IBM NTC-2014 -- Traffic Analyzer [Salted Crypts]

Objective:

We will provide information about the public transport system (buses) of a city by using the data collected from the GPS of the cell-phones of those using the mobile app. We will provide our users with the information on the bus schedules, prediction of arrival times of buses, average waiting time on a bus stop, bus frequency on a bus stop.

Our app depends only on the data collected from the users and hence, will update automatically, without the need to feed bus schedules or bus routes to it. It will also provide real time data to its users, without the need for any added infrastructure to sustain it.

Requirements for running the program:

1. python 2.7
2. Python Libraries:
   a. PIL
      (sudo apt-get install python-pillow)
   b. Pygame
      (sudo apt-get install python-pygame)
   c. Kivy
      (sudo apt-get install python-kivy)

*If Kivy requires cython then run: sudo apt-get install cython. If installation of Kivy fails, use *auxiliary_request.py* instead of *request.py*

* The above software has been tested on linux machine

How to run program:

1. python cluster.py -- Acts as the central server responsible for acting on aggregate data. It will start clustering and identifying the buses.
   1. Red Dots are the identified buses by our clustering algorithm.
   2. Rectangles are the actual buses.
   3. Blue dots are the complete set of data points available to us.
   4. The bus stops are numbered from 0-11.

The plot screen which shows up can be used to identify how our algorithm works.

The default database is DatabaseAlt_new.db and Bus_new_2.db.
To run on the 2nd database, Uncomment the part containing the name -- DatabaseAlt_new_2.db and Bus_new_2.db, and comment out the default databases.

2. python request.py -- Acts as the interface used by the mobile users to request information. Run this after 1 minute of running the cluster.py to allow the algorithm to learn the routes of the buses. It needs to be re-run for every request. A set of sample requests are mentioned in the next section.

In case of no such bus or inability to detect such bus, the plot window will show no plotted points or will intantly disappear.

The actual bus routes can be found in ‘helper_functions/bus_routes_list.txt’.

3. Newone.apk – It is the app responsible for collecting the location and speed of users. It can be transferred to an android platform [4.0 and above] and can be installed in it. It displays – latitude, longitude and current speed of the user. Newone.zip conatins the source code for the app

Test Cases for request.py:

Source [left Colum], Destination[Right Column] -- (in the app interface)

1. Source- 2 Destination- 9
2. Source- 8 Destination- 1
3. Source- 11 Destination- 7

Data Collection:

The mobile app will collect the speed and location of each of its users using the GPS in their phones. For this we have made an android application which displays the latitude, longitude and speed (Newone.apk).

Since real time data was not available, for our demonstration we have generated database (DatabaseAlt_new) containing synthetic data based on the following method:

1. The Synthetic data is generated by program synthetic_data.py.
2. Initially four bus routes which cover twelve bus stops are created. All buses have thirty people and each bus stop has twenty people.
3. The number of people boarding and deboarding a bus follows Poisson's distribution. The parameter of Poisson's distribution (λ) has been defined individually for each bus and bus stop
4. λ of a bus stop gives the average number of people who deboard there from any bus. Similarly the λ of a bus gives the average number of people who board the bus from any bus stop
5. The coordinates of the people in the bus is lie within some small radius (2^0.5 pixel) and the differences of speed are tolerated up to 10 units going in the same direction as bus
6. The people on the bus stops have speeds up to 30 units and random directions
7. Also we have created random points on streets with speeds comparable to buses with random directions. These act as other vehicles (not buses) plying on the same roads. This acts as noise to our clustering algorithms.

*In real time the database will be populated by the data collected from the android app that we have developed
** Synthetic data located in /database/DatabaseAlt_new
*** Database Attributes = Identification Number, Time Stamp, Speed, Location (x & y coordinate), Direction, Street

Application Modules :

The following modules work on the data aggregated in the database. Based on this, they identify buses plying on various routes, which helps us to calculate arrival times of buses on a bus stop, average waiting time and bus frequency on a bus stop:

1. cluster.py [works on the central database and acts as our server]

a. It plots points listed in database and acts as server.
b. Identifies the clusters among the points based on clustering algorithm and classify clusters as either buses or pedestrians or any other vehicle.
d. Plots the buses calculated based on clusters
e. It also shows the bus frequency and average wait time of each station

* It also plots the actual buses so that we may compare the locations of the actual buses and calculated buses graphically.

Working of Clustering Algorithm:

1. In each iteration, points within 20 pixel radius of every bus stop are sent to a cluster function which returns the possible clusters which may form buses
2. Cluster function filters out points whose speed is below cut-off (30 units). Then it creates a graph with edges between points whose distances are less than 7 units, difference in speed is less than 10 units and are in similar directions. Then the connected components created are returned as different sets.
3. The above process is done for each bus stop.
4. The obtained set of clusters is compared with existing calculated buses (The set of points which we have identified as travelling in the same bus. Initially there are no buses) If the intersection of the bus with cluster is above a certain threshold we will replace the existing bus with that cluster.
5. All the remaining clusters are assigned as new buses
6. After each iteration of time stamp, average coordinates of points in the bus are written into the database which is used by request.py and also used for plotting purposes
7. Whenever the average speed of points in a bus falls below 30 and it is 5 pixel radius of a bus stop, the bus stop is added in the bus route of the particular bus. This bus route is also updated after each iteration and written into the database. This bus route is used by request.py to predict the optimum bus.
8. In every time-step, for every existing bus, we run preserve () on its cluster() to preserve the accuracy of the cluster denoting the bus. It would protect against issues such as – people getting down at locations which is not a bus stop, trailing car wrongly identified as a part of the bus, etc.
9. The preserve() re-clusters the busses, picking up the largest cluster which should also be larger than the threshold value for a bus size.
10. Some buses may be duplicated but this will not affect the service provided to request.py.

* Since a bus may also have very few people in it (below the threshold that we have set), our algorithm will wrongly discard it as not being a bus, but will later re-identify it once it fills up again.

2. request.py [Mobile app interface used by user to obtain required information]

1. It runs parallel with cluster.py
2. request.py takes origin and destination bus stops (through GUI) as input
c. Selects all the buses having origin and destination in its bus route. For each bus identified it calculates expected time taken by bus to reach origin and adds expected time taken to reach destination from source based on the route.
d. It outputs the identity and current location of the optimum bus. The optimum bus is the bus which takes the least amount of time to reach source station and finally reach the destination.

*Allow cluster.py to run for (1 min) so that it may learn the correct routes of buses

** Here we also take as input the source of the requester, but in real time, it would be determined by the requesters coordinates. So, only the destination would be required.

<u>Databases Maintained By Application:</u>

DatabaseAlt_new: The current database used by the cluster.py
DatabaseAlt_new_2: Alternate database that can be used
Bus_new: Contains the positions of the actual bus for DatabaseAlt_new for each time stamp
Bus_new_2: Contains the position of actual bus for the alternate database (database-2)
BusPos: It contains the current position of the busses identified by the algorithm (Populated during the execution of cluster.py)
BusRoute: It contains the sequence of stops made by the identified buses. (Populated during the execution of cluster.py)