TSCluWin: Trajectory Stream Clustering over Sliding Window

Jiali Mao, Qiuge Song, Cheqing Jin(✉), Zhigang Zhang, and Aoying Zhou

Institute for Data Science and Engineering,
School of Computer Science and Software Engineering,
East China Normal University, Shanghai, China
maojl1231@163.com, {sugar_song,zzg22936}@sina.com,
{cqjin,ayzhou}@sei.ecnu.edu.cn

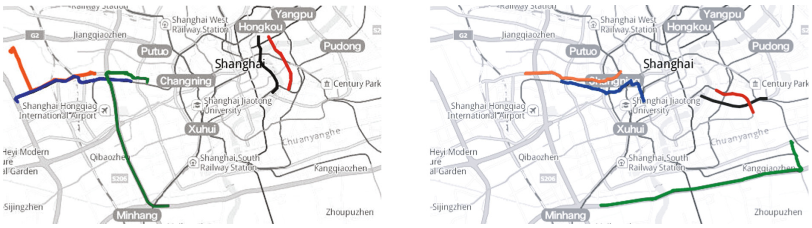
Abstract. The popularity of GPS-embedded devices facilitates online monitoring of moving objects and analyzing movement behaviors in a real-time manner. Trajectory clustering acts as one of the most important trajectory analysis tasks, and the researches in this area have been studied extensively in the recent decade. Due to the rapid arrival rate and evolving feature of stream data, little effort has been devoted to online clustering trajectory data streams. In this paper, we propose a framework that consists of two phases, including a *micro-clustering* phase where a number of micro-clusters represented by compact synopsis data structures are incrementally maintained, and a *macro-clustering* phase where a small number of macro-clusters are generated based on micro-clusters. Experimental results show that our proposal is both effective and efficient to handle streaming trajectories without compromising the quality.

1 Introduction

With the vigorous development of modern mobile devices and location acquisition technologies, the positions of moving objects are collected continuously in a streaming fashion. For instance, the taxis embedded with GPS sensors transmit their current location information to a data center frequently, so that the taxi-company is capable of processing taxi-hailing requests efficiently. Effective analyzing trajectory data stream fosters a broad range of critical applications, such as location-based social network, route recommendation [16], intelligent transportation management [5], road infrastructure optimization, etc.

As an important trajectory analysis task, clustering attempts to group a large amount of trajectories into a few comparatively homogeneous clusters to find the representative paths or common moving trend shared by different objects [6–8, 12, 14, 15, 21]. While trajectory data keep updating rapidly in stream scenarios, it is intrinsically quite difficult to track the cluster changing. For example, Fig. 1(a) illustrates a small example of five taxis' trajectories from 9:00 to 9:30 a.m. There exist two clusters, the left three taxis have the similar driving itinerary, while the rest two behave similarly. But in the next period (from 9:30 to 10:00 a.m.), as shown in Fig. 1(b), the left three trajectories are no longer in one cluster.

S.B. Navathe et al. (Eds.): DASFAA 2016, Part II, LNCS 9643, pp. 133–148, 2016.
DOI: 10.1007/978-3-319-32049-6_9



(a) During (9:00, 9:30) (b) During (9:30, 10:00)

Fig. 1. Movement distribution

Hence, it is imperative to consider a model that focuses on the evolving feature of stream data. The sliding-window model that eliminates the obsolete data and only keeps most recent W tuples in the stream can satisfy this demand.

However, it is challenging to cluster trajectory data streams due to the following reasons. First, a trajectory in the stream may evolve as time progresses. Second, a suitable processing algorithm must be with strict time- and space- complexities since the stream volume is huge and the arrival rate of data is rapid. Finally, as the old tuples progressively expire, the algorithm must be capable of updating the synopsis data structure dynamically. So far, although there exists abundant work on incremental clustering for trajectories, or on data stream clustering over the sliding-window model, to the best of our knowledge, the issue of online clustering trajectory data stream over the sliding-window model has not been addressed yet. This issue is non-trivial since the existing work cannot be adopted to tackle it in a straightforward way. For the work on trajectory incremental clustering [8,14,15], the expired records cannot be discarded swiftly upon the arbitrary arrival of new ones, which influence the quality of clustering result. The work on the data stream clustering over the sliding-window model is upon the scenario where each tuple must be a "*full*" entry [1], whereas each tuple in the trajectory data stream is only a "*part*" of an entry.

We propose a two-phase framework to tackle this issue, including the micro-clustering phase and the macro-clustering phase. During the micro-clustering phase, a number of micro-clusters, represented by a novel synopsis data structure, are maintained incrementally. During the macro-clustering phase, a handful of macro-clusters are built upon the micro-clusters according to a clustering request over the given time horizon. One major novelty of this paper is the construction of two synopsis data structures, called $\underline{\text{T}}$emporal Trajectory Cluster $\underline{\text{F}}$eature (TF) and $\underline{\text{E}}$xponential Histogram of Temporal Trajectory Cluster $\underline{\text{F}}$eature (EF). The contributions of this paper are summarized below.

– We study the issue of online trajectory data stream clustering over the sliding-window model. To the best of our knowledge, there exists no prior work on this topic. Moreover, neither trajectory clustering algorithm nor data stream clustering algorithm upon the sliding-window model can be adopted to tackle this issue directly.

- We propose a two-phase scheme to tackle this issue, with the usage of two novel synopsis data structures (TF and EF) to summarize a cluster of trajectories, and to process the expired tuples.
- We conduct a comprehensive series of experiments on a real dataset to manifest the efficiency and the effectiveness of our proposal, as well as the superiority to other congeneric approach.

The remainder of this paper is organized as follows. In Sect. 2, we review the latest work related to our research. In Sect. 3, the problem is defined formally. In Sect. 4, we outline and analytically study TSCluWin scheme. In Sect. 5, a series of experiments are conducted on a real dataset to evaluate our proposal. Finally, we conclude this article in brief in Sect. 6.

2 Related Work

In this section, we briefly conduct a systematic review over the related work in two relevant areas: data stream clustering and trajectory stream clustering.

Data Stream Clustering. The existing work on data stream clustering are classified into two kinds: one-pass approach and evolving approach. The former constructs clusters by scanning the data stream only once, while the latter can also view the evolving process over time [24]. The work upon the sliding-window model belongs to the latter.

Babcock et al. studied the clustering issue over the sliding-window model with the focus on theoretical bound analysis [3]. Aggarwal et al. developed CluStream algorithm to cluster large evolving data streams based on the pyramid model [1]. Aggarwal et al. further proposed UMicro algorithm to deal with uncertain data streams [2]. Although [1,2] cannot deal with the sliding-window model directly, they can view the evolving process of the data stream. Zhou et al. [24] presented SWClustering algorithm to track the evolution of clusters over the sliding-window model by using a novel synopsis data structure, called EHCF, which combines the exponential histogram with the temporal cluster features to handle the in-cluster evolution and capture the distribution of recent records. Jin et al. proposed the cluUS algorithm to incrementally maintain uncertain tuples based on Uncertain Feature (UF) structure with the combination of the sliding-window model [10]. However, none of the above methods can deal with trajectory data straightforwardly due to different scenarios. In such scenario, each tuple in the data stream is an entry, while each tuple in the trajectory data stream is only a part of an entry.

Trajectory Stream Clustering. A few research work has been conducted for incremental trajectory clustering [8,14,15] in the static trajectory data set. Li et al. proposed the concept of Moving Micro-Cluster to catch the regularities of moving objects, which can accommodate the updates of moving objects and be well suited for handling large datasets [14]. Jensen et al. proposed a scheme for continuously clustering of moving objects, which employs an object dissimilarity across a period of time, and exploits an incrementally maintained clustering feature CF [8]. Li et al. proposed an incremental trajectory clustering framework, called TCMM, which includes micro-clusters maintenance based on the process of simplifying new trajectories into directed line segments, and macro-clusters generating by clustering the micro-clusters [15]. However, during the course of micro-clustering, TCMM has to accumulate a certain amount of new trajectory point data to obtain the simplified sub-trajectory by using MDL method. In addition, due to the effect of obsolete data, after continuously absorbing more records and merging the most similar pair of micro-clusters, the centers of micro-clusters will shift gradually, which more or less degrades the effectiveness of resulting clusters. More specifically, incremental clustering approaches barely consider the temporal aspects of the trajectories and cannot scale up to mine massive trajectory streams. Yu et al. [22] proposed CTraStream for clustering trajectory data stream to extract some patterns similar to convoy pattern [9]. It includes online line segment stream clustering and the update process of closed trajectory clusters based on TC-Tree index.

There also exists some approaches that are to some degree orthogonal to our work but deserve to be mentioned. Various techniques about trajectory simplification and compression have been studied in real-time trajectory tracking [11], but they refer to the problem of minimizing the amount of the position data that are communicated and stored. Due to high computational overhead to attain optimal line simplification, they are not fit to online cluster the rapidly changing stream data in the limited memory. More recent achievements have been reported for continuous query processing over trajectory stream [13,17,19,20,23], but they are unsuitable for identifying the clusters in streaming trajectories. Different from the aforementioned approaches, we focus on online capturing the clustering change in a certain temporal window, and eliminating the effect of obsolete data, which fit the feature of continuous trajectory stream.

3 Problem Statement

We define some notations in this section. Let S denote a stream that contains a totally ordered infinite records of the moving objects in the form of $(o^{(i)}, p^{(i)})$, where $p^{(i)}$ is the location of an object $o^{(i)}$ at the time stamp i in 2-D space (i.e., $p^{(i)} = (x_i, y_i)$). In general, the stream S contains multiple objects. The trajectory of one object is defined below.

Definition 1 (Trajectory). *The trajectory for an object o, denoted as Tr_o, is a sub-sequence of S affiliated to o, denoted as $\{(p_1, t_1), (p_2, t_2), \ldots\}$. Such records arrive in chronological order, i.e., $\forall i < j$, $t_i < t_j$. A line segment L_i refers to a line connecting two temporal adjacent points, i.e., L_i is denoted as (p_i, p_{i+1}). Correspondingly, the trajectory is also denoted as $\{(L_1, t_1), (L_2, t_2), \ldots\}$.*

The goal of trajectory clustering is to divide all trajectories into clusters in terms of their pair-wise distances. Although the Euclidian distance is commonly used in the spacial data management field, it is inappropriate to measure spatial proximity in the scenario with bi-directional property. Hence, we employ the distance measure based on adapted Hausdorff distance [18]. The distance between two trajectories is regarded as the longest path from each line segment to the closest line segment of another trajectory, i.e., $dist(Tr_a, Tr_b) = \max(D(Tr_a, Tr_b), D(Tr_b, Tr_a))$, where $D(Tr_a, Tr_b) = \max_{L' \in Tr_a} \min_{L'' \in Tr_b}(DL(L', L''))$. Note that $DL(L', L'')$ returns the maximal distance between two line segments after alignment, i.e., $DL(L', L'') = \max(dl(L', L''), dl(L'', L'))$, where $dl(L', L'') = \max(\min(||p'_s, p''_s||, ||p'_s, p''_e||), \min(||p'_e, p''_s||, ||p'_e, p''_e||))$. Here, p'_s and p'_e denote the starting and ending positions of L', p''_s and p''_e denote the starting and ending positions of L'', and $||p'_s, p''_s||$ denotes the length of the shortest path between p'_s and p''_s.

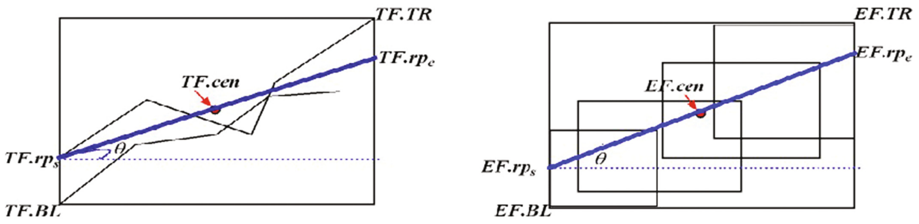
We study the sliding-window model in this paper, where only the most recent W records in S are considered. Due to the infeasibility to keep all trajectories in memory, it is necessary to summarize original data in memory by using a compact synopsis data structure. Our first synopsis data structure, Temporal Trajectory Cluster Feature (TF), summarizes the features of sets of incoming line segments at different intervals.

Definition 2 (Temporal trajectory cluster Feature, (TF)). *Given a set of consecutive line segments $\{L_1, L_2, \ldots, L_n\}$, which is a sub-sequence of a stream S. TF is of the form (SC, SA, BL, TR, n, t).*

- *SC: the linear sum of the line segments' center points;*
- *SA: the linear sum of the line segments' angles;*
- *BL: the bottom left corner of the MBR (Minimal Bounding Rectangle);*
- *TR: the top right corner of the MBR;*
- *n: the number of line segments;*
- *t: the timestamp of the most recent line segment;*

Note that MBR is the minimal bounding rectangle of all line segments contained in a TF. Figure 2(a) illustrates an example of the MBR for two black polyline, we can draw a line segment to represent the moving pattern of all the line segments in TF. Firstly, we obtain the central point (denoted as $TF.cen$) and the angle (denoted as θ) of that line segment by calculating them with $\frac{SC}{n}$ and $\frac{SA}{n}$ respectively. Then, we plot a line across the central point, along that angle, and finally extend it to reach the borders of MBR. The intersection points are treated as the starting and ending points of the representative line segment. For example, the blue line segment with the starting point (denoted as $TF.rp_s$) and the ending point (denoted as $TF.rp_e$) is regarded as the representative line segment in Fig. 2(a).

Property 1 (Additive property). Let $TF(C_1)$ and $TF(C_2)$ denote two TF structures for two sets C_1 and C_2 separately, $C_1 \cap C_2 = \emptyset$. We can construct $TF(C_1 \cup C_2)$ based on $TF(C_1)$ and $TF(C_2)$. The new entries SC, SA and n are equal to the sum of the corresponding entries in $TF(C_1)$ and $TF(C_2)$.



(a) TF.rp (b) EF.rp

Fig. 2. Representative line segment

The new entry t is computed as $\max(TF(C_1).t, TF(C_2).t)$. Moreover, the corners of the new TF can be computed based on two original corners straightforwardly.

As a TF may consist of multiple line segments, and they will go out of the window one by one in the future, which necessitates a structure to deal with the expired line segments. Exponential Histogram (EH) is well-known to deal with the sliding-window model, where all the tuples in the data stream are divided into a number of buckets according to the arrival time [4]. Inspired by EH, we devise a novel structure, called Exponential Histogram of Temporal Trajectory Cluster Feature (EF) below.

Definition 3 (Exponential Histogram of Temporal Trajectory Cluster Feature (EF)). *Given a user-defined parameter ϵ, EF is a collection of TFs on some sets of line segments $C_1, C_2, \ldots$ with the following constraints:*

1. *$\forall i, j$, $i < j$, any line segment in C_i arrives earlier than that in C_j;*
2. *$|C_1| = 1$. $\forall i > 1$, $|C_i| = |C_{i-1}|$ or $|C_i| = 2 \cdot |C_{i-1}|$;*
3. *At most $\lceil \frac{1}{\epsilon} \rceil + 1$ TFs are placed in each level.*

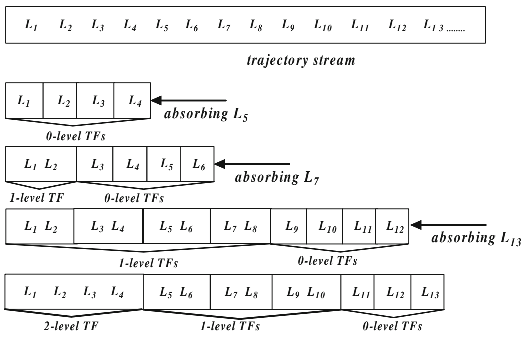


Fig. 3. Process of incorporating line segments into an EF

Theorem 1. *Given an EF that contains n_i tuples, and a user-defined parameter ϵ, the amount of obsolete tuples is within $[0, \epsilon n_i]$, and the number of TFs is at most $(\frac{1}{\epsilon} + 1)(\log(\epsilon n_i + 1) + 1)$.*

Proof. Given an EF of n_i tuples, the parameter ϵ, and the oldest TF of n_s tuples, then $\frac{1}{\epsilon}(1 + 2 + 4 + \ldots + n_s) \leq n_i$. Hence, $n_s \leq \epsilon n_i$ holds. In addition, an EH structure with the window size n_i and the parameter ϵ can be constructed. Each TF maps to a bucket in EH structure. Since EH Structure computes an ϵ-deficient synopsis using at most $(\frac{1}{\epsilon} + 1)(\log(\epsilon W + 1) + 1)$ buckets [4], where W represents the window size, there are at most $(\frac{1}{\epsilon} + 1)(\log(\epsilon n_i + 1) + 1)$ TFs in an EF.

EF is maintained in the following way. When a new line segment is incorporated into the existing EF, a new 0-level TF will be generated for it at first. Once the number of 0-level TFs in EF exceeds the threshold $(\lceil \frac{1}{\epsilon} \rceil + 1)$, the two oldest 0-level TFs are merged to generate a 1-level TF. Note that such merging operation may repeat several times for higher levels. Figure 3 gives an example about how to maintain an EF with $\epsilon = \frac{1}{3}$. It means at most four TFs are kept at each level. When L_5 arrives, a new 0-level TF is generated, which adds to five 0-level TFs. Then, a 1-level TF is generated by merging TF($\{L_1\}$) and TF($\{L_2\}$). Similar operation occurs when L_7 arrives. Moreover, the arrival of L_{13} triggers the merging of TF($\{L_9\}$) and TF($\{L_{10}\}$), and further triggers the merging of TF($\{L_1, L_2\}$) and TF($\{L_3, L_4\}$).

Likewise, we obtain a representative line segment for an EF. The central point (denoted as $EF.cen$) and the angle (denoted as θ) of the representative line segment are the weighted mean of all TFs respectively, i.e., $(\sum_i TF_i.cen \times TF_i.n)/(\sum_i TF_i.n)$ and $(\sum_i TF_i.\theta \times TF_i.n)/(\sum_i TF_i.n)$. The corners of EF can also be computed based on the corners of all TFs involved directly. Figure 2(b) illustrates an example about the MBRs for a group of TFs contained in an EF, and the generated blue representative line segment $(EF.rp_s, EF.rp_e)$.

TF and EF synopsis structure allows us to accurately extract and incrementally maintain the feature of micro-clusters at the different intervals. Meanwhile, such synopsis structures with the combination of sliding-window model can safely eliminate the expired records.

4 General Framework of TSCluWin

In this section, we propose a scheme to cluster trajectory streams over the sliding-window model, called Trajectory Streams Clustering based on sliding Window (TSCluWin). TSCluWin is comprised of two components, including a micro-clustering phase and a macro-clustering phase. During the first phase, appropriate statistical information of the micro-clusters are extracted and maintained incrementally, as shown in Algorithm 1. Note that each micro-cluster is represented by an EF structure (Definition 3), and each bucket in an EF is a TF (Definition 2) that represents the summary statistics of a set of trajectory segments at each interval. During the second phase, a small number of macro-clusters are

Algorithm 1. EFs Generating and Maintenance (abbr.EFGM) $(\epsilon, \gamma, \delta, k, S, W)$

```
 1: $Z \leftarrow \emptyset$;
 2: Initialize $Z$;
 3: for each line segment $L_x$ in $S$ do
 4:    Find the most similar EF $h$ for $L_x$;
 5:    Let $d_1$ and $d_2$ denote the length of $h.rp$ and $L_x$ respectively;
 6:    if $(\exists h, dist(L_x, h.rp)/(d_1 + d_2) \leq \gamma \wedge (h.t - L_x.t < \delta W))$ then
 7:       $Insert(L_x, h, \epsilon)$;
 8:    else
 9:       if $(|Z| = k)$ then
10:          Let $\tau$ denote the average participating records of all EFs in current window;
11:          if $(\exists h_e, (h_e.t \text{ expires}) \vee ((|t_{current} - h_e.t| \geq \frac{W}{2}) \wedge (h_e.n \leq \tau)))$ then
12:             $Z \leftarrow Z - \{h_e\}$;
13:          else
14:             if ((find the most similar EF pair $(h_i, h_j)$) $\wedge$ $(h_i.t - h_j.t < \delta W)$) then
15:                Merge$(h_1, h_2, \epsilon)$;
16:             end if
17:          end if
18:       end if
19:       Create a new EF $h_n$ for $L_x$;
20:       $Z \leftarrow Z \cup \{h_n\}$;
21:    end if
22: end for
23: return $Z$;
```

generated based on EFs maintained in memory by invoking traditional weighted clustering techniques.

4.1 Maintenance of EFs

The goal of micro-clustering phase is to handle new tuples in the stream and discard expired ones. Given the window size W, at any time stamp t_c, only the records in $[t_c - W + 1, t_c]$ are active, while the records arriving before $t_c - W + 1$ are expired. Algorithm 1 shows the main framework to generate and maintain EFs. Let Z represent the set of all generated EFs. Initially, Z is emptied, and subsequently k EFs are generated one after another when continuously receiving k line segments, i.e., we create an EF respectively for each line segment through an initialization process, and regard the line segment itself as the representative line segment of such EF (line 2 in Algorithm 1). At most k EFs can be kept in memory at any time, not the trajectory data per se.

When a new line segment L_x arrives, we attempt to find its nearest EF h in terms of the spatial proximity and temporal closeness. Hence, we set a time tolerance threshold δ $(0 < \delta \leq \frac{1}{2})$ to assess the temporal closeness between L_x and the existing EFs. Only EF with the greatest spatial proximity over recent time period (time span within δW) is regarded as the appropriate EF to absorb L_x.

Algorithm 2. Insert(L_x, h, ϵ)

```
1: Generate $TF_0(\{L_x\})$ for $h$;
2: $h.t \leftarrow L_x.t$;
3: for $l = 0$ to $L$ do
4:    if $(|TF_l| < \lceil \frac{1}{\epsilon} \rceil + 2)$ then
5:       break;
6:    else
7:       Merge two oldest $l$-level TFs into one $(l + 1)$-level TF;
8:    end if
9: end for
10: if (the oldest $L$-level TF($TF_{lst}$) of $h$ expires) then
11:    Drop $TF_{lst}$ from $h$;
12: end if
13: return $h$;
```

Also, we use a distance threshold γ ($\gamma \leq 1$) to assess the spatial proximity between L_x and it's nearest EF. Let $dist(L_x, h.rp)$ denote the distance between L_x and h's representative line segment, d_1 and d_2 denote the length of $h.rp$ and L_x separately. If $dist(L_x, h.rp)/(d_1 + d_2) \leq \gamma$, L_x is absorbed by h and the entries of h are adjusted accordingly based on L_x, the detail procedure as shown in Algorithm 2. Otherwise, a new EF h_n that only contain single line segment L_x will be created on condition that the number of EFs is less than k. When the number of EFs exceeds k, we need to take into account eliminating the expired EFs or merging EFs to make room for the new created EF.

4.2 Elimination of Expired Records and Merging of EFs

To eliminate the adverse effect of expired records, when a line segment L_x is incorporated into its nearest EF h, h must be checked to discard obsolete TFs (line 10 in Algorithm 2). Moreover, when the number of EFs exceeds the given threshold k, we not only remove the expired EFs, but also filter out EFs with the earlier updated time and fewer participating trajectory line segments, which no longer contribute to subsequent clustering. We set τ equal to the average participating records of all EFs in current window, and the least recent updated EFs that participating records are smaller than τ will be discarded (line 11 in Algorithm 1).

If none of the aforementioned eliminating criteria are met, we try to find the most similar EFs pair to merge until the number of EFs meets the space constraints, as shown in Algorithm 3. Similarly, only the nearest EF pair within the time span δW will be merged into a new EF. This is rational since two EFs with the greatest spatial proximity in the most recent period, are more similar in actual situation than that only take spatial proximity into account. The merging process is akin to the process of incorporating line segments into an EF. Once the number of corresponding $l - level$ TFs in two EFs exceeds $\lceil \frac{1}{\epsilon} \rceil + 1$, the oldest $l - level$ TFs need to be merged into a $(l + 1) - level$ TF. Such operation will cascade to level $l = 0, 1, 2, \ldots,$ until all TFs of two EFs are handled.

Algorithm 3. Merge(h_i, h_j, ϵ)

```
1:   for $l = 0$ to $L$ do
2:       if $(|h_i.TF_l| + |h_j.TF_l| > \lceil \frac{1}{\epsilon} \rceil + 1)$ then
3:           Merge two oldest $l - level$ TFs of $h_i$ and $h_j$ into a new $(l+1) - level$ TF of $h_{new}$;
4:       else
5:           $h_{new}.TF_l$ is comprised of $h_i.TF_l$ and $h_j.TF_l$;
6:       end if
7:   end for
8: $Z \leftarrow Z - \{\{h_i\} \cup \{h_j\}\} \cup \{h_{new}\}$;
9: return $Z$;
```

However, the computation overhead of finding the closest EF pair is costly, as a nested loop to calculate and compare the distance between all pairs of EFs is inevitable. The cost of such step is $O(k^2)$. Due to the evolving feature of data and the high update cost, Tree-based indexes cannot be adapted to trajectory streams. We opt for a strategy to speed up this process. For each EF h_i, we maintain its closest EF h_{c_i} and the minimal distance d_s between h_i and h_{c_i}. When receiving a new line segment L_x, we attempt to search the closest EF h_{n_1} and the second closest EF h_{n_2} for it. Meanwhile, the distance between h_{n_1} and h_{n_2} (denoted as $dist(h_{n_1}.rp, h_{n_2}.rp)$) is computed and compared with d_s of h_{n_1}, if $dist(h_{n_1}.rp, h_{n_2}.rp) \leq d_s$, h_{n_1}'s original closest EF is replaced with h_{n_2}. If the closest EF (h_{n_1}) of L_x cannot satisfy the defined spatial proximity and temporal closeness, a new EF h_n that only containing L_x will be created, and h_{n_1} is intuitively regarded as the closest EF for h_n. Only when the closest EF h_{c_i} of an EF h_i is eliminated or merged into the other EF, we need to search the nearest EF for h_i by the distance calculation between EFs. In this way, when searching the closest EF pair to merge, we simply need to traverse the closest EF h_{c_i} of all EF h_i, to find the EF pair with the shortest distance. As a result, the cost of searching the closest EF pair to merge is reduced to $O(k)$.

4.3 Macro-Cluster Creation

Given a time horizon of length len and the current time t_c, we can draw curves to reveal the relation between the amount of trajectory segments that have been absorbed in micro-clusters and the interval from current time t_c to $t_c - len$. Through curve graph, the analysts can better differentiate the micro-clusters to further discover the meaningful characteristic of micro-cluster.

Additionally, given the clustering request over specific time horizon, we can further explore macro-clusters based on the previously generated micro-clusters. As mentioned above, the micro-clusters are represented by the representative trajectory line segments of EFs. All EFs in the user-specific time interval $[t_c - len, t_c]$ are treated as pseudo line segments, and re-clustered to produce macro-clusters by using a variant of DBScan algorithm [12,15]. If the user-defined time horizon exceeds the current window size, we will find the most recent EFs that

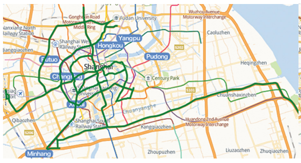


Fig. 4. Movement distribution of taxis (Color figure online)

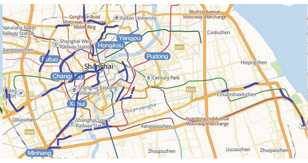


Fig. 5. Micro-clustering results (Color figure online)

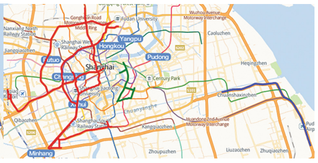


Fig. 6. Macro-clustering results (Color figure online)

maintained in main memory, and the historical EFs that stored on disk within the given time horizon. Subsequently, the representative trajectory line segments of such EFs can be offline re-clustered to generate the macro-clusters by using DBScan algorithm.

4.4 Performance Analysis

The space complexity of TSCluWin is as follows. Given the error parameter ϵ, the maximal number of EF k, the window size W, and the number of line segments absorbed in the i-th micro-cluster n_i, then $\sum_{i=1}^{k} n_i = W$, and the number of obsolete records is within $[0, \epsilon W]$. There are at most $(\frac{1}{\epsilon}+1)(\log(\epsilon n_i+1)+1)$ TFs in an EF. The total number of TFs in k EFs is at most $k(\frac{1}{\epsilon}+1)(\log(\epsilon n_i+1)+1)$. In addition, the number of TFs required by merging two EFs is $(\frac{1}{\epsilon}+1)(\log(\epsilon(n_i+n_j))+1)$. As a consequence, the total required memory (the total number of TFs) is $O(\frac{k}{\epsilon}\log(\epsilon\lceil\frac{W}{k}\rceil))$.

Concerning time complexity, the cost of incorporating a new line segment L_x into the nearest EF involves lines 4, 11 and 14 in Algorithm 1. The cost of line 4 (finding the closest EF for L_x) is simply $O(k)$. At line 11, when the number of EF exceeds k, the cost of removing obsolete EF is $O(k)$. At line 14, the cost of computing the shortest distance between EFs and merging a pair of EFs is $O(k)$ (as illustrated in Sect. 4.2). Consequently, the per-record processing cost is $O(k)$, the total processing cost of dealing with W records is $O(kW)$.

5 Experiments

In this section, we conduct extensive experiments to evaluate the clustering performance and efficiency of our proposed method by comparing against TCMM algorithm (the work most similar to our proposal) on a real life dataset. Though TCMM employs the micro- and macro-clustering framework to cluster trajectory data incrementally, it doesn't take the temporal aspects of the trajectories into account and cannot fit for clustering trajectory stream. All codes written in Java, are conducted on a PC with Intel Core CPU 3.1 GHz and 8.00 GB RAM. The operating system is Windows 8.1. In the experiments, TSCluWin maintains the same number of micro-clusters as that of TCMM. Unless mentioned otherwise, the parameters are set below, $\epsilon = 0.5$, $\gamma = 0.75$, $\delta = \frac{1}{2}$, and $d_{max} = 800$.

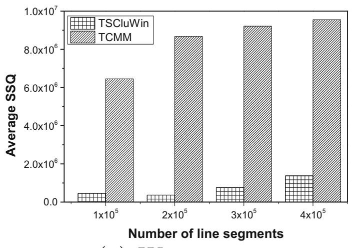
5.1 Dataset

We use a real-life dataset about the trajectories of the taxis in Shanghai, China. This dataset contains the GPS logs of about 30000 taxis during three months (October, November, December) in 2013, covering 93 % main road network of Shanghai. Figure 4 shows the movement distribution of partial taxis (in green) during the interval (from 10:30 to 11:00 a.m.) on October 7th. After micro-clustering phase, 33 micro-clusters are extracted (in blue). As shown in Fig. 5, they capture most traffic flows in Fig. 4. Given the interval (from 10:40 to 11:00 a.m.) on October 7th, 33 micro-clusters are grouped into 3 macro-clusters (in red, green, blue respectively) after clustering by DBScan algorithm, as shown in Fig. 6.

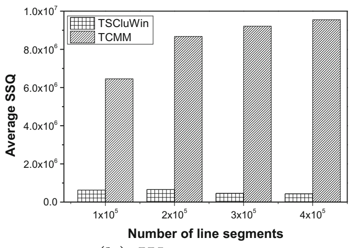
5.2 Effectiveness

We quantify the clustering effect according to the sum of square distance (SSQ). For each cluster C_i, we compute the sum of square distance between any two line segments' center points (denoted as $L_j^i.cen$, $0 < j \leq n_i$) in C_i and the centroid (denoted as Cen_i) of C_i. Therefore, SSQ is calculated with $\sum_{i=1}^{K} dist^2(L_j^i.cen, Cen_i)$, where K denotes the number of clusters (micro-clusters or macro-clusters). Generally, a small SSQ value means the high-quality clustering result.

At first, we report the performance upon different window sizes. To improve credibility, both algorithms are executed for ten times and the average SSQs are reported. Figure 7 shows the average SSQ obtained by both algorithms as time progresses. We observe that TSCluWin (the left bar) always behaves better than TCMM (the right bar) when the window size is set to 160,000 and 330,000 respectively. Since the obsolete records are promptly eliminated by TSCluWin, the micro-clusters are maintained relatively compact with fewer records whenever the cluster center drifts. Conversely, since TCMM does not consider eliminating the influence of the expired records, a micro-cluster may continuously



(a) $W = 160,000$



(b) $W = 330,000$

Fig. 7. Quality comparison

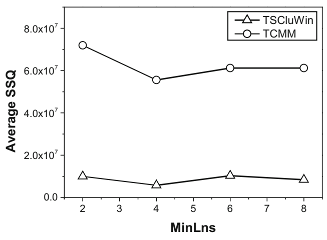


Fig. 8. Average SSQ versus *MinLns*

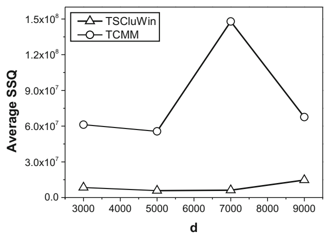


Fig. 9. Average SSQ versus d

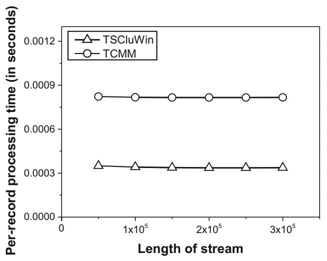


Fig. 10. Execution time versus length of stream

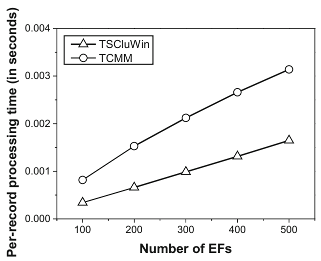


Fig. 11. Execution time versus number of EFs

increase on the boundary rather than be split into multiple small micro-clusters. Additionally, the gap of SSQ between two algorithms varies since the underlying distribution of the positional stream data is always changing.

In order to verify the robustness of TSCluWin in the presence of uncertainty, we proceed to study the effect of input parameters ($MinLns$, d) on the macro-clustering results. Figures 8 and 9 show the average SSQ obtained by TSCluWin and TCMM when varying the values of $MinLns$ and d respectively. Note that the same parameters $MinLns$ and d are used by TSCluWin and TCMM. According to Figs. 8 and 9, TSCluWin is superior to TCMM in all situations, and both algorithms attain the best quality when $MinLns = 4$ and $d = 5,000$.

5.3 Execution Time

Figure 10 shows the execution time of TSCluWin and TCMM. The per-record processing time of TSCluWin fluctuates smoothly, and keeps superior to that processed by TCMM with the progression of trajectory stream. The quicker implementation of TSCluWin is due to that micro-clustering is executed on the original trajectory data (without disregard any trajectory point). While TCMM needs to partition trajectories by using MDL method before micro-clustering phase, which consumes additional waiting time of partitioning accumulated trajectory data into sub-trajectories. Figure 11 shows the per-record processing time when varying the amount of micro-clusters. Both approaches scale linearly with the amount of micro-clusters, since the distance computation cost of finding

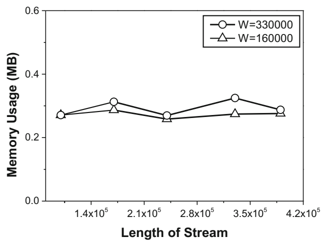


Fig. 12. Memory usage versus length of stream

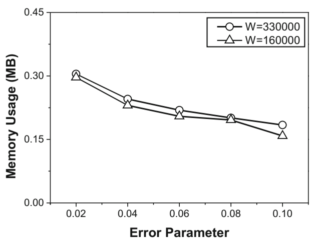


Fig. 13. Memory usage versus parameter ϵ

the most similar micro-cluster for incoming line segment keeps growing as the number of micro-clusters increases.

5.4 Memory Usage

Figure 12 shows the memory footprint (in MBytes) of TSCluWin when the window size W is set to 160,000 and 330,000 respectively. We can see that the memory usage fluctuates with the progression of the trajectory stream. For $W = 330,000$, when the number of incoming records exceed 160,000, the memory usage of TSCluWin approach decreases at first and then gradually rises with the new records. The same change trend happens when the number of incoming records exceed 330,000. The main reason is that the number of TFs drops along with eliminating of the expired records in terms of two erasure criteria (as illustrated in Sect. 4.2). Similarly, when $W = 160,000$, the same finding can be observed when the amount of incoming records exceeds 160,000.

Figure 13 shows the memory usage of TSCluWin by tuning the value of parameter ϵ. We can observe that a larger window size enables more TFs stored in memory and thus leads to the larger memory footprint. In addition, when the value of ϵ increases from 0.02 to 0.1, the memory usage decreases significantly. It is due to that ϵ decides the amount of expired records within the sliding window. In the current window, with the increase of ϵ, more obsolete records are eliminated, and fewer TFs are stored in memory.

6 Conclusion

In this paper, we propose an efficient method to cluster evolving trajectory streams over the sliding-window model, called TSCluWin. It consists of two components, a micro-clustering component that extracts the summary of trajectory stream in the window, and a macro-clustering component that re-clusters the previously extracted summaries according to user's request. Specifically, We define two synopsis data structures (EF and TF) to maintain the most recent cluster changes of trajectory stream in memory. We validate our proposal against

TCMM algorithm for effectiveness and efficiency by conducting extensive experiments on a real dataset, and show that our proposal is efficient in coping with trajectory stream and outperforms the baseline approach.

Acknowledgement. Our research is supported by the 973 program of China (No. 2012CB316203), NSFC (U1501252, U1401256, 61370101 and 61402180), Shanghai Knowledge Service Platform Project (No. ZF1213), Innovation Program of Shanghai Municipal Education Commission(14ZZ045), and Natural Science Foundation of ShanghaiNo. 14ZR1412600).

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: VLDB, pp. 81–92 (2003)
2. Aggarwal, C.C., Yu, P.S.: A framework for clustering uncertain data streams. In: ICDE, pp. 150–159 (2008)
3. Babcock, B., Datar, M., Motwani, R., Callaghan, L.: Maintaining variance and k-medians over data stream windows. In: PODS, pp. 234–243 (2003)
4. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. In: SODA, pp. 635–644 (2002)
5. Duan, X., Jin, C., Wang, X., Zhou, A., Yue, K.: Real-time personalized taxi-sharing. In: DASFAA (2016)
6. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. KDD **96**, 226–231 (1996)
7. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: ACM SIGKDD, pp. 63–72. ACM (1999)
8. Jensen, C.S., Lin, D., Ooi, B.C.: Continuous clustering of moving objects. IEEE TKDE **19**(9), 1161–1174 (2007)
9. Jeung, H., Yiu, M.L., Zhou, X., Jensen, C.S., Shen, H.T.: Discovery of convoys in trajectory databases. PVLDB **1**(1), 1068–1080 (2008)
10. Jin, C., Yu, J.X., Zhou, A., Cao, F.: Efficient clustering of uncertain data streams. Knowl. Inf. Syst. **40**(3), 509–539 (2014)
11. Lange, R., Dürr, F., Rothermel, K.: Efficient real-time trajectory tracking. VLDB J. **20**(5), 671–694 (2011)
12. Lee, J., Han, J., Whang, K.: Trajectory clustering: a partition-and-group framework. In: ACM SIGMOD, pp. 593–604. ACM (2007)
13. Li, X., Ceikute, V., Jensen, C.S., Tan, K.: Effective online group discovery in trajectory databases. IEEE TKDE **25**(12), 2752–2766 (2013)
14. Li, Y., Han, J., Yang, J.: Clustering moving objects. In: ACM SIGKDD, pp. 617–622 (2004)
15. Li, Z., Lee, J.-G., Li, X., Han, J.: Incremental clustering for trajectories. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5982, pp. 32–46. Springer, Heidelberg (2010)
16. Liu, H., Jin, C., Zhou, A.: Popular route planning with travel cost estimation. In: DASFAA (2016)
17. Nehme, R.V., Rundensteiner, E.A.: SCUBA: scalable cluster-based algorithm for evaluating continuous spatio-temporal queries on moving objects. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 1001–1019. Springer, Heidelberg (2006)

18. Roh, G.-P., Hwang, S.: NNCluster: an efficient clustering algorithm for road network trajectories. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5982, pp. 47–61. Springer, Heidelberg (2010)
19. Sacharidis, D., Patroumpas, K., Terrovitis, M., Kantere, V., Potamias, M., Mouratidis, K., Sellis, T.K.: On-line discovery of hot motion paths. In: EDBT, pp. 392–403(2008)
20. Tang, L.A., Zheng, Y., Yuan, J., Han, J., Leung, A., Hung, C., Peng, W.: On discovery of traveling companions from streaming trajectories. In: ICDE, pp. 186–197 (2012)
21. Wang, W., Yang, J., Muntz, R.R.: STING: a statistical information grid approach to spatial data mining. VLDB **97**, 186–195 (1997)
22. Yu, Y., Wang, Q., Wang, X., Wang, H., He, J.: Online clustering for trajectory data stream of moving objects. Comput. Sci. Inf. Syst. **10**(3), 1293–1317 (2013)
23. Zheng, K., Zheng, Y., Yuan, N.J., Shang, S.: On discovery of gathering patterns from trajectories. In: ICDE, pp. 242–253 (2013)
24. Zhou, A., Cao, F., Qian, W., Jin, C.: Tracking clusters in evolving data streams over sliding windows. Knowl. Inf. Syst. **15**(2), 181–214 (2008)