Shortest Path Algorithm Based On Historical Traffic Data

Zarraf Ahmed
Id: 011 123 041

Injamamul Moin Khan
Id: 011 131 164

Shaibal Das
Id: 011 131 143

Sonia Kamal
Id: 011 142 055

A thesis in the Department of Computer Science and Engineering presented
In partial fulfillment of the requirements for the Degree of
Bachelor of Science in Computer Science and Engineering

United International University
Dhaka, Bangladesh

Under the supervision of
Prof. Dr. Mohammad Nurul Huda
Professor & Coordinator - MSCSE
Department of Computer Science and Engineering

July 2018

Declaration

We are certifying that this thesis Shortest Path Algorithm Based On Historical Traffic Data is our own work, based on our personal study and research under the supervision of Prof. Dr. Mohammad Nurul Huda, Professor & Coordinator - MSCSE, Department of Computer Science and Engineering, United International University, Bangladesh. I confirm that:

- This work was done wholly or mainly while in candidature for a BSc degree at United International University.
- This thesis has not previously been submitted for a degree at United International University, this has been clearly stated.
- We have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication.

_____

Zarraf Ahmed
Id: 011 123 041
Computer Science and Engineering

_____

Injamamul Moin Khan
Id: 011 131 164
Computer Science and Engineering

_____

Shaibal Das
Id: 011 131 143
Computer Science and Engineering

_____

Sonia Kamal
Id: 011 142 055
Computer Science and Engineering

Certificate

I do hereby declare that the research works embodied in this thesis entitled "**Shortest Path Algorithm Based On Historical Traffic Data**" is the outcome of an original work Under my supervision.

I further certify that the dissertation meets the requirements and the standard for the degree of BSc in Computer Science and Engineering.

_____

Prof. Dr. Mohammad Nurul Huda
Professor & Coordinator - MSCSE,
Department of Computer Science and Engineering,
United International University,
Dhaka Bangladesh.

Abstract

Traffic jam is a common situation in the capital cities and big towns especially in rising countries such as Bangladesh. In this condition, people are missing their valuable time of their busy platform by getting stacked in weighty traffic. Moreover, any faithful traffic blocking avoidance or prediction tool for providing real time traffic jam data and path selection is not available in Bangladesh. Shortest path traffic data is a challenging assignment that can be used for traffic predicting and improving traffic flow. In this paper, we offered an intelligent system using special types of cost function, which will fix optimal paths of lowest travel budget with duration and distance considering both shortest path and historical traffic data which was offer different time of structures in 24 hours. We have compared all possible paths along with the shortest path to fix the optimal path.

Acknowledgement

We would like to thank our thesis supervisor Prof. Dr. Mohammad Nurul Huda, Professor & Coordinator - MSCSE, Department of Computer Science and Engineering, United International University, Dhaka, Bangladesh. We feel thankful to our honorable supervisor for helping us. Deep knowledge and serious interest of our supervisor in the field of Data Mining prejudiced us to carry out our thesis.

Finally, we would like to thank our parents for their much consciousness and supports to carry our thesis properly.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter1

Introduction

Traffic management system in the field of road transportation is an important part of intelligent transportation system. Total number of registered vehicles in Bangladesh is approximately 4 million whereas effective traffic management system, strategic preparation and direction of the transport structure facilities are not up to the mark. In intelligent transport system, traffic movement data analysis is becoming very important issue. In road network system, traffic congestion is a regular occurrence in situation of Bangladesh especially in the capital city; mainly because of huge number of vehicles, riding at the same time, e.g. busy hours (7:00 am-12:00 pm and 4:00 pm- 10:00 pm). Repairs of structure of the roads or increase of roads for decreasing traffic jam are always time consuming, expensive and difficult job. Thus, vehicle drivers may use the traffic blocking prevention mechanism to save the time and cost by dodging traffic jam. Shortest path computation is an important function in modern car navigation systems. This function helps a driver to figure out the best route from his current position to destination. Typically, the shortest path is computed by offline data presorted in the navigation systems and the weight (travel time) of the road edges is estimated by the road distance or historical data. Real-time adaptive traffic control system displays road traffic and adjusts the traffic signal in real-time. By using advance information technology, it is very easy to notice and record how long it takes a vehicle to move from one place to another place. However, providing such kind of mechanism would not be very easy job because this will consider not only the shortest path from source to destination, but also the traffic situations, which are difficult to predict and change dynamically. Therefore, a fixed budget model and traditional shortest path finding algorithms such as Dijkstra, Bellman Ford, Floyd warshall, A star search etc. are not computationally capable to provide such kind of solutions. However, some meta-heuristic methods such as genetic algorithm (GA) is available that can provide such tools considering the real-time issues as they have adaptive nature with the altering environment.

1.1 Motivation

Traffic congestion is a kind of problem that we do not want to face. This is a worldwide problem and the situation is getting too hard to manage. But, we have to face it every day. People have been trying different approaches to solve this problem but that not seems too good enough. One recent approaches that has been introduced by the data scientists to contribute solve this problem is through mining traffic data. Traffic data mining is a new concept that has been introduced by the data scientists. The basic idea is to get information from the road traffic through some sensors and use that information to get the situation improved. Lately, there has been lots of paper that studied to utilize this traffic data mining

to make some effective works. We have studied some of those papers and got inspired to work on this field.

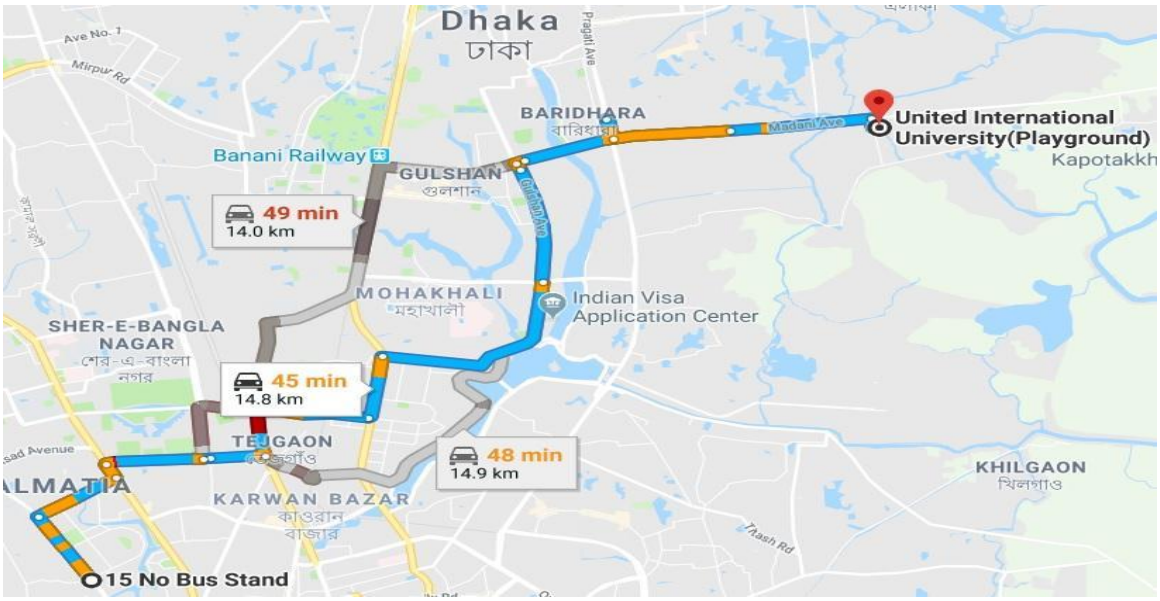

Figure 1.1: Shortest Path from The Road Traffic.

1.2 Machine Learning

Machine learning is one types of artificial intelligence that helps computer to learn short of any open program. It is one of the subsection of computer science. Machine learning is a kind of similar to data mining. Machine learning discovers the study of algorithms that can learn from and make forecasts on data. Machine learning algorithms are often classified as being supervised or unsupervised. They learn from previous calculations to produce consistent, repeatable decisions. Machine learning is used in Web search, spam filters, recommender systems, fraud detection, drug design, and many other applications. Machine learning is given so much importance because it helps in forecasting performance and recognizing patterns that humans with their partial capability can't predict. Things like rising volumes and varieties of available data, calculation processing that is economical and more powerful, and rational data storage. All of these things mean it's possible to quickly and mechanically make models that can analyze larger, more multifarious data and provide faster, more precise results even on a very big calculation.

1.3 Big Data

Traffic data mining in most of the time involves dealing with 'Big Data'. The term big data is a very well-known perception which often referred to as 3 V's such as Volume, Velocity, and Variety. One of the most impressive areas where this concept is taking place is in the area of machine learning. The general idea is instead of instructing the computer what to

do, we are simply going to throw data at the problem and tell the computer to figure out for itself. The tracking of data or information from the real world is denoted to as volume. Velocity is the availability of the dataset with respect to the real-time. And when a dataset consists different types of data types for example, Numeric, Date Time, image, audio, video etc. then that specifies the variety within the dataset. The dataset we used was collect from a historical dataset. It had a large volume and there were different types of data types like we already declared. So, we can say that our dataset was a big data.

1.4 Data Mining

Data mining studies data from different observations and précises it into useful info. It is basically a calculation process of finding patterns in big data sets. The goal of data mining process is gaining information from a data set and converts it into knowledge. User's studied the data from different phase, can classify and précised it. The knowledge extraction can be used in several places like Market Analysis, Production Control, Science Exploration and many more things. Data can be any numbers, or text that can be treated by a computer. Data is being increasing day by day. Whatever we are doing whether liking a picture on Facebook or sending an e-mail, it creates data. That's how we have these days a lot amount of data. Data mining is the method of finding unknown information and designs from a vast database. Mainly in data mining we try to abstract patterns or information from a dataset and build a reasonable structure by converting the dataset. It is a test of data mining to select the planning for working with dataset because it differs with dataset.

1.5 Traffic Data Mining

There were several papers that studied traffic data. The main motivation was to give a prediction of shortest path about the circumstances when traffic delays. Some paper analyzed understanding traffic flow congestion in real time. Now, all of these papers can be used to travel delays. There was a paper that presented optimal path computation by using traffic data. We often use the GPS to figure out the shortest path of our destination. Most of the time working with traffic data involves dealing with big data as the dataset is often collected in real time with short break. We will discuss in detail about big data in chapter 4 in our thesis.

1.6 Thesis Contributions

The contribution includes a couple of developments which are summarized below,

- At first, we select an area in Dhaka city which is Dhanmondi and we draw a manual map for our work.
- Also that, we build a Python code for collecting traffic data, which was collected data by using google direction matrix API key and make a dataset.

- We build an another python code for clustering data simulating traffic condition. It is capable of creating sub-datasets from the main dataset and showing those sub-datasets with regard to Date, Day, Time, Source, Destination, Edge, Distance, Duration(original), Distance(current), Traffic condition.
- Also that sub-dataset can be saved in different formats such as, comma-separated values (CSV) file that stores data in plain text, Attribute-Relation File Format (ARFF) which is an ASCII text file that describes a list of instances sharing a set of attributes and Excel format. Afterwards these files can be used to take the research further by using different other tools.
- We create all possible shortest path using many types of algorithm and also create graph and all paths which are possible.
- Then, we compared all possible path with shortest path and predict the best path.
- We have also applied several regression models on some of our sub-dataset to understand and figure out the relationship among the attributes presented within the dataset. This kind of work will be very significant for the upcoming future to generate a prediction for the road traffic.

1.7 Organization of the Thesis

This thesis organized as follows:
Chapter 2 describes the related papers that we have studied, which actually build up the importance of mining traffic data. The chapter presented their work in different sections.

Chapter 3 provides a brief introduction of shortest path algorithm and describes different types of regression models that have been used in machine learning to mine information. Specially the chapter focuses on those algorithms which we have applied in our research and can be useful in the related field.

Chapter 4 describes the information and structure of our collected dataset. Here, we've also discussed about data visualization and showed some graphical representation of the data.

Chapter 5 aims to describe the experimental results that we've found by using some of our sub-dataset. The results that we've found demonstrated through some tables to understand the result comparison.

Chapter 6 presents the conclusion, summarizes the thesis contribution and discusses the future work.

Chapter 2

Related Work

In current years, various types of data mining methods are being applied for mining historical traffic data. Many researchers are connecting themselves to the contribution of adaptive fastest path, forecasting traffic flow on the road for development of the modern world. In this chapter we describe the related papers that we have studied, which actually build up the importance of mining traffic data. The chapter presented their work in different sections. From these papers, we get so much help to research our work.

2.1 Traffic Flow Comprehension

Ashratuz Zavin and Adnan Sharif [1] they have proposed an intelligent system with a cost function using Ant Colony Optimization (ACO) and a meta-heuristic method, which will calculate finest paths of lowest travel cost considering both as historic and real time traffic data and different time gaps of a day. It will also vigorously re-route the path in case of weighty blocking through travel time for evading unusual situations. Investigational results show that the designed algorithm of the planned system performs accordingly with reliable real time traffic prediction and its proposed routes provide improved triangulation and may save cherished time. Thammasak Thianniwet and Satidchoke Phosaard [2] examined an alternative way to mechanically classify the road traffic congestion levels that was highly stable with road user's decisions. The data gathering applied a GPS device, a webcam, and a survey. Initially J48 algorithm used to learn movement patterns of a vehicle and lastly the user's decision. They separated the obstruct level in three groups (light, heavy, and jam) to relate their classifier with user's decision. Jalil Kianfar and Praveen Edara [3] inspects the presentation of hierarchical clustering, K-means clustering, and general mixture model clustering for isolating traffic data to free flow and jammed flow rules. Brian et al. [4] in this examination they presented and tried to develop traffic capacity forecasting model for two sites on Northern Virginia's Capital Beltway. For forecasting 15 min into the forthcoming, they examined four models for the freeway traffic flow forecasting problem. Jaros law Rzeszotko and SinhHoa Nguyen [5] concentrated on forecasting average car velocities on selected streets from immediate velocities of a selected group of cars driving through the city which was the third problem of a competition in June 2010 by Tom-Tom. The spatial-temporal personality of the data available in the competition raises a range of interesting sub-problems, from which the most immediate one is the problem of finding a cars route through a street graph. An operational solution is discussed in the paper, consisting of mutual usage of the well-known R-Tree data structure and a heuristic established by the authors for establishing the lane through which the car is driving. This was demonstrated as a regression problem and solved using a neural network trained with the tough back broadcast algorithm. Furthermore, tests were carried out to measure the error of different neural network designs when execution this task. To select the absolute

best network architecture, an experiment was carried out relating neural networks with different sets of features as inputs. And the result of rms error is 9.1563.

2.2 Adaptive Path Computation

A new method of mining traffic data to calculate an adaptive reckless path on a road system has introduced [6] on 2007. A road system separating algorithm was presented in this paper. The paths calculated by their algorithm are not only fast given a set of dynamic situations but also reflect noticed driving partialities. Jiawei Han, Zhenhui Li, and Luan Tang [7] focused on mainly three parts: (1) stirring object pattern mining, (2) route data mining, and (3) road traffic data mining. The main thing of this paper is shortest path calculation. Another goal is forecasting on moving vehicle endpoint. Bayes classification and recurrent pattern improve for effectiveness. Recurrent patterns one of the basic patterns that notice frequently visited routes. The test of common pattern is expected locations and evolution time. In the second part, route data mining, we focus on path clustering, classification and outlier detection. For path clustering, many high dimensional data clustering approaches can be easily modified if we give each timestamp as one length.

Chapter 3

Data Visualization

For our research work we needed a dataset. In this chapter we discuss about our collected dataset. Collecting data was not an easy task for us because no governmental or non-governmental organizations will not give us the traffic data without any administrative permission. So we collect the data with the help of the google map API. We took the data of seven days from 6:00am to 11:45pm with different timestamp with the interval of fifteen minutes. After collecting data, we give some label to the data with the help of clustering algorithm and finally classify them. On the other hand, we create a graph from the dataset and perform several algorithms to find the shortest paths and all possible paths with particular source and destination.

3.1 Initial Step of Collecting Data

When we start working about this topic, we find out many peoples are working on other country's traffic data, but no one work on Bangladeshi Traffic Data as detail. In this situation we are decided to collect Bangladeshi Traffic data, but it's a challenging task for us. Then we contact BRTA for Dhaka city traffic data but they unable to provide because they can't provide any data without purpose of Government work. Then we thinking about other process which was describe about next section.

3.2 Point Out Latitude Longitude On Manual Map

At first we decided an area in Dhaka city which was Dhanmondi and we starting draw a manual Traffic Map over Dhanmondi. In this process google map helping us. Our Traffic Map is similar like google Map.

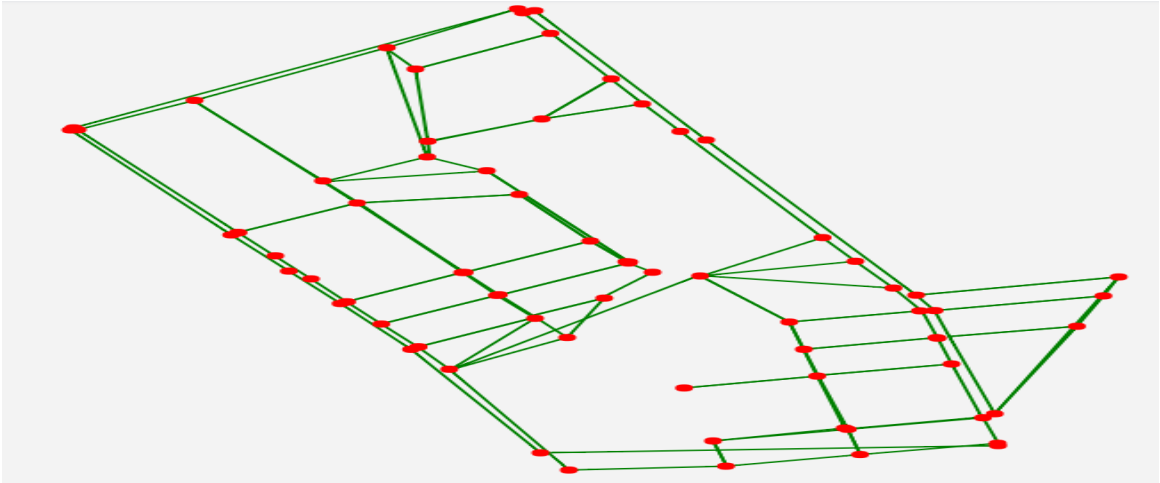

Figure 3.2: Sample of Manual Map.

After building the manual Map, we find out and numbering all the junction points, all Nodes and all Edges on the manual map. Then we create a CSV file on MS Excel and create a table, the attributes are:

Edge	Source Latitude	Source Longitude	Destination Latitude	Destination Longitude

Then we Browse google map and select pin point per junction's point and collect their Latitude and Longitude and entry on CSV file.

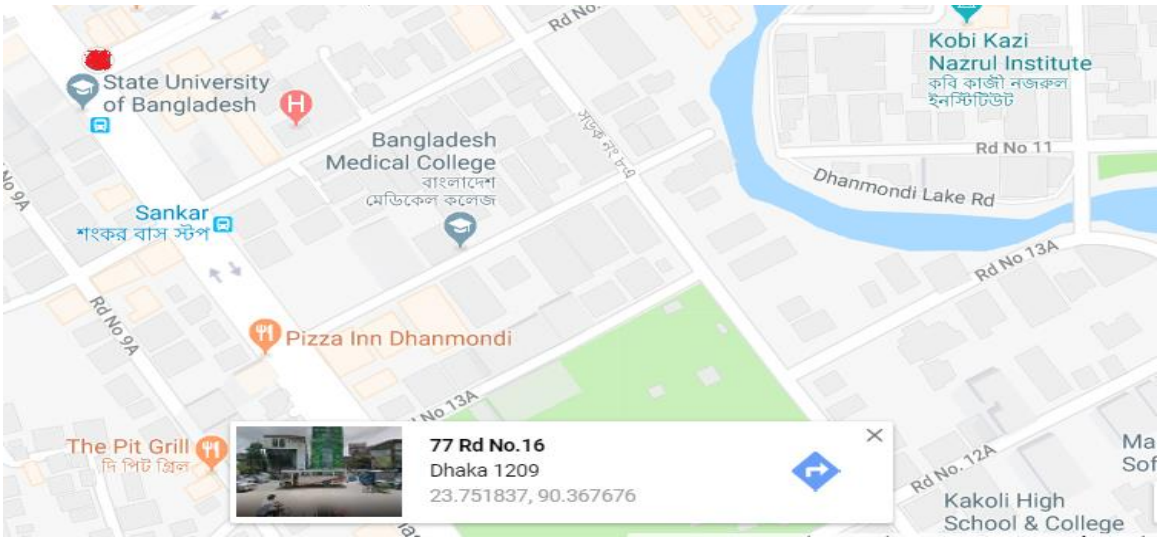

Figure 3.2.1: Pin Point's Latitude and Longitude.

In this process we collect total 75 nodes and 126 edges source to destination Latitude and Longitude on Dhanmondi area. Here one Edge means one Junction point to other Junction point. **Example:** Dhanmondi 27 Bus Stop to Dhanmondi 15 Bus Stop.

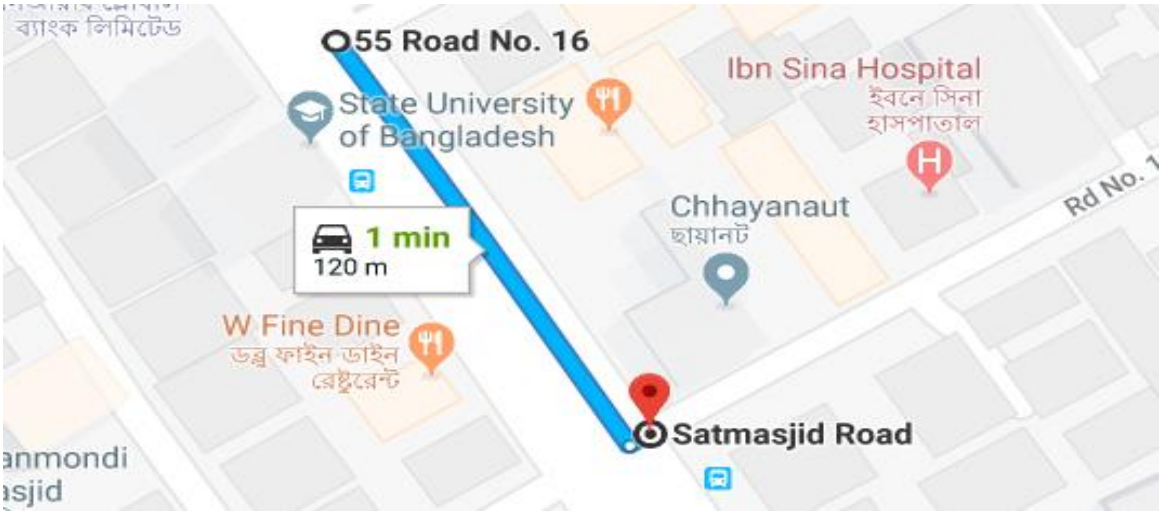

Figure 3.2.2: Edge Source to Destination.

After finishing the task, we create this types of Data set which sample is shown below:

	A	B	C	D	E
1	Edge	Source Latitude	Source Longitude	Destination Latitude	Destination Longitude
2	E1	23.751815	90.367631	23.756405	90.375087
3	E2	23.756405	90.375087	23.754901	90.372881
4	E3	23.754901	90.372881	23.75286	90.369638
5	E4	23.75286	90.369638	23.751721	90.367665

Figure 3.2.3: Sample Data Set.

3.3 Introduce Google Map API

A Google Maps API (Application Programming Interface) is a set of methods and tools that can be used for building software applications. Google Maps API allows you to display maps on your web site Map Data. Google explore Three types of products which was:

- Maps
- Routes
- Places

In our Thesis we consider Route because it's Help users find the best way to get from A to Z with full data and real-time traffic. In Google Map Platform Routes are three types of features which was:

- **Directions**- Get directions for transit, biking, driving, and walking. Calculate current or future travel times based on real-time traffic.
- **Distance Matrix-** Deliver travel times and distances for one or more locations.
- **Roads-** Create precise itineraries by determining the route a vehicle has traveled and the nearest roads along each point of the vehicle's journey.

3.4 How to Collect Data by Google Map API

- At first we go to console.developers.google.com then we create a project.

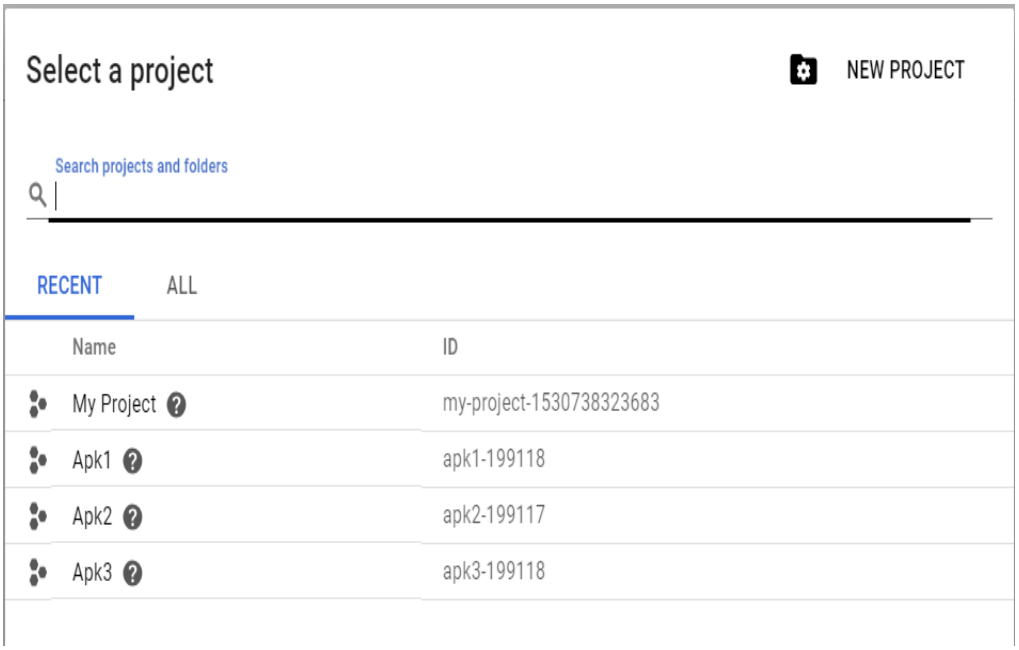

Figure 3.4: Project Creation.

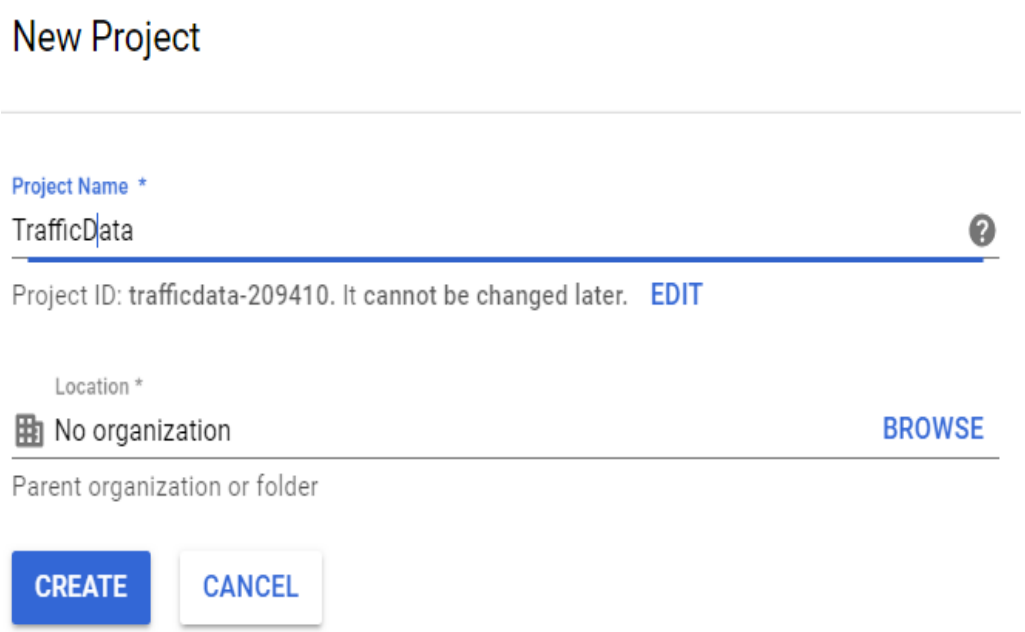

Figure 3.4.1: Project Name Selection.

- In the library section search for Distance Matrix API and enable it. After that the API key is ready for use.
- Go to credentials and from create credentials select API key.

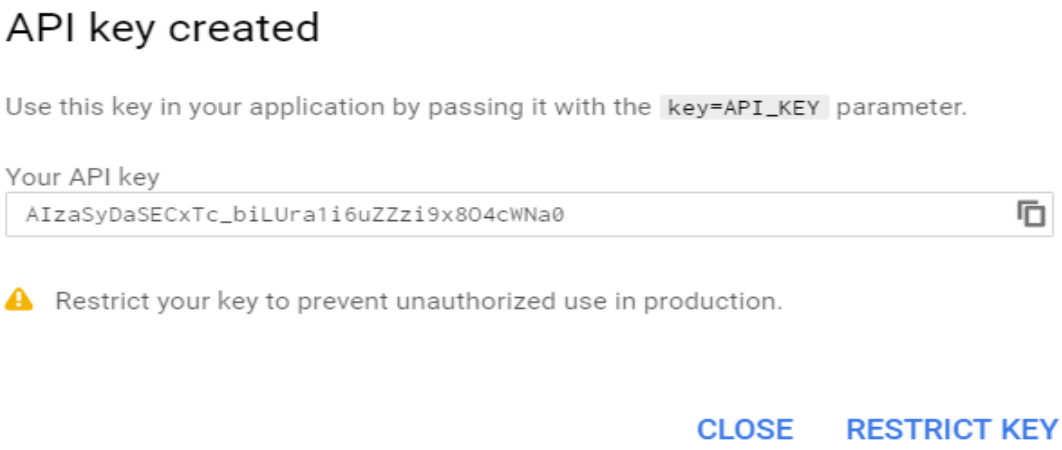

Figure 3.4.2: Create Credentials for Select API Key.

- Then test the API key with any source and destination with this URL.

https://maps.googleapis.com/maps/api/distancematrix/json?origins=Boston,MA|Charlestown,MA&destinations=Lexington,MA|Concord,MA&mode=driving&traffic_model=best_guess&departure_time=now&key=YOUR_API_KEY

As a results it will show you a JSON


```
{
   "destination_addresses" : [ "48 Rd No. 6A, Dhaka 1205, Bangladesh" ],
   "origin_addresses" : [ "63 Rd No. 7A, Dhaka 1205, Bangladesh" ],
   "rows" : [
      {
         "elements" : [
            {
               "distance" : {
                  "text" : "354 ft",
                  "value" : 108
               },
               "duration" : {
                  "text" : "1 min",
                  "value" : 21
               },
               "duration_in_traffic" : {
                  "text" : "1 min",
                  "value" : 21
               },
               "status" : "OK"
            }
         ]
      }
   ],
   "status" : "OK"
}
```


- If the JSON status is OK then it's valid.

- Parse the JSON and save “destination_addresses”, "origin_addresses", "distance", "duration", "duration_in_traffic" in a CSV file. Also create “Day”, ”Date”, “Time” columns in the CSV file.
- So each time we hit the URL, the data will be stored in the csv file.

	A	B	C	D	E	F	G	H	I
1	Date	Day	Time	Source	Destination	Edge	Distance(meter)	Duration(Original)sec	Duration(Current)sec
2	Mar 23 2018	Fri	6:00am	Dhaka 120	Dhaka 1205, B	E1	915	180	98
3	Mar 23 2018	Fri	6:00am	haka 1205	I, Dhaka 1209,	E2	333	115	79
4	Mar 23 2018	Fri	6:00am	Dhaka 120	6, Dhaka 1209,	E3	401	76	41
5	Mar 23 2018	Fri	6:00am	Dhaka 120	tmasjid Road, I	E4	237	53	23
6	Mar 23 2018	Fri	6:00am	nasjid Road	A, Dhaka 1205	E5	524	109	55

Figure 3.4.3**:** Sample Data Stored.

- We have collected data for seven days and 72 different time intervals from 6:00am to 11:45pm. So total number of instances become 63,503.

3.5 Data Clustering

Let’s see the Attributes types:

Attributes	Type
Date	Nominal
Day	Nominal
Time	Nominal
Source	Nominal
Destination	Nominal
Distance(meter)	Numeric
Duration(Original)sec	Numeric
Duration(Current)sec	Numeric
Edge	Nominal

Table 3.5: Attributes Types.

3.5.1 Data Pre-processing

In data pre-processing phase we have filled the missing values with different imputation technics such as mean, mode, most occurred values and many others. Special type's imputation technics have been used to impute numerical values which is called KNN imputation.

3.5.2 K-means Clustering Implementation

Considering the "Duration(Current)sec" as the main attributes for our clustering which is a numeric attribute, we have used the K-means++ clustering in order to label all instances of the datasets. Considering four clusters Low, Medium, High and Extreme, we have used this formula to evaluate the initial centroid values for each edge.

- *Centroid_value = []*
- *Max_edge = find maximum value*
- *Quartile = Max_edge / Number_of_cluster*
- *Centroid_value.append(Quartile)*
- *For I =1 to Number_of_cluster-1*

  Centroid_value.append(Quartile(I+1))*

After selecting the initial centroid value for four clusters, we simulate the code with maximum iteration of 30 times and for distance measure we have used the Euclidian distance. But it takes 13 iterations to reach the finalize the clustering. After clustering our datasets looks like this.

A	B	C	D	E	F	G	H	I	J
Date	Day	Time	Source	Destination	Edge	Distance(meter)	Duration(Original)sec	Duration(Current)sec	Traffic
Mar 23 2018	Fri	6:00 AM	, Dhaka 1209,	Dhaka 1205, E	E1	915	180	98	Low
Mar 23 2018	Fri	6:00 AM	Dhaka 1205, E,	Dhaka 1209,	E2	333	115	79	Low
Mar 23 2018	Fri	6:00 AM	, Dhaka 1209,)	, Dhaka 1209,	E3	401	76	41	Low
Mar 23 2018	Fri	6:00 AM	), Dhaka 1209,	masjid Road,	E4	237	53	23	Low
Mar 23 2018	Fri	6:00 AM	masjid Road,	A, Dhaka 1205	E5	524	109	55	Low

Figure 3.5.2: Clustering Dataset.

Attributes Table:

Attributes	Values
Date	Mar 24 2018, Mar 25 2018, …., Mar 30 2018.
Day	Fri, Sat, Sun, ……. Thu.
Time	6:00am, 6:15am, ……,11:45pm.
Source (Address)	77 Rd No.16, Dhaka 1209, Bangladesh, ……..
Source (Node)	n1, n2, n3, ……., n75
Destination (Address)	10 Old 27, Dhaka 1205, Bangladesh, ……
Destination (Node)	n1, n2, n3, ……., n75
Edge	e1, e2, e3, ……..…., e126
Distance	915 m, 534 m, ……..….
Duration (Original)	180 s, 220s, ……..….
Traffic	Low, Medium, High, Extreme.

Table 3.5.2: Attribute table with sample Data.

3.6 Visualization

We have followed several methods to visualize our data. We have plotted bar chart, scatter plot, histogram for our attributes or features.

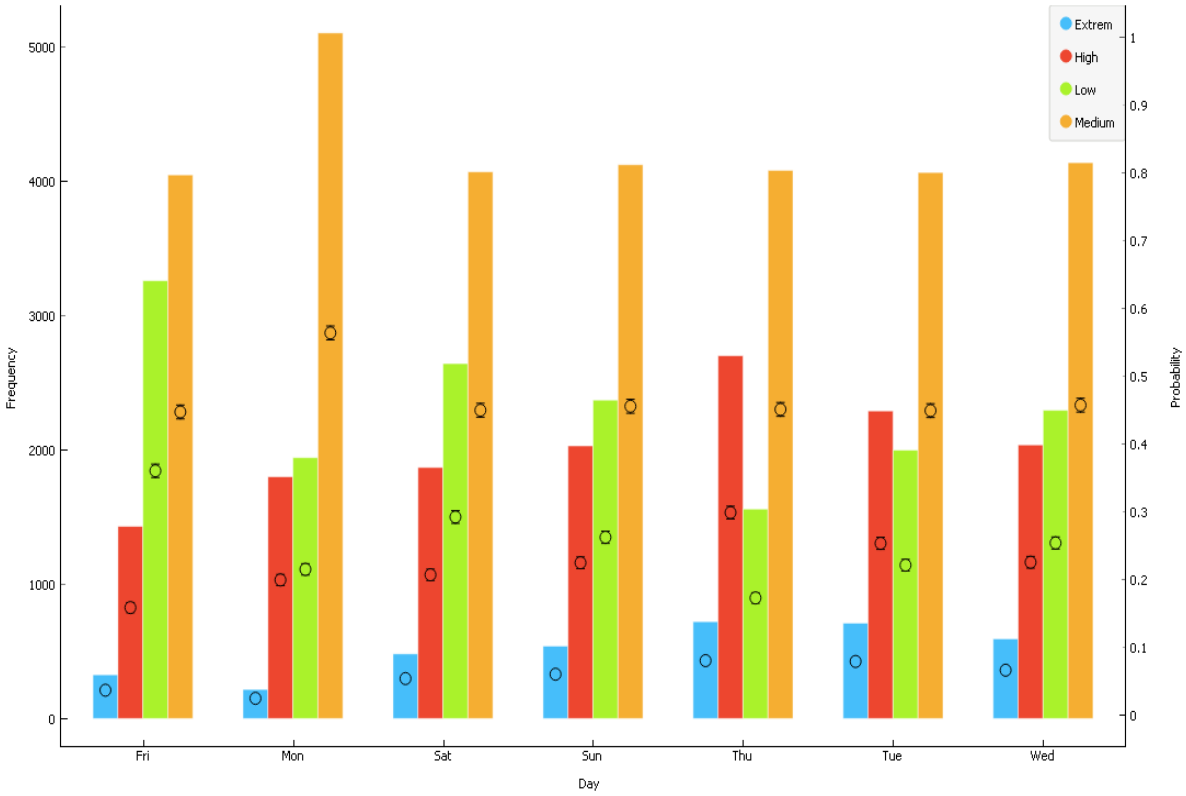

Figure 3.6: Days vs traffic bar chart with distribution.

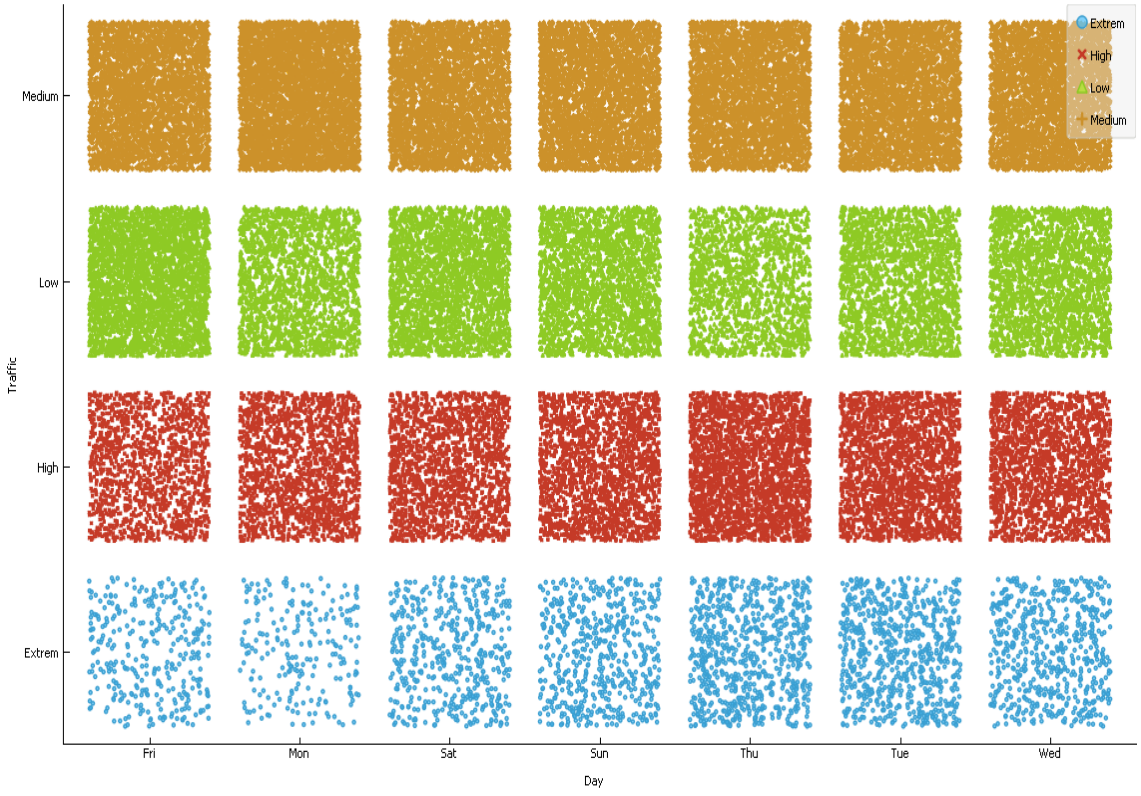

Figure 3.6.1: Days vs traffic scatter plot.

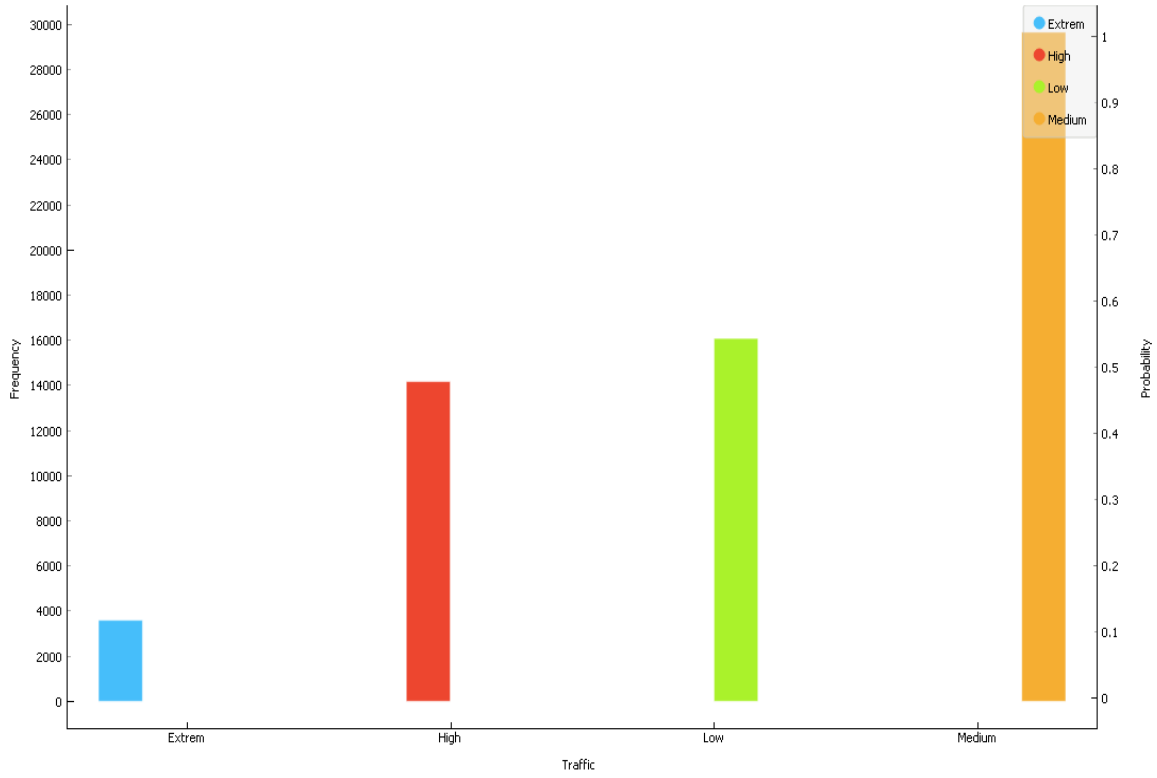

Figure 3.6.2: Traffic bar chart with frequency and distribution.

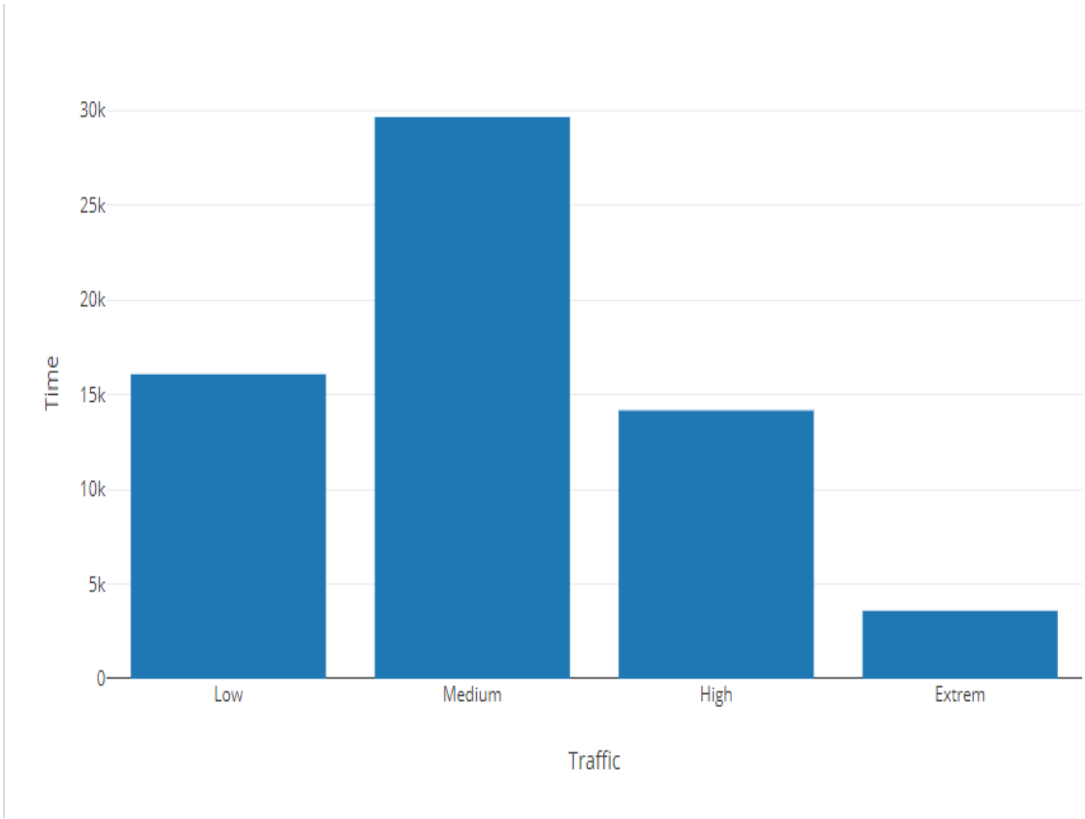

Figure 3.6.3: Traffic vs time histogram.

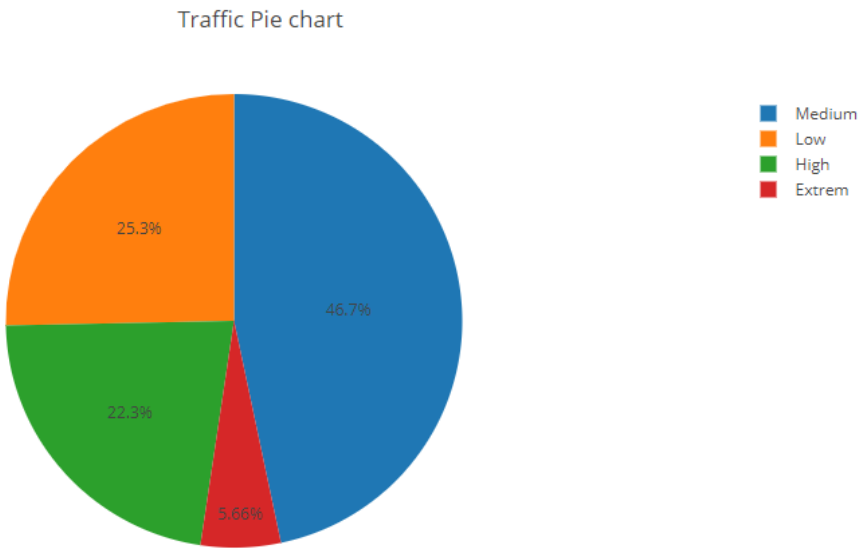

Figure 3.6.4: Traffic pie chart.

3.7 Graph Creation and Find Paths

From manual map that we draw earlier in section: 4.2, we create a graph with 75 nodes and 126 directed edges. We put the distances as the weight for each edges. With the help of *networkx* library for python we have implemented all shortest path algorithms and also find all paths from source to destination.

3.7.1 Implementing Modified Genetic Algorithm

As we know from above description of genetic algorithm from section 3.5 that it has five different functions.

- Initial population
- Fitness function
- Selection
- Crossover
- Mutation

For finding the shortest path we need some modification for specific functions.
Initial population – We create some paths from source to destination and randomly select some paths as the initial population.
Fitness function – For measuring the fitness for each path we have calculate the total length of each path. The path with the lowest length is the fittest path.

$$F(x) = \sum_{i=1}^{n}(length(\mathrm{ni}, ni + 1))$$

Where n = length of the paths, n = node

Selection – In selection we use the fitness function to calculate the fitness of each paths in the initial population and select two best fitted path as the parent path.
Crossover – The biggest modification occurs in crossover function. It has some steps. From two parent paths find the intersecting nodes. This intersecting nodes indicate the cut point for each chromosome. Then perform one-point crossover or multipoint crossover or uniform crossover with the intersecting nodes and generate child's.

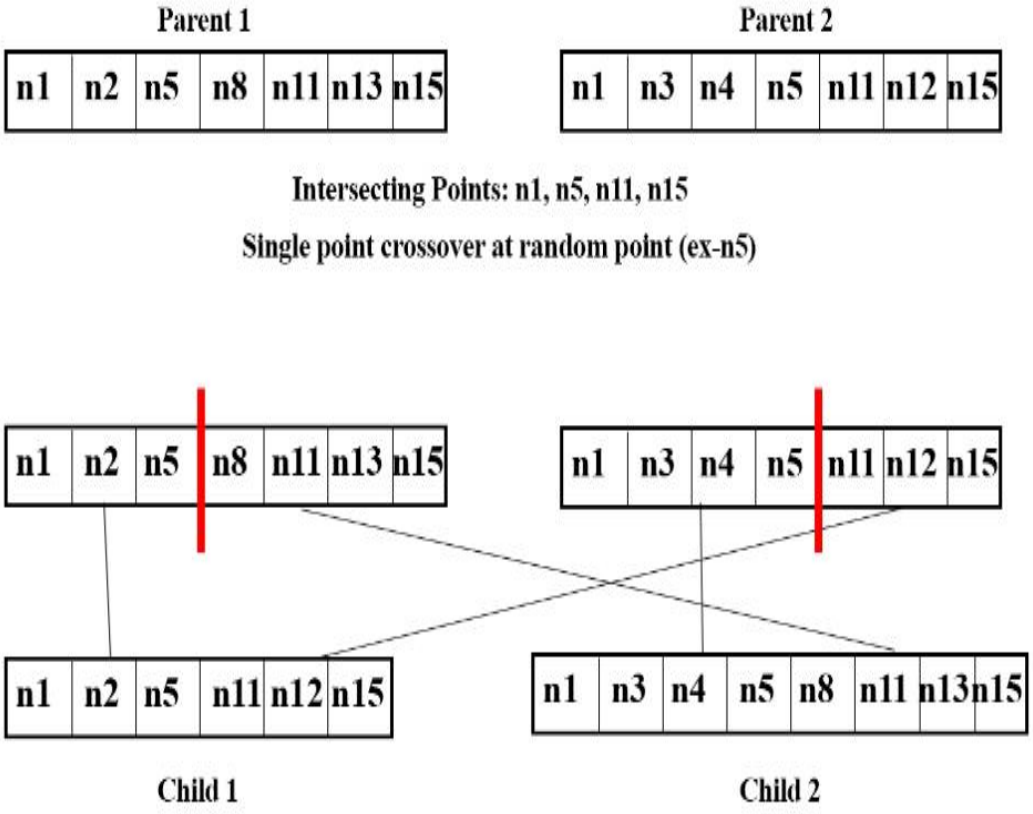

Figure 3.7.1: Crossover Process.

Mutation – For mutation function we have checked if there is any particular cycle's in the child's. If we found any cycle, then remove it.

3.7.2 Shortest Path Result

As we said earlier our graph consists of 75 nodes (n1, n2, n3, n4, ……, n75) along with 126 weighted edges (e1, e2, e3, e4, ……., e126). In order to find shortest path, we have used Dijkstra, Bellman-ford, Floyed-warshall, A* and Genetic algorithms. In genetic algorithm we have used 50 maximum iterations. Considering 'n2' as the source and 'n75' as destination the result are giver below.

```
--------Dijkstra algorithms--------
---Path: ['n2', 'n3', 'n44', 'n45', 'n75']
---Length: 825

---------Bellman-ford algorithms---------
---Path: ['n2', 'n3', 'n44', 'n45', 'n75']
---Length: 825

---------Floyed-warshall algorithms----------
---Path: ['n2', 'n3', 'n44', 'n45', 'n75']
---Length: 4788

---------A star algorithms--------------------
---Path: ['n2', 'n3', 'n44', 'n45', 'n75']
---Length: 825

---------Genetic algorithms(50 iter)--------------------
---Path: ['n2', 'n3', 'n44', 'n46', 'n47', 'n48', 'n49', 'n45', 'n75']
---Length: 1326
```

Figure 3.7.2: Shortest Path Result.

3.8 Dataset Classification

Classification is a data mining technique that assigns categories to a collection of data in order to aid in more accurate predictions and analysis. We have discussed about classifications earlier in section 1.2 and section 1.4.

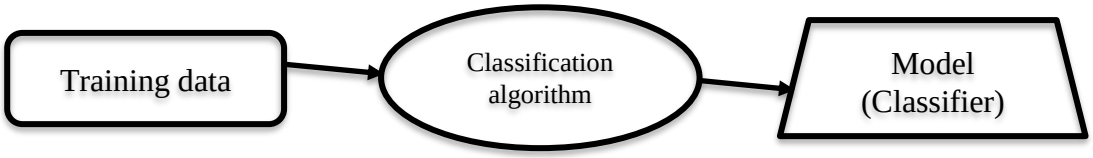

Figure 3.8: Dataset Classification Process.

3.8.1 Data Pre-processing

In data pre-processing phase we have filled the missing values with different imputation technics such as mean, mode, most occurrence and many others. Special type's imputation technics have been used to impute numerical values which is called KNN imputation. We removed the columns which are less important such as "destination_addresses", "origin_addresses", "distance", "duration", "date" etc.

3.8.2 Building Model

From different classification algorithm we have used:

- Decision Tree (J48)
- Naïve Bayes
- Adaboost
- Bagging

Pythons scikit learn library and weka helps us a lot to build the models. We have used six types of test data:

- Use whole training dataset.
- 10 fold cross-validation.
- 20 fold cross-validation.
- 50% split training data.
- 75% split training data.
- Providing test data.

Using	Accuracy	TP rate	FP rate	Precision	Recall	F measure	MCC
Using training set	75.7826%	0.758	0.142	.754	0.758	0.744	0.626
Cross validation (10fold)	70.0192%	.700	.174	0.693	.700	.690	.534
Cross validation (20fold)	70.3908%	0.704	0.173	0.697	0.704	0.693	0.539
Percentage split (50%)	70.1027%	.701	0.186	.698	0.701	.691	.532
Percentage split (75%)	70.4334%	.704	.184	.701	0.704	0.695	0.537

Table 3.8.2: Classifier J48.

Using	Accuracy	TP rate	FP rate	Precision	Recall	F measure	MCC
Using training set	71.5231%	0.715	0.178	0.709	.715	0.704	.556
Cross validation (10fold)	71.4175%	0.714	0.179	.708	.714	.702	.554
Cross validation (20fold)	71.3672%	.714	.179	.708	.714	.702	.554
Percentage split (50%)	71.4475%	0.714	0.177	0.708	.714	.704	.554
Percentage split (75%)	71.3593%	.714	.178	.707	.714	.703	.553

Table 3.8.2.1: Classifier Naïve Bayes.

Using	Accuracy	TP rate	FP rate	Precision	Recall	F measure	MCC
Using training set	81.7822 %	0.818	0.111	0.818	0.818	0.813	0.720
Cross validation (10fold)	74.3843 %	0.744	0.150	0.738	0.744	0.737	0.605
Cross validation (20fold)	74.6284 %	0.746	0.149	0.741	0.746	0.740	0.608
Percentage split (50%)	72.6852 %	0.727	0.162	0.719	0.727	0.718	0.575
Percentage split (75%)	73.6143 %	0.736	0.157	0.731	0.736	0.729	0.590

Table 3.8.2.2: Classifier Adaboost.

Using	Accuracy	TP rate	FP rate	Precision	Recall	F measure	MCC
Using training set	99.2489 %	0.992	0.006	0.993	0.992	0.992	0.989
Cross validation (10fold)	98.4143 %	0.984	0.011	0.984	0.984	0.984	0.976
Cross validation (20fold)	98.6269 %	0.986	0.009	0.986	0.986	0.986	0.980
Percentage split (50%)	89.9061 %	0.899	0.062	0.900	0.899	0.898	0.848
Percentage split (75%)	96.6301 %	0.966	0.024	0.967	0.966	0.966	0.949

Table 3.8.2.3: Classifier Bagging.

3.8.3 Test Data from Paths

In order to create test dataset from the shortest path and all possible paths we have followed some methods. Average traffic speed in Dhaka has reduced from 21km per hour to 7km per hour in the last 10 years as a result of the rising traffic congestion on roads and unplanned and uncontrolled growth of Dhaka, according to World Bank data. The data also reveals that the current traffic speed is slightly above the average walking speed that ultimately eats up 3.2 million working hours a day. The number

of residents in Dhaka rose from 3 million in 1980 to 18 million now. And, if the current trend continues, the number will surpass 35 million and the traffic speed will come down to 4km per hour by 2035. Let us assume we have a path from "n1" to "n70" and the user start his at 10:00am at morning at Friday.

n1	n2	n3	n7	n12	n18	n27	n45	n52	n70

In previous after data pre-processing phase our training dataset has only four attributes which are "Day", "Time", "Edge" and "Traffic". We have removed the other attributes which was unnecessary for our model creation. So, to make the test datasets we have to use the same attributes list. The average speed of vehicles in Dhaka city is 7km/h which is shown in the previous article. We have followed some steps to create test dataset.

- At first we select the source node and destination node and then find the edge between them. We had a predefined dataset which consists of source node, destination node and the edge between them. So source and destination of the first instance of our test dataset is "n1" and "n2" and the edge between them is "e1" which we found in the predefined dataset of nodes and edges. Second instance's source is "n2" and destination is "n3" and we found the edge is "e2" Simultaneously we create the other source and destination attributes and find the edges between them and inserted into our dataset.

Day	Time	Edge	Traffic
		e1	
		e2	
		e3	
		……	

Table 3.8.3: Find Edge.

- We inserted "Fri" as the key string of Friday into the Day attribute.

Day	Time	Edge	Traffic
Fri		e1	
Fri		e2	
Fri		e3	
……..		……	

Table 3.8.3.1: Day Attribute.

- Now we have to fill the time attribute. So we need a calculation. For the first instances, the edge is "e1". The distance of "e1" is 915m. Which we got from a

predefined dataset. This dataset consists of the distance (in meter) between two nodes.

A	B	C
source	destination	distance
n1	n2	915
n2	n3	333
n3	n4	401
n4	n5	237
n5	n6	524

Figure 3.8.3: Sample Data set for distance.

First value of the "Time" attribute will be 11:00am. But to find the second value of the "Time" instance we have make some calculation. First we need find the time that cost to travel from "n1" to "n2". Average speed of vehicle is 7km/h which is 7000m/3600s. So time need to cross 915m is $(3600 * 915)/7000$ which is 471s.And 471 second is 8 minutes to be approximate. Adding this time with our staring time is 10:08pm. Now we have declared earlier that we have divided the time into 15 minutes' interval. So we have to decide the time of our second instance. Whether the time is 10:00am or 10:15am. For this decision we have used the distance calculation which is called Euclidian distance measurement which was used in KNN clustering. And after measuring the distance the closest timestamp is 10:15pm. So we insert 10:15pm to the second instance in our Time attribute. Simultaneously we have calculated the other values of time attributes.

Pseudocode:

1. Find the distance of previous instance.
2. Find previous instance timestamp.
3. Find the next timestamp (e.g. 10:15am)
4. Calculate the travelling time with (3600*distance)/7000 this formula.
5. Add the time with the previous instance time.
6. Find that the added time is in next timestamp or current timestamp using distance measure.
7. Add the timestamp to the Time attribute.

Day	Time	Edge	Traffic
Fri	10:00am	e1	
Fri	10:15am	e2	
Fri	10:30am	e3	
……..	…….	……	

Table 3.8.3.2: Timestamp to the Time attribute

Chapter 4

Experimental Results

The result of our research is to find an optimal route for a source and destination. To do that we have to compared all the paths along with the shortest and longest paths. As a result, we may find the optimal path based on the traffic condition of the roads. We have already builds classification models from the dataset. And let's assume that we have selected the best model. The have to select the source, destination, day and starting time.

4.1 Gathering All Paths

At first gathered all possible paths and shortest path from source to destination that user provide. To find shortest path we used shortest path algorithms that we have discussed earlier in chapter 3. To find all possible path we have used *networkx* library for python. The function called *all_simple_paths* helps us to find all possible paths. Here is some example of all possible paths. (Using "n1" as source and "n72" as destination).

n1	n2	n3	n8	n15	……..	n45	n63	n53	n72

n1	n2	n3	n11	n28	……..	n51	n52	n53	n72

n1	n2	n6	n9	n22	……..	n61	n69	n70	n72

Each pair of source and destination has average 50 possible paths with unique length values.

4.2 Making Test Dataset and Find Prediction

The process of making test dataset from a path has been discussed in section 4.7.3. So each pair of source and destination has average 50 datasets approximate. Now we need to make prediction of the test datasets and set the predicted values to the datasets. As we said earlier our class attribute of the training dataset is "Traffic" which is a nominal attribute and the we have types of class value, "Low", "Medium", "High" and "Extreme" which have shown in attributes Table So after predicting values of test datasets our test datasets will looks like this.

Day	Time	Edge	Traffic
Fri	10:00am	e1	Low
Fri	10:15am	e2	Low

Fri	10:30am	e3	Medium
……..	…….	……	……

Table 4.2: Predicting Values.

4.3 Comparison

To compare the predicted dataset from different paths from source to destination we have used a cost function to evaluate the traffic cost for each dataset. The dataset with lowest cost will be the optimal dataset and also the optimal path will be the path from where the dataset created. Let's assume after prediction we have a full dataset like this.

Day	Time	Edge	Traffic
Fri	10:00am	e1	Low
Fri	10:15am	e2	Low
Fri	10:30am	e4	Medium
Fri	10:30am	e5	Low
Fri	10:30am	e6	Low
Fri	10:45am	e9	Medium
Fri	11:00am	e10	High
Fri	11:00am	e22	High
Fri	11:45am	e32	High
Fri	12:00pm	e50	Medium

Table 4.3: Prediction full Dataset.

So we have used numerical values for each of the predicted traffic values.

Traffic	Numerical value
Low	1
Medium	2
High	3
Extreme	4

Table 4.3.1: Traffic Values.

Now we have calculate the total from our predicted traffic value.

$$f(x) = \sum_{i=1}^{l} 1 + \sum_{i=1}^{m} 2 + \sum_{i-1}^{h} 3 + \sum_{i=1}^{e} 4$$

Where *l, m, h, e* are the number of Low, Medium, High and Extreme.

Traffic	Numerical values
Low	1
Low	1
Medium	2
Low	1
Low	1
Medium	2
High	3
High	3
High	3
Medium	1
	Total = 18

Table 4.3.2: Predicted Traffic Values.

We simultaneously run the whole process for our all predicted datasets from the paths. And it gives us some numerical total. So we select the path with lowest total is the optimal path.

4.4 Practical Example

We have selected “n2” as source and “n72” as destination and starting time is 11:00am in the morning. So here are some pictures of our python code simulation.

nde	Type	Size	Value
0	list	23	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
1	list	22	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
2	list	24	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
3	list	23	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
4	list	23	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
5	list	22	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
6	list	22	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
7	list	21	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
8	list	21	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
9	list	20	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
10	list	22	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
11	list	21	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
12	list	24	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
13	list	23	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
14	list	15	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n11', ...]
15	list	17	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n28', ...]
16	list	16	['n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9', 'n10', 'n28', ...]
17	list	25	['n2', 'n3', 'n4', 'n31', 'n25', 'n27', 'n32', 'n33', 'n30', 'n28', ...
18	list	24	['n2', 'n3', 'n4', 'n31', 'n25', 'n27', 'n32', 'n33', 'n30', 'n28', ...
19	list	26	['n2', 'n3', 'n4', 'n31', 'n25', 'n27', 'n32', 'n33', 'n30', 'n28', ...

Figure 4.4: Python code simulation Results.

We have only showed 20 paths here but practically it gives us 93 unique paths from source to destination.

Index	Day	Edge	Time	Traffic
0	4	2	22	Medium
1	4	64	23	Medium
2	4	67	24	Medium
3	4	74	25	Medium
4	4	62	26	Medium
5	4	60	27	Medium
6	4	36	28	Medium
7	4	37	29	Medium
8	4	38	30	Medium
9	4	42	31	Medium
10	4	44	32	Medium
11	4	45	33	Medium
12	4	119	34	Low

Figure 4.4.1**:** Python code simulation Results with Attributes.

This is the simple test dataset for a particular path. To make our work easy we have replaced the test attributes with numerical values. E.g.

Day	**Numerical value**
Fri	1
Sat	2
Sun	3
Mon	4
Tue	5
Wed	6
Thu	7

Table 4.4: Simple test Dataset.

We have done this for all other attributes.

Final Result

The summation of 93 paths are given below.

```
['Mon', 'Jul', '', '9', '05:37:18', '2018']
Shortest path value is: 19
Other paths values are: [40, 39, 40, 39, 37, 36, 35, 34, 35, 34, 36, 35, 38, 37, 27, 31, 29, 44, 43, 44, 43,
41, 40, 39, 38, 40, 39, 40, 39, 42, 41, 21, 42, 41, 42, 41, 39, 38, 37, 36, 38, 37, 38, 37, 40, 39, 19, 52, 51,
52, 51, 49, 48, 47, 46, 48, 14, 48, 47, 50, 49, 39, 43, 41, 50, 49, 50, 49, 47, 46, 45, 44, 46, 45, 46, 45, 48,
47, 27, 48, 47, 48, 47, 45, 44, 43, 42, 44, 43, 44, 43, 46, 45, 25]
```

Figure 4.4.2: Summation of 93 paths.

It has shown that there is a path with total value of 14 which is lower than the shortest path value 19. So the optimal path is the path with value 14. Although our simulation did not give us a good result. Sometimes it shows that the shortest path is the optimal path. But in some cases when the traffic is mostly danced, our simulation gives us a good result.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In this thesis, at first we took a historical big dataset from the Dhanmondi area using google map direction API. Then to solve the problem we have built up a python code using Json library which was help us to collecting traffic data of Dhanmondi area. Then we built up another python code with k-means clustering to cluster the dataset according to our need. For further experiment we improved our system to save the sub datasets as excel, csv and arff format. And afterwards we applied some shortest path algorithm such as Dijkstra, Bellman-ford, Floyd Warshall, A star search, Genetic algorithm to find out the shortest path, which we have discussed in detail in chapter 3. Then we classified the data using some classifier. Then we compared all possible path with shortest path to find the best path.

5.2 Future Work

In future, we would like to extend our research by using real time traffic data. We will also work on weather, accident data and we will try to predict some future patterns of traffic congestion. Also we are willing to make a proposal plan to implement this kind of work in our Dhaka city.

References

[1] Ashratuz Zavin and Adnan Sharif, “Towards Developing an Intelligent System to Suggest Optimal Path Based on Historic and Real-time Traffic Data’’, 20th International Conference of Computer and Information Technology (ICCIT), 22-24 December, 2017, Dhaka Bangladesh.
[2] Thammasak Thianniwet and Satidchoke Phosaard, “Classification of Road Traffic Congestion Levels from GPS Data using a Decision Tree Algorithm and Sliding Windows”, World Congress on Engineering 2009 Vol I WCE 2009, July 1 - 3, 2009, London, U.K.
[3] Jalil Kianfar and Praveen Edara, “A data mining approach to creating fundamental traffic flow diagram”, Procedia - Social and Behavioral Sciences-Volume 104, 2 December 2013, Pages 430 439,2nd Conference of Transportation Research Group of India (2nd CTRG).
[4] Brian L. Smith and Michael J. Demetsky, Z Fellow, ASCE, “Traffic flow forecasting: comparison of modeling approaches”, Journal of Transportation Engineering, vol. 123, no. 4, pp. 261-266, 1997.
[5] Jaros law Rzeszotko and SinhHoa Nguyen, “Machine Learning for Traffic Prediction”, Concurrency Specification and Programming, vol. 119, no. 3-4, pp. 407-420, August 2012.
[6] Hector Gonzalez, Jiawei Han, Xiaolei Li, Margaret Myslinska, John Paul Sondag, “Adaptive Fastest Path Computation on a Road Network: A Traffic Mining Approach”, 33rd international conference on Very large data bases, Pages 794-805 Vienna, Austria September 23 - 27, 2007.
[7] Jiawei Han, Zhenhui Li, and Luan Tang, “Mining Moving Object, Trajectory and Traffic Data”, Lecture Notes in Computer Science, vol. 5982, pp. 485-486, April 2010.
[8] https://brilliant.org/wiki/dijkstras-short-path-finder/
[9] https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm
[10] https://www.geeksforgeeks.org/dynamic-programming-set-16-floyd-warshall-algorithm/
[11] https://www.geeksforgeeks.org/a-search-algorithm/
[12] https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
[13] https://developers.google.com/maps/documentation/distance-matrix/intro
[14] http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html

Appendix A

For our simulation we have divided our task to multiple function with multiple python files. Some of the main functions are given below.

Task-1: Parse JSON form Google Map Distance Matrix API

"urlJsonParse" function has four parameters source latitude, source longitude, destination latitude, and destination longitude which we read from a predefined CSV file called `'latitude longitude for Dhanmondi area - Sheet1.csv'` and it returns a list of values which includes source address, destination address, distance, duration (average), duration (current). *"main"* function collects the data and make a data frame with *pandas* library and store it a CSV file named **traffic.csv**. We have a TXT file called **API.txt** which stores Google Map API keys. Initially we have used 10 API keys per day. The keys are stored line by line. *"main"* function read the **API.txt** file and put it into the map url.

```
import urllib.request, json
import numpy as np
import pandas as pd
import time
from pathlib import Path

def urlJsonParse(s_lat,s_lang,d_lat,d_lang,key):
    values = []
    url = "https://maps.googleapis.com/maps/api/distancematrix/json?units=imperial&orig
ins="+s_lat+","+
    s_lang+"&destinations="+d_lat+","+d_lang+"&mode=driving&traffic_model=best_guess&de
parture_time=now&key="+key
    with urllib.request.urlopen(url) as url:
        data = json.loads(url.read().decode())
        values.append(str(data['status']))
        if "OVER_QUERY_LIMIT" in values:
            return values

        elif "REQUEST_DENIED" in values:
            return values
        else:
            values.append(data['origin_addresses'][0])
            values.append(data['destination_addresses'][0])
            values.append(data['rows'][0]['elements'][0]['distance']['value'])
            values.append(data['rows'][0]['elements'][0]['duration']['value'])
            values.append(data['rows'][0]['elements'][0]['duration_in_traffic']['value'
])
            return values

def main(times,url_hit_count):
    file = open("API-KEY.txt","r")
    apix = file.read().split("\n")
```

```
    file.close()
    table = pd.read_csv('latitude longitude for Dhanmondi area - Sheet1.csv')
    table = table.fillna('dot')
    source_lat = table['Source Lat']
    source_lang = table['Source Lang']
    dest_lat = table['Destination Lat']
    dest_lang = table['Destination Lang']
    edge = table['Edge']
    #print(dest_lang.size)
    columns = ['Date','Day','Time','Source','Destination','Edge','Distance(meter)','Dur
ation(Original)sec','Duration(Current)sec','Traffic']
    main_data = pd.DataFrame(columns = columns)
    localtime = time.asctime( time.localtime(time.time()) )
    ss,dd,ff,gg,hh = localtime.split(' ')
    all_list = [[] for i in range(9)]
    j = 0
    while j < dest_lang.size:
        if(dest_lang[j] == 'dot'):
            print('Empty value')
        else:
            key_index = 0
            key_size = len(apix)
            while key_index < key_size:
                values = urlJsonParse(str(source_lat[j]),str(source_lang[j]),str(dest_l
at[j]),str(dest_lang[j]),str(apix[key_index]))
                if "OK" in values:
                    s = values[1]
                    d = values[2]
                    dis = values[3]
                    dura = values[4]
                    dura_t = values[5]
                    all_list[0].append(dd+' '+ff+' '+hh)
                    all_list[1].append(times)
                    all_list[2].append(s)
                    all_list[3].append(d)
                    all_list[4].append(edge[j])
                    all_list[5].append(dis)
                    all_list[6].append(dura)
                    all_list[7].append(dura_t)
                    all_list[8].append(ss)
                    url_hit_count+=1
                    break
                key_index+=1
            if key_index == key_size:
                print("!!!!All api key limit exceed. Enter the keys!!!!")
                break
        j+=1
    main_data['Date'] = all_list[0]
    main_data['Day'] = all_list[8]
    main_data['Time'] = all_list[1]
    main_data['Source'] = all_list[2]
    main_data['Destination'] = all_list[3]
    main_data['Edge'] = all_list[4]
    main_data['Distance(meter)'] = all_list[5]
    main_data['Duration(Original)sec'] = all_list[6]
    main_data['Duration(Current)sec'] = all_list[7]
```

```
    my_file = Path("traffic.csv")
    if my_file.is_file():
        previous = pd.read_csv('traffic.csv')
        frames = [previous,main_data]
        result = pd.concat(frames)
        result.to_csv('traffic.csv', encoding='utf-8', index=False)
        print("-----Data Concated with previous data")
    else:
        main_data.to_csv('traffic.csv', encoding='utf-8', index=False)
        print("-----There is no previous data, So new data file has been created")

    return url_hit_count
```

Next we have created a file called *main.py* which. This file includes one functions called "*timeIncrement"* which was to make the timestamp with 15 minutes' intervals. And later we have imported the previous python file *traffic.py* to make the whole works done. We have also included a text file called **log.txt** to observe the performance of our program. This program is simulated from 6.00am to 10:45pm non-stop. After running the program, we got the data in a CSV file.

```
import time
import traffic as tr

def timeIncrement(value):
    if "am" in value:
        x = value.split(":")
        if "45" in value:
            if "12" in value:
                x[0] = "1"
            else:
                x[0] = str(int(x[0])+1)
        y = x[1].split("am")
        if "45" in value:
            y[0] = "00"
        else:
            y[0] = str(int(y[0])+15)

        if "12" in value:
            y[1] = "pm"
        else:
            y[1] = "am"

    if "pm" in value:
        x = value.split(":")
        if "45" in value:
            if "12" in value:
                x[0] = "1"
            else:
                x[0] = str(int(x[0])+1)
        y = x[1].split("pm")
        if "45" in value:
            y[0] = "00"
```

```
        else:
            y[0] = str(int(y[0])+15)

        if "12" in value:
            y[1] = "am"
        else:
            y[1] = "pm"


    b = x[0]+":"+y[0]+y[1]
    return b
 file = open("API-KEY.txt","r")
apix = file.read().split("\n")
file.close()

print("_______API KEY VALIDITY CHECK_____")
print("-----------------------------------")
i = 0
while i < len(apix):
    values = tr.urlJsonParse("23.756405","90.375087","23.754901","90.372881",str(apix[i
]))
    if "OK" in values:
        print("API key:",i+1," is valid")
    else:
        print("API key:",i+1," is Invalid")
    i+=1

choose = input("\nDo u want to continue[y/n]")
if choose == "y":
    tims  = input("Enter 1st local time(eg:12:00pm):")
    file = open("log.txt","w")
    file.write("-------------------------------------------------------------------------
\n")
    file.write("Starts time = "+tims+"\n")
    task = int(input("Enter the number of tasks:"))

    file.write("Tasks = "+str(task)+"\n")
    i = 0
    url_hit = int(input("Previous URL hits:"))
    while i < task:
        t = str(i+1)
        print("---Task "+t+" starts")
        file.write("---Task "+t+" starts\n")
        url_hit = tr.main(tims,url_hit)
        file.write("---#URL Hitted:"+str(url_hit)+"\n")
        print("---Completed task "+str(i+1)+" for time:"+tims)
        file.write("---Completed task "+str(i+1)+" for time:"+tims+"\n")
        tims = timeIncrement(tims)
        print("---Task stopped for 15 minute")
        if i != (task-1):
            time.sleep(10)
        i+=1
    file.write("-------------------------------------------------------------------------
\n")
    file.close()
```

Task-2: Clustering Dataset.

As we have said earlier that we used K-Means clustering. So we have created a *k-means.py* file. `meanFunction` function used for find initial clusters. After clustering it saves the clustered data to a CSV file called `clustered_data.csv`.

```
import numpy as np
import pandas as pd
import math

clusters = ['Low','Medium','High','Extrem']

def meanFunction(candidate_centers,prev_centers):
    i = 0
    while i < len(candidate_centers):
        if candidate_centers[i]:
            j = 0
            sum = 0
            while j < len(candidate_centers[i]):
                sum = sum + candidate_centers[i][j]
                j+=1
            prev_centers[i] = sum/len(candidate_centers[i])
        i+=1
    return prev_centers


# ================================================================================
# centers[w,x,y,z] list_of_values[all values of E1,E2....En]
# ================================================================================
def findClusters(centers,list_OF_values):
    score = []
    center_candidate = [[] for i in range(4)]
    i = 0
    while i < len(list_OF_values):
        j = 0
        temp = []
        while j < len(centers):
            temp.append(math.sqrt((centers[j]-int(list_OF_values[i]))*(centers[j]-
int(list_OF_values[i]))))
            j+=1
        temp2 = np.asarray(temp)
        score.append(clusters[temp2.argmin()])
        center_candidate[temp2.argmin()].append(temp2.min())
        i+=1
      print(meanFunction(center_candidate,centers))
findClusters(centers[0],list(edge_values[0][0]))


datasets = pd.read_csv('Main_datasets\main_datas.csv')
edges = datasets['Edge'].unique()
edge_values = [[] for i in range(edges.size)]
```

```
i = 0
while i < edges.size:
    temp = datasets['Duration(Current)sec'].where(datasets['Edge']==edges[i])
    edge_values[i].append(temp.dropna())
    i+=1


# ============================================================================
# Find center for each Edges
# ============================================================================

centers = [[] for i in range(edges.size)]
i = 0
while i < len(edge_values):
    values = np.asarray(edge_values[i][0])
    quarters = np.max(values)/5
    centers[i].append(quarters)
    j=1
    while j < 4:
        centers[i].append(quarters*(j+1))
        j+=1

    i+=1

traffic = [[] for i in range(edges.size)]
i = 0
while i < len(edge_values):
    traffic[i] = findClusters(centers[i],list(edge_values[i][0]))
    i+=1



trafic_list = []
i = 0
while i < len(traffic[0]):
    j = 0
    while j < len(traffic):
        trafic_list.append(traffic[j][i])
        j+=1
    i+=1
datasets['Traffic'] = trafic_list
datasets.to_csv('clustered_data.csv', encoding='utf-8', index=False)
```

Task-3: Shortest path algorithms.

With the help of the *networkx* library we have implemented Dijkstra, bellman ford and a star algorithm. But we could not found any Floyed warshall algorithm. So we implemented our own code.

```
path = []
def ConstructPath(p, i, j):
    i,j = int(i), int(j)
```

```
    if(i==j):
      path.append("n"+str(i+1))
    elif(p[i,j] == -30000):
      print (i,'-',j)
    else:
      ConstructPath(p, i, p[i,j]);
      path.append("n"+str(j+1))
    return path

def predValues(graph):
    v = len(graph)
    p = np.zeros(graph.shape)
    for i in range(0,v):
        for j in range(0,v):
            p[i,j] = i
            if (i != j and graph[i,j] == 0):
                p[i,j] = -30000
                graph[i,j] = 30000

    for k in range(0,v):
        for i in range(0,v):
            for j in range(0,v):
                if graph[i,j] > graph[i,k] + graph[k,j]:
                    graph[i,j] = graph[i,k] + graph[k,j]
                    p[i,j] = p[k,j]

    return p
```

Code for Genetic algorithm that we implement of our own.

```
import networkx as nx
import numpy as np
import pandas as pd
import itertools as it
from collections import Counter

def fitnessFunc(G,path):
    total = 0
    i = 0
    j = 1
    while i < len(path):
        if j > len(path)-1:
            break
        else:
            weight = G.get_edge_data(path[i],path[j])
            if weight is None:
                return False
            else:
                w = G.get_edge_data(path[i],path[j])['weight']
                total+=int(w)
            j+=1
        i+=1
    return total


def initialPopulation(all_paths,population_size):
```

```
    indexs = []
    population = []
    i=0
    while i < population_size:
        index = np.random.randint(0,len(all_paths))
        if index not in indexs:
            indexs.append(index)
            population.append(all_paths[index])
            i+=1
    return population


def percentCalculation(G,chromosoms,percent):
    fit_value = fitnessFunc(G,chromosoms[0]) + fitnessFunc(G,chromosoms[1])
    avg = ((fit_value/2)/100)*percent
    #per = (mean/100)*percent

    return (fit_value/2)-avg



def chromCutMarge(G,chrom1,chrom2,cut_point_1,cut_point_2):
    offspring1 = []
    offspring2 = []
    for i in range(len(cut_point_1)+1):
        if i == 0:
            offspring1.append(chrom1[:cut_point_1[i]].tolist())
            offspring2.append(chrom2[:cut_point_2[i]].tolist())
        elif i == len(cut_point_1):
            offspring1.append(chrom1[cut_point_1[i-1]:].tolist())
            offspring2.append(chrom2[cut_point_2[i-1]:].tolist())
        else:
            offspring1.append(chrom1[cut_point_1[i-1]:cut_point_1[i]].tolist())
            offspring2.append(chrom2[cut_point_2[i-1]:cut_point_2[i]].tolist())
    child1 = offspring1[0]
    child2 = offspring2[0]
    c = 0
    for i in range(len(offspring1)-1):
        if c == 0:
            child1 = child1 + offspring2[i+1]
            child2 = child2 + offspring1[i+1]
            c = 1
        else:
            child1 = child1 + offspring1[i+1]
            child2 = child2 + offspring2[i+1]
            c = 0

    return child1,child2


def mutationFunction(G,path):
    rand1 = [np.random.randint(1,len(path)-1),np.random.randint(1,len(path)-1)]
    rand1.sort()
    all_paths = []
    for s in nx.all_simple_paths(G, source=path[rand1[0]], target=path[rand1[1]-1]):
        all_paths.append(s)
    fit_new = []
```

```
    for value in all_paths:
        fit_new.append(fitnessFunc(G,value))

    path[rand1[0]:rand1[1]] = all_paths[fit_new.index(min(fit_new))]
    #print("Before fit: ",fit," After fit: ",min(fit_new))
    return path



def crossOver(G,chromosoms,percent,max_cut_point):


    #print("Fitness:",fitnessFunc(G,chromosoms[0])," ",fitnessFunc(G,chromosoms[1]))
    avg_fitness = percentCalculation(G,chromosoms,percent)
    #print("avg fit: ",avg_fitness)
    #cut_points = []
    chrom1 = np.asarray(chromosoms[0])
    chrom2 = np.asarray(chromosoms[1])
    intrsct = np.intersect1d(chrom1,chrom2)
    if max_cut_point >= intrsct.size:
        return False
    else:
        #print("interscting points:",intrsct)
        childs = []
        for i in range(1,max_cut_point+1):

            cut_point_values = []
            if i == 1:
                cut_point_values = intrsct

                for j in cut_point_values:
                    cut_point_1 = []
                    cut_point_2 = []
                    cut_point_1.append(np.where(chrom1 == j)[0][0])
                    cut_point_2.append(np.where(chrom2 == j)[0][0])
                    ch1,ch2 = chromCutMarge(G,chrom1,chrom2,cut_point_1,cut_point_2)
                    childs.append(ch1)
                    childs.append(ch2)

            else:
                for item in it.combinations(intrsct,i):
                    cut_point_values.append(list(item))
                #print("i is :",i,"combination is:",cut_point_values)
                for j in cut_point_values:
                    cut_point_1 = []
                    cut_point_2 = []
                    for k in j:
                        cut_point_1.append(np.where(chrom1 == k)[0][0])
                        cut_point_2.append(np.where(chrom2 == k)[0][0])
                    cut_point_1.sort()
                    cut_point_2.sort()
                    ch1,ch2 = chromCutMarge(G,chrom1,chrom2,cut_point_1,cut_point_2)
                    childs.append(ch1)
                    childs.append(ch2)
        new_fitness = []
        for kids in childs:
            if fitnessFunc(G,kids) == False:
```

```
                new_fitness.append(np.inf)
            else:
                new_fitness.append(fitnessFunc(G,kids))
        final = []
        uniq_fit = np.unique(np.asarray(new_fitness))
        uniq_fit = np.sort(uniq_fit)
        ch1_fit = [uniq_fit[0],uniq_fit[1]]

        for u in range(2):
            for v in range(len(new_fitness)):
                if new_fitness[v] == ch1_fit[u]:
                    final.append(childs[v])
                    break
        return final



datasets = pd.read_csv('data-distance-update.csv')
graph=[]

for row in datasets.iterrows():
    index, data = row
    graph.append(data.tolist())

G = nx.DiGraph()

i = 0
while i < len(graph):
    G.add_edges_from([(graph[i][0],graph[i][1])], weight=graph[i][2])
    i+=1
source = "n1"
target = "n75"

all_paths = []



for path in nx.all_simple_paths(G, source=source, target=target):
    all_paths.append(path)

peoples = initialPopulation(all_paths,4)

fitness_values = []
for i in peoples:
   fitness_values.append(fitnessFunc(G,i))

chromosom = []
for i in range(2):
    chromosom.append(peoples[fitness_values.index(min(fitness_values))])
    fitness_values.remove(min(fitness_values))


generation = int(input("Enter the number of the generation: "))
max_cut_point = int(input("Enter the maximum number of the chromosome cutting point:
"))
percent = int(input("Enter the Parcentage value for maching fitness: "))
```

```
print("Initial selected population fitnes are : ",fitnessFunc(G,chromosom[0])," and
",fitnessFunc(G,chromosom[1]))

for i in range(generation):
    childs = crossOver(G,chromosom,percent,max_cut_point)
    mute_child1 = mutationFunction(G,childs[0])
    mute_child2 = mutationFunction(G,childs[1])
    if fitnessFunc(G,mute_child1) < fitnessFunc(G,childs[0]):
        chromosom[0] = mute_child1
    else:
        chromosom[0] = childs[0]

    if fitnessFunc(G,mute_child2) < fitnessFunc(G,childs[1]):
        chromosom[1] = mute_child2
    else:
        chromosom[1] = childs[1]
    print("Generation: ",i+1," completes")
    print("Fitness of new born chids are: ",fitnessFunc(G,chromosom[0])," and
",fitnessFunc(G,chromosom[1]))
```

Task-4: Comparison

We have a file called *classification_1.py* which include all classification algorithm with the help of the *scikit-learn* library. We did not include the file in here. We just simply go to the comparison phase. So for comparison we create a file called *path_finder*. *py* file. It includes the other files *shortest_path.py* and c*lassification_1.py* to collect the other data. In *path_finder.py* we have created test data and make the comparison.

```
import numpy as np
import pandas as pd
import networkx as nx
import shortest_path_algos.main as spm
import classification.classification_1 as clf
import time
import math

localtime = time.asctime( time.localtime(time.time()) )
ss = localtime.split(' ')
print(ss)

avg_duration_pMeter = 3600/7000 #KMPH


def comparisonValue(predicted_data):
    total = 0
    traffic_data = predicted_data["Traffic"]
    for i in traffic_data:
        if i == "Low":
            total = total+1
        elif i == "High":
```

```
            total = total+3
        elif i == "Medium":
            total = total+2
        else:
            total = total+4
    return total


def next_time_finder(curr_time):
    mints = curr_time.split(":")[1]
    hrs = int(curr_time.split(":")[0])
    a,b,c,d = mints
    mints = int(a+b)
    am_pm = c+d
    if hrs == 11 and mints == 45:
        hrs = 12
        mints = 0
        if am_pm == "am":
            am_pm = "pm"
        else:
            am_pm = "am"
    else:
        if mints == 45:
            mints = 0
            if hrs == 12:
                hrs=1
            else:
                hrs+=1
        else:
            mints += 15
    return hrs,mints,am_pm



def timeCalculation(start_time,distance_list):
    mints = start_time.split(":")[1]
    hrs = int(start_time.split(":")[0])
    a,b,c,d = mints
    mints = int(a+b)
    am_pm = c+d

    time_list = []
    vs = mints
    next_hrs,next_mints,next_am_pm = next_time_finder(start_time)

    for values in distance_list:
        tme = math.ceil(avg_duration_pMeter*int(values))
        mints = tme+mints

        if next_mints == 0:
            next_mints = 60

        if  abs(mints-next_mints) < abs(mints-vs):


            hrs = next_hrs
            am_pm = next_am_pm
```

```
            if next_mints == 60:
                mints = 0
                vs = 0
                next_hrs,next_mints,next_am_pm = next_time_finder(str(hrs)+":"+str(mint
s)+str(mints)+am_pm)
                time_list.append(str(hrs)+":"+str(mints)+str(mints)+am_pm)
            else:
                mints = next_mints
                vs = next_mints
                next_hrs,next_mints,next_am_pm = next_time_finder(str(hrs)+":"+str(mint
s)+am_pm)
                time_list.append(str(hrs)+":"+str(mints)+am_pm)
        else:
            if vs == 0:
                time_list.append(str(hrs)+":"+str(vs)+str(vs)+am_pm)
            else:
                time_list.append(str(hrs)+":"+str(vs)+am_pm)

    return time_list



def testDataCreate(path,start_time,datasets,path_data):


    j = 0
    edge,source,dest,times,day,date,distance,duration_avg,duration_in_traf = [],[],[],[
],[],[],[],[],[]

    for i in range(1,len(path)):

        day.append(ss[0])

        dist = shortest_path_data['distance'][np.where((shortest_path_data['source'] ==
 path[j]) & (shortest_path_data['destination'] ==path[i]))[0][0]]
        distance.append(dist)
        edge.append(datasets['Edge'][np.where((datasets['Source'] == path[j]) & (datase
ts['Destination'] == path[i]))[0][0]])
        j+=1
    times = timeCalculation(start_time,distance)

    d = {'Day': day,'Time':times,'Edge':edge,'Traffic':np.nan}
    test_data = pd.DataFrame(data=d)
    return test_data



shortest_path_data = pd.read_csv('all_paths_graph.csv')

source = 'n2'
destination = 'n72'
start_time = '11:00am'

dj_path,dj_len,bell_path,bell_length,flw_path,flw_len,astar_path,astar_len,all_paths =s
pm.shortest_path(shortest_path_data,source,destination)

datasets = pd.read_csv('clustered_data.csv')
```

```
s_data,d_data = shortest_path_data['source'].tolist(),shortest_path_data['destination']
.tolist()

new_s_data,new_d_data = [],[]
for i in range(int(datasets['Source'].size/shortest_path_data['source'].size)):
    new_s_data = new_s_data + s_data
    new_d_data = new_d_data + d_data

datasets['Source'],datasets['Destination'] = new_s_data,new_d_data

dds = testDataCreate(dj_path,start_time,datasets,shortest_path_data)

datasets_1,dds,classes = clf.dataPreprocess(datasets,dds)

predicted_data = clf.classifing(datasets_1,dds,classes)

sp_value = comparisonValue(predicted_data)
print("Shortest path value is:",sp_value)

all_paths_values = []

for p in all_paths:
    dds = testDataCreate(p,start_time,datasets,shortest_path_data)

    datasets_1,dds,classes = clf.dataPreprocess(datasets,dds)

    predicted_data = clf.classifing(datasets_1,dds,classes)

    all_paths_values.append(comparisonValue(predicted_data))

print("Other paths values are:",all_paths_values)
```

Here is some example of CSV file that we used.

latitude longitude for Dhanmondi area - Sheet1.csv

Edge	Source Lat	Source Lar	Destinatio	Destinatio	distance
E1	23.7518	90.3676	23.7564	90.3751	915
E2	23.7564	90.3751	23.7549	90.3729	333
E3	23.7549	90.3729	23.7529	90.3696	401
E4	23.7529	90.3696	23.7517	90.3677	237
E5	23.7517	90.3677	23.7477	90.3704	524

main_datas.csv

	A	B	C	D	E	F	G	H	I
1	Date	Day	Time	Source	Destination	Edge	Distance(meter)	Duration(Original)sec	Duration(Current)sec
2	Mar 23 2018	Fri	6:00am	Dhaka 120	Dhaka 1205, B	E1	915	180	98
3	Mar 23 2018	Fri	6:00am	haka 1205,	L, Dhaka 1209,	E2	333	115	79
4	Mar 23 2018	Fri	6:00am	Dhaka 120	6, Dhaka 1209,	E3	401	76	41
5	Mar 23 2018	Fri	6:00am	Dhaka 120	tmasjid Road, I	E4	237	53	23
6	Mar 23 2018	Fri	6:00am	nasjid Roac	A, Dhaka 1205	E5	524	109	55

data-distance-update.csv

source	destinatic	distance
n1	n2	915
n2	n3	333
n3	n4	401
n4	n5	237
n5	n6	524
n6	n7	125
n7	n8	122
n8	n9	117

all_paths_graph.csv

source	destinatic	distance
n1	n2	915
n2	n3	333
n3	n4	401
n4	n5	237
n5	n6	524
n6	n7	125
n7	n8	122

clustered_data.csv

	A	B	C	D	E	F	G	H	I
1	Date	Day	Time	Source	Destination	Edge	Distance(meter)	Duration(Original)sec	Duration(Current)sec
2	Mar 23 2018	Fri	6:00am	Dhaka 120	Dhaka 1205, B	E1	915	180	98
3	Mar 23 2018	Fri	6:00am	haka 1205,	L, Dhaka 1209,	E2	333	115	79
4	Mar 23 2018	Fri	6:00am	Dhaka 120	6, Dhaka 1209,	E3	401	76	41
5	Mar 23 2018	Fri	6:00am	Dhaka 120	tmasjid Road, I	E4	237	53	23
6	Mar 23 2018	Fri	6:00am	nasjid Roac	A, Dhaka 1205	E5	524	109	55