INCREMENTAL CLUSTERING FOR TRAJECTORIES

BY

ZHENHUI LI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Adviser:

Professor Jiawei Han

Abstract

Trajectory clustering has played a crucial role in data analysis since it reveals underlying trends of moving objects. Due to their sequential nature, trajectory data are often received *incrementally*, e.g., continuous new points reported by GPS system. However, since existing trajectory clustering algorithms are developed for static datasets, they are not suitable for incremental clustering with the following two requirements. First, clustering should be processed efficiently since it can be frequently requested. Second, huge amounts of trajectory data must be accommodated, as they will accumulate constantly.

An *incremental clustering framework for trajectories* is proposed in this paper. It contains two parts: online micro-cluster maintenance and offline macro-cluster creation. For online part, when a new bunch of trajectories arrives, each trajectory is simplified into a set of directed line segments in order to find clusters of trajectory subparts. Micro-clusters are used to store compact summaries of similar trajectory line segments, which take much smaller space than raw trajectories. When new data are added, micro-clusters are updated incrementally to reflect the changes. For offline part, when a user requests to see current clustering result, macro-clustering is performed on the set of micro-clusters rather than on all trajectories over the whole time span. Since the number of micro-clusters is smaller than that of original trajectories, macro-clusters are generated efficiently to show clustering result of trajectories. Experimental results on both synthetic and real data sets show that our framework achieves high efficiency as well as high clustering quality.

To my family for all their love.

Acknowledgments

First, I would like to thank Professor Jiawei Han for his guidance and insightful comments for my research work. Second, I would like to thank all the faculties and colleagues in DAIS (Data and Information System) research group at the University of Illinois, Urbana-Champaign. Especially, I thank Jae-Gil Lee and Xiaolei Li for their help throughout the writing of the thesis.

Table of Contents

List of Figures

Chapter 1

Introduction

In recent years, the collection of trajectory data has become increasingly common. GPS chips implanted in animals have enabled scientists to track their study objects as they travel. RFID technology installed in vehicles has enabled traffic officers to track road traffic in real-time. With such data, trajectory clustering is a very useful task. It discovers movement patterns that help analysts see overall trends in the trajectories. For example, analysis of bird feeding and nesting habits is an important task. With the help of GPS, scientists can tag and track birds as they fly around. Such tracking devices report the trajectories of animals on a continual basis (*e.g.*, every minute, every hour). With such data, scientists can study the movement habits (*i.e.*, trajectory clusters) of birds.

One important property with tracking application is the *incremental* nature of the data. The data will grow to be in huge size as time goes by. Consider the following real case of moving vehicle data which is used in experiment evaluation.

Example 1 *A taxi tracking system tracks the real-time locations of more than 5,000 taxis in San Francisco. With the sensor installed on each taxi, the system is able to receive information about current location(longitude and latitude) of each taxi with a precise timestamp. The system accumulates the updated data every minute. After a single day, the system will collect totally 7.2 million points with 1,440 points for each taxi. After a week, the number of points will be accumulated to 50.4 million points.*

For static data sets, there are many existing trajectory clustering algorithms developed. However, to the best of our knowledge, none of them targeted at solving clustering problem for incremental huge trajectory data as pointed out in Example 1. Gaffney *et al.* [9, 8] proposed a probabilistic clustering technique for trajectories. The problem with this statistical approach is that it considers trajectories as a whole. But in the real cases, one trajectory can be very long and complicated while subparts of different trajectories may share similar paths. These common paths of sub-trajectories could be interesting trajectory clusters. Lee *et al.* [12] proposed a trajectory clustering algorithm TRACLUS based on the partition-and-group framework. This is the first work that mines clusters from a sequence of sub-trajectories. It first partitions trajectories into several line segments with least information loss, then group them into clusters. The followed work of Lee *et al.* [13, 11] on trajectory outliers detection and trajectory classification based on the idea of sub-trajectories shows that it is necessary and important to mine interesting knowledge on partial trajectories rather than on the whole trajectories. However, neither of these algorithms is able to handle the case when the input data is continuously updated since they require the complete input data be available. Facing continuous data, previous methods will take long time to retrieve all the data and re-compute the trajectory cluster over the whole huge data set. If the users want to track real-time clusters every hour, it is almost impossible to finish computation within the time period threshold, especially considering the data size still keeps growing every minute. Therefore, trajectory data must be accommodated incrementally.

A naïve solution to the incremental trajectory clustering problem is to re-compute the clusters over the whole data set when a clustering request is evoked. But, it can become quite expensive after huge data has accumulated. And in real world scenarios, the application will be running for a long time (*e.g.*, months or years), to efficiently compute trajectory clusters will be a major obstacle. So it is desirable to adjust the clusters *incrementally* as new data coming in.

An important point to notice is that *new data will only affect local shifts*. It will not have big influence on clusters in the areas which are far away from the local area of new data. So, a more sensible approach to accommodate huge amount of data is to maintain and adjust *micro-clusters* of the trajectory data. Micro-clusters are tight clusters over small local regions. Due to their small sizes, they are more flexible to changes in the data source. Yet they still achieve the desired space savings of clusters by summarizing extremely similar input trajectories. These properties make them suitable for incremental clustering.

This work proposes an *incremental Trajectory Clustering using Micro- and Macro-clustering* framework called TCMM. It makes the following contributions towards an incremental trajectory clustering solution. First, trajectories are simplified by partitioning into line segments to find the clusters of sub-trajectories. Second, micro-clusters of the partitioned trajectories are computed and maintained incrementally. Micro-clusters hold and summarize similar trajectory partitions at very fine granularity levels. They use very little space and can be updated efficiently. And finally, micro-clusters are used to generate the macro-clusters(*i.e.*, final trajectory clusters).

The TCMM framework is truly incremental in the sense that micro-clusters are incrementally maintained as more and more data are received. Because their granularity level is low, they can adjust to all types of change in the input data. The number of micro-clusters is much smaller than that of the original input data. When the user wants to compute the full trajectory clusters, micro-clusters are combined together to form the macro-clusters in higher granularity level.

The rest of this paper is organized as follows. Section 2 formally defines the problem and gives an outline of the TCMM framework. Sections 3.1 and 4 discuss the micro-clusters and the macro-clusters, respectively. Experiments are shown in Section 5. Related work is analyzed in Section 6. Finally, the paper concludes in Section 8.

Chapter 2

General Framework

2.1 Problem Statement

The data to be studied in this work will be in the context of an *incremental data source*. That is, new batches of trajectory data will continuously be fed into the clustering algorithm (*e.g.*, from new data recordings). The goal is to process such data and produce clusters *incrementally* and *not* have to re-compute from scratch every time.

Let the input data be represented by a sequence of time-stamped trajectory data sets: $\langle I_{t_1}, I_{t_2}, \ldots \rangle$ where each I_{t_i} is a set of trajectories being presented at time t_i. Each $I_{t_i} = \{TR_1, TR_2, \ldots, TR_{n_{TR}}\}$ where each TR_j is a trajectory. A single trajectory TR_j is often represented as a polyline, which is a sequence of connected line segments. It can be denoted as $TR_j = p_1 p_2 \ldots p_{len_j}$, where each point p_i is a time-stamped point. TR_j can be further simplified to derive a new polyline with fewer points while its deviation from the original polyline is below some threshold. The simplification techniques have been studied extensively in previous work [12, 5] . In this paper, we use the simplification technique in our previous paper [12]. Simplified trajectory is represented as $TR_j^{simplified} = L_1 L_2 \ldots L_n$, where L_i and L_{i+1} are connected directed line segments (i.e., trajectory partitions).

Given such input data, the goal is to produce a set of clusters $O = \{C_1, C_2, \ldots, C_{n_C}\}$. A *cluster* is a set of directed trajectory line segments $C_i = \{L_1, L_2, \ldots, L_{ln}\}$, where L_k is a directed line segment from certain simplified trajectory $TR_j^{simplified}$ at certain time stamp t_i. Because we do clustering on line segments rather than whole trajectories, the clusters we find are actually sub-trajectory clusters, which are the popular paths visited by many moving objects.

2.2 TCMM Framework

Figure 2.1 shows the general data flow of TCMM. The x-axis represents the progress of time and the y-axis shows the progress of data processing. As the figure illustrates, input data are received continuously.

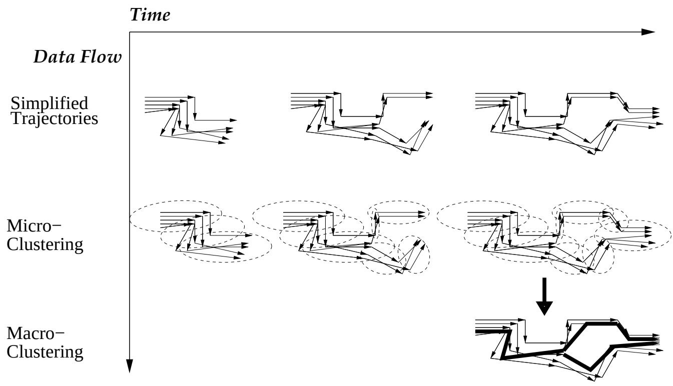


Figure 2.1: The Framework

The first step is micro-clustering. Because there is an infinite data source, it is impossible to store all the preprocessed input data and compute clusters from them on request. To solve this problem, this work introduces the concept of *trajectory micro-clusters*. The term "micro" refers to the extreme tightness of the clusters. The idea is to only cluster at very fine granularity. Hence, the number of micro-clusters is much larger than that of final trajectory clusters. Figure 2.1 shows the micro-clusters in the second row. Section 3.1 will discuss them in detail.

The second step is macro-clustering, which will be discussed in detail in Section 4. Compared to the micro-clustering step, which are updated constantly as new data is received, the macro-clustering step is *only* evoked after receiving the user's request of trajectory clusters. This step will then use the micro-clusters as input.

Chapter 3

Trajectory Clustering using Micro- and Macro-Clustering

3.1 Trajectory Micro-Clustering

As newly arrived trajectories will only affect local clustering result, trajectory micro-clusters (or just micro-clusters) are introduced here to maintain a fine-granularity clustering. Micro-clusters (defined in Section 3.1.1) are much more restrictive than the final clusters in the sense that each micro-cluster is meant to only hold and summarize the information of local partitioned trajectories. Micro-clustering will enable more efficient computation of final clusters comparing with computation from original line segments.

Algorithm 1 Trajectory Micro-Clustering

```
1: Input:New trajectories $I_{t_{current}} = \{TR_1, TR_2, \cdots, TR_{nTR}\}$ and existing micro-clusters $MC = \{MC_1, MC_2, \ldots, MC_{n_{MC}}\}$.
2: Parameter: $d_{max}$
3: Output: Updated $MC$ with new trajectories inserted.
4: Algorithm:
5: for every $TR_i \in I_{t_{current}}$ do
6:    for every $L_j \in TR_i$ do
7:       Find the closest $MC_k$ to line segment $L_j$ /* Section 3.1.2 */
8:       if $distance(L_j, MC_k) \leq d_{max}$ then
9:          Add $L_j$ into $MC_k$ and update $MC_k$ accordingly
10:      else
11:         Create a new micro-cluster $MC_{new}$ for $L_j$;
12:         if size of $MC$ exceeds memory constraint then
13:            Merge micro-clusters in $MC$ /* Section 3.1.3 */
14:         end if
15:      end if
16:   end for
17: end for
```

Algorithm 1 shows the general work flow of generating and maintaining micro-clusters. It proceeds as follows. After a batch of new trajectories arrive, we compute the closest micro-cluster

MC_k for each line segment L_i in every trajectory. If the distance between L_i and MC_k is less than a distance threshold (d_{max}), L_i will be inserted into MC_k. Otherwise, a new micro-cluster MC_{new} will be created for L_i. If the creation of the new micro-cluster results in the overload of the total number of micro-clusters, some micro-clusters will be merged. The rest of this section discuss these steps in detail.

3.1.1 Micro-Cluster Definitions

Each trajectory micro-cluster will hold and summarize a set of partitioned trajectories, which are essentially line segments.

Definition 1 (Micro-Cluster) *A* trajectory micro-cluster (or micro-cluster) *for a set of directed line segments $L_1, L_2, \cdots, L_N$ is defined as the tuple: (N, LS_{center}, LS_θ, LS_{length}, SS_{center}, SS_θ, SS_{length}), where N is the number of line segments in the micro-cluster, LS_{center}, LS_θ, and LS_{length} are the linear sums of the line segments' center points, angles and lengths respectively, SS_{center}, SS_θ, and SS_{length} are the squared sums of the line segments' center points, angles and lengths respectively.*

The definition of trajectory micro-cluster is an extension of the cluster feature vector in *BIRCH* [16]. The linear sum LS represents the basic summarized information of line segments(*i.e.*, center point, angle and length). The square sum SS will be used to calculate the tightness of micro-cluster which will be discussed in Section 3.1.3. The additive nature of the definition makes it easy to add new line segments into the micro-cluster and merge two micro-clusters. Meanwhile, the definition is designed to be consistent with the distance measure of line segments in Section 3.1.2.

Also, every trajectory micro-cluster will have a *representative line segment*. As the name suggests, this line segment is the representative line segment of the cluster. It is an "average" of sorts.

Definition 2 (Representative Line Segment) *The* representative line segment *of a micro-cluster is represented by the starting point s and ending point e. s and e can be computed from the micro-*

cluster features.

$$s = (center_x - \frac{\cos\theta}{2}len, center_y - \frac{\sin\theta}{2}len)$$
$$e = (center_x + \frac{\cos\theta}{2}len, center_y + \frac{\sin\theta}{2}len)$$

where $center_x = LS_{center_x}/N$, $center_y = LS_{center_y}/N$, $len = LS_{length}/N$, *and* $\theta = LS_\theta/N$.

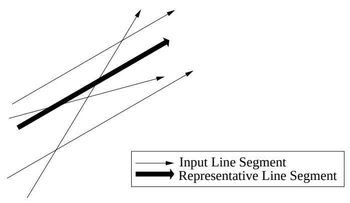


Figure 3.1: Representative Line Segment

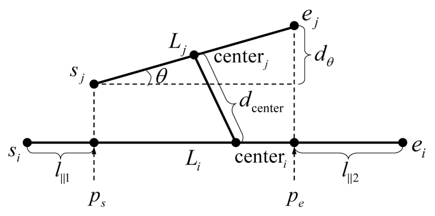


Figure 3.2: Line Segments Distance

Figure 3.1 shows an example. There are four line segments in the micro-cluster, which are drawn in thin lines. The representative line segment of the micro-cluster is drawn in a thick line.

3.1.2 Creating and Updating Micro-Clusters

When a new line segment L_i is received, the first task is to find the closest micro-cluster MC_k that can absorb L_i (*i.e.*, Line 7 in Algorithm 1). If the distance between L_i and MC_k is less than the distance threshold d_{max}, L_i is then added to MC_k and MC_k is updated accordingly; if not, a new micro-cluster is created (*i.e.*, Line 8 to 11 in Algorithm 1). This section will discuss how these steps are performed in detail.

Before proceeding, the distance between a line segment and a micro-cluster is defined. Since a micro-cluster has its representative line segment, the distance is in fact defined between two line segments, which is composed of three components: the center point distance (d_{center}), the angle distance (d_θ) and the parallel distance ($d_{||}$) . The distance is adapted from a similarity measure used in the area of pattern recognition [10], which is a modified line segment Hausdorff distance. The similar distance measure is also used in [12]. Different from [12], we use component d_{center} instead of $d_\perp$. The reason to choose d_{center} is because it is a more balanced measure between d_θ and $d_{||}$ and it is easier to adapt the concept of extent, which will be introduced in Section 3.1.3.

Let s_i and e_i be the starting and ending points of L_i; similarly for s_j and e_j with L_j. Without loss of generality, the longer line segment is assigned to L_i, and the shorter one to L_j. Figure 3.2 gives an intuitive illustration of the distance function.

Definition 3 *The distance function is defined as the sum of three components:*

$$dist(L_i, L_j) = d_{center}(L_i, L_j) + d_\theta(L_i, L_j) + d_{\|}(L_i, L_j)$$

The center distance:

$$d_{center}(L_i, L_j) = \| \ center_i - center_j \ \| \ ,$$

where $\| \ center_i - center_j \ \|$ *is the Euclidean distance between center points of* L_i *and* L_j*.*
The angle distance:

$$d_\theta(L_i, L_j) = \begin{cases} \| \ L_j \ \| \times \sin(\theta), & 0^o \le \theta < 90^o \\ \| \ L_j \ \|, & 90^o \le \theta \le 180^o \end{cases} ,$$

where $\| \ L_j \ \|$ *denote length of* L_j*,* $\theta(0^o \le \theta \le 180^o)$ *denote the smaller intersecting angle between* L_i *and* L_j*. Note that the range of* θ *is not* $[0^o, 360^o)$ *because* θ *is the value of smaller intersecting angle without considering the direction.*
The parallel distance:

$$d_{\|}(L_i, L_j) = \min(l_{\|1}, l_{\|2}),$$

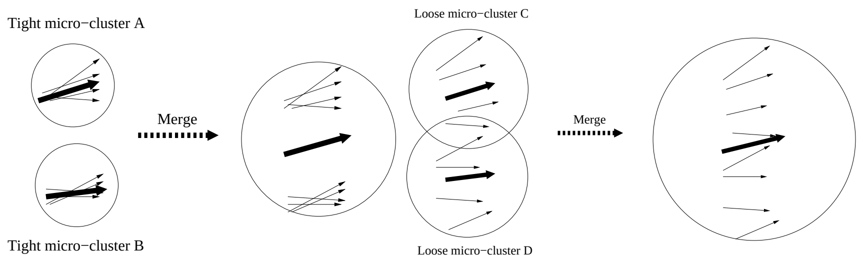
where $l_{\|1}$ *is the Euclidean distances of* p_s *to* s_i *and* $l_{\|2}$ *is that of* p_e *to* e_i*.* p_s *and* p_e *are the projection points of the points* s_j *and* e_j *onto* L_i *respectively.*

After finding the closest micro-cluster MC_k, if the distance from L_i is less than d_{max}, L_i is inserted into it, and the linear and square sums in MC_k are updated accordingly. Because they are just sums, the additivity property applies and the update is efficient. If the distance between the nearest micro-cluster and L_i is bigger than d_{max}, a new micro-cluster will be created for L_i. The initial measures in the new micro-cluster is simply derived from line segment L_i (*i.e.*, center point, theta, and length).

3.1.3 Merging Micro-Clusters

In real world applications, storage space is always a constraint. The TCMM framework faces this problem with its micro-clusters as shown in Line 12 to 13 of Algorithm 1. If the total space used by micro-clusters exceeds a given space constraint, some micro-clusters have to be merged to satisfy the space constraint. Meanwhile, if the number of micro-clusters keeps increasing, it will affect the efficiency of algorithm because the most time-consuming part is finding the nearest micro-cluster. And what is most important, it may be unnecessary to keep all the micro-clusters since some of the micro-clusters may become closer after several rounds of updates. Therefore, the algorithm demands merging close micro-clusters when necessary to speed up efficiency and save storage. Obviously, pairs of micro-clusters that contain similar line segments are better candidates for merging because the merge results in less information loss.

One way to compute the similarity between two micro-clusters is to calculate the distance between the representative line segments of the micro-clusters. Though intuitive, this method fails to consider the tightness of the micro-clusters. Figure 3.3 shows an example that how tightness



(a) Merging tight micro-clusters (b) Merging loose micro-clusters

Figure 3.3: Merging micro-clusters

might effect distance between two micro-clusters. Figure 3.3(a) shows two tight micro-clusters and the micro-cluster after merging them. Figure 3.3(b) shows the case for two comparatively loose micro-clusters. We can see that micro-cluster A and micro-cluster C have same representative line segments, and so do micro-clusters B and D. Thus the distance between micro-cluster A and B should be the same as that between micro-clusters C and D if we measure the distance only using representative line segments. In this case, the chance to merge micro-clusters A and B is equal to that of merging micro-clusters C and D. However, we actually prefer merging micro-clusters C and D. There are two reasons: on one hand, if both micro-clusters are very tight, they may not be good candidates for merging because it would break that tightness after the merge. On the other hand, if they are both loose, it may not do much harm to merge them even if their representative line segments are somewhat far apart. Hence, a better approach would be to consider the *extent* of the micro-clusters and use that information in computing the distance between micro-cluster.

In the following parts, we will first introduce the way to compute micro-cluster extent, then give definitions of the distance between micro-clusters with extent information. Lastly, we will discuss how to merge two micro-clusters.

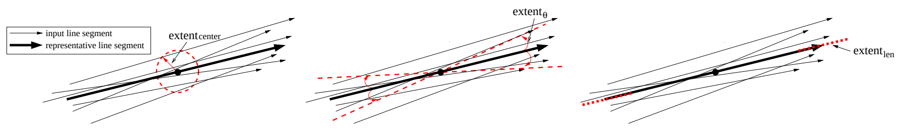
Micro-Cluster Extent The extent of a micro-cluster is an indication of its tightness. Recall that micro-clusters are represented by tuples of the form: $(N,\ LS_{center},\ LS_{\theta},\ LS_{length},\ SS_{center},\ SS_{\theta},\ SS_{length})$, which maintain linear and square sums of center, angle and length. The extent of the micro-cluster also includes three part $extent_{center}$, $extent_{\theta}$ and $extent_{length}$ to measure the tightness of three basic facts of a trajectory micro-cluster. The extents are the standard deviation that calculated from its corresponding LS and SS. We have the following lemma from [16].

Lemma 1 *Given a set of distance values, $D = (d_1, d_2, ..., d_n)$. Let $LS = \sum_{i=1..n} d_i$, and $SS = \sum_{i=1..n}(d_i)^2$. The standard deviation of the distances is $\sigma = \sqrt{\frac{n \times SS - (LS)^2}{n^2}}$.*

Using Lemma 1, we give a formal definition for extent of a micro-cluster:

$$extent_{\alpha} = \sqrt{(N \times SS_{\alpha} - LS_{\alpha}^2)/N^2}$$

where symbol α represents *center*, θ, or *length* and N is the number of line segments in the micro-cluster.

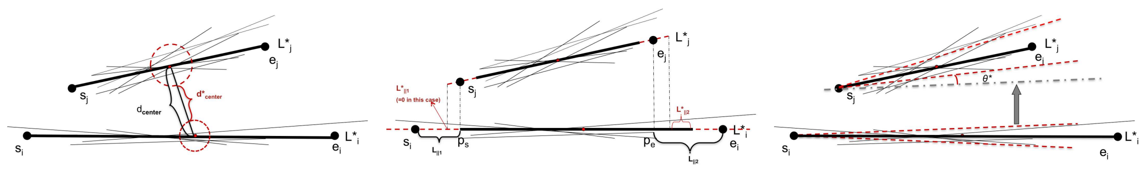


(a) Center extent (b) θ extent (c) Length extent

Figure 3.4: Micro-Cluster Extent

To give an intuition of extent concept, Figure 3.4 shows an example of $extent_{center}$, $extent_{\theta}$ and $extent_{length}$. Figure 3.4(a) states that "most" center points of the line segments stored in this micro-cluster are within the circle of radius $extent_{center}$. Figure 3.4(b) illustrates that "most" angles vary within a range of $extent_\theta$ and Figure 3.4(c) reflects the uncertainty of length.

Micro-Cluster Distance with Extent With the extents properly defined, we can now incorporate them into the distance function. Recall that the intention of extent was to adjust the distance function based on the tightness of micro-clusters. For instance, let $d_{1,2}$ be the distance between micro-clusters MC_1 and MC_2 according to the distance function defined previously. If these two micro-clusters are both "tight" (*i.e.*, having zero or very small extent), then $d_{1,2}$ indeed represents the distance between them. However, if these two micro-clusters are both "loose" (*i.e.*, having large extent), then their "true" inter-cluster distance should actually be *less* than $d_{1,2}$. This is because the line segments at the borders of the two micro-clusters are likely to be much closer than $d_{1,2}$. With respect to merging micro-clusters, this allows loose micro-clusters to be more easily merged and vice-versa. The adjustment of the distance function using extent is relatively simple. Whenever possible, extent is used to reduce the distance between the representative line segments of micro-clusters.



(a) Center distance with extent (b) Parallel distance with extent (c) Angle distance with extent

Figure 3.5: Line Segments Distance with Extent

To measure the distance between micro-cluster i and micro-cluster j, it is equivalent to measure the distance $d^*(L_i^*, L_j^*)$ between the representative line segments L_i^* with $extent^i$ and L_j^* with $extent^j$. Figure 3.5 shows an intuitive example of distance measure with extent. For example, in Figure 3.5(a), the distance between the centers is the distance between representative line segments minus the center extents of two micro-clusters. The formal definition is given as follows based on the modification of distance measure between line segments (*i.e.*, Definition 3). To avoid the redundancy in presentation, the symbols explained in Definition 3 are not repeated in Definition 4.

Definition 4 *The distance between L_i^* and L_j^* contains three parts: center distance d^*_{center}, angle* *distance d^*_θ and parallel distance $d^*_\parallel$.*

$$dist(L^*_i, L^*_j) = d_{center}(L^*_i, L^*_j) + d_\theta(L^*_i, L^*_j) + d_\parallel(L^*_i, L^*_j)$$

The center distance:

$$d^*_{center}(L^*_i, L^*_j) = \max\Big(0, \|center_i - center_j\| - extent^i_{center} - extent^j_{center}\Big)$$

The angle distance:

$$\theta^* = \theta - (extent^i_\theta + extent^j_\theta)$$

$$d^*_\theta(L^*_i, L^*_j) = \begin{cases} \| L^*_j \| \times \sin(\theta^*), & 0^o \le \theta^* < 90^o \\ \| L^*_j \|, & 90^o \le \theta^* \le 180^o \end{cases}$$

The parallel distance:

$$d^*_\parallel(L^*_i, L^*_j) = \max\Big(0, \min(l_{\parallel 1}, l_{\parallel 2}) - (extent^i_{length} + \overline{extent^j_{length}})/2\Big),$$

where $\overline{extent^j_{length}}$ is the projection of $extent^j_{length}$ onto L^*_i.

Note that the distances defined between two representative line segments with extent are smaller than those defined between two original ones. And the distance may be equal to zero when there is an overlap between representative line segments with extent.

Merging Algorithm The final algorithm of merging micro-clusters is as follows. Given M micro-clusters, the distance between any two micro-clusters is calculated. They are then sorted from the most similar to the least similar. The most similar pairs are the best candidate for merging since merging them result in the least amount of information loss. They are merged until the number of micro-clusters satisfy the given space constraints.

Chapter 4

Trajectory Macro-Clustering

The last step in the TCMM framework produces the overall trajectory clusters. While micro-clustering is processed with a new batch of data comes in, macro-clustering is evoked *only when* it is called upon by the user.

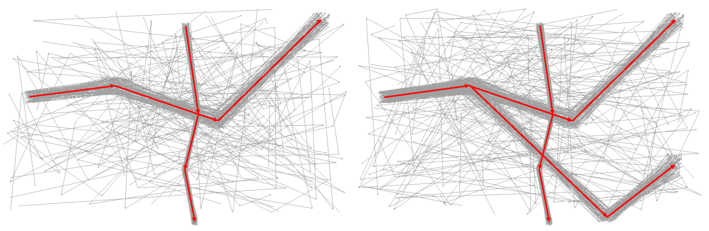
Since the distance between micro-clusters is defined in Definition 4, it is easy to adapt any clustering method on spatial points. We simply need to replace the distancce between spatial points with the distance between micro-clusters. In our framework, we use density-based clustering [7], which is also used in TRACLUS [12]. The clustering technique in macro-clustering step is the same as the clustering algorithm in TRACLUS. The only difference is that macro-clustering in TCMM is performed on the set of micro-clusters rather than the set of trajectory partitions as in TRACLUS. The micro-clusters are clustered through a density-based algorithm which discovers maximally "density-connected" components, each of which forms a macro-cluster.

Chapter 5

Experiments

This section tests the efficiency and effectiveness of the proposed framework under a variety of conditions with different datasets. The TCMM framework and the TRACLUS [12] framework are both implemented using C++ and compiled with gcc. All tests were performed on a Intel 2.4GHz PC with 2GB of RAM.

5.1 Synthetic Data



(a) Micro-clusters at snapshot 1 (b) Micro-clusters at snapshot 2

Figure 5.1: Micro-clusters from synthetic data

As a simple way to quickly test the "accuracy" of TCMM, synthetic trajectory data is generated. Objects are generated to move along pre-determined paths with small perturbations ($< 10\%$ relative distance from pre-determined points). 15% trajectories are random noises added to the data. Figure 5.1 shows the result of incremental micro-clustering at two different snapshots. Figure 5.1(a) shows raw trajectories in gray; one can clearly see the trajectory clusters. The extracted micro-clusters are drawn with red/bold lines; they match the intuitive clusters. Figure 5.1(b) shows the trajectories and extraction results for a later snapshot. Again, they match the intuitive clusters.

5.2 Real Animal Data in Free Space

Next, clusters are computed from deer movement data [1] in Year 1995. This data set contains 32 trajectories with about $20,000$ points in total. The dataset size of animal is considerably small due to the high expense and technological difficulties to track animals. But it is worth studying animal data because the trajectories are in free space rather than on restricted road network. In Section 5.3, a further evaluation on a much larger vehicle dataset containing over $7,000$ trajectories will be conducted.

To the best of our knowledge, there is no any other incremental trajectory clustering algorithm. So the results of TCMM will be compared with TRACLUS [12], which does trajectory clustering over the whole data set. Since micro-clusters in TCMM summarize original line segments information with some information loss, the clustering result on micro-clusters might not be as real as TRACLUS. So the cluster result from TRACLUS is used as a standard to test the accuracy of TCMM. Meanwhile, it is important to show the efficiency against TRACLUS while both results are similar.

We adapt performance measure, sum of square distance (SSQ), from CluStream [1] to test the quality of clustering results. Assume that there are a total of n line segments at the current timestamp. For each line segment L_i, we find the centroid (*i.e.*, representative line segment) C_{L_i} of its closest macro-cluster, and compute $d(L_i, C_{L_i})$ between L_i and C_{L_i}. The SSQ at timestamp is equal to the sum of $d^2(L_i, C_{L_i})$ and the average SSQ is SSQ/n.

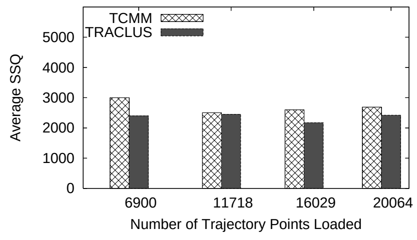


Figure 5.2: Effectiveness Comparison (Deer)

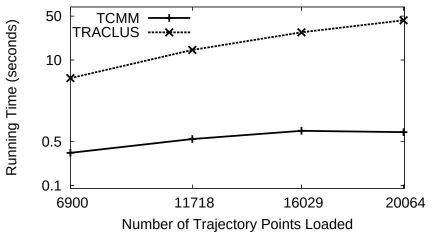


Figure 5.3: Efficiency Comparison (Deer)

As shown in Algorithm 1, there is only one parameter d_{max} in micro-clustering step and we set it to 10. The parameter sensitivity is analyzed and discussed in Section 5.4. For macro-clustering and TRACLUS, they use the same parameters ε and $MinLns$. Here, ε is set to 50 and $MinLns$

[1]http://www.fs.fed.us/pnw/starkey/data/tables/

is set to 8.

Figure 5.2 shows the quality of clustering results. Comparing with TRACLUS, the average SSQ of TCMM is slightly higher. In the worst case, the average SSQ of TCMM is 2% higher than TRACLUS. But the processing time of TCMM is significantly faster than TRACLUS. To process all the $20,000$ points, TCMM only takes 0.7 seconds while TRACLUS takes 43 seconds. The reason is that it is much faster to do clustering over micro-clusters rather than over all the trajectory partitions. With the deer dataset, at last, the number of trajectory partitions (3390) is much more than the number of micro-clusters (324) in total.

5.3 Real Traffic Data in Road Network

Real world GPS recorded data from a taxi company in San Francisco is used to test the performance of TCMM. The data set is huge and keeps growing as time goes by. It contains 7,727 trajectories($100,000$ points) of taxis as they travel around the city picking up and dropping off passengers.

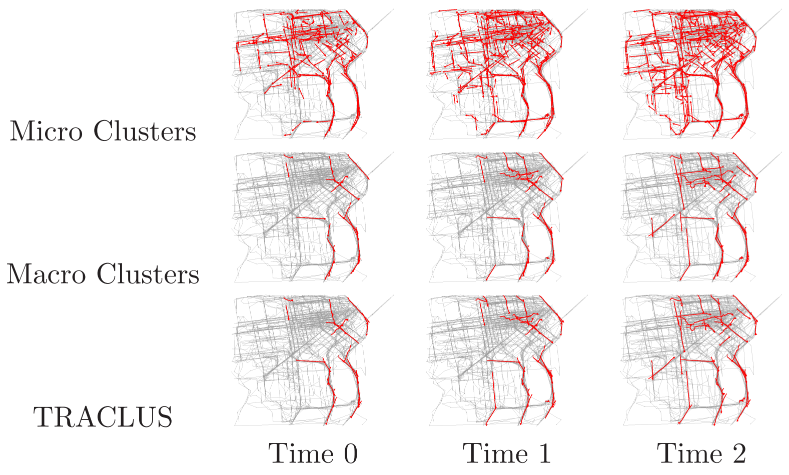


Figure 5.4: Taxi Experiment

Figure 5.4 shows the visual clustering result of taxi data. First row and second row show the micro-clusters (d_{max} set to 800) and macro-clusters (ε set to 50 and $MinLns$ set to 8). Last row shows cluster result from TRACLUS. Time 0, 1, and 2 correspond to the timestamps respectively when 52317, 74896, and 98002 trajectory points have been loaded. As we can see from Figure 5.4, the results from TCMM and TRACLUS are similar except very few differences. The similar

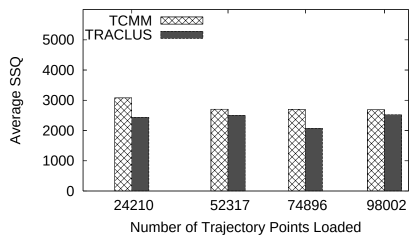


Figure 5.5: Effectiveness Comparison(Taxi)

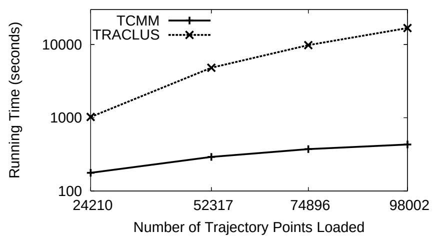


Figure 5.6: Efficiency Comparison(Taxi)

clustering performance is further proved in Figure 5.5, where the average SSQ of TCMM is only slightly higher than that of TRACLUS (2% higher in worst case and 1.4% higher on average).

Regarding to efficiency issue, Figure 5.6 shows the time needed to process the data in 4 increments with TCMM and TRACLUS. Compared to previous data sets, TRACLUS is substantially slower this time due to the larger data set size. To process all the data, TRACLUS takes about 4.6 hours while TCMM only takes about 7 minutes to finish. This is because the number of trajectory partitions (52,600) is much larger than the number of micro-clusters (2,013). It means that TCMM is much more efficient than TRACLUS as data set is getting bigger, while at the same time, the effectiveness remains the same as TRACLUS.

5.4 Parameter Sensitivity

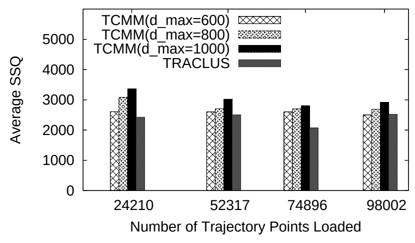


Figure 5.7: Effectiveness with d_{max}

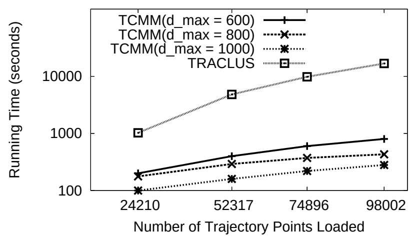


Figure 5.8: Efficiency with d_{max}

The micro-clustering step of TCMM has the nice property that it only requires one parameter: d_{max}. A large d_{max} builds micro-clusters that are large in individual size but small in overall quantity, whereas a small d_{max} has the opposite effect. If we set $d_{max} = 0$, TCMM is actually TRACLUS because each line segment will form a micro-cluster itself. Then the macro-clustering applied on micro-clusters is exactly the one applied on original line segments. Therefore, the smaller the d_{max} is, the better the quality of clustering should be but the longer processing time is needed. At the same time, if we set d_{max} larger, the algorithm runs faster but loses more information in micro-clustering. Hence there is a trade-off between effectiveness and efficiency.

We use taxi datasets to study the parameter sensitivity of our algorithm. Figure 5.7 and Figure 5.8 show the performance of TCMM with different d_{max}. We can see that when $d_{max} = 600$, the average SSQ is closer to that of TRACLUS, which shows that it has more similar performance as TRACLUS. But it also takes longer time to do clustering when $d_{max} = 600$. However, comparing with TRACLUS, the time spent on incremental clustering is still significantly shorter.

Chapter 6

Related Work

Clustering has been studied extensively in machine learning and data mining. A number of approaches have been proposed to process *point* data in various conditions , such as k-means [14], BIRCH [16, 3] and OPTICS [2]. The micro-clustering step in TCMM share the idea of micro-clustering in BIRCH [16]. However, BIRCH [16] cannot handle trajectory clustering. The clustering feature in TCMM has been extended to exactly describe a line-segment cluster by including three kinds of information. The data bubble [3] is an extension of the BIRCH framework and introduces the idea of the extent. TCMM also uses the extent in its micro-cluster, but the definition has been changed to accommodate trajectories.

Trajectory clustering has been studied in various contexts. Gaffney *et al.* [9, 4, 8] proposes several algorithms for model-based trajectory clustering. TRACLUS [12] is a trajectory clustering algorithm which performs density-based clustering over the entire set of sub-trajectories. However, all of these algorithms cannot efficiently handle *incremental* data. They are not suitable for incremental data since clusters are re-calculated from scratch every time.

CluStream [1] studies clustering dynamic data streams. Our method adapts its micro-/macro-clustering framework for trajectory data. However, our method so far handles only incremental data but not trajectory streams. This is because sub-trajectory micro-clustering has to wait for nontrivial number of new points accumulated to form sub-trajectories, which needs addition buffer space and waiting time. Moreover, the processing of sub-trajectories is more expensive and additional processing power is needed for real time stream processing. Thus, the extension of our framework for trajectory streaming left for future research.

Ester *et al.* [6] proposes the Incremental DBSCAN algorithm, which is an extension of DBSCAN for incremental data. Here, the final clusters are directly updated based on new data. We believe our two-step process is more flexible since any clustering algorithm can be employed for macro-clustering, whereas IncrementalDBSCAN is dedicated to DBSCAN. More recently, Sacharidis *et al.* [15] discusses the problem of online discovering hot motion. The basic idea is to delegate part of the path extraction process to objects, by assigning to them adaptive lightweight filters that dynamically suppress unnecessary location updates. Their problem is different from ours in two ways: first, they are trying to find recent hot paths whereas our clusters target at whole time span; and second, they require the objects in a moving cluster to be close enough to each other at any time instant during a sliding window of W time units but we are more from geometric point of view to measure the distance between trajectories.

Chapter 7

Discussion

The TCMM framework is general in the sense that it can accept many modifications to handle different incremental data source. For example, when new batches of trajectory data received during the time interval $[t_{i-1}, t_i]$, *i.e.*, $I_{t_i} - I_{t_{i-1}}$, they are categorized into two types: (i) continuing parts of existing trajectories and (ii) beginning parts of brand-new trajectories. Our framework can handle both types of new data, but these types are not distinguished at first. Thus, the moving objects' identities are not included or required in the input data at this point.

Previously, micro-clusters stored line segments without remembering the corresponding trajectories. It makes the micro-clusters incapable of linking existing trajectories to new ones. However, if the micro-clusters remembered the IDs of the corresponding trajectories, it would be easy to establish the links. The modification is as follows: in addition to the existing features in every micro-cluster, the corresponding trajectory IDs of the line segments are also recorded. In other words, each micro-cluster records which trajectories contribute to its being. During the macro-clustering stage, these IDs are used to conditionally link micro-clusters. If two micro-clusters do not share any IDs, then they will not be connected in the density clustering. It will have the effect of creating new clusters for new trajectory beginnings.

Notice that these modifications comes with a cost. With regards to space, each micro-clusters will have to hold a set of trajectory IDs. With regards to efficiency, set intersection operations will have to be performed during macro-clustering. Both these costs are straightforward but nontrivial. Some techniques such as bitmap index could be implemented to improve the performance.

Chapter 8

Conclusions

In this work, we have proposed the TCMM framework for incremental clustering of trajectory data. It uses a two-step process to handle incremental datasets. The first step maintains a flexible set of micro-clusters that is updated continuously with the input data. Micro-clusters compress the infinite data source to a finite manageable size while still recording much of the trajectory information. The second step, which is on-demand, produces the final macro-clusters of the trajectories using the micro-clusters as input. Compared to previous static approaches, the TCMM framework is much more flexible since it does not require all of the input data at once. The micro-clusters provide a summary of the trajectory data that can be updated easily with any new information. This makes it more suitable for many real world application scenarios.

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB'03*.

[2] M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *SIGMOD'99*.

[3] M. M. Breunig, H.-P. Kriegel, P. Kröger, and J. Sander. Data bubbles: Quality preserving performance boosting for hierarchical clustering. In *SIGMOD'01*.

[4] I. V. Cadez, S. Gaffney, and P. Smyth. A general probabilistic framework for clustering individuals and objects. In *KDD'00*.

[5] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a line or its character. In *The Ameican Cartographer*, 1973.

[6] M. Ester, H. P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in data warehousing environment. In *VLDB'98*.

[7] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *KDD'96*.

[8] S. Gaffney, A. Robertson, P. Smyth, S. Camargo, and M. Ghil. Probabilistic clustering of extratropical cyclones using regression mixture models. In *Technical Report UCI-ICS 06-02*, University of California, Irvine, Jan. 2006.

[9] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *KDD'99*.

[10] M. K. Leung J. Chen and Y. Gao. Noisy logo recognition using line segment hausdorff distance. In *Pattern Recognition*, 2002.

[11] X. Li J.-G. Lee, J. Han and H. Gonzalez. Traclass: Trajectory classification using hierarchical region-based and trajectory-based clustering. In *VLDB'08*.

[12] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: A partition-and-group framework. In *SIGMOD'07*.

[13] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. Trajectory outlier detection: A partition-and-detect framework. In *ICDE'08*.

[14] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proc. 5th Berkeley Symp. Math. Statist, Prob.*, 1:281–297, 1967.

[15] D. Sacharidis, K. Patroumpas, M. Terrovitis, V. Kantere, M. Potamias, K. Mouratidis, and T. Sellis. On-line discovery of hot motion paths. In *EDBT '08*.

[16] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *SIGMOD'96*.