ST-OPTICS: A spatial-temporal clustering algorithm with time recommendations for taxi services

by

Avni Malhan

Under the supervision of

Dr. Viswanath Gunturi

Dr. Vinayak Naik

Indraprastha Institute of Information Technology, Delhi

June 2017

ST-OPTICS: A spatial-temporal clustering algorithm with time recommendations for taxi services

By

Avni Malhan

Submitted
in partial fulfilment of the requirements for the degree of
Master of Technology

to

Indraprastha Institute of Information Technology, Delhi
June 2017

Certificate

This is to certify that the thesis titled **"ST-OPTICS: A spatial-temporal clustering algorithm with time recommendations for taxi services"** being submitted by **Avni Malhan** to the Indraprastha Institute of Information Technology Delhi, for the award of the Master of Technology, is an original research work carried out by her under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

Dr. Viswanath Gunturi and Dr. Vinayak Naik

Department of Computer Science

June, 2017 **IIIT, Delhi**

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my supervisors Dr. Viswanath Gunturi and Dr. Vinayak Naik for giving me the opportunity to work on this project under them. The door to both the professors' office was always open whenever I had a doubt regarding my research or writing. They consistently provided with exemplary guidance and steered me in the right the direction whenever I needed it.

I would also like to thank my parents, siblings and my friends for providing me with unfailing support and continuous encouragement throughout the journey of my thesis.

Abstract

Taxis have become the most popular and preferred mode of transport because of the comfort and convenience they offer. In this work, we design a novel algorithm, called ST-OPTICS, which helps taxi fleet owners to manage their cars by recommending the nearest hotspot of pickup locations based on the current time. It will be useful for personal as well as shared taxi drivers. ST-OPTICS is a density-based algorithm to find spatio-temporal clusters by analyzing the pickup trends of spatially close locations. In contrast to the existing clustering algorithm, it handle noise points in a different way and forms clusters of varying density based on spatial and temporal closeness. Recommendations from ST-OPTICS will consist of locations along with the timings during which they'll behave as pickup hotspots. This algorithm was evaluated on Singapore's taxi cab data. Our results indicated a superior performance of the proposed ST-OPTICS algorithm.

Contents

List of Figures

Chapter-1

Introduction

Taxis have become the most common and preferred mode of transport for both, short and long routes. Due to increasing traffic many people avoid driving and prefer taking cabs instead as they are easy to get at economical rates. This increasing demand for taxis increased employment as a lot of drivers are getting linked either with cab aggregators like Ola and Uber or with taxi rentals to increase their incomes. Although some drivers are able to earn a good amount, many fail to even earn a descent living as they don't get enough rides.

Few drivers roam around the city for hours looking for passengers, while many prefer waiting near some busy places, like airports or outside some big malls. Due to this we get taxis in abundance at such places, which doesn't give every driver a chance to take a passenger. Even if he gets, he might have to wait for long. It will not be of much help to get a ride for a short route after roaming around or waiting for quite long as short routes don't get much profit. This further leads to mismanagement of taxis as they'll be in excess at some places, while very less at others, resulting in increased waiting time for the passengers. To solve these problems, location recommendation is needed.

While a lot of work has already been done in the field of location recommendation for taxi drivers, none of them recommend time along with the locations for pickups. Simply recommending top three or top five locations as done by partitioning clustering will not help cab drivers to find passengers efficiently. Also, recommending just the hotspot locations will not help them decide when to be there. Thus, it is very important to recommend time with each location.

Our algorithm, ST-OPTICS is a density-based clustering algorithm which recommends locations along with the timings when the drivers can expect good number of pickups. This will also help people wanting to take a cab by reducing the waiting time.

Chapter-2

Related Work

This chapter provides details on the related work done with respect to the problem statement of this thesis.

2.1 Location recommendation for taxi drivers based on spatial data

Some significant work has been done by researchers in the field of location recommendation.

Nazneen N. [4] designed a booking system with the main aim of increasing the revenue for taxi drivers. They created an application where passengers and taxi drivers can register themselves. The taxi driver will send a request for pickup location recommendation and his current location. He'll get a recommendation by analysing next pickup location distance, pickup count and fare. To improve results, more information is collected from the rides and analysed. Yuan et al. [5] worked with a similar motive to maximize profit. He proposed a recommender system to help taxi drivers reach some nearby location where they're likely to find passengers. It also recommended nearby locations to people where they can easily find vacant taxis. For this, a probabilistic model is designed and fed with the information collected by GPS systems installed in taxis. Their method has 10% improvement than the baseline methods and 67% precision of the recommendations. Another work done by mining GPS trajectories is by Hsueh et al. [6] who devised a recommender system for next cruising location based on four main factors. These factors include the distance between current and the recommended location, waiting time for next passenger, expected fare for the trip and based on the drivers experience, most likely location given the current passengers drop-off location. They created a location-to-location graph model and evaluated the proposed system on a real world dataset from CRAWDAD.

2.2 Location recommendation for spatial-temporal data

Some other works focussed on clustering techniques. Zhang et al. [14] proposed an algorithm to recommend pickup points based on spatio-temporal clustering. Their framework mainly consists of pre-processing and real-time recommendations. Pre-processing is performed to get spatial-temporal clusters using hierarchical clustering algorithms. Based on taxis current location and the already obtained results, candidate pickup points are obtained. These are then ranked and top-5 locations are recommended to the taxi driver. Another work done on spatial-temporal data is the ST-DBSCAN algorithm by Birant et al. [7]. It is a density-based clustering algorithm which is an extension to DBSCAN. Existing DBSCAN algorithm has been modified in terms of identification of core points, noise points and adjacent clusters. It

discovers clusters on the basis of spatial, non-spatial and temporal values of the objects. Min et al. [15] focussed on designing a framework for spatio-temporal clustering algorithm using spatio-temporal statistics methodology and intelligence computation technology. This framework accounts for spatio-temporal autocorrelations and heterogeneity characteristics. It was applied to China's annual average temperature details over the period 1951-1992. In comparison to existing methods, their results showed that the framework was effective and considers the coupling in space and time.

Chapter-3

Background Study

3.1 Clustering

"Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters)" [11].

Clustering methods can be broadly classified as –

Partitioning Method- It divides the dataset of 'n' objects into 'k' partitions, called clusters. Each cluster contains atleast one element and k<=n.

Hierarchical Method- It creates a hierarchical decomposition of the objects present in the dataset. Based on how this decomposition is done, it can be classified as agglomerative or divisive approach.

Density-based Method- It helps create non-linear shaped clusters based on density. The basic idea of this technique is that the cluster keeps growing till the density of the neighbourhood exceeds some threshold value, where neighbourhood is defined by a fixed radius.

Model-based Method- In this model, data is considered as coming from a mixture of two or more components or clusters where each component is described by a density function.

Constraint-based Method- It finds clusters that satisfy user-specified preferences or constraints.

We focus on finding the optimal locations for taxi stands by clustering points which are spatially and temporally close, irrespective of the clusters' shape. Thus, we use density-based clustering method. The two main algorithms for density-based method are DBSCAN and OPTICS.

3.1.1 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN is a density-based algorithm which is robust to outliers. It requires two parameters [9]-

- ***Eps*** – Maximum radius of the neighborhood
- ***MinPts*** – Minimum number of points in the Eps-neighborhood of a point.

Algorithm 3.1 DBSCAN

Input:

- D: Dataset
- Eps: radius for neighbouring points
- minPts: minimum points within Eps distance for a point to become a core-point.

Output: Clusters based on spatial closeness.

```
1.  DBSCAN(D, eps, minPts)
2.      C=0
3.      For each point p in D
4.          If p is visited
5.              Continue next point
6.          Mark p as visited
7.          neighborPts = regionQuery(P,eps)
8.          If sizeof(NeighborPts) < minPts
9.              Mark p as NOISE
10.         Else
11.             C = next cluster
12.             expandCluster(p, NeighborPts, C, eps, minPts)
```

Algorithm 3.2 expandCluster(p, NeighborPts, C, eps, minPts)

```
1.  Add p to cluster C
2.  For each point p' in NeighborPts
3.      If p' is not visited
4.          Mark p' as visited
5.          NeighborPts' = regionQuery(p', eps)
6.          If sizeof(NeighborPts') >= minPts
7.              NeighborPts = NeighborPts joined with NeighborPts'
8.      If p' is not yet member of any cluster
9.          Add p' to cluster C
```

Algorithm 3.3 regionQuery(p, eps)

```
1.  Return all points within p's eps-neighbourhood
```

3.1.2 Ordering Points To Identify the Clustering Structure (OPTICS) algorithm

OPTICS is also a density based clustering algorithm for spatial data. Its basic approach is similar to that of DBSCAN, but DBSCAN algorithm does not work well over clusters with different densities, while OPTICS can [1]. The algorithm for OPTICS is given below [10] [17].

Algorithm 3.4 OPTICS

Input:

- D: Dataset
- Eps: radius for neighbouring points
- minPts: minimum points within Eps distance for a point to become a core-point.

```
1.  Function optics(D, Eps, minPts)
2.          for each unprocessed point p of D
3.                  N = getNeighbors(G,p, eps)
4.                  mark p as processed
5.                  p.reachability-distance = UNDEFEINED
6.                  Set core-distance of p
7.                  output p to the ordered list
8.                  if (p.core-distance != UNDEFINED)
9.                          Seeds = empty priority queue
10.                         update(N, p, Seeds, eps, Minpts)
11.                         for each next q in Seeds
12.                                 N' = getNeighbours(G,q, eps)
13.                                 mark q as processed
14.                                 Set core-distance of q
15.                                 output q to the ordered list
16.                                 if(q.core-distance != UNDEFINED)
17.                                         update(N', q, Seeds, eps, Minpts)
```

Algorithm 3.5 update(N', q, Seeds, eps, Minpts)

```
1. Function update(N,p,Seeds,eps,Minpts)
2.         core_dist = p.core-dist
3.         For each o in N
4.                 If (o is not processed)
5.                         new-reach-dist = max(core_dist, dist(p,o))
6.                         If (o.reachability-distance == UNDEFINED)
7.                                 o.reachability-distance = new-reach-dist
8.                                 Seeds.insert(o, new-reach-dist)
9.                         Else
10.                                If (new-reach-dist < o.reachability-distance)
11.                                        o.reachability-distance = new-reach-dist
12.                                        Seeds.move-up(o, new-reach-dist)
```

Algorithm getNeighbours(G,p,eps)

1. Return all points within p's eps-neighbourhood

We can then extract clusters from this obtained list.

3.2 Distance Metrics

Distance metrics are used to compute the distance between two time series.

3.2.1 Euclidean Distance

The Euclidean distance or Euclidean metric is the straight-line distance between two points. Euclidean and Manhattan distances are two most popular distance metrics to measure similarity between the data items of a cluster.

If $p = (p_1,p_2,p_3,\ldots,p_n)$ and $q = (q_1,q_2,q_3,\ldots,q_n)$,

Then distance is defined as -

$$\boldsymbol{d(p,q)} = \sqrt{(\mathbf{p_1} - \boldsymbol{q_1})^2 + (\mathbf{p_2} - \boldsymbol{q_2})^2 + \cdots + (\mathbf{p_n} - \boldsymbol{q_n})^2} = \sqrt{\sum_{i=1}^{n} (\mathbf{p_i} - \boldsymbol{q_i})^2}$$

These distance metrics assumes that the i^{th} point of one sequence is aligned with the i^{th} point in the other as shown in Figure 3.0.1 which produces a pessimistic dissimilarity measure [3].

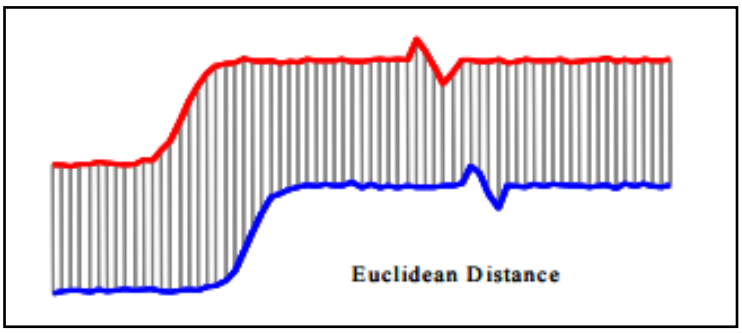


Figure 3.0.1 - Euclidean distance series alignment

On the other hand, a non-linear alignment allows more intuitive distance measure to be calculated [3].

3.2.2 Dynamic Time Warping

Dynamic Time Warping or DTW algorithm is a dynamic programming approach to find the similarity between two temporal sequences which may vary in speed. it This algorithm is useful in detecting similarity where series change with time like the walking pattern of two people. Any series which can be converted to linear can be analysed using DTW [12].

It computes the optimal match between the given sequences by warping them non-linearly in time dimension as in Figure 3.0.2. In spite of its computational lethargy, it is superior to Euclidean distance for classification and clustering of time series [3]. Algorithm to compute DTW distance is shown in Figure 3.0.3.

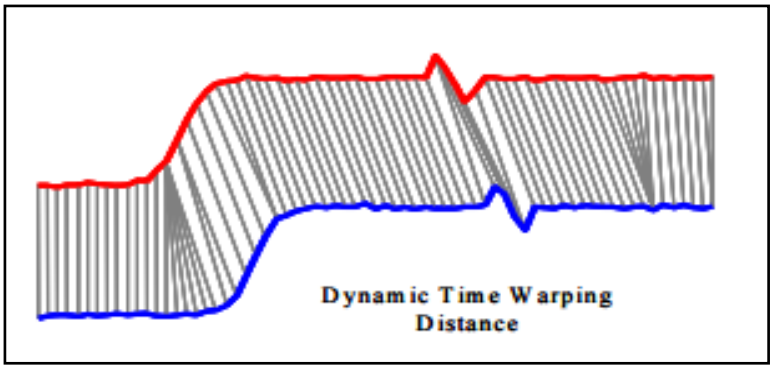


Figure 3.0.2 - DTW distance series alignment

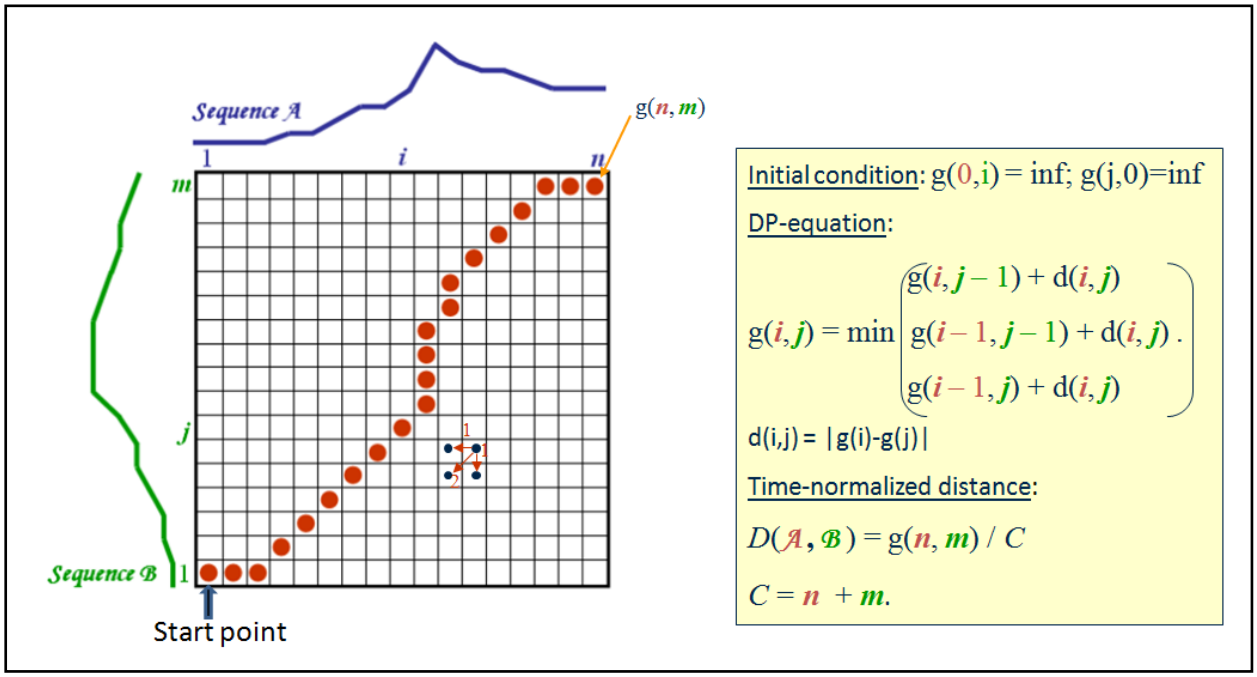


Figure 3.0.0.3 – Dynamic Time Warping algorithm

3.2.3 Haversine formula

It computes the great-circle distance between two points on a sphere ignoring ellipsoidal effects. It gives the shortest distance over the earth's surface and is computed by the following formula [13]-

$$a = sin^2\left(\frac{\Delta\emptyset}{2}\right) + cos\emptyset_1 \cdot cos\emptyset_2 \cdot sin^2\left(\frac{\Delta\gamma}{2}\right)$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

Where, $\emptyset_1 and\ \emptyset_2$ are the latitudes (in radians), $\Delta\emptyset$ is the difference in latitudes (in radians), $\Delta\gamma$ is difference in longitudes (in radians), R is earth's radius (= 6371km).

Chapter-4

Problem Statement

This chapter formally defines the problem of recommending pickup hotspots during different times of the day. Input is a Time Aggregated Graph (TAG) and output will be the locations along with the time-intervals during which these locations are in high-demand.

We start by explaining the problem statement by detailing the input and output followed by an example.

4.1 Input

Input is the demand of taxi services in Singapore during different times of the day. The data files are in .csv format and each of them contains information about the pickups made on a single day starting from 12:00:00AM till 11:59:59PM. Subset of fields from these data files are shown in Table 4.1, where StartDate and StartTime tells when the trip begins and EndDate and EndTime tells when it ends. Pickup and drop locations are in the form of latitude and longitude as StartLat, StartLon, EndLat and EndLon.

ID	StartDate	StartTime	EndDate	EndTime	StartLon	StartLat	EndLat	EndLon
6130B1201150000	15/01/2012	00:00:00	15/01/2012	00:20:00	103.89291	1.37295	103.83627	1.42011

Table 4.1- Trip file format

Using this file and the road-network of Singapore (available as Singapore.osm), we have created a time-aggregated graph (Section 6.1), which will be given as an input to the algorithm.

4.2 Output

Based on spatial and temporal aspects, we will recommend locations for taxi stands. Each recommendation will also have a recommended time with it, which will tell the taxi drivers that when it will be most profitable for them to reach a particular recommended place. In simple words, we study the hourly pickup pattern over a period of days and then cluster the points based on their location and pickup trend followed during each hour of the day.

As the demand changes with time, the location will also change. It will be highly beneficial for taxi drivers as their average waiting time for the next ride can be minimized by reaching the nearest pickup-hotspot during that time at the earliest.

4.3 Example

We show an example to give the reader a visual idea of how things will work. Consider the following graph –

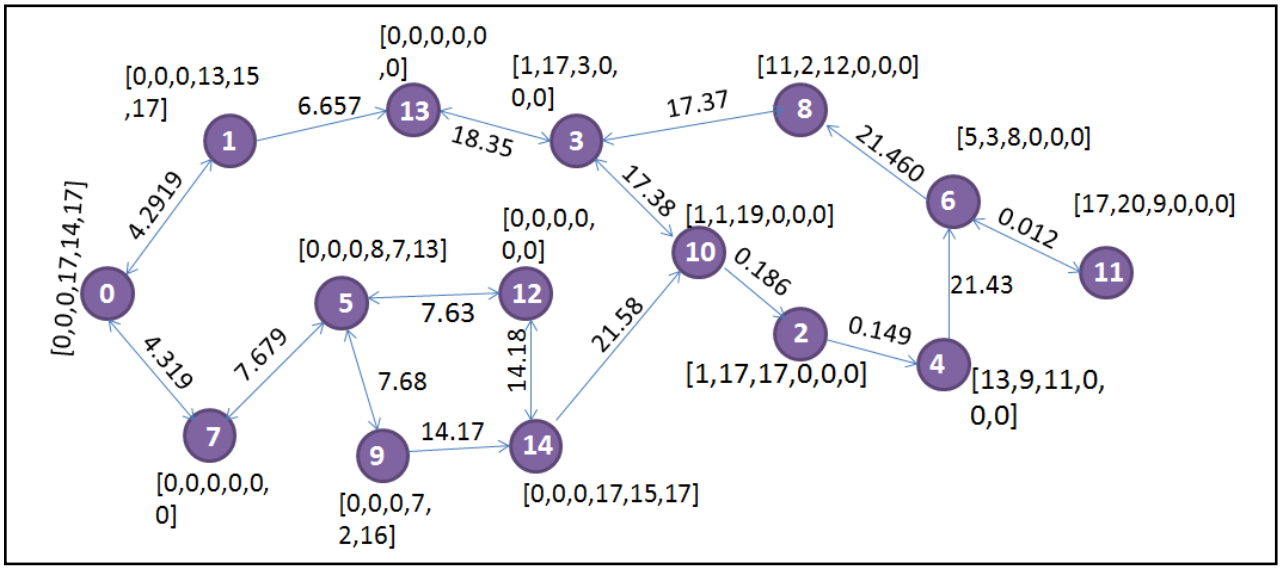


Figure 4.0.1 - Example of input Time Aggregated Graph

Let Figure 4.0.1 is the created graph for some country after processing the data as explained in Section 6.1. Vertices are the road-end points and edges are the directions in which they are connected. Edge weights are the cost to travel between the connected vertices and the list of six values at each vertex denote the number of pickups made during each hour from that point assuming that taxis are working for six hours only. After applying the designed algorithm, we'll get clusters like in Figure 4.0.2.

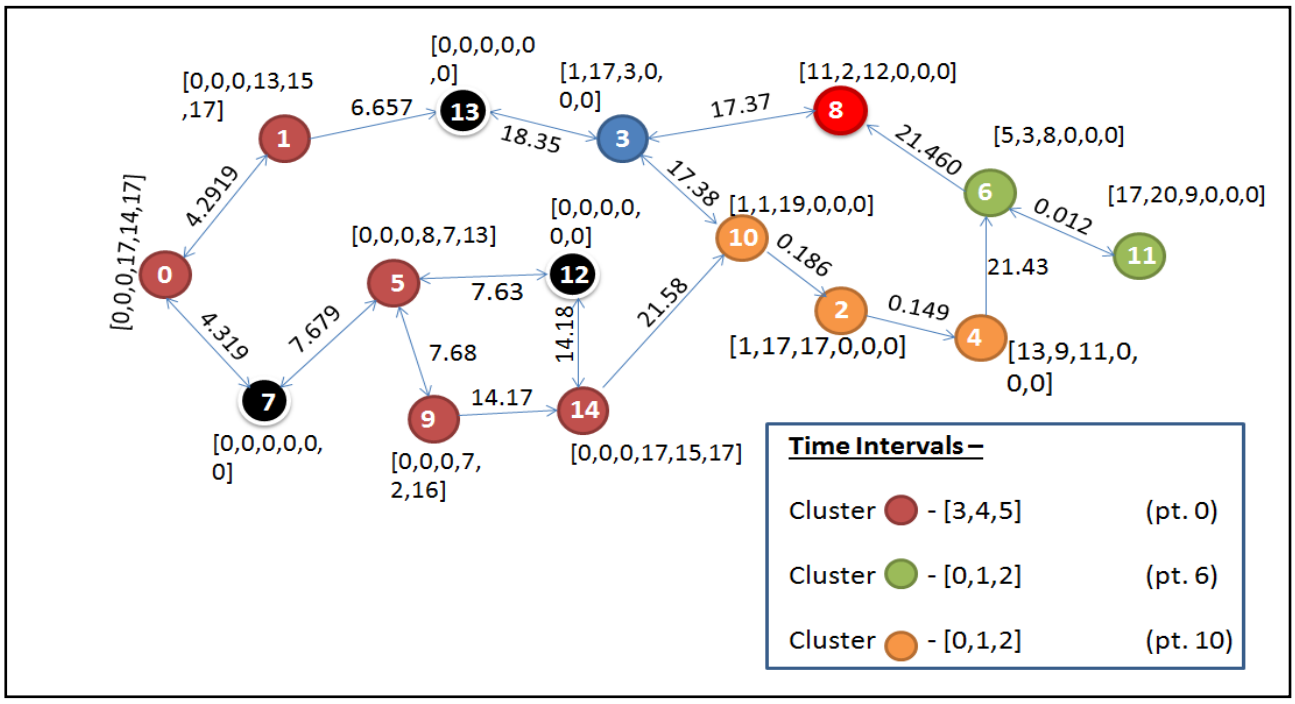


Figure 4.0.2 - Output graph

Figure 4.0.2 shows the noise points in black and clusters formed by all other colours. We are considering only six time intervals which have been mapped from 0-5. Final recommendation consists of cluster-maroon with a time recommendation from 3:00:00AM-5:59:59AM; cluster-green from 12:00:00AM-2:59:59AM and cluster-orange from 12:00:00AM-2:59:59AM. Blue and red clusters have been formed but they won't be recommended. Execution trace for this example has been explained in section 5.3.

Next we compute the centroids for each cluster which will be the recommended location and map them on the city's map.

Chapter-5

Proposed Approach

In this chapter we explain the details of proposed solution. We first provide the key idea of ST-OPTICS and later explain the detailed algorithm. Lastly, we provide an execution trace of the algorithm with an example.

5.1 Key idea of the algorithm

To run ST-OPTICS algorithm, we first create a Time Aggregated Graph (TAG) and then perform clustering based on spatial closeness and temporal similarity. Our algorithm is a combination of DTW and OPTICS, modified to work on a directed weighted graph. While OPTICS requires two parameters, eps and minPts, clustering by ST-OPTICS requires three parameters, eps, minPts and dtw_threshold. Neighbourhood set for a point will contain points which are both, spatially as well as temporally similar making it work on spatial-temporal data.

After forming the clusters, we run an algorithm to compute the timings during which each of these places will be pickup hotspots. Final recommendation consists of location along with the timings after pre processing the obtained results.

5.2 Details of the algorithm

5.2.1 Spatial Clustering

We modified the existing OPTICS algorithm to work on the created graph for clustering spatial points. There are mainly two parameters used in OPTICS eps and minPts (explained in section 3.1.2). We modified how neighbourhood set was being formed with respect to these parameters.

5.2.1.1 Modifications to the existing algorithm

- ST-OPTICS finds the neighbouring points using Dijkstra's algorithm on the created graph. In simple words, we are finding spatially close points via shortest path algorithm on road network, since edge weights are the road distances computed using Haversine Formula (explained in section 3.2.3). In order to reduce space and time complexity, we modified Dijkstra's algorithm. Instead of traversing and inserting all the nodes in our priority queue, as done in traditional Dijkstra's, we insert only those which lie within a distance of eps from the source node.
  This can be explained by a simple example. Let's assume that we run Dijkstra's algorithm on a graph with eps distance of 2km. The next vertex we're about to traverse has a distance of 2.5km from the source vertex. Since we traverse the vertices in the increasing order of their distance from the source node, it is obvious that the remaining vertices will have a minimum distance of 2.5km which is greater than eps distance. So instead of traversing all the remaining points, we halt the algorithm here and form the set of spatially close points by the already traversed vertices. The algorithm for modified Dijkstra's to find the neighbourhood set is in getNeighbours function given in Algorithm 5.3.

- MinPts is the minimum number of points that should lie within Eps distance from the source vertex for it to become a core point [16]. Since each vertex contains a list with number of pickups made from there, we modified the existing definition of minPts by considering it as the total pickups made from the points which lie within Eps distance. For instance, we have three points A, B and C within eps distance of X with the total pickup count of 30, 20 and 15 respectively during the entire day, making a total of 65 pickups. If MinPts or threshold value was 50, and since 65>50, X becomes a core-point. If total pickups made would have been less than 50, then X would be categorized as a noise point and we would have moved on to the next point in the database.
  Considering these parameters, our neighbourhood set will be formed based on spatial aspect. Eps distance will take care of spatial closeness, while minPts will make sure that the area has sufficient number of pickups made to form a cluster.

- We modified the **update function** (algorithm 3.5) for it to work on a directed graph properly. The need for this modification along with the change made can be explained via the following example. Assume Eps to be 4km.

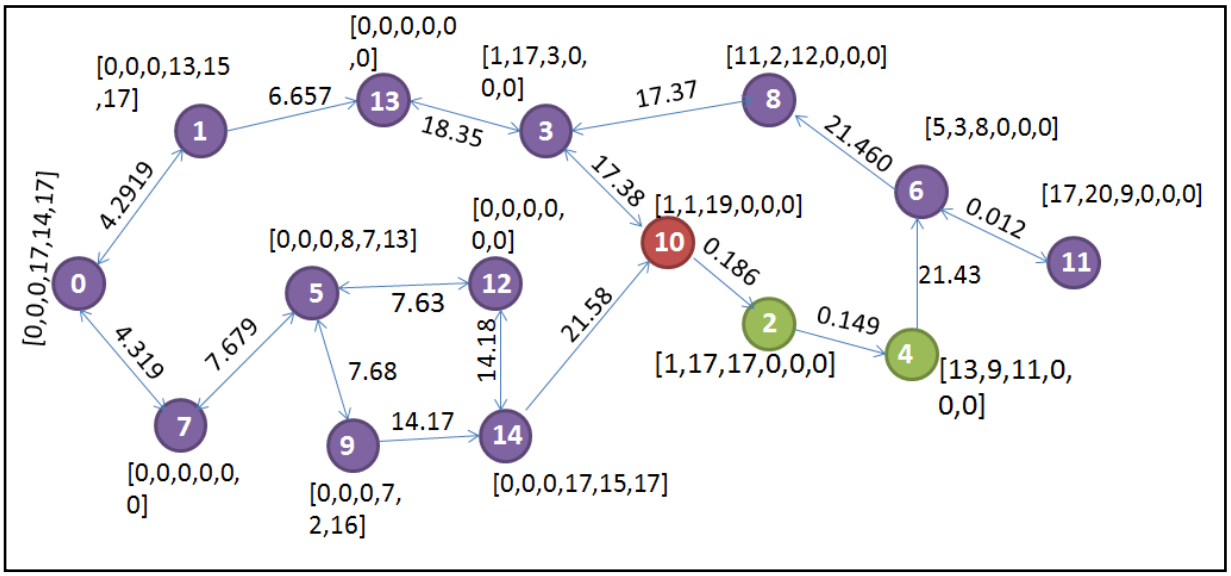


Figure 5.0.1 - Understanding modified update function

Assume we start forming a cluster from node2. This node will be marked processed and its reachability distance will be set to undefined. The neighbourhood set will contain node4. We stop over there since node6 has a distance of 21.43km which is greater than Eps distance (4km). If node10 is the next chosen point from the dataset, node2 is its neighbour. If it becomes a core-point, then it'll not be beneficial to recommend two nearby locations. Had we started from node10, then node10, node2 and node4 could have formed a cluster. To take care of such cases, we modified the existing update function as:

Reachability distance of node10 will be set to undefined. Distance of node2 will be changed to maximum of core-distance and distance between node10 and node2. Node2 and node4 (if there in ordered list) will be placed after node 10.

Please note that this modification will only hold if node10 is the starting node and has exactly one neighbour, which is node2 in this case. This pseudocode for this modified algorithm is in Algorithm 5.4.

5.2.2 Temporal Clustering

While recommending a pickup place to anyone, temporal aspect, that is, time is very important. What if you have a cluster with points which are spatially close, but with completely different pickup patterns? It won't make sense to group points under one cluster if we won't be able to find any common pickup patterns from them. Thus, we need clusters which are spatially as well as temporally similar to find some interesting pickup trends followed at different places.

ST-OPTICS uses Dynamic Time Warping to compute temporal similarity. The final neighbourhood set will be an intersection of spatial as well as temporal neighbourhood set.

5.2.2.1 Modifications to the existing algorithm

- DTW threshold - We have used one parameter, named *DTW threshold* to detect if series are temporally similar or not. After computing the neighbourhood set of spatially close points, we compute the DTW distance between the source vertex and the points lying in this set. Points having distance smaller than the threshold value stay and rest are removed. This forms the final neighbourhood set, after which it is checked if the source point is a core-point or not.

- Penalty – it contains binary values (0 or 1) for each neighbouring point. While computing the neighbourhood set, if a point lays very close, say within 750m from the source node, we assign it a value of 1, else 0. While computing the DTW distance, points with the penalty value as 1 will get their DTW distance reduced to 60% of the current value. This is because 750m means that points are very close. So we can reduce the impact of temporal dissimilarity as even if the pattern is a bit different, the driver can cover this small distance. Also, it won't affect the location of the recommended point much and it'll be useless to recommend a location just a few meters away.

5.2.3 Complete algorithm for spatial-temporal clustering

Algorithm 5.1 is the complete modified algorithm which works on spatio-temporal data.

Algorithm 5.1 SPATIO_TEMPORAL

Input:

- G: Graph created after pre-processing the data
- Eps: radius for neighbouring points
- minPts: minimum points within Eps distance for a point to become a core-point.
- dtw_threshold: maximum DTW distance between a node and its source node.

Output: Clusters based on spatial and temporal closeness.

1. Procedure SPATIO_TEMPORAL
2. ResultList = ST-Optics(G, Eps, minPts, dtw_threshold)

End procedure

Algorithm 5.2 ST-Optics

```
1.  Function ST-Optics (G, Eps, minPts, dtw_threshold)
2.      for each unprocessed point p of DB
3.          N = getNeighbors(G,p, eps)
4.          For each n in N
5.              dtw_distance = dtw_distance(n,p)
6.              if (penalty for n is 1)
7.                  dtw_distance = dtw_distance*0.6
8.              if (dtw_distance > dtw_threshold)
9.                  Remove n from N
10.         mark p as processed
11.         p.reachability-distance = UNDEFINED
12.         set core-distance of p
13.         output p to the ordered list
14.         if (p.core-distance != UNDEFINED)
15.             Seeds = empty priority queue
16.             ModifiedUpdate(N, p, Seeds, p.core-distance)
17.             for each next q in Seeds
18.                 N' = getNeighbours(G,q, eps)
19.                 For each n in N'
20.                     dtw_distance = dtw_distance(n,q)
21.                     if (penalty for n is 1)
22.                         dtw_distance = dtw_distance*0.6
23.                     if (dtw_distance > dtw_threshold)
24.                         Remove n from N'
25.                 mark q as processed
26.                 Set core-distance of q
27.                 output q to the ordered list
28.                 if(q.core-distance!= UNDEFINED)
29.                     ModifiedUpdate(N', q, Seeds, q.core-distance)
```

Algorithm 5.3 getNeighbours : finds the neighbourhood set of a point or a vertex

Input:

- G: weighted graph
- p: source vertex
- eps: radius for neighbouring points

Output: Neighbourhood set of p

```
1.  function getNeighbours(G, p, eps)
2.      create a list visited and a priority queue with sorted distance values named
        min_dist
3.      Insert p into min_dist
4.      while(min_dist.size()>0)
5.          q= extract first node from min_dist
6.          Add q to visited
7.          prev_dist = q.distance
8.          N' = list of vertices which are directly connected to q
9.          for each i in N'
10.         if (!visited.contains(i))
11.             dist = prev_dist+computeDistance(q,i)
12.             if (dist<eps && !neighbour.contains(i))
13.                 Insert a with all its details into min_dist
14.                 Insert a in neighbour
15.             else if (dist<eps && neighbour.contains(i))
16.                 if (neighbour.get(i).distance>dist)
17.                     Update distance of i in neighbour
18.                 Insert a with all its details into min_dist
19.         Insert q to visited
20.         Remove q from min_dist
21.     Return neighbour
```

Algorithm 5.4 ModifiedUpdate : traverses the neighbourhood set and update details in Seeds

Input:

- neighbour: Neighbourhood set
- src: Source vertex
- seeds: details about nodes forming a cluster
- core_dist: core distance

Output: updated *seeds* list

```
1. function ModifiedUpdate(neighbour, src, seeds, core_dist)
2.         for each o in neighbour
3.                 new_reach_dist = max(core_dist, distance computed above from src to
   o)
4.                 if (o is not processed)
5.                         if (o.reachability_distance == UNDEFINED)
6.                                 o.reachability_dist = new_reach_dist
7.                                 Insert o with complete details in seeds
8.                         else
9.                                 if (o.reachability_dist> new_reach_dist)
10.                                        Update distance of o to new_reach_dist in seeds
   and distance list
11.                 else if (o is processed && neighbour.size==1 &&
   src.reachability_distance == UNDEFINED)  //check that if src is the starting node
12.                         if (o.reachability_distance == UNDEFINED)
13.                                 Update distance of src to new-reach-dist and extract
   nodes till next undefined value in ordered list. Add these nodes after the current node.
14.         return seeds
```

Ordered list contains node id, reachability distance and core distance of each node. We then extract the clusters and find the timings during which the clusters in *ordered list* will be in demand.

5.2.4 Recommended time intervals

Assume that you get a list of places which are pickup hotspots, but you have no idea about the time. No driver can wait all day at a place for passengers. Along with the location, only if they get an idea about the timings too, will a recommendation be useful. Timings recommended should be those, during which the pickups made are high.

5.2.4.1 Parameters used

We have three parameters which define a pickup hotspot. They help decide whether a particular time slot should be recommended or not based on the cluster size and number of pickups made.

- Time_threshold - this is the minimum number of pickups which should be made from a cluster during an hour, for that particular time to get recommended.

- alpha – this parameter changes the time threshold for different clusters. Having a same threshold value for two clusters A and B with five and fifty pickup locations respectively would be unfair. For example, if threshold value is 20, then 20 pickups made in an hour from cluster A with five nodes is a significant count. But 20 pickups made from a cluster with fifty nodes is negligible as the size of the cluster will also be large. No taxi driver will prefer waiting near cluster B. To handle this, we have used a parameter named alpha, which increases the threshold value as per the number of nodes in a cluster.

- Max_points – these are the maximum number of points in a cluster beyond which new time_threshold value will be computed.

  New time threshold will be -

New time threshold = time_threshold + (alpha* (total_nodes-max_points)/max_points)

Example: If number of points in a cluster are 30, time_threshold is 15, alpha is 2 and max_points are 10 nodes, then-
As total points>10 (max_points), so we compute the new time threshold value as –
(15 + (2* 20 / 10)) = 19. Thus intervals (each of one hour) during which the pickup-count >=19, will get recommended. If no such time interval exists, the cluster will get discarded.

5.2.4.2 Algorithm

- We run a while loop over all the clusters and find the total pickups made during each hour.
- Time_threshold value will be updated based on the number of points in the cluster.

- Whichever intervals cross the time threshold value, will get added to the recommended timing list.
- For instance, we have a cluster with three points namely A, B and C with pickup lists as [1,5,6,0.3], [2,6,6,1,2] and [1,4,7,0,1] respectively. Suppose the pickups are from 12:00:00AM till 4:59:59AM. If time_threshold value is 10 and no update is needed, then only 1:00:00AM-2:59:59AM will be recommended, as the pickups made during these timings from all the points crosses our threshold value.

Algorithm 5.5 TimeInterval

Input:

- Detail about final clusters, including all the nodes with detail about total pickup during each hour.
- time_threshold: minimum pickups made during a time interval for it to get recommended.
- Alpha: increase in time_threshold value depending on the number of nodes in a cluster.
- Max_points: the maximum points after which new threshold value will be computed.

Output: Recommended time for each cluster

```
1. function TimeInterval (clusters with pickup details, time_threshold, alpha, max_points)
2.         for i in range (0, cluster_number)
3.                 if(total_pickups > min_pickups)
4.                         time_threshold=time_threshold+(alpha*(total_pickups-max_points)/max_points
5.                 for j in range (0, elements_in_cluster_i)
6.                         for k in range (0,23):
7.                                 Sum[k] : sum[k]+element[j].no_pickup.get(k)
8.                 for k in range(0,23):
9.                         if (sum[k]> time_threshold)
10.                                Add k to recommended_list for cluster i
11.                Sum = new ArrayList();
12.        return recommended_list
```

The complexity of this algorithm will be O(n^2).

5.2.5 Post processing the results

Some post processing has been done after getting the time recommendations for refining the results.

Case 1: if some clusters do not behave like pickup-hotspots at all during the day-

In case we get no time recommendation for some cluster, which means either this cluster is an outlier or a few points with very less pickups during each hour have been grouped together. In this case, we will discard the cluster.

Case 2: if some cluster gets time recommendation for just one hour during the entire day-

To refine our results and improve its quality, we have discarded those clusters which are recommended for just one hour during the entire day.

5.2.6 Application of ST-OPTICS in shared taxis

This algorithm can be used for personal as well as shared taxis. Personal taxi driver can get a recommendation based on his current location and time. In case of shared taxis, the driver can get his first ride around the recommended location. As soon as he gets a passenger, he'll have details about the drop location. So we can recommend hotspot locations lying near to the path taken. Since we have locations along with timings, based on the current time we can easily suggest if there are any nearby pickup places. This will increase their probability of finding a passenger, thus increasing the money earned.

5.3 Working Example

Input – Graph created by processing the data obtained from osm file and trip files as explained in Section 6.1.

Consider the graph shown in Figure 5.0.2. We have to find different pickup hotspots along with their active hours.

Let eps = 15km, minPts = 50, dtw_threshold = 25, time_threshold = 15, alpha = 0.2, max_points = 5.

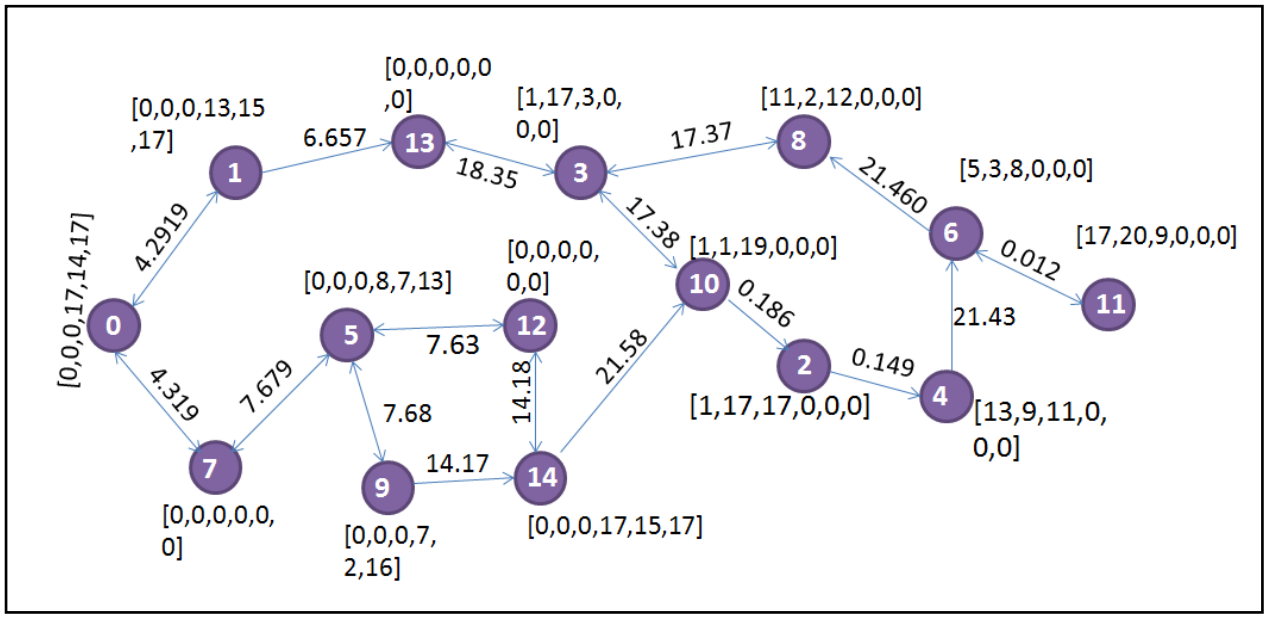


Figure 5.0.2 - Working Example

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
u	u	u	u	u	u	u	u	u	u	u	u	u	u	u
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 4.4.1 - Reachability Distance

Table 4.4.2 - Ordered List

Step 1: Start with node0. Neighbourhood set, N = [node1, node7, node13, node5]. Update the penalty table while computing distance for the neighbouring points. As no point lies within 750m from node0, so penalty will be 0 for all.

Step 2: DTW distances are as follows-

- dtw(node0, node1) = 5.0
- dtw(node0, node7) = 48
- dtw(node0, node13) = 48
- dtw(node0, node5) = 20

As threshold is 25, thus node7 and node13 get discarded. Thus, N = [node1, node5]

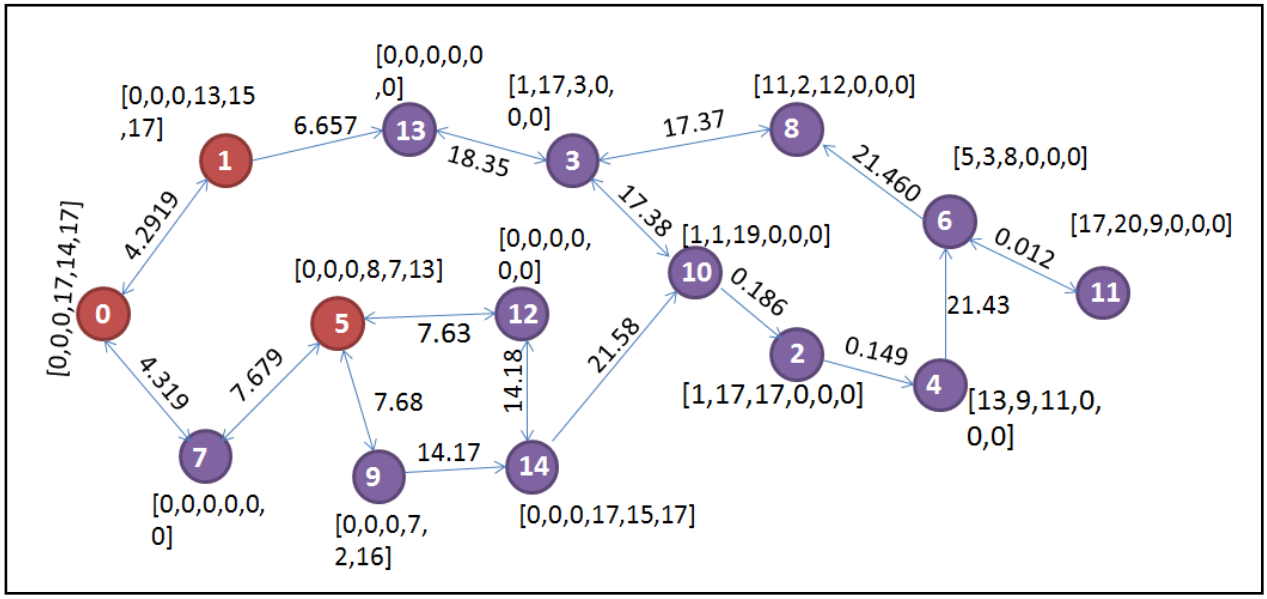


Figure 5.0.3 – Intermediate Cluster-0

Step 3: mark node0 as processed and set its reachability-distance to UNDEFINED.

Step 4: Update its core-distance add it to the ordered list.

Reachability Distance

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p	u	u	u	u	u	u	u	u	u	u	u	u	u	u
UND	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Ordered List

node0														

Step 5: Check if its core-distance is undefined or not. In this case, we have a total pickup count = 121>minpts. Thus node0 is a core-point with core distance of 4.2919km.

Step 6: Now create a new list named Seeds and update it with values obtained by expanding N.

Seeds

- Core distance = 4.2919.
- Run *for loop* and traverse each of the vertices in N.
  - v = node1.
    - Node1 is unprocessed, so we set its reachability distance to maximum of core distance and its distance from node0, that is, max(4.2919, 4.2919) which is 4.2919. Add it to seeds list.
  - v = node5.
    - We check if it's processed or not. Since it is unprocessed, we set its reachability distance to max (4.2919, 11.998) = 11.998.

Seeds

node1, (4.2919)	node5, (11.998)

- Seeds list has two values after the update function. We then run for loop on Seeds list to expand our cluster by expanding the neighbouring points.
- n = node1.
  - Its neighbourhood set, N' = [node0, node13, node7]
  - Compute DTW distance of points in N' with node1.
    - dtw (1,0) = 5.0
    - dtw (1,13) = 45.0
    - dtw (1,7) = 45.0
  - As dtw(1,13) and dtw(1,7) are greater than the threshold value, they'll be discarded. Updated N' = [node0].
  - We mark node1 as processed and find its core-distance.
  - Then we add it to the ordered list.

Reachability Distance

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p	p	u	u	u	u	u	u	u	u	u	u	u	u	u
UND	4.2919	-	-	-	11.998	-	-	-	-	-	-	-	-	-

Ordered List

node0	node1													

- Since core distance of node1 is 4.2919km and not undefined, we'll update seeds list with N'.
- Run for loop and traverse all nodes in N'.
  - a = node0.
    - ➔ Since node0 is processed, we'll check if size of N' is 1 and the reachability-distance of node1. As reachability-distance is not undefined, so move on.

- n = node5.
  - Neighbourhood of node5, N' = [node12, node7, node9, node0].
  - Compute the DTW distance of node5 with all nodes in N'.
    - dtw(node5, node12) = 28.0
    - dtw(node5, node7) = 28.0
    - dtw(node5, node9) = 9.0
    - dtw(node5, node0) = 20.0

  - Thus, N' = [node9, node0] as node12 and node7 get discarded.
  - Mark node5 as processed and find its core-distance.
  - Add it to the ordered list.

Reachability Distance

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p	p	u	u	u	p	u	u	u	u	u	u	u	u	u
4.2919	4.2919	-	-	-	11.998	-	-	-	-	-	-	-	-	-

Ordered List

node0	node1	node5												

- Core distance of node5 is 7.6867, so we again call the update function on N'.
- Traverse all neighbours:
  - a = node9.
    - ➔ Its reachability distance is max (7.6867, 7.6867) = 7.6867.
    - ➔ Since the node is unprocessed and not in seeds, we update its reachability distance to 7.6867 and add it to seeds list.

  - a = node0.
    - ➔ Reachability distance = max(11.9991, 7.6867) = 11.9991
    - ➔ As node5 has more than one neighbour, we leave it as it is.

Seeds

node1, (4.2919)	node5, (11.998)	node9, (7.6867)

- n = node9
  - Neighbourhood of node9, N' = [node5, node14].
  - Compute the DTW distance of node9 with all nodes in N'.
    - dtw(node9, node5) = 9.0
    - dtw(node9, node14) = 12.0

- As both these values are less than dtw_threshold value, N' = [node5, node14].
- Mark node9 as processed.
- Find its core-distance and add it to the ordered list.

Reachability Distance

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p	p	u	u	u	p	u	u	u	p	u	u	u	u	u
UND	4.2919	-	-	-	11.998	-	-	-	7.6867	-	-	-	-	-

Ordered List

node0	node1	node5	node9											

- Core distance of node9 is 7.6867, so we again call the update function on N'.
- Traverse all neighbours:
  - a = node5.
    - ➔ Reachability distance = max(7.6867, 7.6867) = 7.6867
    - ➔ Since the node is already processed and its distance is not undefined, we leave it as it is.

  - a = node14.
    - ➔ Reachability distance = max(14.1733, 7.6867) = 14.1733
    - ➔ Since the node is unprocessed and not in seeds, we update its reachability distance to 14.1733 and add it to seeds list.

Seeds

node1, (4.2919)	node5, (11.998)	node9, (7.6867)	node14, (14.1733)

- n = node14.
  - Neighbourhood of node14, N' = [12]
  - Compute the DTW distance of node9 with nodes in N'.
  - dtw(node9, node12) = 49.0

  - As 49.0 is greater than dtw_threshold value, so it'll be discarded. N' = [].
  - Mark node9 as processed and add it to the ordered list.

- Find the core distance for node14. It is UNDEFINED, so we move on to the next node in seeds list.

As we don't have any other node in Seeds, we stop here. A new unprocessed point is chosen from the graph till all the nodes get processed.

We then extract clusters by assigning a cluster number to points starting from UNDEFINED value till next UNDEFINED based on the distance chosen.

After extraction, the first cluster can look like the one shown in maroon in Figure 5.0.4.

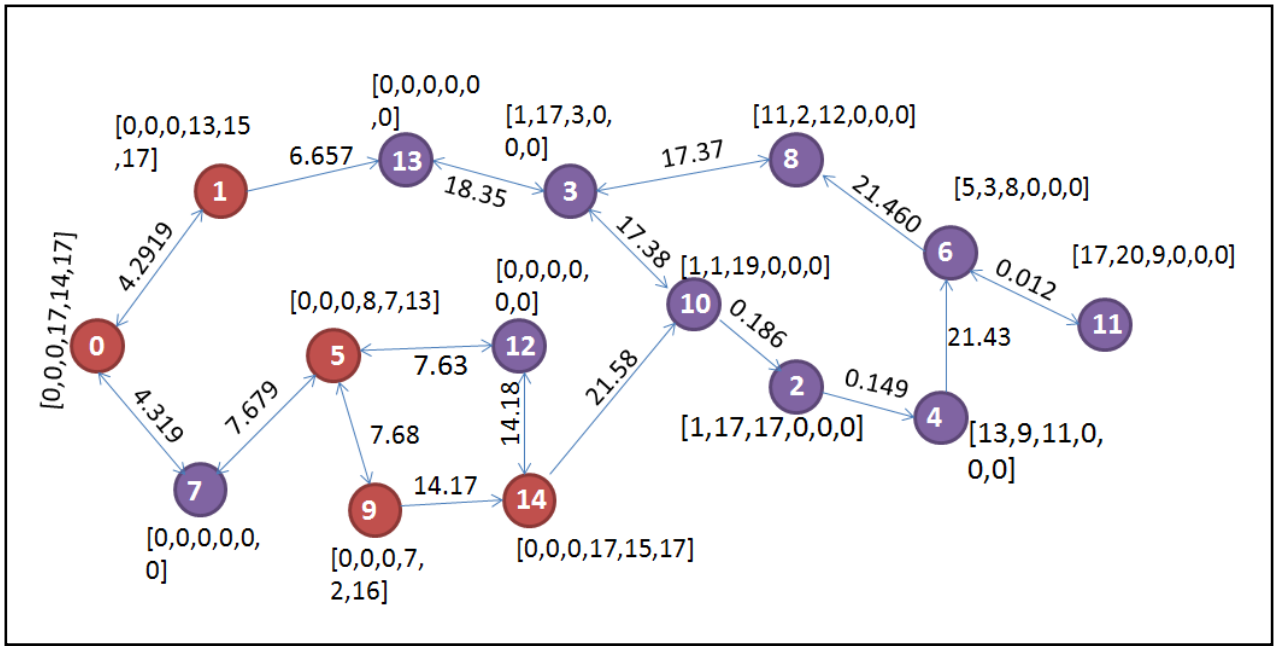


Figure 5.0.4 - Final cluster-0

On completing the algorithm, you'll get the result as follows –

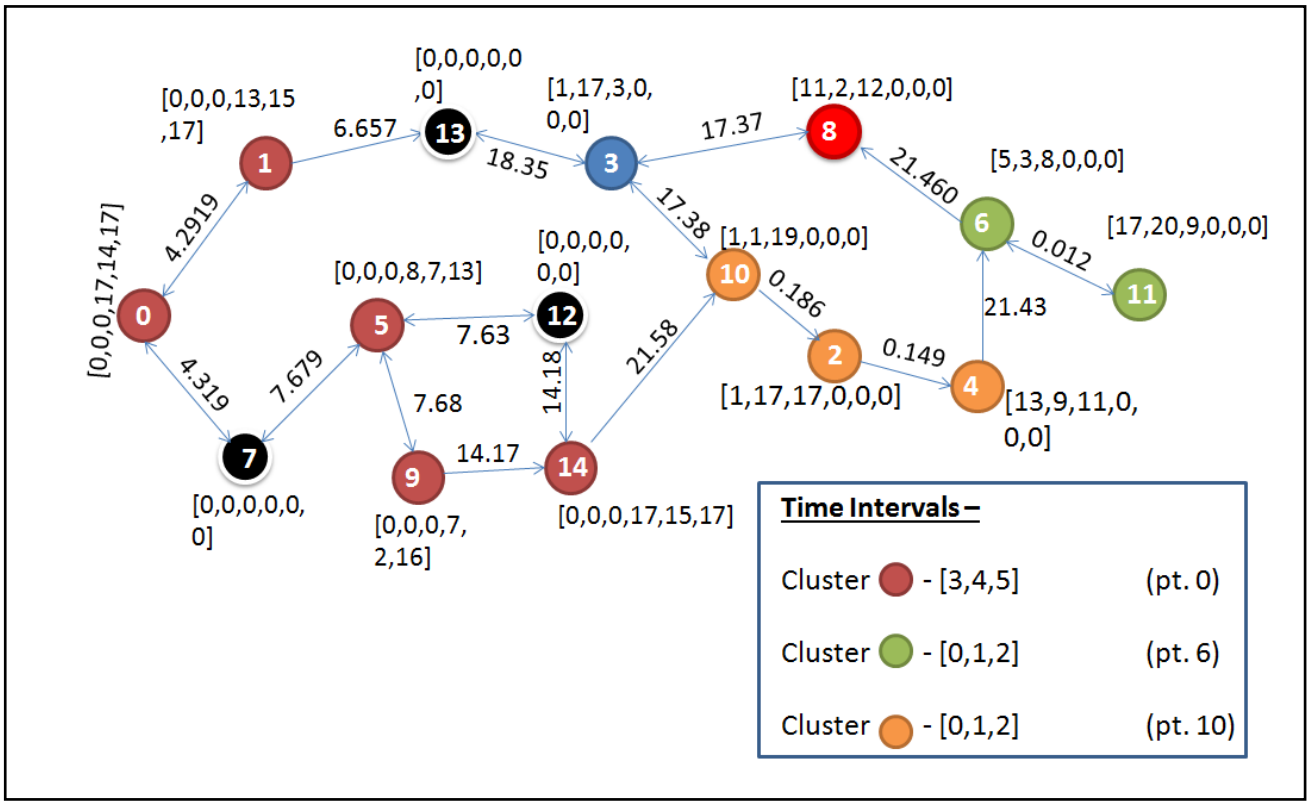


Figure 5.0.5 - Resultant clusters

Figure 5.0.5 shows different clusters by different colours. Black means noise or outliers. Time recommendations in the box have been made as explained in algorithm 5.5. Final recommended list contains cluster-maroon, cluster-green and cluster-orange with the timings from 3:00:00AM-5:59:59AM, 12:00:00AM-2:59:59AM and 12:00:00AM-2:59:59AM respectively.

As time_threshold is 15, we observe that 1:00:00AM-1:59:59AM (2nd time slot) for cluster-blue also gets recommended. But as mentioned in section 5.2.5, clusters with no recommendation and only one slot recommendation will be discarded, thus cluster-red and cluster-blue have been removed from the final list.

Chapter-6

Experimental Evaluation

6.1 Pre-processing the data

This section explains the pre-processing done to create the final Time Aggregated Graph on which ST-OPTICS algorithm worked.

6.1.1 Pre-processing trip files

- Each trip file has details about pickups made from 00:00:00AM till 23:59:59PM.
- Along with latitude and longitude of the location, we store a list of 24 values which contains the pickup counts from a location during each hour as shown in figure 6.1.1. Slots are numbered from 0 to 23.
- Value in a particular slot corresponding to a location keeps incrementing whenever we find a pickup made during that time interval. For instance, if a pickup has been made at 4:15PM, then the value in the fifth slot (starting from 0) corresponding to this particular point gets incremented by one. If this location doesn't already exists in the list, then add a new entry to the list and update the details.

Now we have created a list with details about the pickup locations along with number of pickups made during each hour.

latitude	longitude	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Figure 6.1.1 – Information stored

6.1.2 Pre-processing osm file

6.1.2.1 Mapping GPS locations to actual road-network locations

- We first extract and store all the nodes from the <nodes> tag.
- Then map the points from the list created in section 6.1.1 to the nearest point fetched from <nodes> tag. This is because trip file has locations fetched from GPS, which

might be approximate locations and not lie on road-network and our graph can only have points present on the road-network.

- This will give us a list of locations on road-network, each with twenty-four values containing pickup counts for each hour. If some location is a non-pickup location throughout the day, then it'll contain a list with all 0's.

6.1.2.2 Graph creation

- For creating a graph, we consider <way> tags in the osm file.
- Each way consists of some nodes arranged in the same order as on actual road-network. Detail about the direction, that is if a road is one-way or two-way is also mentioned.
- To make the graph dense, we store only the end-points of each road and map the intermediate nodes to the closest end point. The pickup count list of the intermediate node gets added to the list of the closest end point.
- Each road-end point also carries the information about its connected vertices. It stores the number of connected vertices along with their node ids.
- All these road-end points will be the vertices of the graph. Edges will be unidirectional from start node to end node if it's a one-way and bi-directional otherwise.
- Each edge is associated with a weight which is the Haversine distance or the road distance (explained in section 3.2.3) between the vertices (locations).

To summarize it all, the created graph can be represented as a time-aggregated graph G(V,E) with a set of vertices V = $\{v_1,..,v_{nv}\}$ and a set of edges E = $\{e_1,..,e_{ne}\}$. Each vertex is a road intersection point and contains the following information –

Node id, latitude of node, longitude of node, number of pickups made during each of 24 hours, number of connected vertices and node id of all connected vertices.

Using this data, a road network graph was created which looks similar to the one shown in figure 4.0.2.

6.2 Experiments

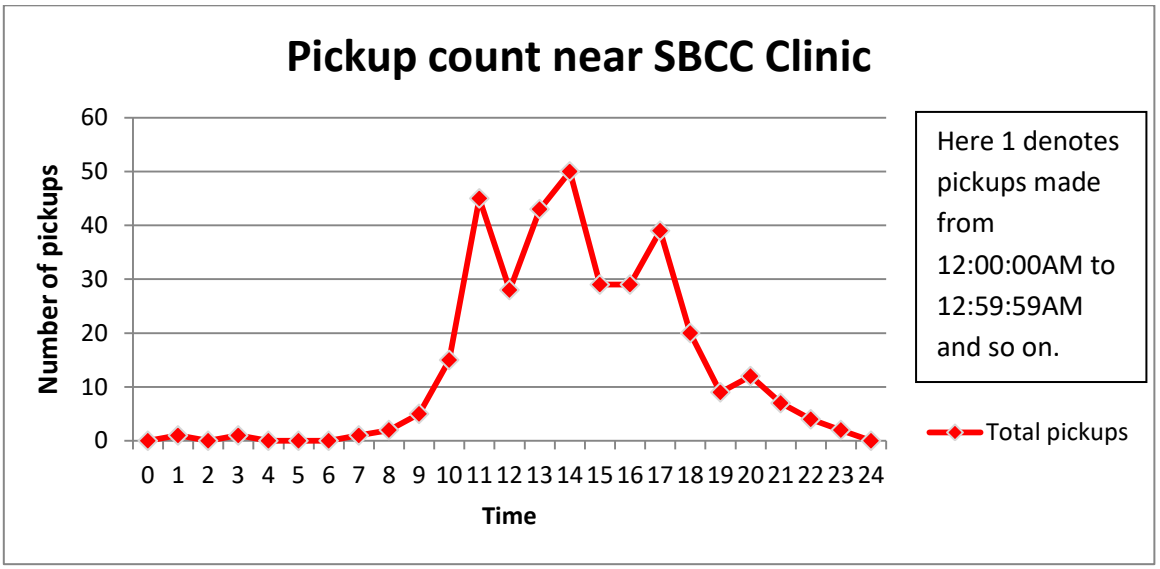
Our main motive is to prove the importance of finding the active hours while recommending a pickup place and show how our algorithm is better than some already existing algorithms.

6.2.1 Importance of time recommendation

The below results are obtained for a period of four days, from Sunday to Wednesday.

1) Recommendation1: Near clinics

Figure 6.2.0.1 - Recommended pickup location near some clinics.



Graph 6.2.1 – Near clinics

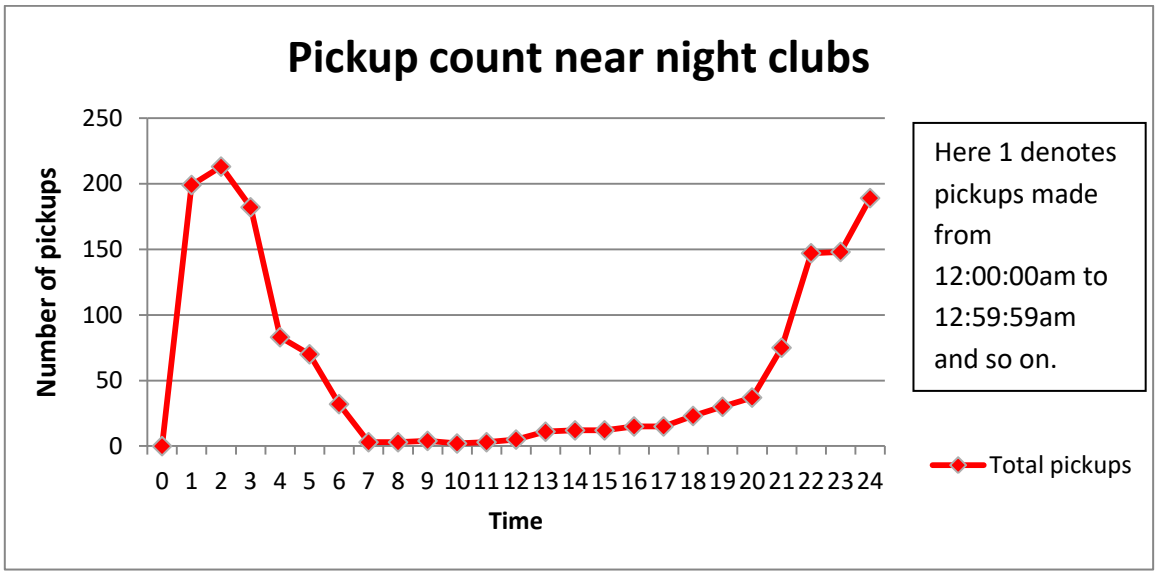
On clustering spatially and temporally similar points, we get a cluster with centroid as shown in Figure 6.2.0.1. It is near to some clinics with a time recommendation from 10AM-11AM, 12PM-2PM and 4PM-5PM. Graph 6.2.1 shows a pickup count from the points clustered in this area. Since many clinics here are closed on Sunday, maximum pickups are made from Monday to Wednesday.

Most of the clinics open in morning from 9AM-12PM and then in the evening around 2PM-5PM while a few like Singapore Sports and Orthopaedic clinic are open from 9AM-6PM. So from the pickup trend, we observed that the most rush hours are between 10AM-1PM and then 4PM-5PM. From the graph, we also see that descent number of pickups have been made during 2PM-4PM, while there are hardly any pickups made after 6pm.

As the pickup count is high around 1PM and most of the clinics get closed around 12.30PM, many doctors might also be taking cabs to go back home or some other place after attending their patients here.

2) Recommendation2: Near clubs

Figure 6.2.0.2 - Recommended pickup location near restaurants and night clubs



Graph 6.2.2 – Near night clubs

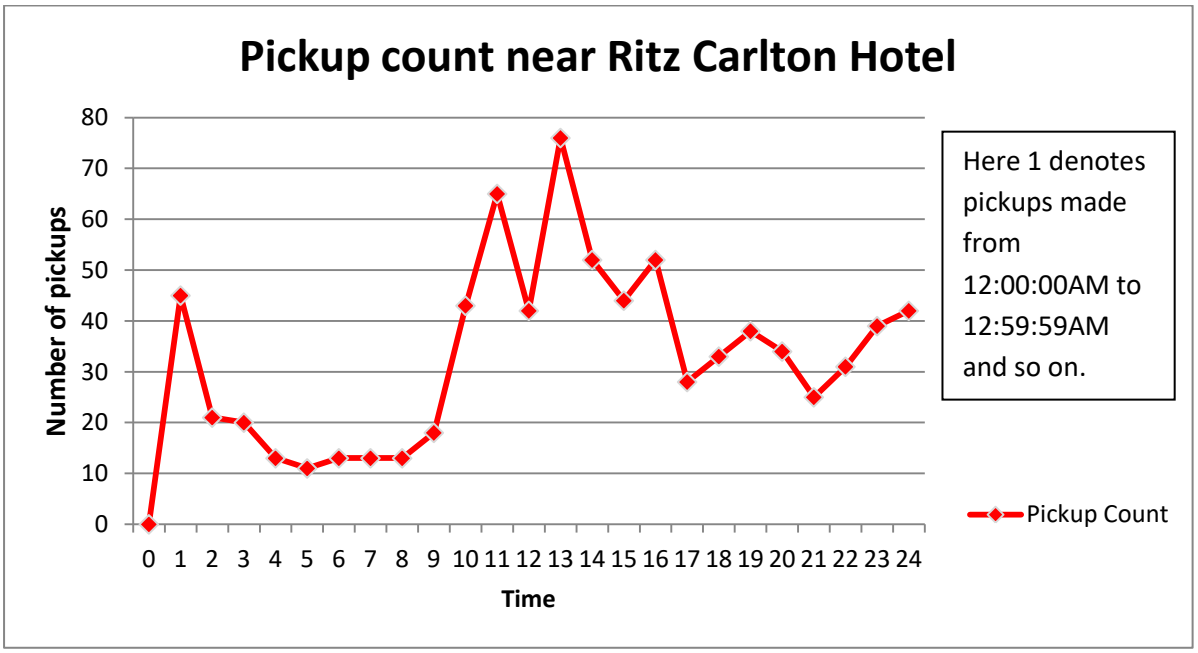
We got one cluster near some dining areas and night clubs in Clarke Quay, Singapore as shown in Figure 6.2.0.2. Green icons denote the various pickup locations which were later mapped to the nearest road-end points. After mapping, we compute the centroid (pink icon) which denotes the recommended location from 8:00:00PM-5:00:00AM.

We have a few restaurants in this area along with many night clubs which are generally open till 4.30AM. The pickup count is high from 8PM but it increases tremendously up to around 150-200 pickups per hour over four days from 10PM till 3AM after which it starts reducing. This is because people in Singapore do not prefer drink and drive and take cabs instead to go back home. The total pickup count of this area during each hour is shown in Graph 6.2.2.

The fact that many people have to go to their work places during weekdays and might not prefer staying till late, justifies the descent pickup count from 8PM-11PM.

3) Recommendation3: Near a Five star hotel

Figure 6.2.0.3 - Recommended pickup location near The Ritz-Carlton Hotel



Graph 6.2.3 – Near The Ritz Carlton Hotel

The Ritz-Carlton Singapore is a 5-star hotel in 7 Raffles Ave. Ritz-Carlton is a very famous hotel chain with 91 hotels around the world. We have a good number of pickups made from this place from 9:00:00AM till 3:00:00PM and then from 11:00:00PM to 1:00:00AM with highest pickups made during 12:00:00PM-1:00:00PM. This pickup pattern is understandable as check out time is 12PM or noon. Few guests take the taxi provided by the hotel but many opt for a personal cab since hotel taxis to airport and other places are very expensive. Graph 6.2.3 shows that good number of pickups have been made around 10AM-12PM too as many guests leave before the checkout time as per their schedule.

Pickups during afternoon and night can be made by guests who went out and explored the city.

4) Recommendation4: Near Singapore zoo and night safari

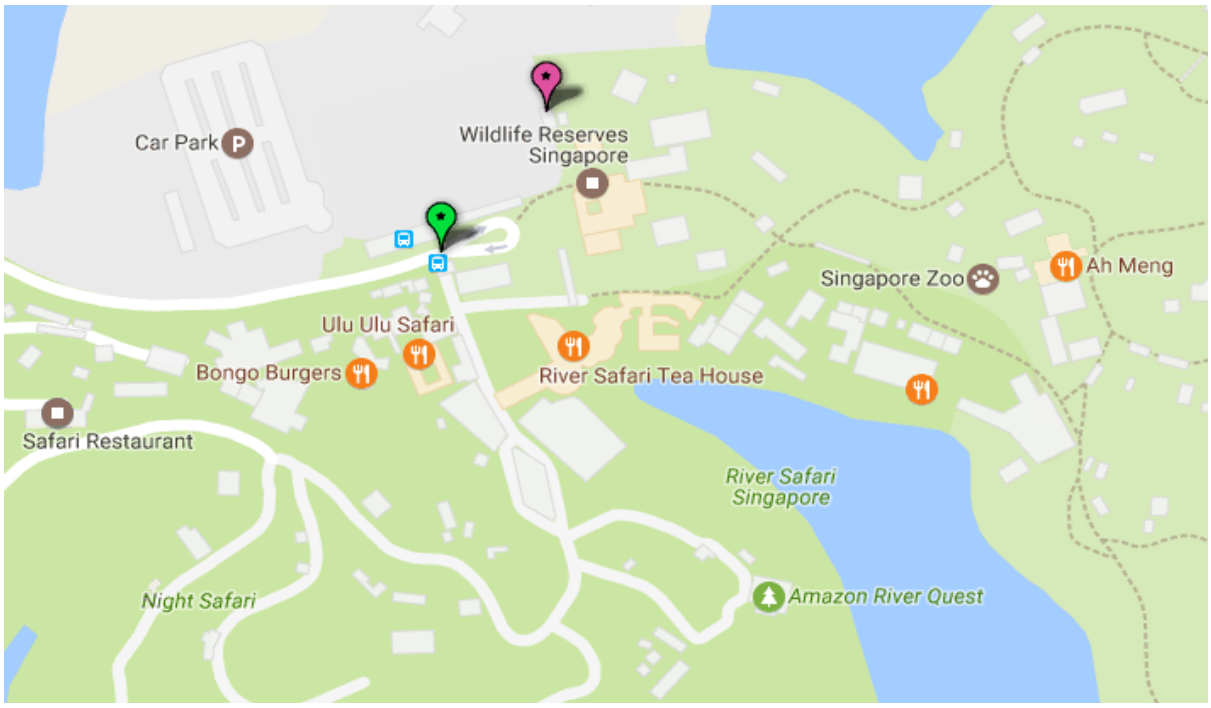
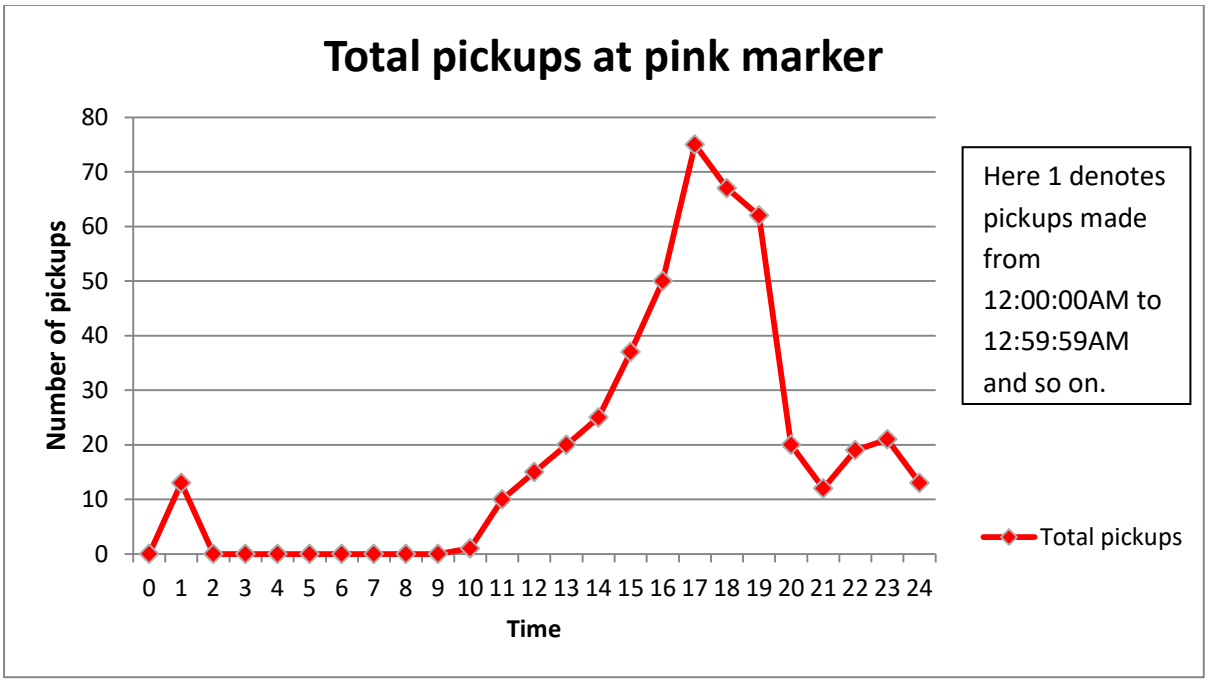
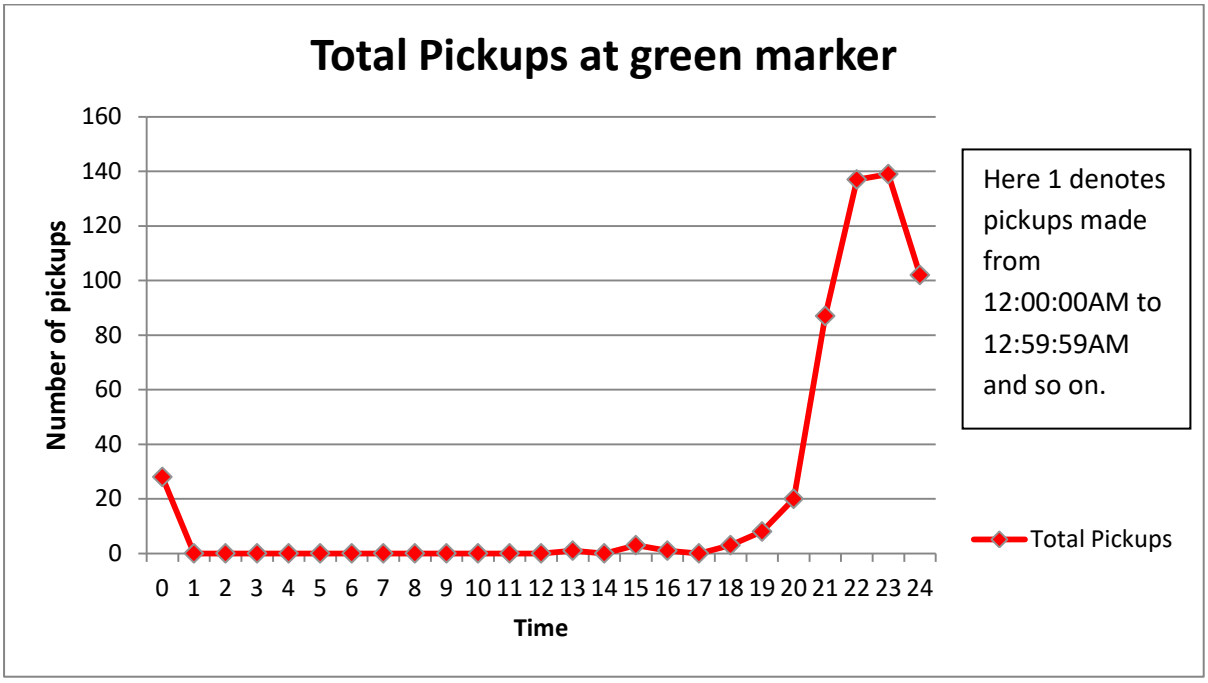


Figure 6.2.0.4 – Near zoo and night safari



Graph 6.2.4



Graph 6.2.5

We have two locations marked in Figure 6.2.0.4. Pink marker is near a taxi stand already present there. This place is a pickup hotspot from 3:00:00PM-7:00:00PM as shown in Graph 6.2.4. This location is around 350m from Singapore Zoo which is open from 8.30AM-6PM. There are a few food joints too where people eat and then take taxis to go back. Around 70 pickups in one hour over a period of 4 days is a big count. Thus, this location can be recommended to taxi drivers for pickups.

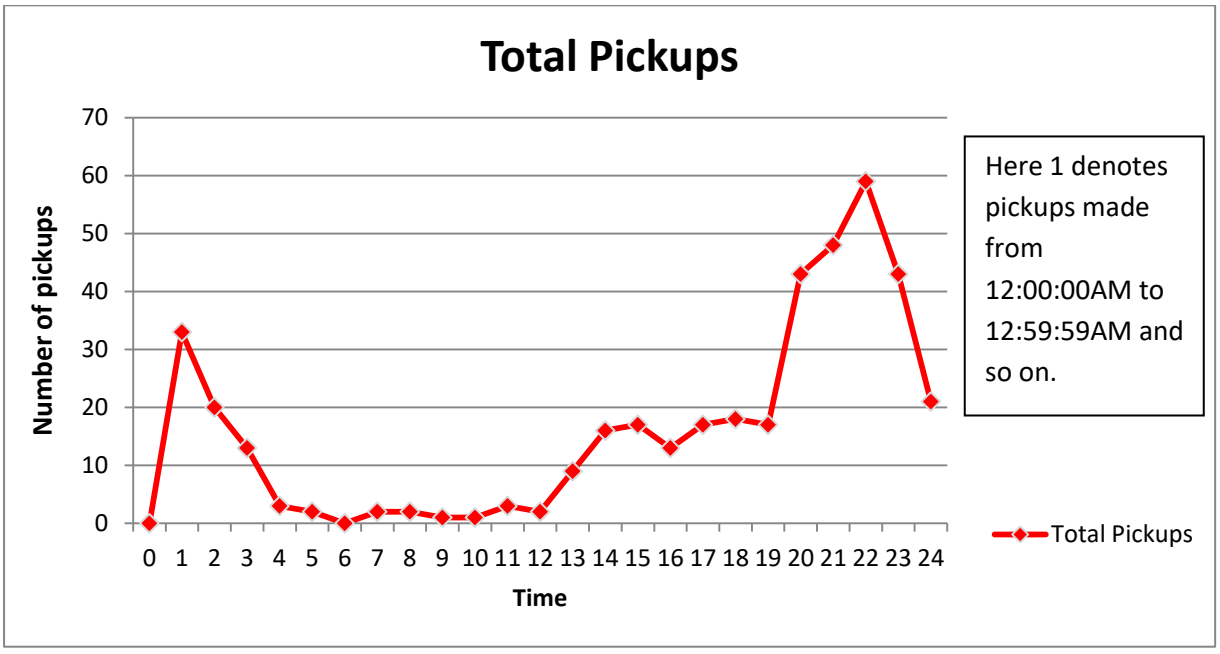
Next is the green marker which is near Ulu Ulu Safari Restaurant, located in Night Safari. Night Safari is open from 7:30PM till 12AM. Pickup time recommendation in this place is from 8PM-12AM where maximum pickups made are from 9PM-11PM. Pickup count can be seen in Graph 6.2.5.

After the safari, many people prefer having food before leaving as there are a good number of options available at pocket-friendly prices.

Instead of placing a permanent taxi stand over here (already present near the pink marker), taxis can come during rush hours so that waiting time for the drivers decrease.

5) Recommendation5: Near MRT station

Figure 6.2.0.5 - Recommended pickup location near Yishun MRT Station



Graph 6.2.6

There are many different and not so common pickup trends found near various Mass Rapid Transit (MRT) stations. One of them is at Yishun MRT station.

This pickup hotspot is on North South line in Yishun, Singapore. The peak hours for taxis from here are from 8PM-12AM. The MRT operates from early morning till 12.30AM. Pickup count can be seen in Graph 6.2.6.

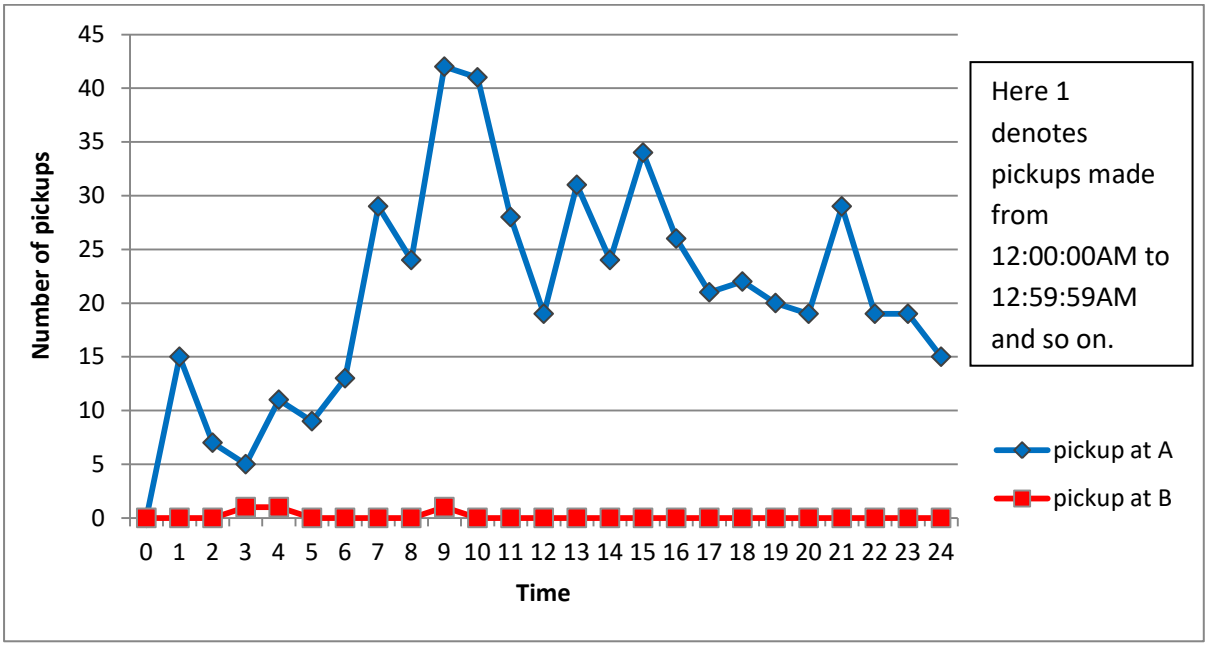
6.2.2 Comparison with OPTICS algorithm

OPTICS is a spatial clustering algorithm which we modified to cluster spatio-temporal data. As stated earlier, simply clustering the nearby points isn't sufficient to recommend a place as they may have completely different pickup patterns. If a driver waits near a place without any knowledge about the pickup trend over there, he'll end up wasting a lot of time and decreasing the money which he could have earned, had he planned wisely. By not considering the pickup trends at places, we might end up clustering dissimilar points and thus getting a big shift in the recommended position.

This can be explained by the following results. These results are obtained by ST-OPTICS but OPTICS failed to find them. It simply merged all of these clusters into one big cluster of spatially close points.

1) **RESULT-1**

Figure 6.2.0.6 – Near park



Graph 6.2.7 – Different pickup counts at point A and point B

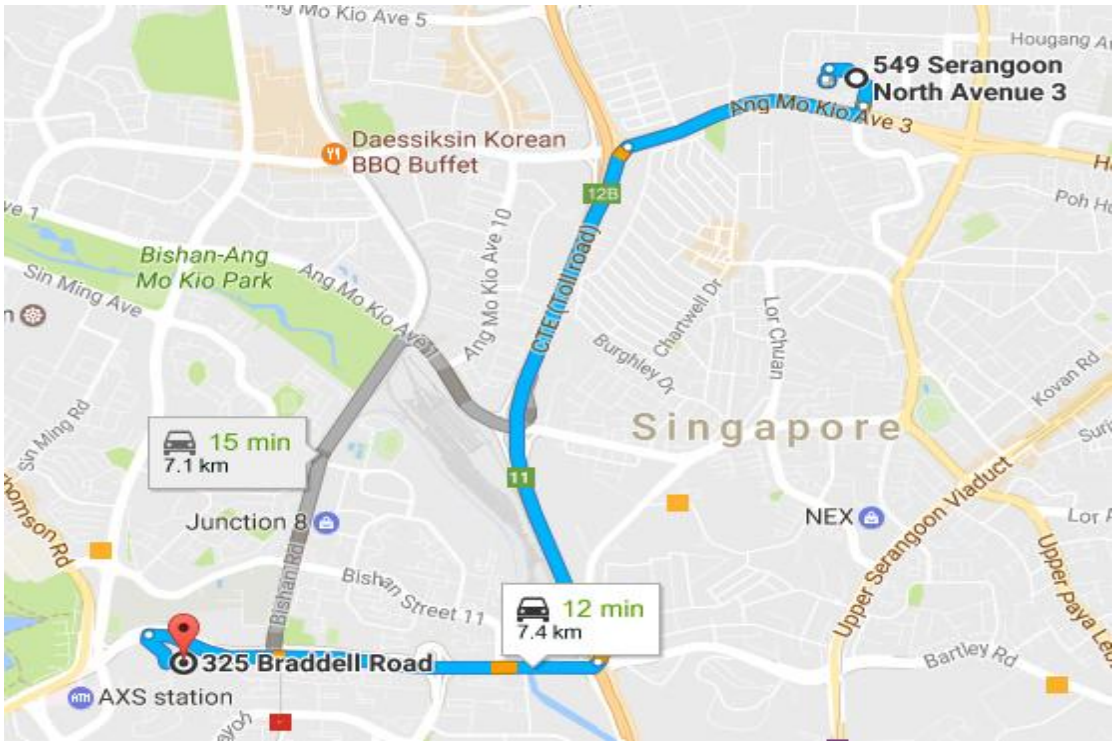


Figure6.2.0.7 - Road Distance

Figure 6.2.0.6 has two points which belong to one big cluster by OPTICS algorithm as they are spatially close but belong to different clusters by ST-OPTICS because of temporal dissimilarity shown in Graph 6.2.7. The road distance between the recommended position of these two clusters in shown in Figure 6.2.0.7 which is around 7.1km by the shortest route. As this distance is quite big, and neither the taxi driver will travel 7km for a passenger nor will the passenger wait for so long.

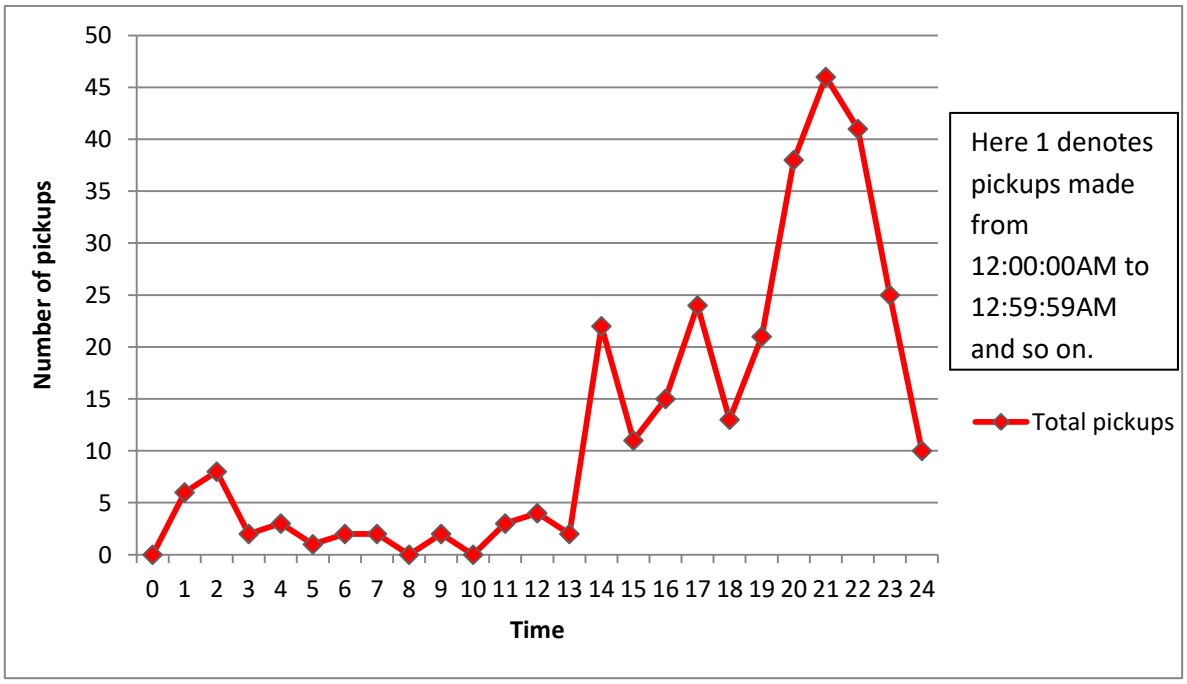
2) RESULT-2

Figure 6.2.0.8 - Near Hotel

As shown in section 6.2.1, ST-OPTICS found a recommendation near Ritz-Carlton. But OPTICS algorithm failed to detect it and merged this location with the big cluster.

3) RESULT-3

Figure 6.2.0.9 – Near Fountain of Wealth



Graph 6.2.8

This location is near to Fountain of Wealth, which is the largest foundation in the world and is located in Suntec City mall. It opens from 10AM-12PM, 2PM-4PM and 6PM-7:30PM. It is believed that who walks around the central base of the foundation three times touching the water would gain some good luck. This is one reason why visitors come here.

People generally prefer visiting it during the evening hours as water show looks interesting when compared to day timings. Since it is located in one of the largest mall, many people prefer eating or shopping after watching the show as the mall is opened till 10PM and then going home. Our algorithm recommended a pickup location at the marked position from 8PM-10PM but OPTICS failed to identify this cluster.

4) RESULT-4

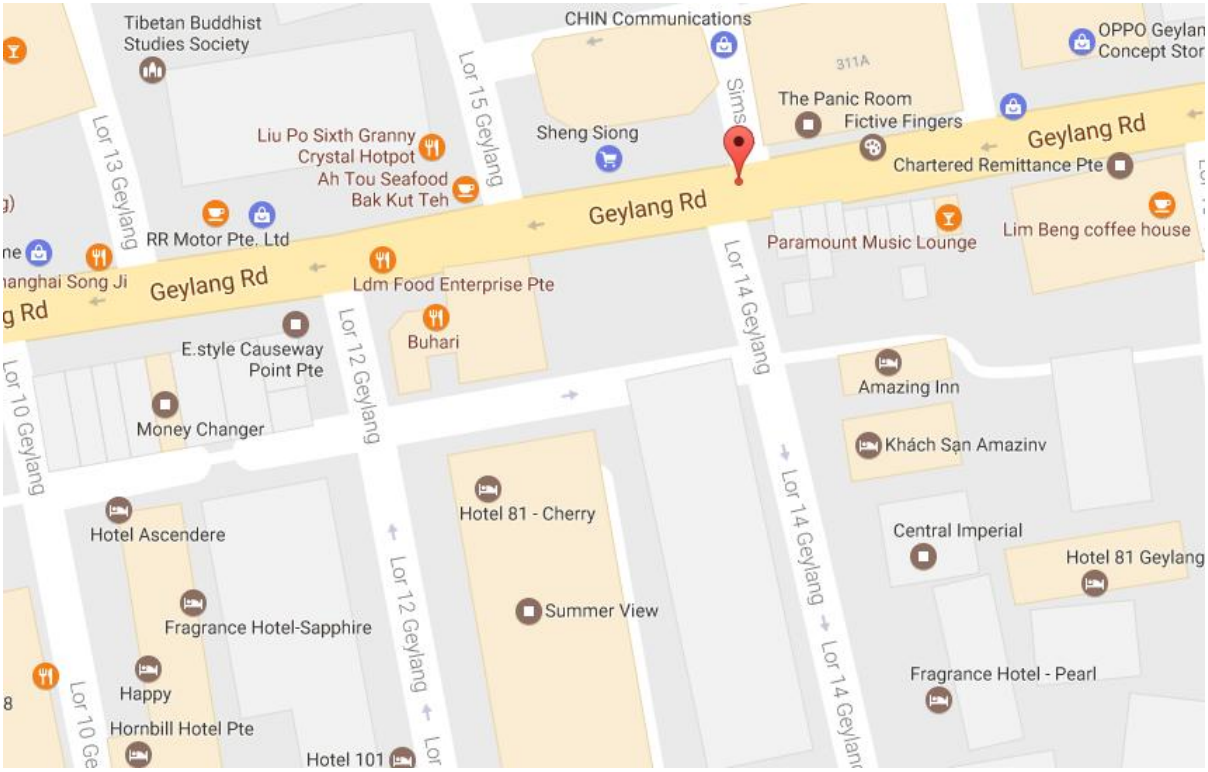
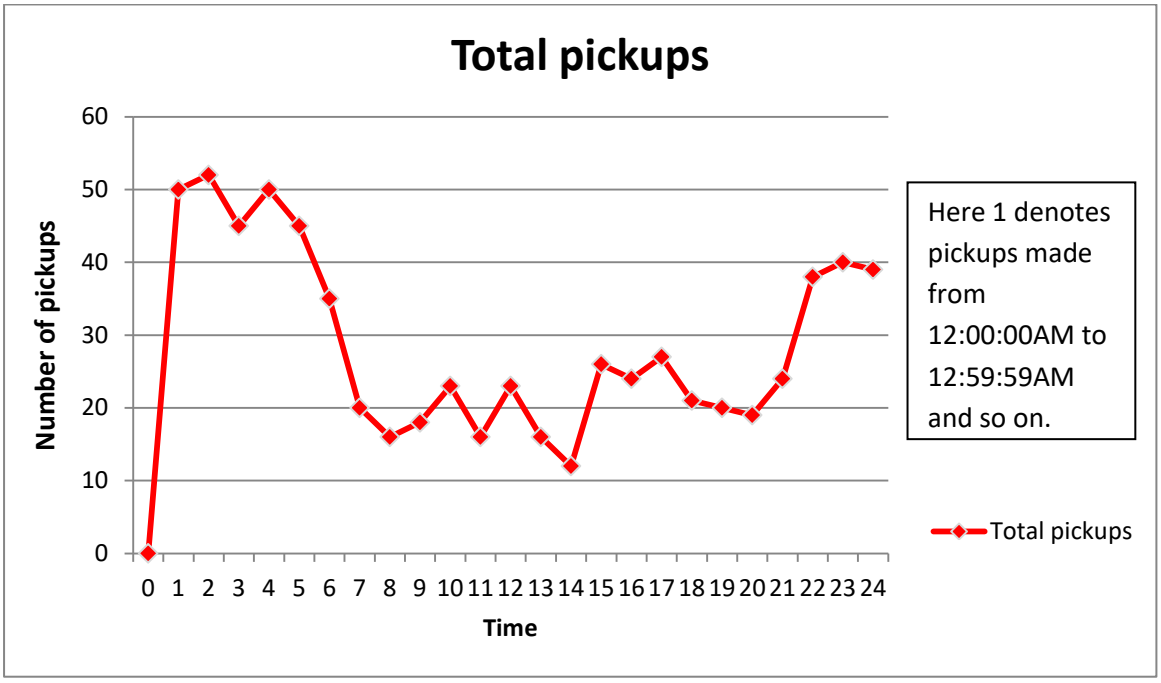


Figure 6.2.0.10 – Near restaurants and lounges of Geylang



Graph 6.2.9

Geylang is a treasure trove of local cuisine. This red marker in Figure 6.2.0.10 is near a few lounges and restaurants which are close to some apartments and budgeted hotels. Pickups are made throughout the day from this place but peak hours are from 10PM-5AM. OPTICS again couldn't detect this cluster.

5) RESULT-5

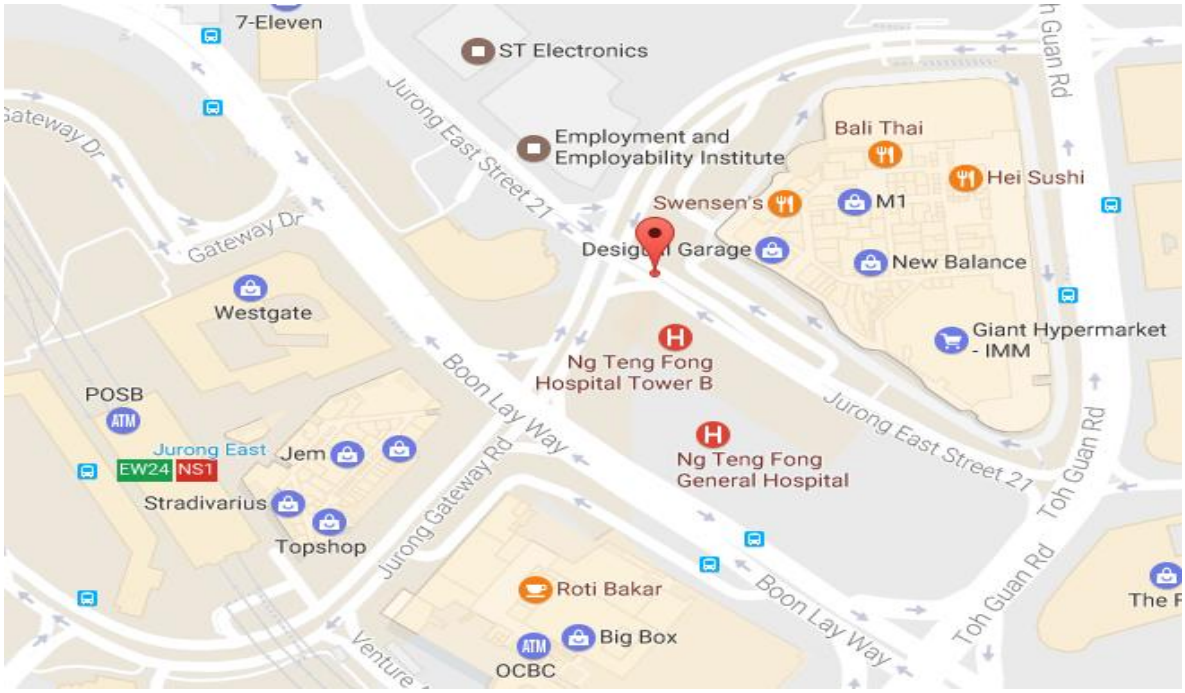
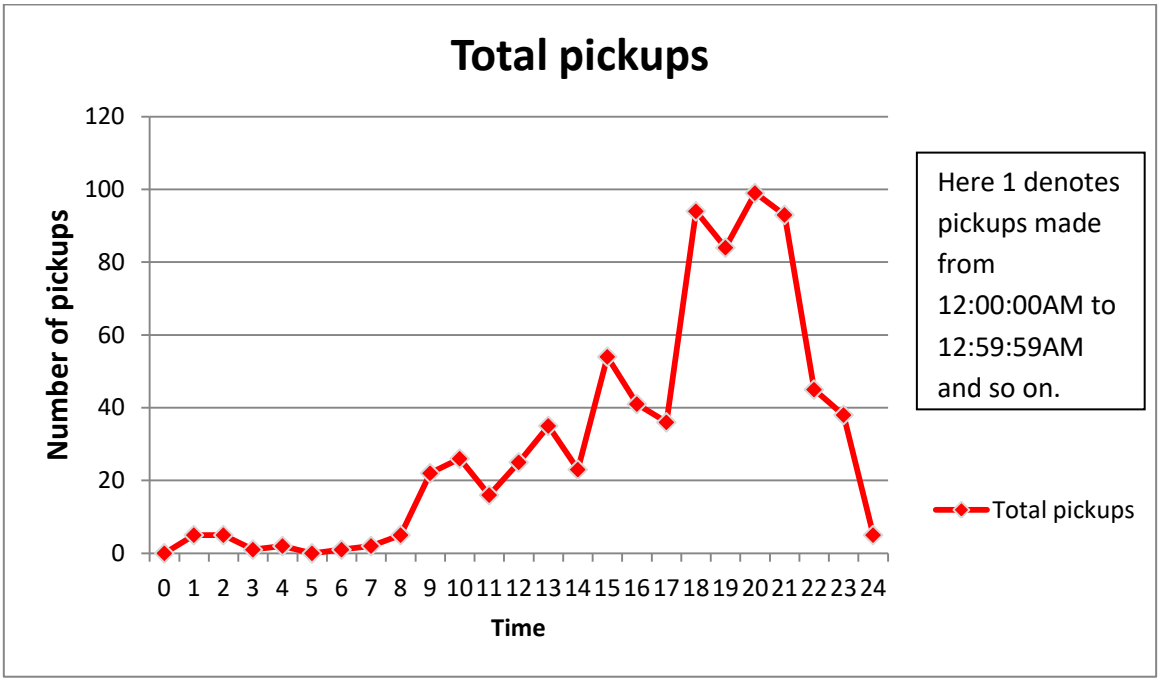


Figure6.2.0.11 - Near shopping mall and companies



Graph 6.2.10

This place is very close to shopping area, hospital and company like ST Electronics. Thus it is pretty much obvious that we'll have many pickups from here. OPTICS couldn't find this as a separate cluster while our algorithm figured out with a time recommendation from 2PM-10PM (approximately). The pickup count can be seen in Graph 6.2.10.

OPTICS did find few interesting clusters like near to the clinics in figure 6.2.0.1 and near night clubs shown in figure 6.2.0.2, but it failed to identify many others. We can thus conclude that there is definitely a need for something more than just spatial closeness while recommending locations. Similarity between temporal and non-spatial aspects is also very important for clustering.

6.2.3 Comparison with existing spatial-temporal algorithm

ST-DBSCAN algorithm is used to cluster data on the basis of spatial, non-spatial and temporal aspects. It handles noise well when compared to original DBSCAN algorithm. One major drawback of this algorithm is that it can't handle clusters of varied densities [2]. We did various experiments to compare ST-DBSCAN with ST-OPTICS. The results showed that ST-OPTICS is better suited for our problem statement.

Experiment 1: Comparison of spatial results obtained by ST-DBSCAN and ST-OPTICS

One main difference between these algorithms comes from the inherited difference between DBSCAN and OPTICS, which is the identification of clusters of varying densities. We did experiments by changing the input parameters given to ST-DBSCAN. Parameters for temporal and non-spatial similarity (eps2 and Δe) were set very high, so that we could focus on clusters formed using spatial aspect only, like DBSCAN algorithm. From the obtained results, we noticed that clusters of different densities couldn't be found.

If the value for eps1 was set a bit high, we could find clusters like in Figure 6.2.0.12, but couldn't detect clusters similar to shown in Figure 6.2.0.13.

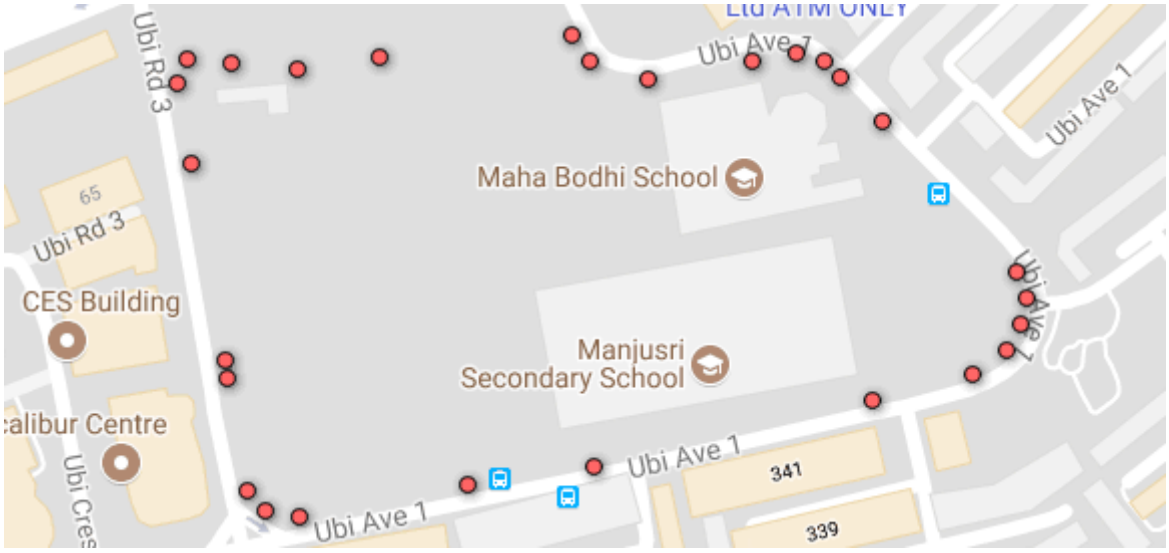


Figure 6.2.0.12 - Loose cluster-Near school

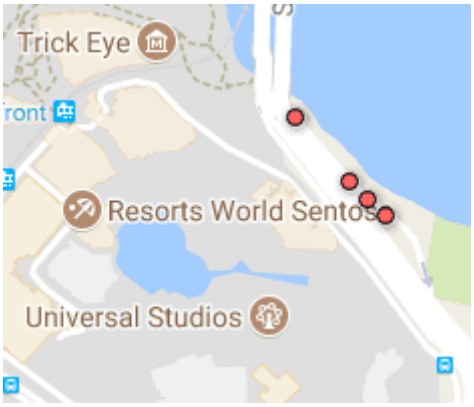


Figure 6.2.0.13 - Tight cluster near Universal Studio

If we decreased the value for eps1 and minPts to find clusters as in Figure 6.2.0.13, then the algorithm failed to find cluster in figure 6.2.0.12.

By setting a low value for eps1, this algorithm is not able to detect loose clusters and merges some points to some other cluster, whereas for a big value, it fails to detect tight clusters. We noticed that merging of points to different clusters mainly happened where two clusters were near to each other. In ST-DBSCAN, we find an intersection of spatial and non-spatial neighbours which forms our final neighbourhood set. It then excludes noise points based on the average of a cluster. So if a point doesn't lie in the spatial neighbourhood, it can't lie in the neighbourhood set and tends to get excluded. Thus, we need to add some variation to DBSCAN algorithm so that we can detect different clusters which are near to each other.

This can be achieved up to an extent by OPTICS. Instead of giving clusters, OPTICS gives a list with reachability and core-distances arranged in an order. From this list, we can extract clusters by setting some distance value. We were able to figure out many such clusters.

Experiment 2: Difference in clusters based on temporal and non-spatial aspects for ST-DBSCAN and ST-OPTICS

Figure 6.2.0.14 - Different clusters because of temporal difference

Since we have parameters like eps2 and Δe in ST-DBSCAN, they don't allow much difference in temporal patterns and deviation of average value for non-spatial aspects and focus on figuring out noise points. Due to this, points which could have been clubbed together got split into different clusters shown by green and pink markers due to a difference in pickup trend in figure 6.2.0.14. On the other hand, ST-OPTICS gives a relaxation for temporal dissimilarity, if points are spatially very close. Thus, by our algorithm, these points got combined into a single cluster which helped in proper time recommendation.

Experiment 3: Results obtained by ST-OPTICS, but not by ST-DBSCAN

Some points which can belong to same cluster by ST-OPTICS are split into different clusters by ST-DBSCAN either due to a difference in spatial or temporal or non-spatial aspect, resulting with a low pickup counts. Low pickup counts give an impression that the location might not be a pickup hotspot and tends to get neglected. If pickup points do not cross the threshold value, then that particular location will not be recommended, which leads to a few important locations getting discarded.

We did experiments to compare the result of our algorithm with ST-DBSCAN by providing them the same input graph and similar parameter values. The input graph was created the same way as explained above. The only difference was that the intermediate nodes were not mapped to road-end points; instead each pickup point and road-end point was a vertex in the graph. Below are a few results to show how ST-OPTICS is better than ST-DBSCAN.

Result-1: Near food joints

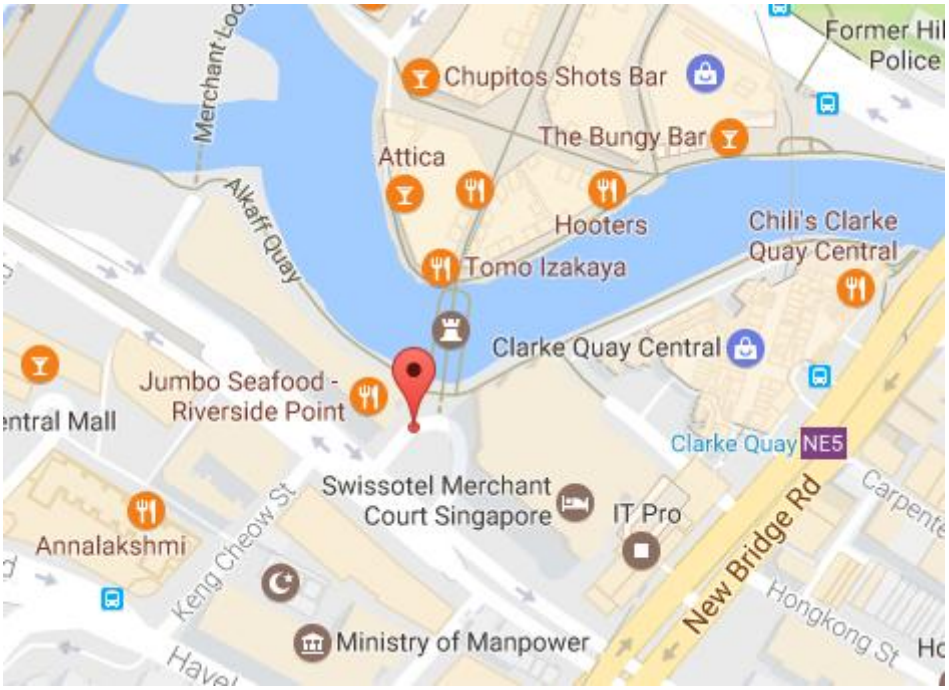


Figure 6.2.0.15 - Near food joints

This recommended location is near some food joints and bars. ST-DBSCAN couldn't combine various points present here and divided them into separate clusters. As individual pickups from each location couldn't cross the threshold value, this place would not be recommended if we had used ST-DBSCAN algorithm.

Result-2: Near schools and bus stands

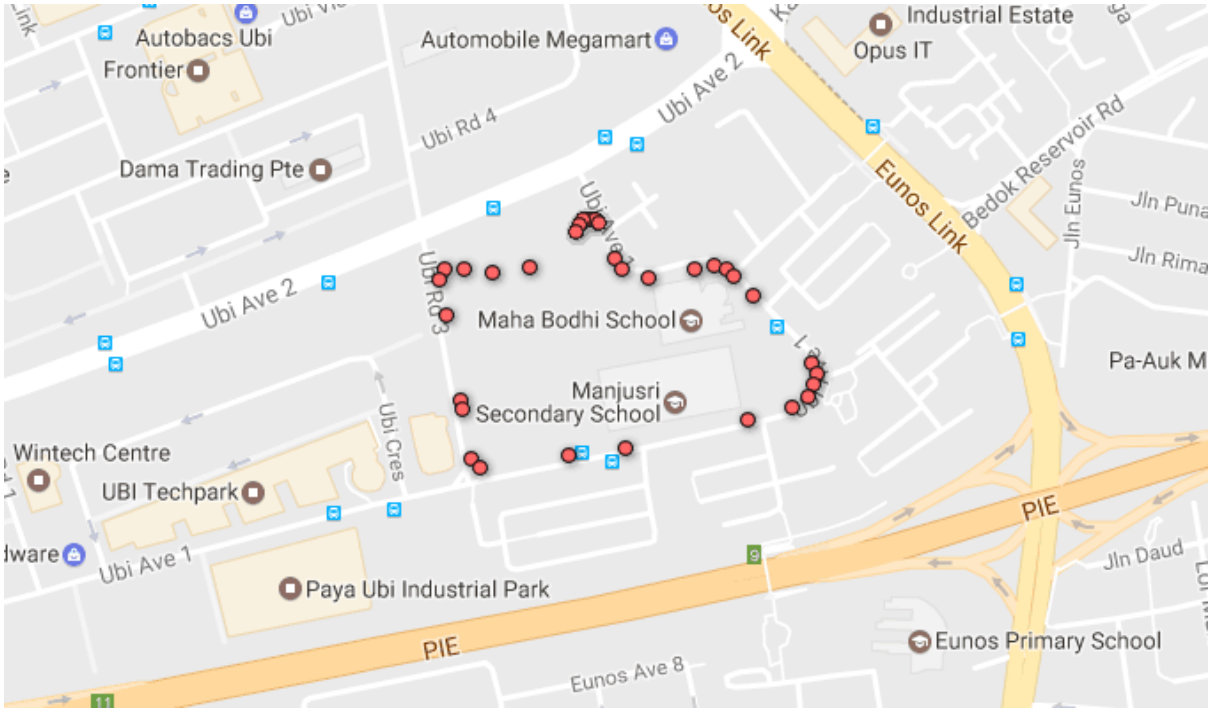


Figure 6.2.0.16 - Near schools and bus stands

These points form a cluster near schools and bus stands by our algorithm, whereas ST-DBSCAN failed to combine them because of difference in pickup patterns. ST-OPTICS gives a relaxation and reduces the temporal value to 60% of the original value, which helped combine these points into one cluster.

Result-3: Near night club

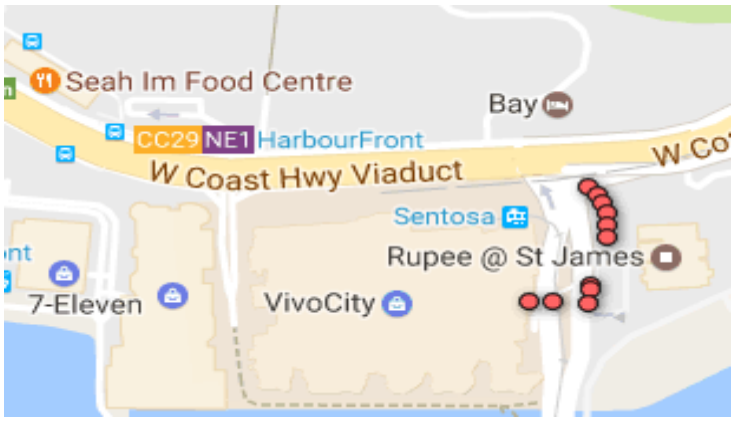


Figure 6.0.2.0.17 - Near night club

These pickup locations are near a night club named Rupee @ St James with a time recommendation from 10PM-3AM by ST-OPTICS. ST-DBSCAN again divided these points into various clusters which resulted in reduced pickup.

From the above results, we can conclude that there are two main differences between ST-OPTICS and ST-DBSCAN. First is the identification of clusters of different densities, which is the inherited difference between DBSCAN and OPTICS. Secondly, ST-DBSCAN focuses more on identifying noise points. It doesn't offer much flexibility and forms clusters for only similar points in all aspects. On the other hand, ST-OPTICS tries to combine points of similar pattern. It tries to group spatially close points even if they differ a bit non-spatially, as recommending locations which are too near to each other won't be beneficial. The time recommendation algorithm takes care of discarding locations which will not be much profitable.

Chapter-7

Conclusion and Future Work

In this thesis, we designed an algorithm named ST-OPTICS which recommended locations along with preferable timings to taxi drivers so that they can reach the nearest pickup hotspot location at that time. OPTICS algorithm was modified for spatial clustering while Dynamic Time Warping algorithm was used for temporal clustering. We further designed an algorithm which computed the time intervals to be recommended for each of these clusters formed. Results were refined by performing some post processing and final recommendations were made. The experiments conducted have shown that this algorithm is highly beneficial. Since the algorithm recommends time along with location, it can be helpful for shared taxi drivers too. They can get an idea about where to find passengers at that time on their way.

As part of the future work, we would like to perform experiments by taking different datasets over weekends, weekdays and festive times and analyse the results. Also, current trip files used were for 2012. We'll try to work on more recent data and compare the results. Lastly, instead of recommending the centroid of clusters, we would use centre of the graph algorithm to recommend location for each cluster.

Bibliography

[1]: Vivek S Ware, Bharathi H N, "Study of Density based Algorithms", International Journal of Computer Applications (0975 – 8887) Volume 69– No.26, May 2013

[2]: Rupanka Bhuyan, Samarjeet Borah, "A Survey of Some Density Based Clustering Techniques", National Conference on Advancements in Information, Computer and Communication (AICC-2013)

[3]: Chotirat Ann Ratanamahatana, Eamonn Keogh, "Making Time-series Classification More Accurate Using Learned Constraints", SDM 2004

[4]: Shaikh Nazneen N, Prof. Kahate S.A, "Recommendation System for Taxi Drivers", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 6, Issue 4, April 2016

[5]: Jing Yuan, Yu Zheng, Liuhang Zhang, Xing Xie and Guangzhong Sun, "Where to Find My Next Passenger?", UbiComp'11 / Beijing, China

[6]: Yu-Ling Hsueh, Ren-Hung Hwang, and Yu-Ting Chen, "An Effective Taxi Recommender System Based on a Spatiotemporal Factor Analysis Model", 2014 International Conference on Computing, Networking and Communications, Mobile Computing & Vehicle Communications Symposium

[7]: Derya Birant, Alp Kut, "ST-DBSCAN: An algorithm for clustering spatial–temporal data", Data & Knowledge Engineering 60 (2007) 208–221

[8]: Wikipedia, clustering, https://en.wikipedia.org/wiki/Cluster_analysis

[9]: Wikipedia, DBSCAN, https://en.wikipedia.org/wiki/DBSCAN

[10]: Wikipedia, OPTICS, https://en.wikipedia.org/wiki/OPTICS_algorithm

[11]: Wikipedia, metric, https://en.wikipedia.org/wiki/Metric_(mathematics)

[12]: Wikipedia, Dynamic Time Warping, https://en.wikipedia.org/wiki/Dynamic_time_warping

[13]: Haversine Distance, http://www.movable-type.co.uk/scripts/latlong.html

[14]: Mingyue Zhang, Jianxun Liu, Yizhi Liu, Zhenyang Hu, Liang Yi, "Recommending Pick-up Points for Taxi-drivers based on Spatio-temporal Clustering", 2012 Second International Conference on Cloud and Green Computing

[15]: DENG Min∗, LIU QiLiang, WANG JiaQiu & SHI Yan, "A general method of spatio-temporal clustering analysis", Science China Press and Springer-Verlag Berlin Heidelberg 2012

[16]: Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander, “OPTICS: Ordering Points To Identify the Clustering Structure”, Proc. ACM SIGMOD’99 Int. Conf. on Management of Data, Philadelphia PA, 1999

[17]: Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander, “OPTICS: Ordering Points To Identify the Clustering Structure”, Proc. ACM SIGMOD’99 Int. Conf. on Management of Data, Philadelphia PA, 1999