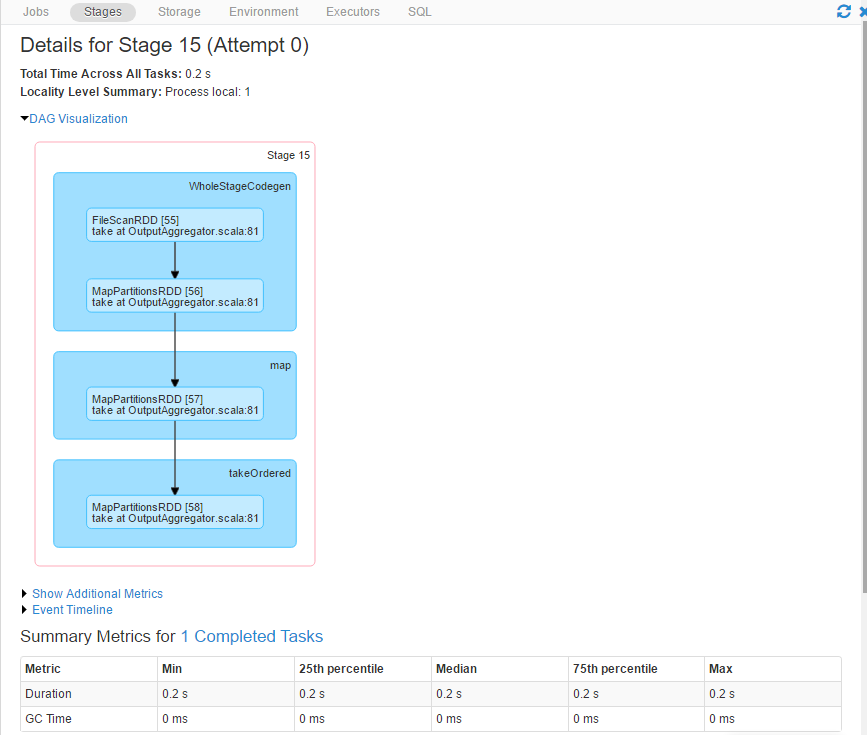


Figure Spark stage example

We observe that the computing speed of queries is better than the other MapReduce technologies like Hadoop.

**Finding whether driver took actual route to destination or not by comparing the actual distance:**

For finding whether the driver took correct route or not we use Google Distance Matrix API. Intially we take the pick up and drop off latitude and longitude and the total travel distance given in the dataset. Next, we have created a HTML file which contains code for JQUERY and AJAX. It takes a file containing the above mentioned fields as input and outputs total count of trips and count of trips for which the actual distance and distance from google matrix distance api are same. In doing so, we take pick-up and dropoff latitude and longitude values and place them in the below URL.(Code for the following can be found in **“distanceAnalytics.html”**)

<http://maps.googleapis.com/maps/api/distancematrix/outputFormat?parameters>

In the above URL output format is the format of return type. The Parameters can be ORIGIN which is the pick-up latitude and longitude and DESTINATIONS which is the drop off latitude and longitude. Also, we should include the Google API Key.

So, when we include the following details in the URL and request to google API, it return a JSON file that contains the distance which is taken as optimal distance between given origin and destinations. We compare the total ride distance from the dataset with optimal distance. If the actual distance is less than optimal distance then ride is said to taken correctly and count of optimalTrip is incremented. Finally, we display the optimalTrip count among the total trips. The obtained result can be used by the users, government, Taxi companies to check and take necessary actions.

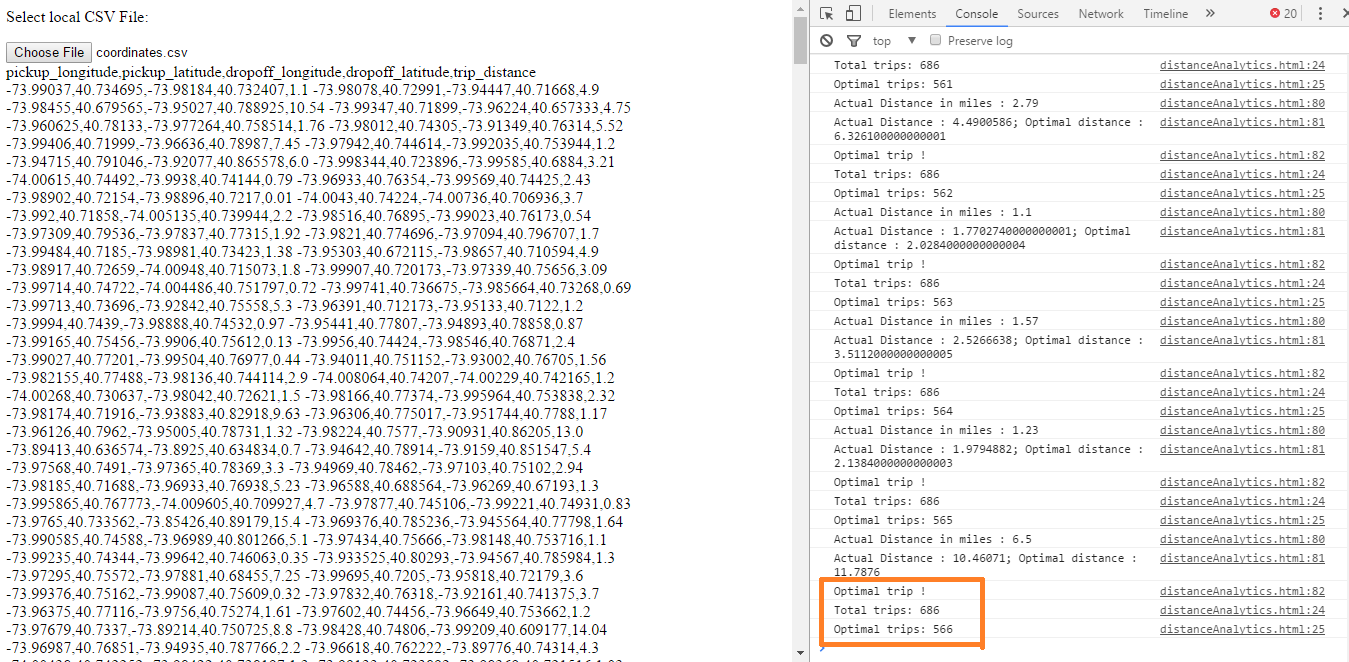
****

Figure Optimal ride details and GUI

**Finding the popular places around the drop-off location using R-tree:**

In this method, we can find the popular nearest neighbors around drop-off location given a range. Since there is lot of data that is to be searched we have used R-tree as a data structure to store the coordinate values. R-tree is very efficient in dealing with spatial access methods like geographical coordinates, rectangles or polygons. They are quick in generating the results for queries such as “Find all hotels within 2km of given location”. It really works well for problems on nearest neighbor search. A minimum bounding rectangle (MBR) is constructed from given coordinates. By using the concept of MBR R-tree eliminates the need to search queries that doesn’t intersect the MBR. This minimizes the time to search queries and increase the speed of execution. From the collected data we have selected coordinates of pickup and drop-off locations and popularity value that we got based on analysis done in phase-1 and stored them in a file. Then we have constructed an R-tree based on the coordinate values with maximum 4 children. We provide a drop off location and distance value within which the Nearest Neighbors are to be found. Then these inputs are sent to R-tree search method which return the tree containing coordinates that satisfy the given query conditions. We then sort the result based on the distance and compare with popularity values. Finally, we display top 5 nearest neighbors to the given drop-off location(Code for the following can be found in **RideAnalytics.java**).

Algorithm:

FindingPopularPlaces(File file){

Read the file;

List input = file;

//create R-tree using the coordinate values in file

createRtree(file);

//provide the point for which nearest neighbors are to be calculated

input the dropoff\_coordinates;

//provide the distance within which the nearest neighbors are to be calculated

Input distanceKm;

//search the R-tree for coordinates that satisfy above input conditions

List result= Search (rtree, dropoff\_coordinates, distanceKm);

//sort the result and compare with popularity values

Compare (result, file);

//display top 5 NN

Display(top\_nn\_values);

}

Search (rtree, dropoff\_coordinates, distanceKm){

//create a variable from which distance is to be computed

Position from = dropoff\_coordinates;

//create rectangle bound using ‘from’ and distanceKm

Rectangle bounds = createBounds(from, distanceKm);

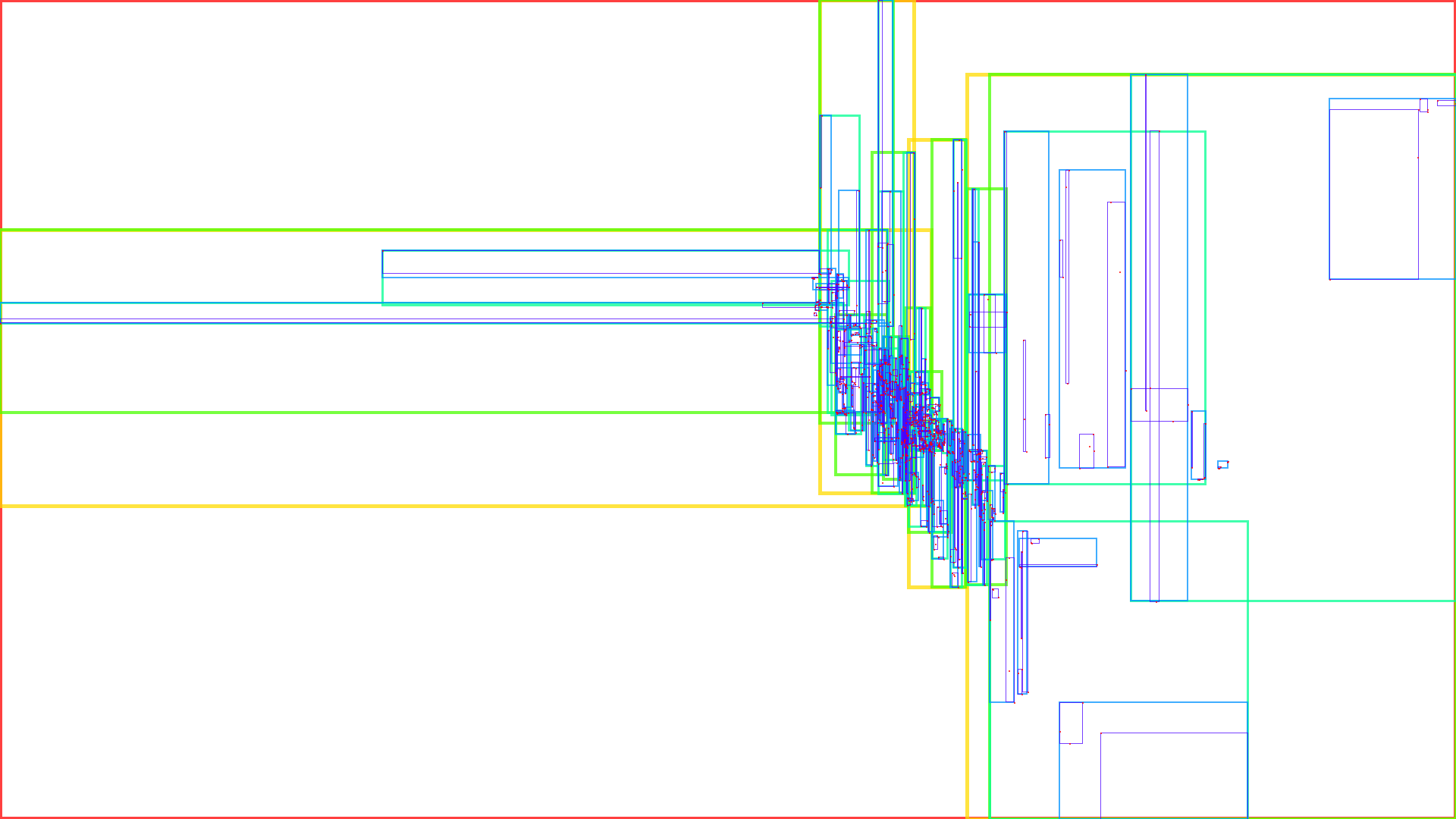
//Search the rtree initially with bounds

tree.search(bounds);

//If the bounds doesn’t overlap the tree return null else refine using exact distance

tree.filter (return if from.getDistanceToKm(position) < distanceKm;

}

****

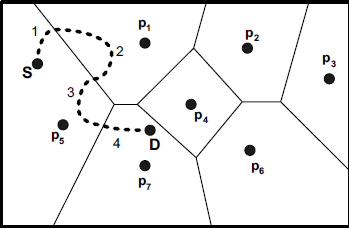
Rtree for dataset

**Finding continuous nearest neighbor between two points using voronoi based network nearest neighbor VN3:**

Existing continuous nearest neighbor queries are using Euclidean distance to find the nearest neighbors between two points. The main disadvantage with this approach is that they use Minkowski distance metrics which cannot be applied on road network which are metric space so, they cannot approximate the exact distances. To overcome these issues, incorporation of VN3 approach can make the continuous nearest neighbors queries to find nearest neighbors of an object efficiently.

In [3], two methods are discussed to solve the problem of finding continuous nearest neighbor in road networks. They are intersection examination and upper bound algorithm which are based on voronoi network nearest neighbor approach.

VN3 technique mainly depends on voronoi diagrams which divides a space into disjoint polygons. In the Euclidean plane, there are set of points called generator points and these generator points are assigned with set of locations called voronoi polygons. In a polygon, nearest neighbor of a point is known as generator of the polygon. In voronoi polygon, the distance between two points is measured as their shortest distance. To find the first nearest neighbor of a query, network voronoi polygon are directly used. To find the correct network distances from query object to the generator points in the candidate set, precomputed distances using Google Matrix Distance API are used. Initially in VN3, a network voronoi diagram for point of interests like hotels, gas stations etc in network are calculated and the resultant network voronoi polygons are stored in a table. Using the network voronoi polygons(NVP) constructed in above stage we can compute the first nearest neighborhood of q by locating NVP that contains q. Next network distance between the border points of the NVP is pre-computed which can be done by using Google matrix distance API. This network distance is used to find the distance between query object and candidate set which is generated from the filter step. Below Figure shows the Voronoi network diagram.



Voronoi Network Diagram (source [3])

**Intersection Examination:**

In Intersection examination approach, KNN’s are computed for the nodes at intersections of the path. This approach states that, if there is a path from A to B and n1,n2 and n3 are the nearest neighbors to A and n4,n5 and n6 are nearest neighbors to B then k nearest neighbors of any object between A and B will be subset of { n1,n2 ,n3, n4,n5 ,n6}provided that there will be no split points on the shortest points between these objects. If there is one or more point of interest between the path from A to B then above statements cannot be applied. We can solve this issue in two ways. Firstly, divide the path into sub paths so that the sub paths do not contain any point of interest. If there is a point of interest n3 in between A to B then line segment is divided as (A, n7)and (n7, B) which requires the computation of nearest neighbor for n3 also. Secondly, similar to the statements defined above we can include the point of interest n7 in candidate set. On each segment, after specifying the k nearest neighbors of the nodes, we should find the location and the k nearest neighbors of the split points. KNN’s obtained are defined as either increasing or decreasing groups which can be used to find the split points. Below are the steps for finding the continuous k nearest neighbors from A to B.

* Divide the line segments into sub segments.
* A candidate list of KNN’s of each node in above divided segments are calculated using VN3 approach.
* By taking A, sort the candidate list and specify whether nearest neighbor is increasing or decreasing.
* Specify the location of the first split point by considering one increasing and one decreasing nearest neighbor. If Oi and Oi+1 are NN’s from sorted list. Distance is defined by the following formula.
* Update the sorted candidate list to reflect their distances to the first split point p1 by adding/subtracting the distance of A and p1 to/from the members that have increasing/decreasing distances to A.
* Now consider the obtained split point as starting node for new segment from split point to B and repeat steps from four to six until new split points has similar KNNs as B.

**Upper Bound Algorithm:**

Intersection examination approach gives a precise set of results but it executes some of the KNN queries unnecessarily. This can cause delay in execution time when dealing with large amount of data. To avoid this unnecessary computation of KNNs, Upper Bound Algorithm can be used. Working of upper bound algorithm is same as intersection examination approach. In Intersection examination, k nearest neighbors are computed for every intersection. Upper bound approach will postpone the calculation of the k nearest neighbors and locations which are needed and it gives efficient performance by decreasing the k nearest neighbor computations. In [4] it was given that, there will be no change in the KNN of a query object if the movement is small. So, it is not required to compute the KNN for some threshold. Given a query object q and oi+1 is one of the KNNs of q, the threshold is defined as ⸹ = min(d(oi+1, q)-d(oj, q)). This threshold provides us the minimum distance between two subsequent KNNs of q. The KNNs of the query object remain same until distance of q is less than ⸹/2.

As an extension to this threshold a new threshold ⸹1 is defined in [3] which can improve the performance of the system. It is explained with help of figure-4.

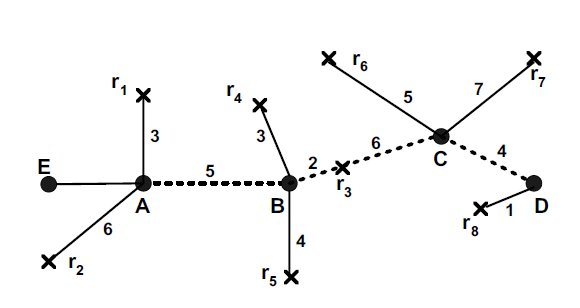


Figure 4 C-KNN query example (source: [3])

In Figure 4 let us consider that a car is travelling from A(source) to B(destination) and we are interested in finding three nearest neighbor restaurants. In doing so, use Intersection examination to find first three nearest neighbors around A and B. The result set would be like for A KNNs are {(r1, 3), (r2,6), (r3,7)}. Now the value of ⸹ = 1. So, the first three KNNs are same for distance of ⸹/2 i.e., 0.5. The next NN query should be given at 0.5 or greater. But as we discussed above that if two candidate list members are both increasing or decreasing, or first one is increasing and other is decreasing then they cannot will not generate any split points on the path. Based on this property UBA algorithm was defined. In this a new threshold is calculated as ⸹1 = min (d(oi, q) - d(oi+1, q)). In this Oi should have increasing distance and Oi+1 should have decreasing distances to the query object. This help in eliminating calculation of NNs that doesn’t generate split point by specifying the minimum difference between distances of NNs that generate split points. From the above figure 4, if the car moves a distance less than ⸹1/2 to its initial position then the KNNs will be the same if it is greater, then a new KNN is set at the intersection point that is immediately before the point that has a distance of ⸹1/2 to the initial position of the car. ⸹1 is always greater than or equal to ⸹.

From figure 4, if a car is travelling in path (D,C,B,A). Initially four nearest neighbors to D and if they have increasing or decreasing distances are calculated using VN3 approach. They are {I(r8,1), D(r6,9),D(r3,10),D(r7,11)} where I is increasing distance and D is decreasing distance from D. Now from above defined technique we select r8 and r6 which are increasing and decreasing distances that satisfy the condition of ⸹1. Now using r8 and r6 we compute the threshold value ⸹1 which will be 9-1 = 8. From this we can know that the 3NN to the travelling car will be the same until distance from initial position D is (8/2)=4 for which (3+1)KNNs are not required to compute from D🡪C path. In the discussion about VN3 it was stated that the first nearest neighbor of a moving point remains the same as long as the point stays in the same NVP. So, based on necessity we can avoid comparing first and second NN. The change in the first neighbor can be identified based on the intersection of the travel path with the NVPs. In this way we can efficiently find the Continuous KNNs in a road network.

The pseudo code for this algorithm is given below:

**Function UpperBoundAlg (Path p)**

Divide P to segments that satisfy the property 1 as P ={n0,n1……, ni}

Start from nj=n0, while nj ≠ ni :

Find (K+1)NN(nj) and their directions

Calculate ⸹1

Find np,nq where ⸹1 between (np,nq)

If(nq = nj+1):

IntersecExam(nj, nj+1)

nj = nj+1

**6. Visual Applications**

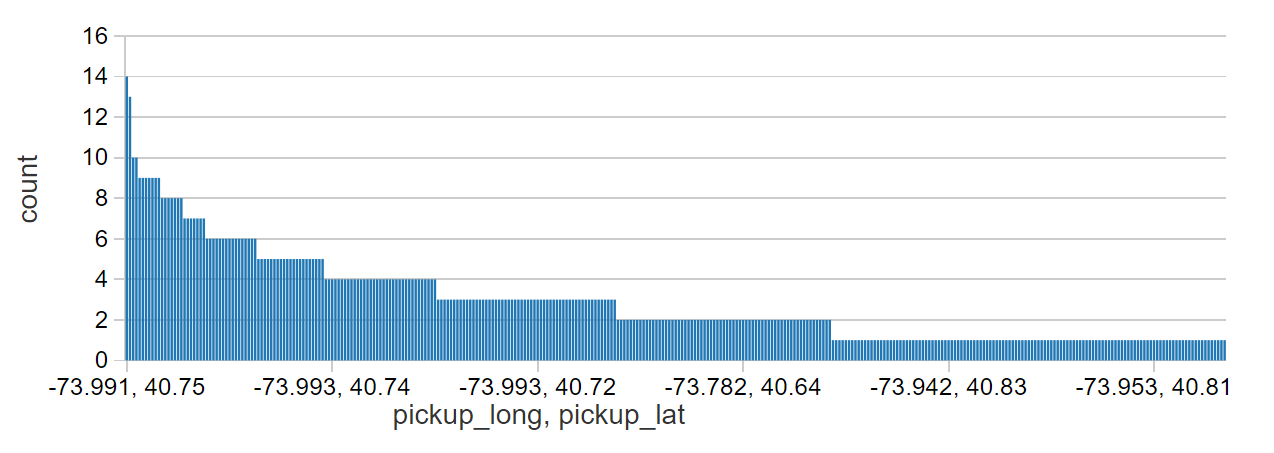
****

Figure 5: Most common pickup locations

We can observe from figure 5 that most common pickup locations are around -73.991, 40.75 coordinates.

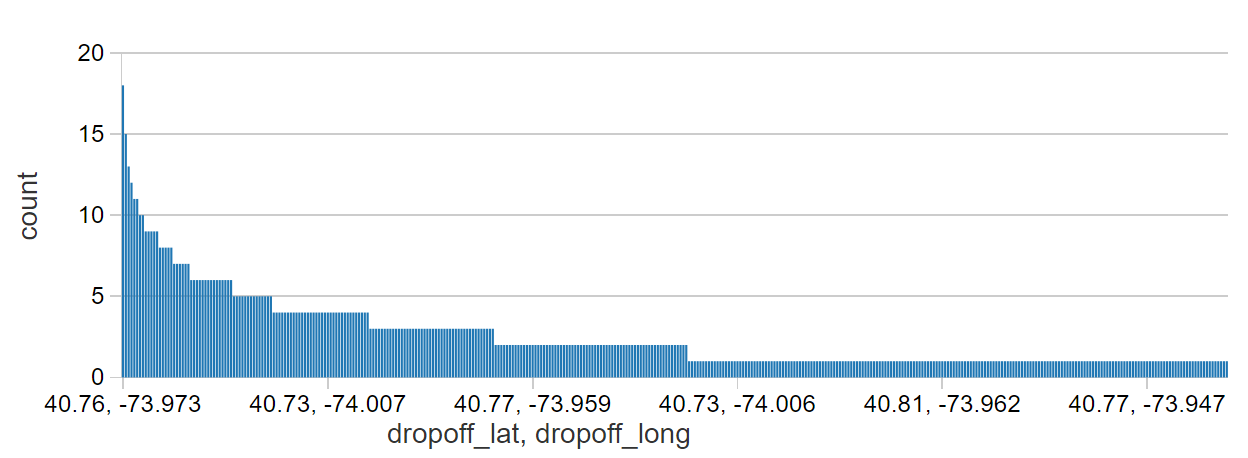
****

Figure 6: Most common drop off locations

We can observe from figure 6 that most common dropoff locations are around -73.973, 40.76 coordinates.

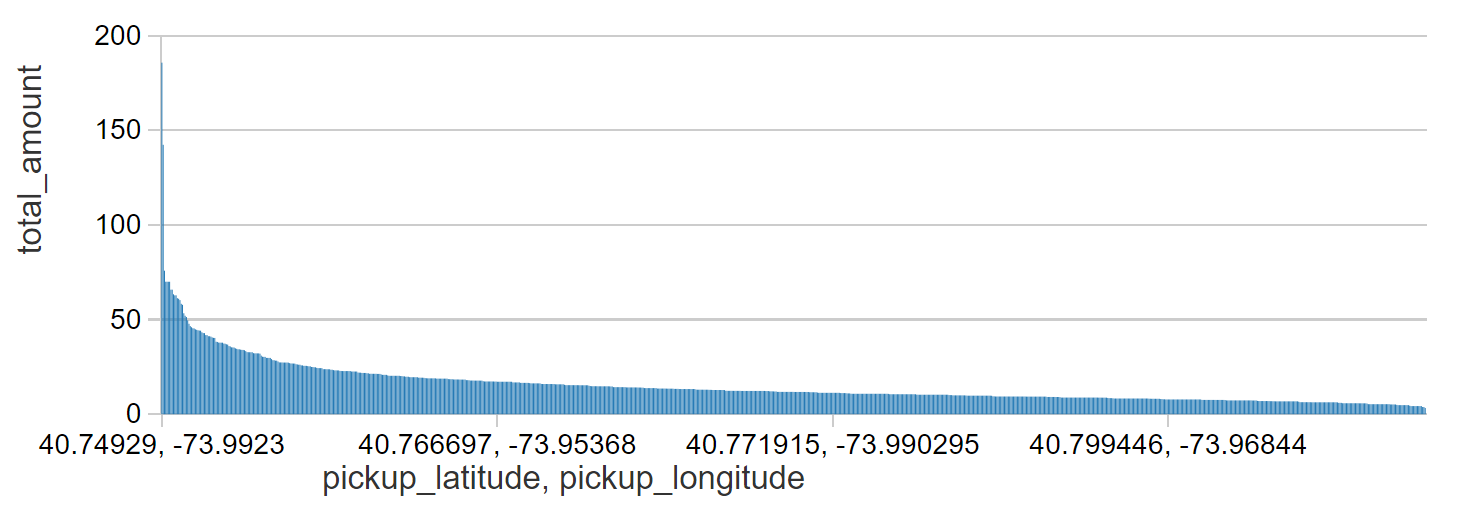
****

Figure 7: Most revenue generated areas for cabs

We can observe from figure 7 that most revenue generated locations are around -73.9923, 40.74929 coordinates.

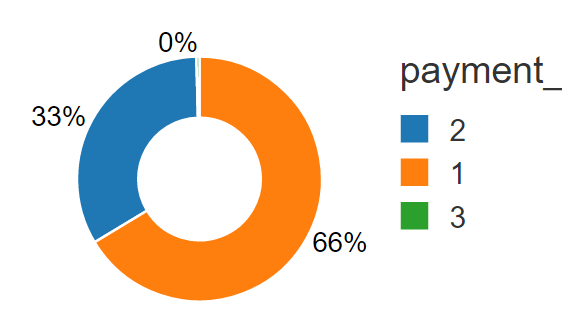
****

Figure 8: Types of payment

We can observe from figure 8 that most payments are made using cards.

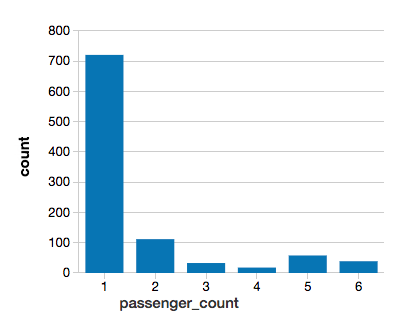


Figure 9: Passenger count vs number of rides

We can observe from figure 9 that rides with one passenger are more in number than rides with other passenger count.

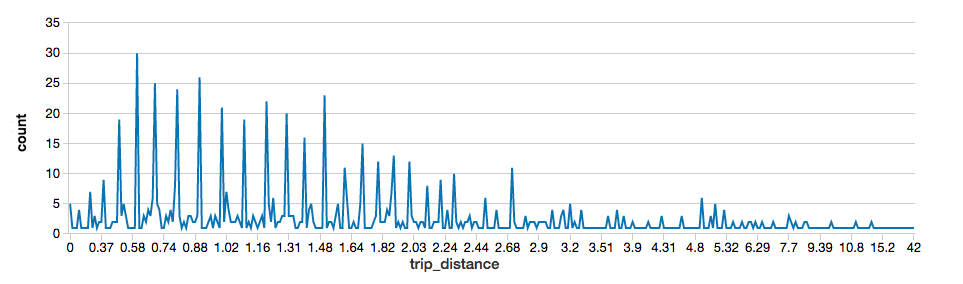


Figure 10: Trip distance vs number of rides

From figure 10 we can observe that Most of the trips are around 0.8 to 2.5 miles.

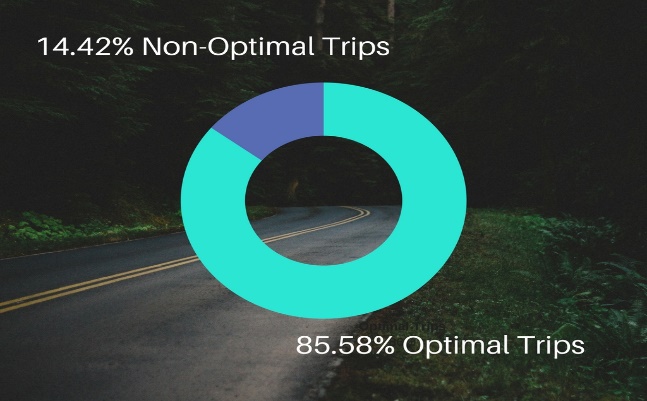


Figure 11: optimal rides vs non-optimal rides

We observe from figure 11 that out of total rides 85.58% rides are optimal rides. That means driver took correct distance to the destination in 85.58% cases.

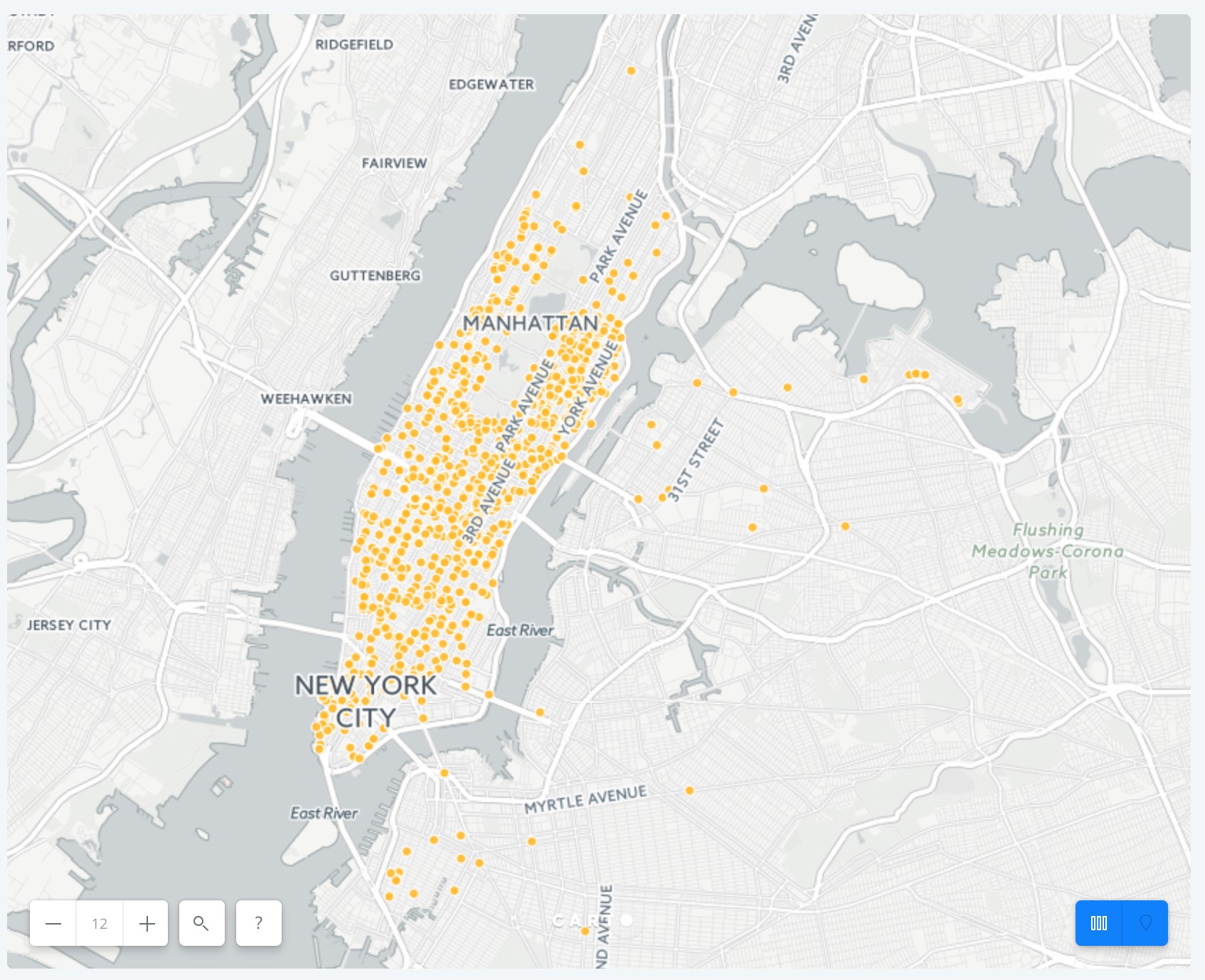


Figure 12 NYC pickup points for optimal trips

We then fetch the pickup points of those optimal trips and plot them on the map. The following is the illustration for it. Most of them are in the Manhattan area.

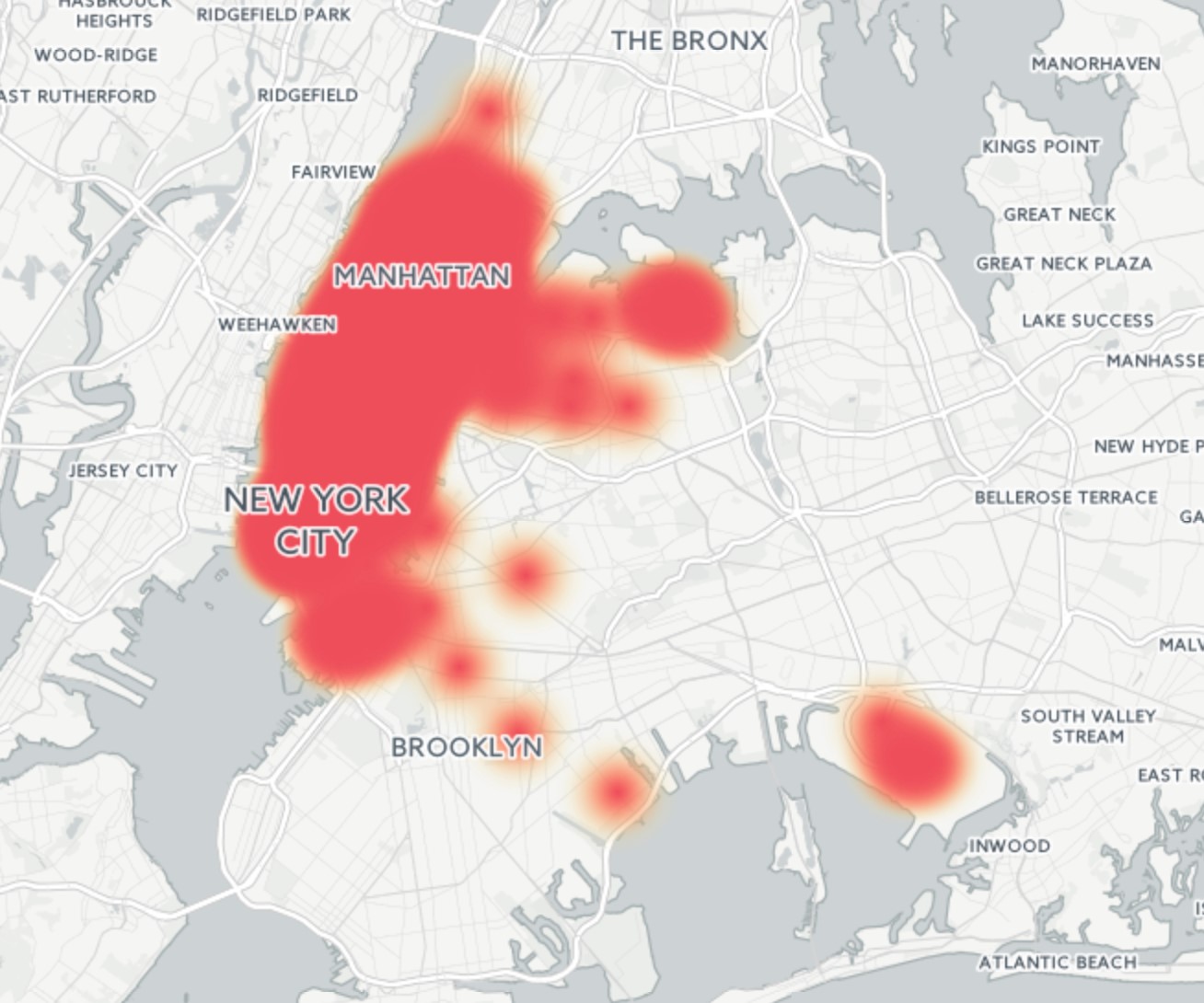
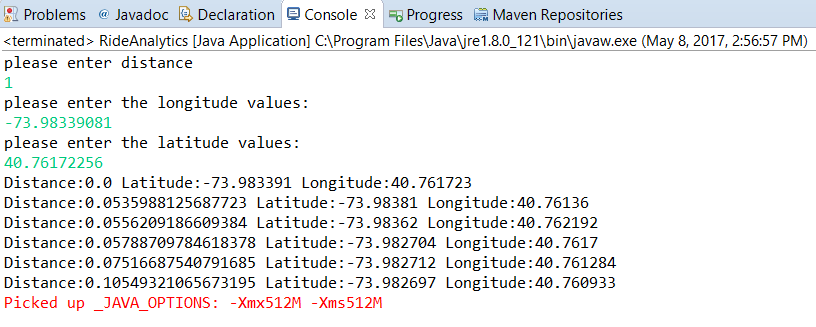


Figure 13 NYC pickup points heat map

A heat-map is generated below for the pickup points in NYC. Red means high pick up density and yellow-orange means low pickup density. It can be seen that almost entire NYC has high pickup density.

**Result of Top 5 Nearest Neighbors at given point:**

****

**7. Experimental Evaluation**

From the above-mentioned methods and techniques, we can see that the performance when dealing with Big Data can be improved to better extent. The speed of execution of analytics using Apache Spark is better than other Big Data platforms like Hadoop. Also, they require less amount of code to be written to perform multiple jobs simultaneously. Also for finding nearest neighbors around certain distance from the given point is calculated by using R-tree data structure. It reduces the storage space by providing efficient storage mechanisms as well as incorporating pruning strategies decreases the result set for computation. This decreases the time to execute and provide better performance of the system. The result distance provided in this method is varies approximately 0.3% to the actual distance. Also, the encoding methods and algorithms provided by Google distance matrix API are very helpful in finding the network distance in a very less time. For finding the Continuous nearest neighbors defined algorithms in the above section Upper Bound Algorithm helps in eliminating unnecessary computation of the nearest neighbors by using VN3 approach. This speed up the process of finding distances by finding KNNs only to query objects that satisfy the threshold condition. In this way we have improved our performance in doing analysis on Ride data on NYC.

**8. Future Work**

In future, we can introduce many features to this application apart from analysis such as finding places which would be perfect to establish business and what type of business. The ability to find routes which are good for going alone and finding routes which are good for carpooling. We can also provide visualization for heated maps and route visualization. Profiting the taxi drivers by allowing them to find places where they can get many customers at different times in a day. Giving details of places which are best for going in a personal vehicle and places which are best for using public transport, which allows you save on your expenditure. The taxi business can profit from using this analysis by providing more number of vehicles to places where there is more requirement and reducing in places where there is less requirement for taxi.

**9. References**

[1] NYC Taxi & Limousine Commission. <http://www.nyc.gov/html/tlc/html/about/about.shtml>

[2] Yufei Tao, Dimitris Papadias, Qiongmao Shen. “Continuous Nearest Neighbor Search”. Proceedings of the 28th VLDB conference, Hong Kong, China, 2002.

[3] Mohammad R. Kolahdouzan and Cyrus Shahabi. “Continuous K Nearest Neighbor Queries in Spatial Network Databases”. Proceedings of the Second Workshop on Spatio-Temporal Database Management (STDBM'04), Toronto, Canada, August 30th, 2004.

[4] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point". In SSTD 2001, Redondo Beach, CA, USA.